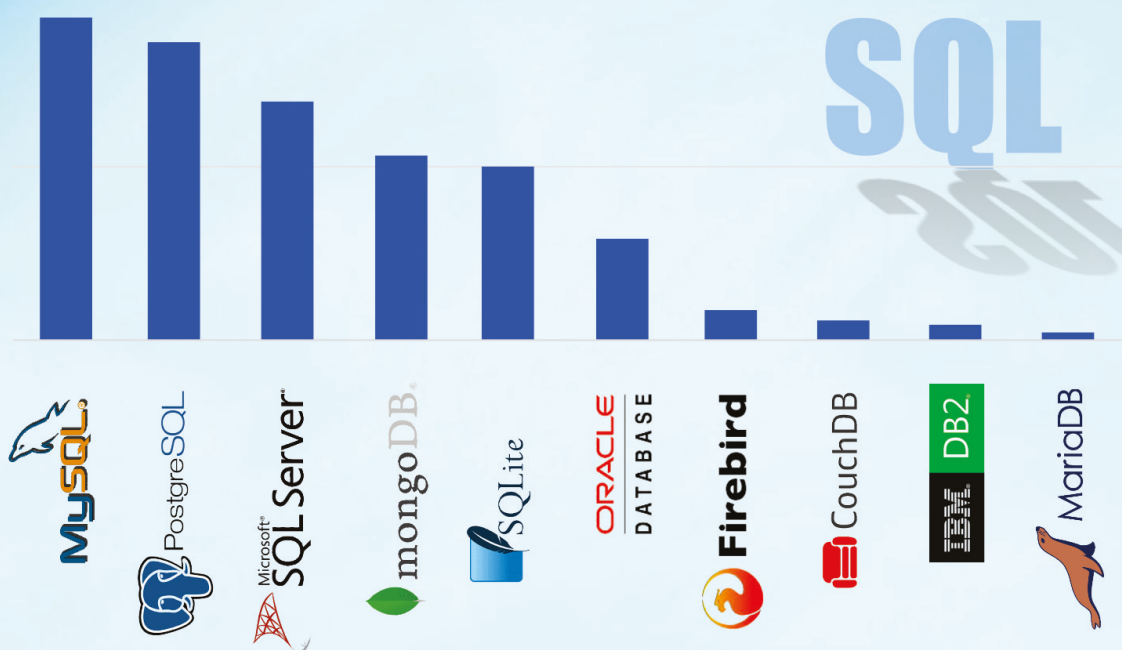


Я.І. Соколовський, М.В. Дендюк,  
І.М. Крошний, І.Б. Пірко, М.М. Паславський

# ОРГАНІЗАЦІЯ БАЗ ДАНИХ



Навчальний посібник

Jaroslav Sokolovskyy, Mykhailo Dendiuk,  
Ihor Kroshnnyy, Ihor Pirko, Mykhailo Paslavskyy

# ORGANIZACJA BAZ DANYCH

Podręcznik akademicki

Podręcznik jest efektem współpracy Gdańskiej Szkoły Wyższej z uczelniami z Ukrainy w ramach realizowanych umów o współpracy naukowej.

„Organizacja baz danych” może być przydatna dla studentów Zarządzania i inżynierii produkcji oraz Administracji w ramach realizacji zajęć dotyczących zarówno informatyki, jak i modułów związanych z cyberbezpieczeństwem.

Podręcznik do samodzielnej nauki omawia teoretyczne aspekty baz danych zarówno klasycznych relacyjnych SQL, jak i nierelacyjnych technologii NoSQL, OLTP i OLAP, w tym zasady pracy z dużymi zbiorami danych. Dużą uwagę poświęcono administrowaniu i ochronie baz danych, podano typowe sytuacje wyjątkowe w MySQL oraz sposoby ich eliminacji. Podręcznik może być wykorzystany do sprawdzenia stanu wiedzy, nauczania praktyk laboratoryjnych, projektowania, kursów. Przeznaczony jest dla studentów szkół wyższych na kierunku Technologie informacyjne, w szczególności specjalnościach: Informatyka, Systemy i technologie informacyjne oraz Inżynieria oprogramowania. Podręcznik przyda się tym, którzy chcą samodzielnie opanować podstawy teoretyczne i umiejętności tworzenia i utrzymywania baz danych.



Ярослав Соколовський, Михайло Дендюк,  
Ігор Крошній, Ігор Пірко, Михайло Паславський

# ОРГАНІЗАЦІЯ БАЗ ДАНИХ

Навчальний посібник



Гданськ 2023

Рекомендовано Вченою радою Національного лісотехнічного університету України протокол № 3 від 30 березня 2023 року

#### Рецензенти

Дияк Іван Іванович

доктор технічних наук, професор, декан факультету прикладної математики та інформатики Львівського національного університету імені Івана Франка

Терещенко Василь Миколайович

доктор технічних наук, професор, завідувач кафедри математичної інформатики Київського національного університету імені Тараса Шевченка

Процах Наталія Петрівна

доктор технічних наук, професор, завідувач кафедри математики і фізики Національного лісотехнічного університету України

Соколовський Я.І., Дендюк М.В., Крошний І.М., Пірко І.Б., Паславський М.М. Організація баз даних: навчальний посібник. / Я.І. Соколовський, М.В. Дендюк, І.М. Крошний, І.Б.Пірко, М.М. Паславський – видавництво Wydawnictwo GSW, 2023. – 466 с.

У навчальному посібнику розглянуто теоретичні аспекти баз даних, як класичних реляційних SQL, так і нереляційних NoSQL, розглянуто технології OLTP та OLAP, в тому числі принципи роботи з великими даними. Значна увага приділяється адмініструванню та захисту баз даних, наведено характерні виняткові ситуації в MySQL та шляхи їх усунення. Посібник може бути використаний для розроблення баз даних MySQL та MongoDB, контролю знань, викладання лабораторних практикумів, курсового та дипломного проектування. Для студентів закладів вищої освіти в галузі знань „Інформаційні технології”, зокрема спеціальностей „Комп’ютерні науки”, „Інформаційні системи та технології” та „Інженерія програмного забезпечення”. Посібник буде корисним бажаним самостійно оволодіти теоретичними основами та навиками розроблення і супроводу баз даних.

Іл. 304, Табл. 10, Бібліографії 39

Jaroslav Sokolovskyy, Mykhailo Dendiuk, Ihor Kroshnyy, Ihor Pirko, Mykhailo Paslavskyy, **Organizacja baz danych. Podręcznik akademicki**

Projekt okładki: Tatiana Samotij

Дизайн обкладинки: Тетяна Самотій

Redakcja i skład tekstu: autorzy

Редагування та верстка: автори

Wydanie pierwsze, Gdańsk 2023

Перше видання, Гданськ (Польща) 2023

Copyright by

© Я.І. Соколовський, М.В. Дендюк, І.М. Крошний, І.Б. Пірко, М.М. Паславський, 2023,

© НАТУ України, 2023,

© Gdańska Szkoła Wyższa, 2023



Wydawnictwo GSW

Gdańska Szkoła Wyższa  
80-875 Gdańsk, ul. Biskupia 24B

☎ (+48) 58 305 08 11, 12  
wydawnictwo@gsw.gda.pl

[www.gsw.gda.pl/wydawnictwo](http://www.gsw.gda.pl/wydawnictwo)

ISBN online 978-83-66270-29-9 Nr kat. [129]

[www.gsw.gda.pl/wydawnictwo/organizacya-baz\\_129](http://www.gsw.gda.pl/wydawnictwo/organizacya-baz_129)





# ЗМІСТ

|                                                                          |           |
|--------------------------------------------------------------------------|-----------|
| <b>Передмова.....</b>                                                    | <b>15</b> |
| <b>Розділ 1. ТЕОРЕТИЧНІ ОСНОВИ БАЗ ДАНИХ .....</b>                       | <b>17</b> |
| <b>Тема 1. Структура баз даних.....</b>                                  | <b>17</b> |
| 1.1. Основні поняття та визначення .....                                 | 17        |
| 1.2. Системи керування базами даних .....                                | 18        |
| 1.3. Загальна архітектура баз даних .....                                | 20        |
| 1.4. Класифікація баз даних .....                                        | 21        |
| 1.4.1. Бази даних за типом доступу (локальні, клієнт-серверні) .....     | 21        |
| 1.4.2. Бази даних за типом керування (реляційні, NoSQL, NewSQL) .....    | 24        |
| 1.5. Тенденції використання різних типів баз даних.....                  | 28        |
| Контрольні запитання .....                                               | 29        |
| <b>Тема 2. Моделі баз даних і моделі даних .....</b>                     | <b>31</b> |
| 2.1. Поняття про моделі баз даних .....                                  | 31        |
| 2.1.1. Міфологічні (концептуальні) моделі .....                          | 32        |
| 2.1.2. Даталогічні (логічні) моделі .....                                | 33        |
| 2.1.3. Фізичні моделі .....                                              | 33        |
| 2.1.4. Взаємозв'язок моделей даних .....                                 | 34        |
| 2.2. Типи моделей даних .....                                            | 35        |
| 2.2.1. Файлова модель .....                                              | 36        |
| 2.2.2. Ієрархічна модель .....                                           | 38        |
| 2.2.3. Мережна модель .....                                              | 42        |
| 2.2.4. Реляційна модель .....                                            | 44        |
| 2.2.5. Постреляційна модель.....                                         | 47        |
| 2.2.6. Багатовимірна модель. OLAP технологія .....                       | 49        |
| 2.2.7. Об'єктно-орієнтована модель .....                                 | 56        |
| 2.2.8. Об'єктно-реляційна модель даних.....                              | 60        |
| 2.3. Проблеми маніпулювання даними та обмеження цілісності<br>даних..... | 61        |
| Контрольні запитання .....                                               | 63        |
| <b>Тема 3. Реляційна модель баз даних .....</b>                          | <b>65</b> |
| 3.1. Обмеження цілісності даних.....                                     | 65        |
| 3.2. Реляційна алгебра.....                                              | 67        |
| 3.2.1. Традиційні операції над множинами.....                            | 68        |
| 3.2.2. Спеціальні реляційні операції .....                               | 70        |
| 3.3. Нормалізація реляційних баз даних .....                             | 75        |
| 3.3.1. Перша нормальна форма .....                                       | 77        |

|                                                              |    |
|--------------------------------------------------------------|----|
| 3.3.2. Друга нормальна форма .....                           | 79 |
| 3.3.3. Третя нормальна форма .....                           | 81 |
| 3.3.4. Нормальна форма Бойса-Кодда.....                      | 82 |
| 3.3.5. Четверта нормальна форма .....                        | 85 |
| 3.3.6. П'ята нормальна форма.....                            | 86 |
| 3.3.7. Доменно-ключова нормальна форма. Денормалізація ..... | 89 |
| 3.3.8. Переваги та недоліки нормалізації .....               | 90 |
| Контрольні запитання.....                                    | 92 |

## **Тема 4. Етапи проектування баз даних.....93**

|                                                                              |     |
|------------------------------------------------------------------------------|-----|
| 4.1. Етап системного аналізу і словесного опису .....                        | 93  |
| 4.2. Етап концептуального (інфологічного) проектування. ER-<br>діаграми..... | 94  |
| 4.2.1. ER-діаграма в нотації Чена .....                                      | 99  |
| 4.2.2. ER-діаграми в нотації Баркера і Мартіна (Information Engineering).... | 103 |
| 4.2.3. Проектування баз даних з використанням методології IDEF1X .....       | 105 |
| 4.2.4. Нотація бази даних UML .....                                          | 109 |
| 4.2.5. CASE-засоби.....                                                      | 112 |
| 4.3. Основи роботи з MySQL Workbench .....                                   | 117 |
| 4.3.1. Проектування моделі бази даних .....                                  | 120 |
| 4.3.2. Підключення MySQL Workbench до MySQL-сервера .....                    | 131 |
| 4.3.3. Імпорт схеми даних з MySQL WorkBench на MySQL-сервер .....            | 134 |
| Контрольні запитання .....                                                   | 138 |

## **Розділ 2. РЕЛЯЦІЙНІ БАЗИ ДАНИХ MySQL .....140**

### **Тема 5. Основи мови SQL .....140**

|                                                  |     |
|--------------------------------------------------|-----|
| 5.1. Роль і місце мови SQL .....                 | 140 |
| 5.2. Основні поняття мови SQL .....              | 141 |
| 5.7.1. Склад мови SQL .....                      | 141 |
| 5.7.2. Оператори мови SQL .....                  | 142 |
| 5.7.3. Типи даних мови SQL .....                 | 143 |
| 5.3. Робота з доменами (Domain).....             | 146 |
| 5.4. Команда створення таблиць.....              | 147 |
| 5.5. Команди модифікації структури таблиці ..... | 152 |
| 5.6. Команда вилучення таблиці.....              | 156 |
| 5.7. Команди модифікації вмісту таблиць .....    | 156 |
| 5.7.1. Команда додавання записів .....           | 156 |
| 5.7.2. Команда редагування записів .....         | 159 |
| 5.7.3. Команда видалення записів .....           | 160 |
| 5.8. Використання індексів.....                  | 161 |
| Контрольні запитання .....                       | 162 |

### **Тема 6. Команда вибірки SELECT.....164**

|                                                                                         |     |
|-----------------------------------------------------------------------------------------|-----|
| 6.1. Формат запису команди SELECT .....                                                 | 164 |
| 6.2. Використання запиту SELECT для вибору записів з однієї таблиці .....               | 165 |
| 6.2.1. Використання оператора AS .....                                                  | 165 |
| 6.2.2. Задавання умови WHERE .....                                                      | 166 |
| 6.2.3. Порівняння за шаблоном LIKE та REGEXP .....                                      | 167 |
| 6.2.4. Використання оператора IN для задавання списку можливих значень .....            | 170 |
| 6.2.5. Задавання діапазону значень оператором BETWEEN .....                             | 170 |
| 6.2.6. Використання оператора IS NULL .....                                             | 171 |
| 6.2.7. Об'єднання декількох записів в результуючий за допомогою групових операцій ..... | 171 |
| 6.2.8. Сортування записів оператором ORDER BY .....                                     | 174 |
| 6.2.9. Отримання діапазону записів оператором LIMIT .....                               | 174 |
| 6.3. Використання SELECT для вибору записів з декількох таблиць ....                    | 176 |
| 6.3.1. Об'єднання внутрішнє .....                                                       | 177 |
| 6.3.2. Об'єднання зовнішнє .....                                                        | 178 |
| 6.3.3. Декартовий добуток CROSS JOIN .....                                              | 180 |
| 6.3.4. Віднімання таблиць .....                                                         | 181 |
| 6.3.5. Додавання таблиць .....                                                          | 182 |
| 6.3.6. Самооб'єднання таблиць .....                                                     | 184 |
| 6.4. Використання запитів SELECT ... INTO .....                                         | 186 |
| 6.5. Використання запитів SELECT в інших запитах .....                                  | 189 |
| 6.5.1. Створення таблиці на основі існуючої .....                                       | 189 |
| 6.5.2. Створення таблиці на основі існуючої за умовою .....                             | 190 |
| 6.5.3. Створення гібридних таблиць .....                                                | 190 |
| 6.5.4. Створення тимчасових таблиць .....                                               | 191 |
| 6.5.5. Створення тимчасових таблиць на основі існуючих .....                            | 191 |
| 6.5.6. Задавання первинних ключів у створених таблицях на основі існуючих .....         | 191 |
| 6.5.7. Додавання записів з іншої таблиці .....                                          | 192 |
| Контрольні запитання .....                                                              | 193 |

## **Тема 7. SQL функції..... 195**

|                                                |     |
|------------------------------------------------|-----|
| 7.1. Оператори загального призначення .....    | 195 |
| 7.1.1. Змінні в MySQL .....                    | 195 |
| 7.1.2. Значення NULL та операції над ним ..... | 196 |
| 7.1.3. Оператори керування в запитах .....     | 198 |
| 7.2. Агрегатні функції .....                   | 201 |
| 7.3. Числові функції .....                     | 203 |
| 7.3.1. Математичні функції .....               | 203 |
| 7.3.2. Функції дати і часу .....               | 205 |
| 7.4. Функції роботи з рядками .....            | 210 |

|                                                    |     |
|----------------------------------------------------|-----|
| 7.4.1. Функції визначення властивостей рядків..... | 210 |
| 7.4.2. Функції перетворення у рядках.....          | 212 |
| 7.4.3. Пошук та видобування частини рядка.....     | 218 |
| 7.4.4. Функції порівняння рядків.....              | 222 |
| 7.4.5. Чутливість до регістру .....                | 224 |
| 7.5. Системні функції.....                         | 225 |
| Контрольні запитання.....                          | 226 |

## **Тема 8. Використання підзапитів в MySQL .....228**

|                                                                 |     |
|-----------------------------------------------------------------|-----|
| 8.1. Визначення підзапитів.....                                 | 228 |
| 8.2. Використання підзапитів в якості розрахункових полів ..... | 229 |
| 8.3. Підзапити в реченні FROM.....                              | 230 |
| 8.4. Вкладений запит в умові WHERE.....                         | 230 |
| 8.5. Вкладений запит в умові HAVING .....                       | 231 |
| 8.6. Оператор EXIST у вкладених запитах .....                   | 232 |
| 8.7. Підзапити з операторами IN, ANY, ALL.....                  | 232 |
| 8.8. Підзапити в операторах маніпулювання даними .....          | 233 |
| 8.8.1. Підзапити в операторі DELETE .....                       | 233 |
| 8.8.2. Підзапити в операторі UPDATE .....                       | 234 |
| 8.8.3. Підзапити в операторі INSERT .....                       | 235 |
| 8.9. Корельовані підзапити.....                                 | 237 |
| Контрольні запитання.....                                       | 238 |

## **Розділ 3. OLTP ТА OLAP СИСТЕМИ .....240**

### **Тема 9. Збережені підпрограми. OLTP технологія .....240**

|                                                                    |     |
|--------------------------------------------------------------------|-----|
| 9.1. Збережені процедури і функції на стороні сервера.....         | 240 |
| 9.2.3. Поняття збережених процедур і функцій.....                  | 240 |
| 9.2.4. Створення процедур і функцій та звертання до них .....      | 241 |
| 9.2.4.1. Збережені процедури.....                                  | 241 |
| 9.2.4.2. Збережені функції.....                                    | 253 |
| 9.2.5. Редагування і видалення збережених процедур і функцій.....  | 254 |
| 9.2. Технологія OLTP. Основні поняття та вимоги до транзакцій..... | 255 |
| 9.2.1. Реалізація транзакцій в SQL .....                           | 259 |
| 9.2.2. Точки відкату .....                                         | 265 |
| 9.2.3. Блокування таблиць.....                                     | 266 |
| 9.2.3.1. Типи таблиць в MySQL.....                                 | 266 |
| 9.2.3.2. Типи блокувань.....                                       | 267 |
| 9.2.3.3. Оператори LOCK TABLES та UNLOCK TABLES .....              | 268 |
| Контрольні запитання.....                                          | 270 |

### **Тема 10. Big Data рішення. OLAP технологія .....272**

|                                   |     |
|-----------------------------------|-----|
| 10.1. Великі дані (Big Data)..... | 272 |
|-----------------------------------|-----|

|                                                                 |     |
|-----------------------------------------------------------------|-----|
| 10.1.1. Принципи роботи з великими даними (VWV) .....           | 273 |
| 10.1.2. Технології і тенденції роботи з Big Data.....           | 274 |
| 10.1.3. Методи і техніка аналізу великих даних.....             | 276 |
| 10.2. Технологія OLAP .....                                     | 277 |
| 10.2.1. Основні поняття OLAP .....                              | 277 |
| 10.2.2. Структура OLAP-куба.....                                | 278 |
| 10.2.3. Ієрархія вимірів OLAP-кубів.....                        | 280 |
| 10.2.4. Операції над гіперкубом .....                           | 280 |
| 10.3. Сховища даних (Data Warehouse).....                       | 283 |
| 10.3.1. Поняття сховища даних .....                             | 283 |
| 10.3.2. Категорії даних, що зберігаються в сховищах даних ..... | 284 |
| 10.3.3. Властивості сховища даних. Вітрина даних .....          | 285 |
| 10.3.4. Глобальне сховище даних .....                           | 286 |
| 10.3.5. Архітектура та наповнення сховищ даних .....            | 287 |
| 10.3.6. Моделі сховищ даних.....                                | 289 |
| 10.3.6.1. MOLAP (Multidimensional OLAP) .....                   | 289 |
| 10.3.6.2. ROLAP (Relational OLAP) .....                         | 291 |
| 10.3.6.3. Інші моделі побудови сховищ даних .....               | 295 |
| 10.4. Інтелектуальний аналіз даних (Data Mining) .....          | 295 |
| Контрольні запитання.....                                       | 297 |

## **Розділ 4. АДМІНІСТРУВАННЯ ТА ЗАХИСТ БАЗ ДАНИХ..... 298**

### **Тема 11. Перегляди, курсори та тригери в MySQL ..... 298**

|                                                         |     |
|---------------------------------------------------------|-----|
| 11.1. Поняття переглядів VIEW .....                     | 298 |
| 11.1.1. Призначення переглядів .....                    | 298 |
| 11.1.2. Переваги використання переглядів .....          | 299 |
| 11.1.3. Обмеження для переглядів в MySQL .....          | 299 |
| 11.1.4. Алгоритми переглядів.....                       | 300 |
| 11.1.5. Створення переглядів.....                       | 300 |
| 11.2. Модифікації переглядів .....                      | 303 |
| 11.2.1. Групові перегляди .....                         | 303 |
| 11.2.2. Перегляди і об'єднання та підзапити .....       | 304 |
| 11.2.3. Перегляди та підзапити.....                     | 305 |
| 11.3. Перегляди в операторах маніпулювання даними ..... | 305 |
| 11.4. Модифікація і видалення переглядів .....          | 307 |
| 11.5. Курсори .....                                     | 308 |
| 11.6. Тригери.....                                      | 311 |
| Контрольні запитання.....                               | 314 |

### **Тема 12. Управління обліковими записами користувачів та виняткові ситуації в MySQL ..... 316**

|                                                                                 |     |
|---------------------------------------------------------------------------------|-----|
| 12.1. Управління обліковими записами користувачів.....                          | 316 |
| 12.2.1. Права (привілеї) в MySQL .....                                          | 316 |
| 12.2.2. Створення користувачів БД.....                                          | 320 |
| 12.2.2.1. Режим командного рядка.....                                           | 320 |
| 12.2.2.2. Режим конструктора.....                                               | 324 |
| 12.2.3. Відміна привілеїв і видалення користувачів.....                         | 328 |
| 12.2. Резервне копіювання і відновлення БД MySQL .....                          | 330 |
| 12.2.1. Копія БД або її частини на тому ж сервері .....                         | 332 |
| 12.2.2. Резервне копіювання і відновлення за допомогою MySQL<br>Workbench ..... | 334 |
| 12.2.2.1. Інструмент Reverse Engineer.....                                      | 334 |
| 12.2.2.2. Майстер експорту та імпорту SQL-даних .....                           | 338 |
| 12.2.2.3. Майстер експорту та імпорту однієї таблиці.....                       | 339 |
| 12.2.2.4. Майстер експорту та імпорту даних результатів .....                   | 343 |
| 12.3. Характерні помилки в MySQL.....                                           | 343 |
| 12.3.1. Діагностика причин виняткових ситуацій .....                            | 343 |
| 12.3.2. Помилки Cannot Connect to Database Server .....                         | 345 |
| 12.3.3. Проблеми при роботі з сервером .....                                    | 347 |
| 12.3.3.1. Причина «server has gone away».....                                   | 347 |
| 12.3.3.2. Помилка Host '...' is blocked .....                                   | 348 |
| 12.3.3.3. Помилка Too many connections .....                                    | 349 |
| 12.3.3.4. Помилка Out of memory .....                                           | 350 |
| 12.3.3.5. Помилка Table 'xxx' doesn't exist.....                                | 352 |
| 12.3.3.6. Помилка Can't initialize character set xxx.....                       | 354 |
| 12.3.4. Інші виняткові ситуації .....                                           | 355 |
| Контрольні запитання .....                                                      | 356 |

## **Розділ 5. БАЗИ ДАНИХ NoSQL .....357**

### **Тема 13. Основи баз даних NoSQL .....357**

|                                                |     |
|------------------------------------------------|-----|
| 13.1. Основні поняття NoSQL.....               | 357 |
| 13.2. Категорії NoSQL .....                    | 358 |
| 13.2.1. Документно-орієнтовані бази даних..... | 358 |
| 13.2.2. Бази даних типу "ключ/значення".....   | 360 |
| 13.2.3. Графові бази даних .....               | 361 |
| 13.2.4. Столпцеві бази даних .....             | 363 |
| 13.3. Основи MongoDB.....                      | 364 |
| 13.3.1. Колекції.....                          | 365 |
| 13.3.2. Документи замість рядків .....         | 366 |
| 13.3.3. Формат JSON.....                       | 369 |
| Контрольні запитання .....                     | 374 |

### **Тема 14. Консоль MongoDB .....376**

|                                     |     |
|-------------------------------------|-----|
| 14.1. Основи роботи з MongoDB ..... | 376 |
|-------------------------------------|-----|

|                                                           |     |
|-----------------------------------------------------------|-----|
| 14.1.1. Початок роботи з MongoDB .....                    | 376 |
| 14.1.2. Типи даних .....                                  | 378 |
| 14.1.3. Колекції і документи .....                        | 379 |
| 14.2. Додавання даних .....                               | 381 |
| 14.2.1. Додавання даних командами .....                   | 381 |
| 14.2.2. Завантаження даних з файлу .....                  | 385 |
| 14.3. Вибірка з БД .....                                  | 386 |
| 14.3.1. Функція find .....                                | 386 |
| 14.3.2. Проекція .....                                    | 390 |
| 14.3.3. Запит до вкладених об'єктів .....                 | 392 |
| 14.3.4. Використання регулярних виразів .....             | 393 |
| 14.3.5. Налаштування запитів та сортування .....          | 395 |
| 14.3.6. Курсори .....                                     | 397 |
| 14.4. Команди групування .....                            | 400 |
| 14.4.1. Кількість елементів в колекції .....              | 400 |
| 14.4.2. Функція distinct .....                            | 401 |
| 14.4.3. Агрегація і метод group .....                     | 402 |
| 14.5. Оператори керування .....                           | 404 |
| 14.5.1. Умовні оператори .....                            | 404 |
| 14.5.2. Логічні оператори .....                           | 408 |
| 14.5.3. Пошук у масивах .....                             | 410 |
| 14.5.4. Оператори \$exists, \$type та \$regex .....       | 412 |
| 14.6. Оновлення даних .....                               | 414 |
| 14.6.1. Метод save .....                                  | 414 |
| 14.6.2. Функція update .....                              | 415 |
| 14.6.3. Оновлення окремого поля .....                     | 416 |
| 14.6.4. Методи updateOne і updateMany .....               | 419 |
| 14.6.5. Оновлення масивів .....                           | 420 |
| 14.7. Видалення даних .....                               | 422 |
| 14.7.1. Видалення поля .....                              | 422 |
| 14.7.2. Видалення елемента з масиву .....                 | 423 |
| 14.7.3. Видалення колекцій і баз даних .....              | 426 |
| 14.8. Встановлення посилань в БД .....                    | 426 |
| 14.8.1. Ручне встановлення посилань .....                 | 427 |
| 14.8.2. Автоматичне зв'язування .....                     | 428 |
| 14.9. Робота з індексами .....                            | 429 |
| 14.10. Керування колекцією .....                          | 433 |
| 14.11. Резервне копіювання і відновлення БД MongoDB ..... | 435 |
| 14.11.1. Резервне копіювання БД .....                     | 435 |
| 14.11.2. Відновлення даних .....                          | 438 |
| Контрольні запитання .....                                | 440 |

## **Тема 15. Графічний клієнт MongoDB Compass ..... 442**

|                                                   |     |
|---------------------------------------------------|-----|
| 15.1. Основні відомості про Compass.....          | 442 |
| 15.2. Підключення Compass до сервера MongoDB..... | 444 |
| 15.3. Створення баз даних та колекцій .....       | 447 |
| 15.4. Додавання документів .....                  | 448 |
| 15.4.1. Імпорт файлу .....                        | 449 |
| 15.4.2. Інтерактивне додавання документу .....    | 450 |
| 15.4.3. Клонування документів.....                | 451 |
| 15.5. Перегляд даних (документів).....            | 452 |
| 15.6. Редагування документів .....                | 454 |
| 15.7. Видалення документів.....                   | 454 |
| 15.8. Видалення колекції та бази даних .....      | 455 |
| 15.9. Консольний клієнт Mongosh .....             | 456 |
| Контрольні запитання.....                         | 457 |

|                                 |            |
|---------------------------------|------------|
| <b>ПРЕДМЕТНИЙ ПОКАЖЧИК.....</b> | <b>459</b> |
|---------------------------------|------------|

|                         |            |
|-------------------------|------------|
| <b>ЛІТЕРАТУРА .....</b> | <b>463</b> |
|-------------------------|------------|



## Перелік скорочень

1НФ – перша Нормальна Форма (1NF)  
2НФ – друга Нормальна Форма  
3VL – Three Valued Logic (тризначна логіка)  
3НФ – третя Нормальна Форма  
4НФ – четверта Нормальна Форма  
5НФ – п'ята Нормальна Форма  
ACID – Atomicity, Consistency, Isolation, Durability (властивості транзакцій)  
ADO – ActiveX Data Objects  
AI – Auto Increment  
ANSI – American National Standards Institute  
BDB – Berkeley Data Base  
BIN – BINary  
BSON – Binary JSON  
CASE – Computer-Aided Software Engineering  
CSV – Comma-Separated Values  
DAO – Data Access Objects  
DB – Data Base  
DCL – Data Control Language  
DDL – Data Definition Language  
DML – Data Manipulation Language  
ER-модель – Entity Relationship model (модель "сутність-зв'язок")  
FK – Foreign Key  
GNU – General Public License  
IDEF1X – Integration DEFinition for information modeling  
ISAM – Indexed Sequential Access Method  
ISO – International Organization for Standardization  
JSON – JavaScript Object Notation  
MOLAP – Multidimensional OLAP  
MVCC – MultiVersion Concurrency Control  
NF2 – Non First Normal Form  
NN – Not Null  
NoSQL – Not Only SQL (не лише SQL)  
NTST – Non-Transaction-Safe Tables  
ODMG-93 – стандарт Object Database Management Group  
OLAP – On-Line Analytical Processing  
OLTP – Online Transaction Processing  
OQL – Object Query Language  
PK – Primary Key

RDO – Remote Data Objects  
ROLAP – Relational OnLine Analytical Processing  
SQL – Structured Query Language  
SSH – Secure SHell  
SSL – Secure Sockets Layer  
TCL – Tool Command Language  
TST – Transaction-Safe Tables  
UML – Unified Modeling Language  
UN – UNsigned  
UQ – UniQue  
UTF8 – Unicode Transformation Format, 8-bit  
VVV – Volume, Velocity, Variety  
XML – eXtensible Markup Language  
YAML – Yet Another Markup Language  
ZF – Zero Fill  
БД – База Даних  
БСКБД – Багатовимірна СКБД  
ВК – Вторинний Ключ  
ДКНФ – доменно-ключова Нормальна Форма  
ІАД – Інтелектуальний Аналіз Даних (Data Mining)  
ІС – Інформаційна Система  
НФ – Нормальна Форма  
НФБК – Нормальна Форма Бойса-Кодда  
ООП – Об’єктно-Орієнтоване Програмування  
ПЗ – Програмне Забезпечення  
ПК – Первинний Ключ  
СКБД – Система Керування Базою Даних  
ФЗ – Функціональна Залежність

# Передмова

Різноманіття типів баз даних (БД) зумовлене, насамперед, різноманіттям задач, для яких використовуються відповідні БД.

Даний посібник надає змогу зорієнтуватися розробнику в класифікації та сферах застосування БД, дозволяє вивчити теоретичні основи БД, а також вдосконалити навички зі створення БД як класичних, реляційних, так і нереляційних, їх супроводу, адміністрування, діагностувати причини виняткових ситуацій, що виникають при роботі з ними, і, відповідно, знайти шляхи вирішення конфліктів.

Адже від вибору типу бази даних, розробленої схеми, її зв'язків залежить подальша коректна робота з даними. І не тільки на стороні сервера, ще більш залежні від розробленої структури БД будуть застосунки користувача.

Для розроблення і супроводу БД на стороні сервера акцент ставиться на середовища, що поширюються під вільною ліцензією, зокрема MySQL Workbench для реляційних БД та MongoDB для NoSQL БД.

У багатьох сферах, таких як у банківській, страховій, біржовій, обліковій, комерційній, тощо, використовується спосіб організації баз даних (БД), при якому система працює в режимі реального часу з короткими, не пов'язаними одна з одною транзакціями, організованими у великий потік. Така онлайнова обробка транзакцій отримала назву OLTP-системи та забезпечує обов'язковий набір властивостей – атомарність, узгодженість, ізолюваність та довговічність (ACID).

Зростаючі об'єми інформації вимагають нових технологій для зберігання, модифікації, отримання та адміністрування «великих даних». Адже такі дані повинні бути горизонтально масштабовані, тобто зберігатися на багатьох ресурсах і забезпечувати хорошу продуктивність їх оброблення.

В останні роки у світі сформувався ряд нових технологій збереження і аналізу великих обсягів даних:

- 1) сховища даних, або склади даних (Data Warehouse);
- 2) оперативна аналітична обробка (On-Line Analytical Processing, OLAP);
- 3) інтелектуальний аналіз даних – ІАД (Data Mining).

Для аналітичного опрацювання «великих даних» використовується технологія OLAP як ключовий компонент організації сховищ даних. Інтелектуальний аналіз даних використовується для підтримки прийняття рішень, заснований на пошуку в даних схованих закономірностей.

Оскільки реляційні бази даних погано пристосовані до роботи з дійсно великими обсягами даних, так як вони спроектовані для роботи на одній машині, тому для розподіленої роботи використовується інша технологія баз даних – NoSQL (Not only SQL) – це не тільки реляційні бази даних.

Такі бази даних добре масштабуються горизонтально, але не зберігають всі принципи SQL (ACID). Їх можна умовно розділити на чотири групи: документо-орієнтовані бази даних, бази даних типу "ключ/значення", графові бази даних та стовпцеві бази даних.

Автори висловлюють сподівання, що даний посібник буде корисний не тільки студентам, які вивчають дисципліну «Організація баз даних і знань», але безпосередньо і розробникам баз даних.

Також автори висловлюють подяку усім колегам, співпраця з якими позитивно впливала на написання посібника, особливо Євгену Бегену з ІТ-компанії SoftServe Confidential за проведений ґрунтовний аналіз робочої програми дисципліни «Організація баз даних і знань» і наданим рекомендаціям щодо вдосконалення її вмісту. Окрему вдячність автори висловлюють шановним рецензентам – д.ф.-м.н., проф. Дияк І.І., д.ф.-м.н., проф. Терещенко В.М., д.ф.-м.н., проф. Процах Н.П. за цінні поради та зауваження, які сприяли покращенню посібника. Ми будемо вдячні всім, хто знайде за можливе подати свої зауваження та конструктивні відгуки про посібник за адресою [dendiuk@ukr.net](mailto:dendiuk@ukr.net). Вони слугуватимуть добрим підґрунтям нашої подальшої роботи.

*З повагою – автори.*

# Розділ 1. ТЕОРЕТИЧНІ ОСНОВИ БАЗ ДАНИХ

## Тема 1. Структура баз даних

### 1.1. Основні поняття та визначення

Сучасну комп'ютерну інформаційну систему (IC) можна розглядати як сукупність апаратних (hardware), програмних (software), мережних (netware), інформаційних (dataware) та міжкомпонентних (middleware) ресурсів (рис. 1.1), які в процесі свого сумісного функціонування забезпечують розв'язок кола задач з отримання, реєстрації, передачі, збереження, захисту, перетворення, подання інформації в певній галузі людської діяльності.

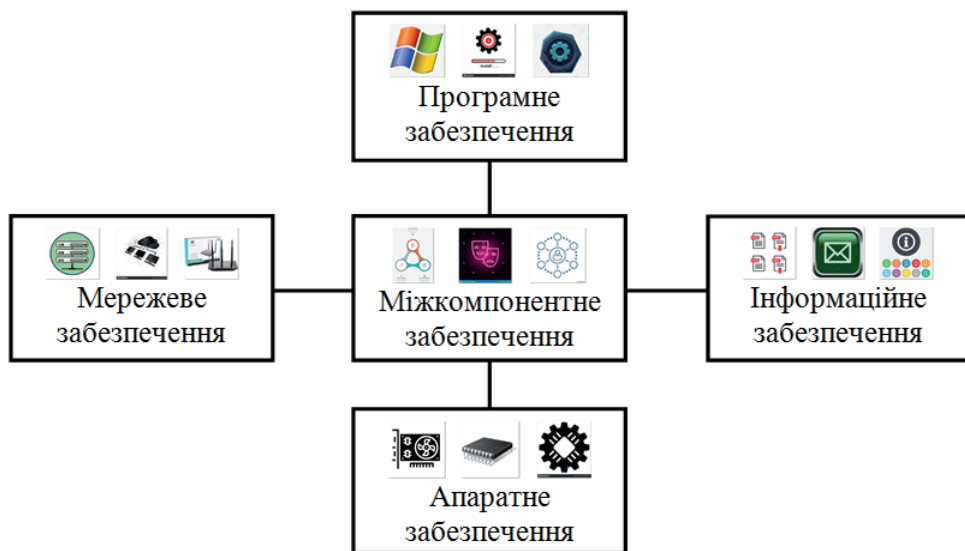


Рис. 1.1. Сучасна інформаційна система

Для розв'язання проблем накопичення, зберігання, відтворення та перетворення інформаційних ресурсів комп'ютерних інформаційних систем призначені сховища, які називаються **базами даних**.

Під **базою даних** (БД) (Database, DB) розуміють сукупність спеціальним чином організованих даних і об'єктів, що відображають предметну область і призначені для спільного використання.

Під *предметною областю* прийнято розуміти частину реального світу, який підлягає вивченню для організації управління і в кінцевому рахунку – автоматизації. Вміст даних у БД відноситься до певної предметної області (ПО).

Для створення бази даних та її супроводу найчастіше досить засобів *Системи Керування Базою Даних* (СКБД) – спеціалізованої програми (частіше комплекс програм), призначеної для організації і ведення бази даних.

Всі бази даних складаються з двох компонентів: структури (схеми) та даних.

## 1.2. Системи керування базами даних

*Система керування базою даних* (СКБД) – це комплекс мовних і програмних засобів, призначений для створення, ведення і сумісного використання бази даних користувачами.

Для кінцевого користувача (та прикладних програм) робота з базою даних напряду неможлива. Всі маніпуляції над даними здійснюють через спеціальні запити, які надсилають до СКБД. СКБД опрацьовує їх і повертає результат. Безпосередньо з базою даних працює виключно СКБД.

### *Основні функції СКБД:*

- керування даними в зовнішній пам'яті (на дисках);
- керування даними в оперативній пам'яті з використанням дискового кешу;
- журналізація змін, резервне копіювання і відновлення бази даних після збоїв;
- підтримка мов БД (мова визначення даних, мова маніпулювання даними).

Зазвичай *СКБД містить наступні компоненти* (рис. 1.2):

- ядро, яке відповідає за управління даними у зовнішній і оперативної пам'яті та журналізацію;
- процесор мови бази даних, що забезпечує оптимізацію запитів на вилучення та зміну даних і створення, як правило, машинно-незалежного виконуваного внутрішнього коду;
- підсистему підтримки часу виконання, яка інтерпретує програми маніпуляції даними, що створюють користувальницький інтерфейс із СКБД;

- сервісні програми (зовнішні утиліти), що забезпечують ряд додаткових можливостей з обслуговування інформаційної системи.



Рис. 1.2. Система керування базою даних

Сучасні СКБД забезпечують функції щодо керування даними, які можна поділити на такі групи:

- **оголошення** даних – створення, зміна та видалення визначень, які описують організацію даних;
- **модифікація** даних – додавання даних, їх редагування та видалення;
- **отримання** даних – надання даних за запитом у формі, яка дозволяє їх безпосереднє використання;
- **адміністрування** даних – реєстрування та відслідковування дій користувачів, дотримання безпеки роботи з даними, забезпечення надійності та цілісності даних, моніторинг продуктивності, резервне копіювання та відновлення даних тощо.

Незважаючи на те, що всі системи керування базами даних виконують одні і ті ж задачі (тобто дають можливість користувачам створювати, редагувати і отримувати доступ до інформації, що зберігається в базах даних), сам процес виконання цього завдання варіюється в широких межах, що породжує різноманіття СКБД.

Власне вибір СКБД визначає, які дані і як будуть зберігатися в БД, як здійснюватиметься доступ до даних, яка мова підтримується, тощо. Істинне також зворотне твердження: які дані і як зберігаються в БД, як здійснюється доступ до даних, яка мова використовується процесором і т.д. визначають тип СКБД.

Тому терміни СКБД та БД зазвичай взаємопов'язані.

Але варто все ж розрізняти базу даних – як *сховище даних*, а СКБД – як засоби для роботи зі сховищем даних.

### 1.3. Загальна архітектура баз даних

Архітектура баз даних – це сукупність її основних функціональних компонентів, а також засобів забезпечення їх взаємодії один з одним. Американським національним інститутом стандартів (ANSI) визначено три архітектурні рівні баз даних (рис. 1.3):

- внутрішній;
- концептуальний;
- зовнішній.

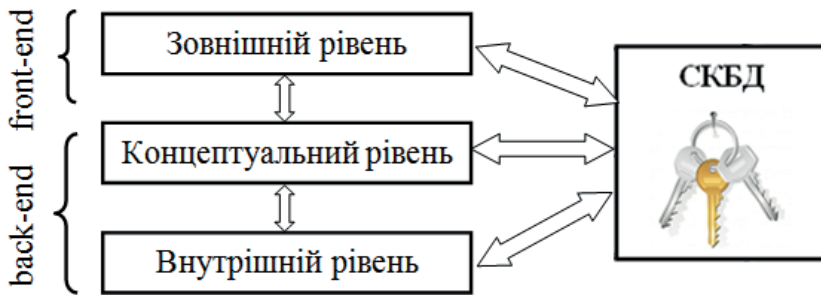


Рис. 1.3. Загальна архітектура баз даних

Принцип незалежності структур, визначених на різних рівнях, є основоположним принципом побудови баз даних.

Такий підхід дозволяє відділити дані від інтерфейсу користувача і дозволяє сконцентрувати ринок ПЗ безпосередньо на покращення своїх продуктів.

**Внутрішній рівень** – фізичне представлення БД в комп'ютері. Цей рівень описує, як інформація зберігається в БД. Внутрішній рівень описує фізичну реалізацію БД, також призначений для досягнення оптимальної продуктивності і забезпечення економного використання дискового простору. На внутрішньому рівні зберігається наступна інформація:

- розподіл дискового простору для зберігання даних та індексів;



- опис подробиць збереження записів (із зазначенням реальних розмірів елементів даних та розміщення записів);
- відомості про стиснення даних і обраних методах шифрування.

**Концептуальний рівень** – це інтегрований опис предметної області, для якої створена БД, незалежний від будь-яких міркувань щодо способу зберігання даних. Цей рівень не містить ніяких відомостей про методи зберігання даних.

На концептуальному рівні представлені такі компоненти:

- всі сутності, їх атрибути та зв'язки;
- накладені обмеження на дані;
- семантична інформація про дані (пов'язана зі значенням, змістом);
- інформація про заходи забезпечення безпеки і підтримки цілісності даних.

Концептуальний рівень підтримує кожне зовнішнє уявлення, в тому сенсі, що будь-які доступні користувачеві дані повинні міститися (або можуть бути обчислені) на цьому рівні.

**Зовнішній рівень** – це представлення даних з точки зору користувачів. Цей рівень описує призначену для користувача частину БД, тобто частина, яка відноситься до кожного користувача. Відповідно до термінології ANSI, уявлення окремого користувача називається **зовнішнім поданням** – це вміст бази даних, яким його бачить певний користувач. Кожне зовнішнє уявлення визначається за допомогою зовнішньої схеми, яка в основному складається з визначень записів кожного з типів, присутніх в цьому зовнішньому поданні.

## 1.4. Класифікація баз даних

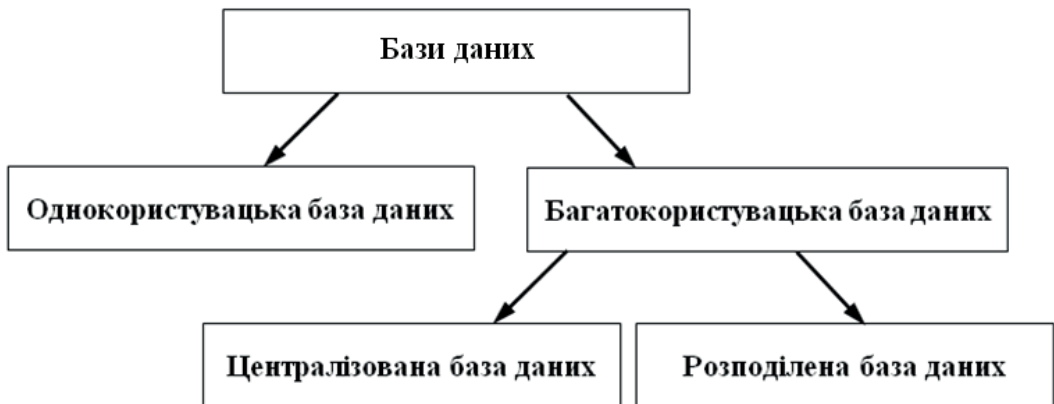
Залежно від задач, які розв'язує інформаційна система, використовуються бази даних різних типів.

### 1.4.1. Бази даних за типом доступу (локальні, клієнт-серверні)

Перш за все, БД за **типом доступу** розрізняють:

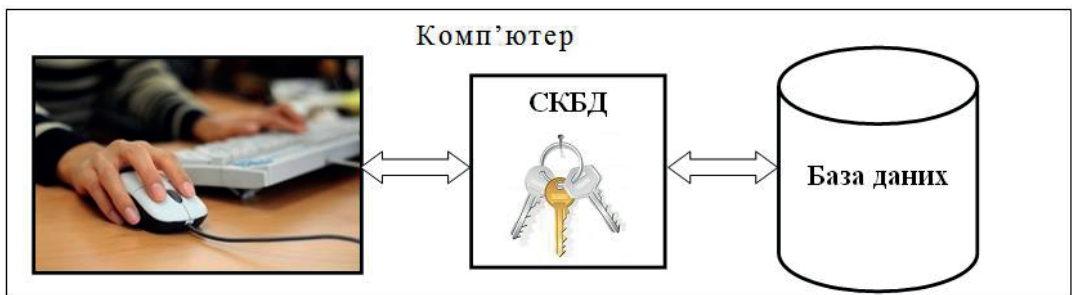
- локальні (однокористувацькі);
- багатокористувацькі,

які, в свою чергу, можуть бути як централізованими, так і розподіленими (рис. 1.4).



*Рис. 1.4. Види баз даних за типом доступу*

Однокористувацька (локальна) БД передбачає розташування всього програмного забезпечення і даних на одному ізольованому комп'ютері (рис. 1.5).



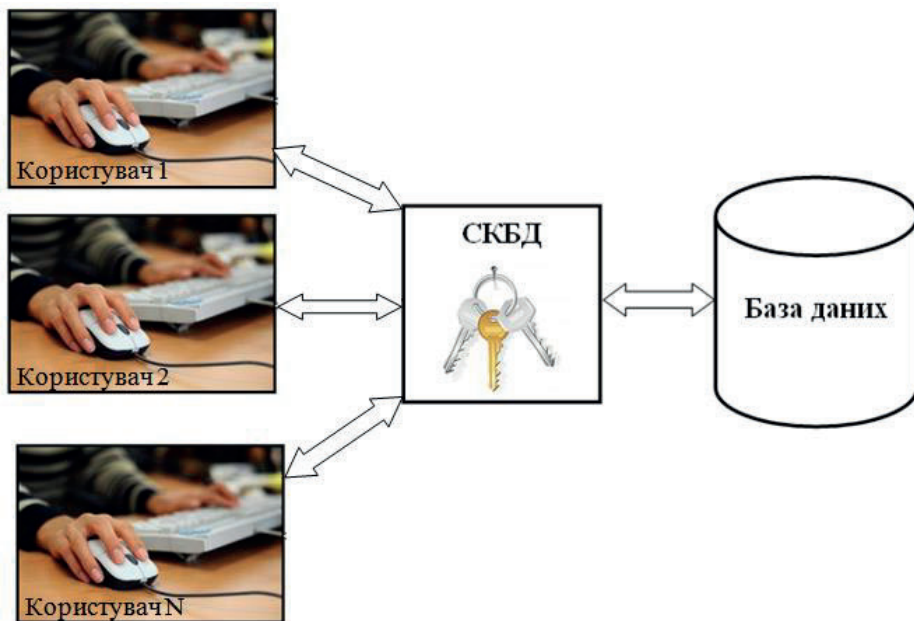
*Рис. 1.5. Однокористувацька баз даних*

Одночасно з такою БД може працювати тільки один користувач. Однак локалізація не заперечує передавання даних з цієї БД іншим користувачам.

Багатокористувацька база даних передбачає доступ до даних мережею багатьох користувачів. В цьому випадку власне база даних може знаходитися як на одному комп'ютері (сервері), так і в кластері, тобто на багатьох серверах.

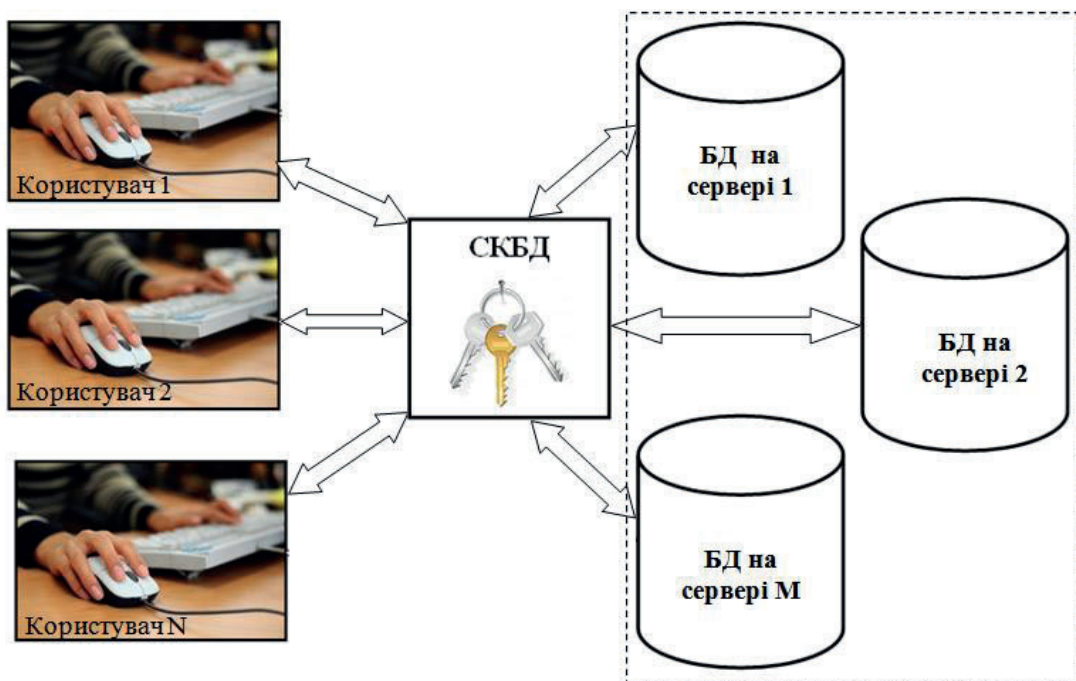
Структура багатокористувацької централізованої БД наведено на рис. 1.6.

За такої архітектури кожен користувач зі свого комп'ютера через локальну або глобальну мережу здійснює роботу зі спільною БД, яка розташована на віддаленому сервері, і де всі функції доступу до даних виконує СКБД.



*Рис. 1.6. Багатокористувацька централізована база даних*

Структура багатокористувацької розподіленої бази даних (рис. 1.7) – це сукупність логічно взаємопов’язаних баз даних, розподілених у комп’ютерній мережі, де:



*Рис. 1.7. Багатокористувацька розподілена база даних*

- кожен вузол – це повноцінна БД сама по собі;
- вузли взаємодіють між собою таким чином, що користувач будь-якого з них може отримати доступ до будь-яких даних у мережі так, як ніби вони знаходяться на його власному комп'ютері.

Кожен вузол сам по собі є системою бази даних. Будь-який користувач може виконати операції над даними на своєму локальному вузлі точно так само, як якщо б цей вузол зовсім не входив у розподілену систему. Розподілену систему баз даних можна розглядати як партнерство між окремими локальними БД на окремих локальних вузлах.

#### 1.4.2. Бази даних за типом керування (реляційні, NoSQL, NewSQL)

БД за *типом керування* поділяють реляційні та інші. Іншими БД можуть бути NoSQL або NewSQL (рис. 1.8).

**Реляційна база даних** – це множина взаємопов'язаних таблиць, кожна з яких містить інформацію про об'єкти певного виду. Кожен рядок таблиці містить дані про один об'єкт, а стовпці таблиці містять різні характеристики цих об'єктів – атрибути.



Рис. 1.8. Види баз даних за типом керування

Рядки таблиці називаються записами. Всі записи таблиці мають однакову структуру та складаються з полів (елементів даних), в яких зберігаються атрибути об'єкта. Кожне поле запису містить одну характеристику об'єкта і має визначений тип даних (рис. 1.9).



Рис. 1.9. Структура таблиці реляційної бази даних

Для ідентифікації записів використовується первинний ключ. *Первинним ключем* називається набір полів таблиці, комбінація значень яких однозначно визначає кожний запис у таблиці. Також таблиці можуть бути з'єднані між собою з допомогою *зовнішніх ключів*. До даних цих таблиць можна отримати доступ багатьма способами, і при цьому реорганізовувати таблиці БД не потрібно.

Приклади реляційних БД: комерційні – MS SQL Server, Oracle; некомерційні – MySQL, PostgreSQL, SQLite.

**NoSQL база даних** – це база даних, в якій, на відміну від більшості традиційних систем баз даних, не використовується таблична схема рядків і стовпців. У цих базах даних застосовується модель зберігання, що оптимізована під конкретні вимоги типу збережених даних. Бази даних таких типів оптимізовані для додатків, які працюють з *величезними обсягами даних*, потребують *низької затримки виконання команд* доступу до даних і гнучких моделей даних. Все це досягається шляхом пом'якшення жорстких вимог до несутеречливості даних, характерних для інших типів БД.

Кількість конкуруючих NoSQL баз даних дуже велика. А причина проста: різні бази даних NoSQL призначені для вирішення різних завдань. Всі NoSQL бази даних можна розділити на чотири основні категорії (рис. 1.10).

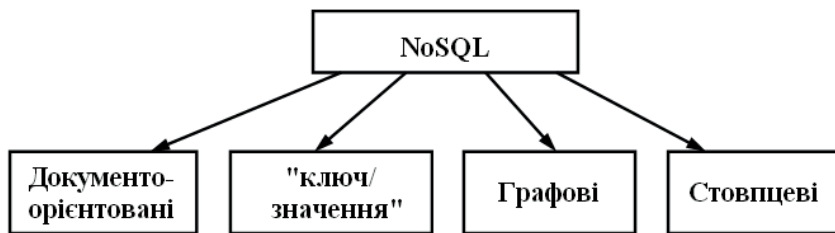


Рис. 1.10. Основні категорії NoSQL баз даних

**Документо-орієнтовані бази даних** – надають можливість зберігання складних вкладених документів, в той час як більшість реляційних баз даних підтримує лише одномірні рядки. Така можливість може стати в нагоді в багатьох випадках, наприклад, при необхідності зберігання в системі інформації про користувача з декількома адресами. При використанні документо-орієнтованої бази даних, в подібному випадку можна просто зберігати складний об'єкт, що включає масив адрес, в той час як в реляційній базі даних вам довелося б створити дві таблиці: одну для інформації про користувача, а другу для адрес.

Приклади СКБД даного типу – CouchDB, Couchbase, MongoDB, eXist, Berkeley DB XML.

**Бази даних типу "ключ/значення"** – зазвичай реалізують найпростішу NoSQL-модель та, по суті, надають розподілену хеш-таблицю, що дозволяє записувати дані за заданим ключем і читати їх назад з його допомогою.

Бази даних типу "ключ / значення" легко масштабуються і відрізняються від інших баз даних значно нижчою затримкою.

Приклад таких сховищ – Berkeley DB, Memcached, Redis, Riak, Amazon DynamoDB.

**Графові бази даних** – призначаються для завдань, у яких дані містять велику кількість зв'язків, наприклад, соціальних мереж, виявлення шахрайства, інформацію про фільми та акторів.

Такі предметні області можна представити у вигляді графів. Хоча граф можна представити і за допомогою реляційної бази даних, це складно і незручно. Якщо потрібні графові дані, краще скористатися спеціалізованою графовою базою даних, яка може зберігати інформацію про граф в розподіленому кластері і дає можливість ефективної реалізації алгоритмів на графах.

Приклади: Neo4j, OrientDB, AllegroGraph, Blazegraph, InfiniteGraph, FlockDB, Titan.

**Стовпцеві бази даних** (BigTable-подібні бази даних) – створюють по файлу для кожного стовпця ваших таблиць. Основна відмінність між стовпцевими і іншими типами баз даних полягає в способі зберігання даних на диску.

Прикладами СКБД даного типу є: Apache HBase, ApacheCassandra, ScyllaDB, ApacheAccumulo, Hypertable.

Насправді SQL та NoSQL скоріше не конкуруючі, а доповнюючі одна одну технології. Використання в одному додатку SQL-рішень, коли потрібно працювати зі складними даними в їх взаємозв'язку, і NoSQL, коли на передній план виходить швидкість роботи з неструктурованою інформацією, абсолютно природна практика.

**Бази даних NewSQL** – клас сучасних реляційних розподілених БД, які прагнуть поєднати в собі переваги NoSQL і транзакційні вимоги класичних реляційних баз даних (рис. 1.11).

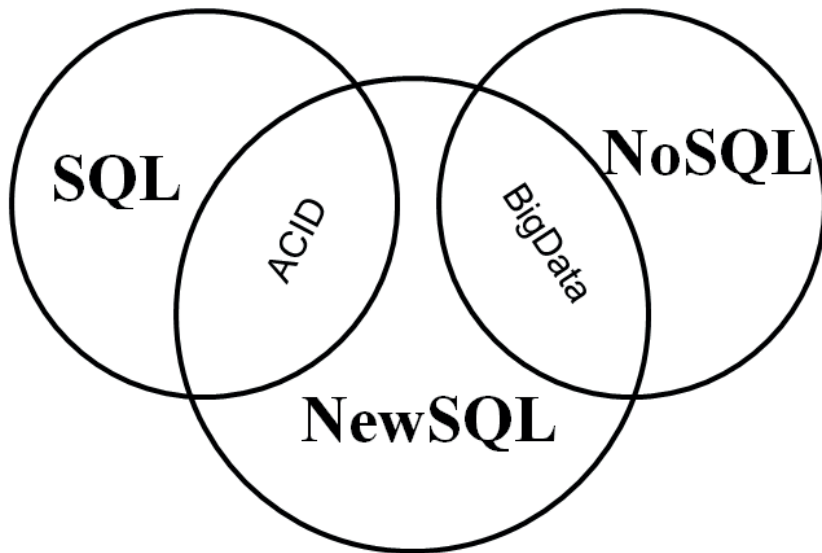


Рис. 1.11. NewSQL бази даних

Потреба в NewSQL виникла в першу чергу у компаній, що працюють з критичними даними, наприклад банківська сфера, яким були необхідні масштабовані рішення, але використання NoSQL не могло надати гарантій

виконання транзакцій і не відповідали вимогам надійності даних. NewSQL дозволяє досягти рівня продуктивності NoSQL-систем, зберігши при цьому підтримку виконання запитів на мові SQL і гарантовану транзакційну цілісність даних. Високу продуктивність NewSQL забезпечує не схожа на традиційні схема внутрішньої архітектури, що комбінує зберігання даних в пам'яті концепції розподіленої організації і розбивкою змісту БД за розділами.

Інакше кажучи, це здорова спроба досягти якоїсь «золотої середини», балансу між двома полюсами: багатством SQL і мінімалізмом NoSQL, спроба поєднати переваги масштабованості і швидкості NoSQL, при цьому максимально зберігаючи плюси широкої функціональності традиційного SQL – головна мета руху NewSQL.

Прикладами СКБД даного типу є: H-Store, VoltDB, Hypertable, NuoDB, MariaDB.

## **1.5. Тенденції використання різних типів баз даних**

Як свідчить аналіз БД за типом керування, вибір тієї чи іншої БД ґрунтується вимогами, які можна розділити на дві категорії:

- надійність роботи та достовірність даних;
- швидкість роботи та об'єм даних.

Надійність роботи та достовірність даних забезпечують реляційні БД, які дедалі частіше називають OldSQL. Але вони не забезпечують швидкості роботи з даними і не призначені для величезних об'ємів даних, тим паче роботи в хмарі. На зміну OldSQL приходять NewSQL, які поволі, але впевнено опановують ринок ПЗ. До предметних областей, де використовуються OldSQL та NewSQL відносяться, в першу чергу, банківська сфера та бухгалтерський облік.

Якщо ж необхідно працювати з величезними об'ємами інформації на великих швидкостях виконання та при цьому не вимагати транзакційну атомарність і узгодженість даних, тоді однозначно вибір належить БД NoSQL. Предметними областями для використання тут слугують соціальні мережі, пошукові системи тощо.



Якщо ж проаналізувати використання в інформаційних системах різних типів БД, то перші три місця займають реляційні БД і лише тоді – NoSQL MongoDB (рис. 1.12).

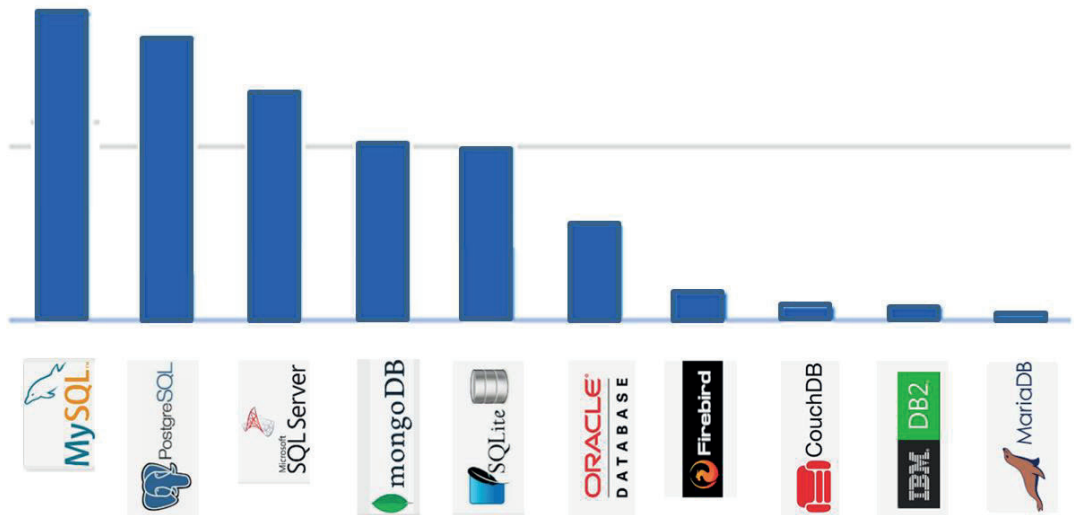


Рис. 1.12. Використання різноманітних БД в інформаційних системах

На закінчення цікаво відмітити, що чим більше популярними стають NoSQL сьогодні, чим більша кількість користувачів приходить в цю популярну індустрію – тим більше можливостей хочуть від нього замовники, тим більше NoSQL ускладнюється і втрачається ідея початкової простоти.

### Контрольні запитання

1. Дайте визначення бази даних.
2. Що таке предметна область?
3. Які основні функції СКБД?
4. Проаналізуйте сукупність основних функціональних компонентів БД.
5. Наведіть класифікацію БД за типом доступу.
6. Охарактеризуйте класифікацію БД за типом керування.
7. Які особливості багатокористувацької БД?
8. Які особливості розподіленої БД?
9. Що таке реляційна БД?
10. Що таке NoSQL БД і для чого вони створені?

11. Які вимоги до NoSQL БД?
12. Які основні категорії NoSQL БД?
13. В чому потреба виникнення NewSQL БД?
14. Проаналізуйте тенденції використання різних типів баз даних.

## Тема 2. Моделі баз даних і моделі даних

### 2.1. Поняття про моделі баз даних

Кожна система БД реалізує ту чи іншу модель даних, яка визначає правила породження допустимих для системи видів структур даних, можливі операції над такими структурами, засоби обмежень цілісності даних. Таким чином, модель даних задає межі множинам всіх конкретних БД, які можуть бути створені засобами цієї системи.

Опис обраної предметної області в термінах моделі даних дозволяє отримати модель бази даних.

**Модель бази даних** – те саме, що і схема бази даних, тобто опису змісту, структури і обмежень цілісності, що використовуються для створення і підтримки бази даних.

У літературі, статтях і в повсякденній мові іноді зустрічається плутанина терміна «модель бази даних» та «модель даних». Співвідношення між цими поняттями аналогічно співвідношенню між мовою програмування і конкретною програмою на цій мові. Модель даних є теорія, або інструмент моделювання, в той час як модель бази даних (схема бази даних) є результат моделювання.

Зазвичай виділяють три рівні моделей БД:

- **Міфологічна (концептуальна)** модель відображає інформацію про предметну область без орієнтації на конкретну СКБД (або навіть на тип передбачуваної до використання СКБД). У зв'язку з цим інколи говорять про існування інфологічної моделі предметної області, а не БД.
- **Даталогічна (логічна)** модель БД – модель логічного рівня, що представляє собою відображення логічних зв'язків між елементами даних незалежно від їх змісту і середовища зберігання. Ця модель будується в термінах інформаційних одиниць, допустимих в тій СКБД, в середовищі якої буде створюватися БД. Етап створення даної моделі називається даталогічним або логічним проектуванням.

- **Фізична** модель БД будується з урахуванням можливостей по організації і зберігання даних, що надаються СКБД та використовуваною програмноапаратною платформою.

### **2.1.1. Міфологічні (концептуальні) моделі**

Після передпроектного обстеження предметної області адміністратор БД поєднує свої уявлення про дані та представлення, отримані в результаті опитування користувачів, дає узагальнений формальний опис майбутньої БД у вигляді концептуальної моделі даних. Ця модель є єдиним інтегрованим описом предметної області і ядром майбутньої ІС.

У моделі повинні бути відбиті поняття і об'єкти реального світу, що повинні складати систему, і відношень між ними. Повнота концептуальної моделі даних залежить від цілей ІС, що проектується. У моделі адміністратор БД може описати тільки ті основні об'єкти, що важливі для ІС, а інші деталі, що ускладнюють розуміння, можуть бути опущені. Цей опис семантики реальної предметної області може бути виконаний з використанням звичайної мови, математичних формул, таблиць, графіків і інших засобів, зрозумілих усім людям, що працюють над проектуванням БД. Такий опис дозволяє одночасно представляти дані та їхню семантику.

Головна мета концептуальної моделі даних – це забезпечення найбільш природних для людини способів збору та представлення тієї інформації, що передбачається зберігати в майбутній БД. Вона повинна допомагати спілкуватися з замовником і уточнювати його вимоги, і, як результат, виробити правильний підхід при практичній розробці БД.

Можна сказати, що концептуальна модель даних відображує погляди, потреби і уявлення всіх користувачів та адміністратор БД про поняття предметної області та дані, що можуть знадобитися в майбутніх програмних застосунках.

Концептуальна модель даних створюється як людино-орієнтована модель даних. Вона повинна бути цілком незалежною від середовища збереження

даних, яким фізично може бути не комп'ютер, а пам'ять людини. Модель повинна бути не суперечливою і не допускати неоднозначного трактування.

### **2.1.2. Даталогічні (логічні) моделі**

Далі концептуальні моделі даних моделі повинні бути відображені в комп'ютер-орієнтовану модель представлення даних на рівні реалізації. Ця модель називається даталогічною або просто логічною моделлю даних.

Вона відображує логічне представлення даних у БД з точки зору користувача і прикладного програміста. Ця модель повинна надати можливість програмам і користувачам здійснювати доступ до збережених даних лише по їх іменах, не піклуючись про фізичне розташування цих даних. Такий доступ здійснюється за допомогою конкретних засобів управління даними, в основному засобами СКБД.

Логічна модель даних повинна бути "зрозумілою" для СКБД, тому ці моделі повинні бути описані мовою опису даних СКБД. Звичайно такий опис також створюється адміністратором БД.

Метою логічного проектування є відображення об'єктів предметної області в абстрактні об'єкти моделі даних таким чином, щоб це відображення не суперечило семантиці предметної області і бажано, щоб воно було ефективним.

### **2.1.3. Фізичні моделі**

Дані зберігаються на зовнішніх пристроях пам'яті, тому фізична організація даних впливає на експлуатаційні характеристики БД.

Фізичний рівень представлення даних відображає уявлення системного програміста-аналітика або адміністратора БД.

Фізичне представлення зазвичай мають незначні розходження з представленням реалізації.

Фізична модель даних створюється з урахуванням можливостей та особливостей конкретної СКБД. При цьому враховуються типи даних, обмеження, правила, які підтримує СКБД.

Зазначимо, що сучасні СКБД пропонують розвинутий інструментарій для розробки і корегування фізичної моделі даних, а також необхідних додаткових структур.

Кінцевий користувач мало звертає увагу на організацію фізичного зберігання інформації – його цікавить логічне представлення даних.

#### **2.1.4. Взаємозв'язок моделей даних**

Таким чином, в загальному випадку представлення даних складається з трьох пов'язаних рівнів:

- 1) концептуальна модель даних;
- 2) логічна модель даних;
- 3) фізична модель даних.

Така архітектура представлення даних дозволяє забезпечити незалежність даних, що зберігаються, від програм, що використовують ці дані.

Фізична незалежність полягає в тому, що носії інформації і способи фізичного збереження можуть змінюватися без зміни прикладних програм. При цьому адміністратор БД змінює лише фізичну модель даних.

Логічна незалежність дозволяє додавати нові елементи або дані, розширювати загальні логічні структури без переробки існуючих програм. Адміністратор БД може підключити до системи будь-яку кількість нових користувачів. При цьому в разі необхідності логічна модель даних може бути доповнена. Зазначені зміни фізичної і логічної моделей даних не будуть помічені існуючими користувачами системи. Незалежність даних забезпечує можливість розвитку системи БД без зміни існуючих програмних застосунків.

Концептуальна модель даних майже не змінюється на всіх етапах життєвого циклу БД. І тільки тоді, коли в тому реальному світі, що моделюється, стануться якісь суттєві зміни, для продовження адекватного відображення предметної області необхідно буде змінити цю модель.

## 2.2. Типи моделей даних

Для розміщення однієї і тієї ж інформації в БД можуть бути використані різні структури і моделі даних. Вибір покладається на користувача, що створює базу, і залежить від багатьох факторів, в тому числі від наявного технічного і програмного забезпечення, визначається складністю задач, що автоматизуються, і обсягом інформації.

**Модель даних** (Data Model) – це система правил, за якими створюють структури даних, здійснюють доступ до даних та змінюють їх. Ці структури даних є незалежні від апаратного й програмного забезпечення (концептуальний рівень архітектури) і не пов'язані з функціонуванням комп'ютера.

Залежно від характеру зв'язків між елементами даних у баз даних розрізняють такі основні типи моделей даних (рис. 2.1):

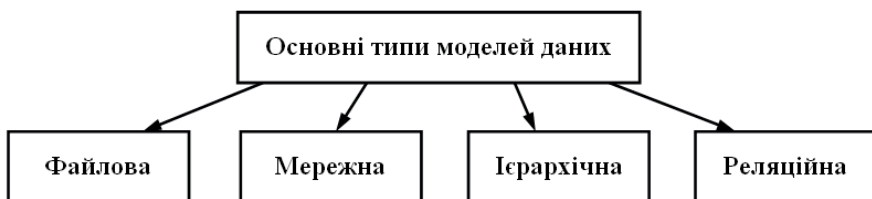


Рис. 2.1. Основні типи моделей даних

- файлову;
- мережну;
- ієрархічну;
- реляційну.

Основне розходження даних моделей полягає в поданні взаємозв'язків між об'єктами.

Крім того, в останні роки з'явилися і стали більш активно впроваджуватися на практиці наступні моделі даних:

- постреляційні;
- багатовимірні;
- об'єктно-орієнтовані.

Розробляються також системи, основані на інших моделях даних, що розширюють відомі моделі. З них можна виокремити:

- об'єктно-реляційні;
- дедуктивно-об'єктно-орієнтовані;
- семантичні;
- концептуальні;
- орієнтовані моделі.

Деякі з цих моделей служать для інтеграції баз даних, баз знань і мов програмування. У деяких СКБД підтримуються одночасно кілька моделей даних.

### 2.2.1. Файлова модель

Найпростіша модель даних реалізується набором незалежних файлів, орієнтованих, як правило, на одну прикладну задачу. Такий підхід використовується у пакетних системах обробки інформації. Модель внутрішньомашинної інформаційної бази є сукупністю незалежних, не пов'язаних між собою локальних інформаційних файлів, які мають лінійну структуру. Інформаційні файли проектується окремо для кожної конкретної задачі чи комплексу задач. Файли складаються з однотипних записів.

Основні типи структур даних файлової моделі – поле, запис, файл. Запис є основною структурною одиницею обробки даних у файлових моделях і одиницею обміну між оперативною і зовнішньою пам'яттю. При описі логічної організації даних кожному файлу привласнюється унікальне ім'я і дається опис структури його записів (рис. 2.2).

| ІМ'Я ФАЙЛУ |              |            |         | Ознака ключа |
|------------|--------------|------------|---------|--------------|
| ПОЛЕ       |              | ФОРМА ПОЛЯ |         |              |
| Позначення | Найменування | Тип        | Довжина | Точність     |

Рис. 2.2. Структура файлу

Опис структури записів включає перелік полів і їх порядок усередині запису. Для кожного поля задається скорочене позначення – ім'я поля, формат поля – тип



збережених даних, довжина поля і точність числових даних. Для полів, що виконують роль унікального (первинного) ключа запису, вказується ознака ключа.

Екземпляр запису є реалізацією запису, що містить конкретні значення полів. Структура запису файлу – лінійна, тобто поля мають єдине значення. Кожен екземпляр запису однозначно ідентифікується унікальним ключем запису. У загальному випадку ключі бувають двох видів: первинні (унікальні) та вторинні.

*Первинний ключ (ПК)* – це одне чи декілька полів, що однозначно ідентифікують запис. Якщо первинний ключ складається з одного поля, він називається простим, якщо з декількох полів – складним ключем.

Вторинний ключ (ВК) – це поле, значення якого може повторюватися в декількох записах файлу, тобто він не є унікальним. Якщо за значенням первинного ключа може бути знайдений один єдиний екземпляр запису, то за вторинним – декілька.

Названі структури даних використовуються у ряді СКБД, що робить ці поняття у визначеному змісті універсальними.

До недоліків файлової моделі даних можна віднести:

- **Надмірність даних** – для розв’язування різних задач управління використовуються одні й ті самі дані. Інформація одного й того самого об’єкта управління розподіляється між багатьма файлами. Це приводить до дублювання даних в інформаційних масивах та неекономного використання пам’яті комп’ютера.
- **Неузгодженість даних** – одна й та сама інформація може розміщуватись у різних файлах. При цьому технологічно складно простежити за внесенням змін одночасно в усі файли. Через це може виникнути узгодженість даних – це коли одне й те саме поле в різних файлах може мати різні значення.
- **Залежність структур даних і прикладних програм** – при файловій організації логічна та фізична структура файлу повинна мати опис у прикладній програмі. При зміні логічної чи фізичної структури файлу прикладна програма має бути модифікована. Але зміни в одній програмі

часто потребують внесення змін до інших інформаційно пов'язаних програм, що призводить до значного збільшення вартості супроводу програмних засобів (іноді вартість супроводу програмних засобів досягає 70 % вартості їх розроблення).

Перевагою файлової моделі даних є досягнення високої швидкості обробки інформації та можливість масштабування (розміщення на багатьох комп'ютерах мережі), що використовується в сучасних NoSQL базах даних.

### 2.2.2. Ієрархічна модель

*Ієрархічна модель* будується у вигляді ієрархічної деревоподібної структури, у якій для кожного головного об'єкта існує кілька підлеглих, а для кожного підлеглого об'єкта може бути тільки один головний. На найвищому рівні ієрархії перебуває кореневий об'єкт. Прикладом ієрархічної структури даних може бути організація каталогів на диску, різного роду класифікації, структура державної влади тощо.

Зв'язки між елементами в ієрархічній моделі даних можна описати за допомогою орієнтованого графа (перевернене дерево) (рис. 2.3).

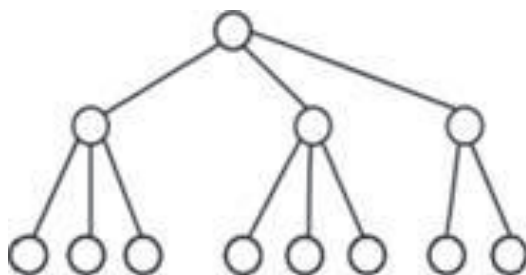


Рис. 2.3. Подання зв'язків в ієрархічній моделі

Ієрархічна БД є упорядкованою сукупністю екземплярів даних типу "дерево" (дерев), що містять екземпляри типу "запис" (записи). Часто відношення спорідненості між типами переносять на відношення між самими записами. Поля записів зберігають власне числові або символічні значення, складають основний зміст БД.

Обхід всіх елементів ієрархічної БД звичайно проводиться зверху вниз і зліва направо.

Основними інформаційними одиницями є поле, сегмент (запис), зв'язок, БД.

**Поле даних** (або просто "поле", в деяких виданнях, аналогічно мережній моделі, також називається "елемент") – мінімальна іменована одиниця даних, доступна користувачу за допомогою СКБД.

**Сегмент** або **запис** становить основну одиницю обробки БД: записи запам'ятовуються, витягуються, видаляються. Визначають тип і екземпляр запису (сегмента).

Тип запису – це іменована сукупність полів даних із зазначенням їх типів.

Екземпляр запису (або просто запис) – деяка сукупність значень елементів в послідовності, що відповідає визначенню типу. Іншими словами, тип запису задає всю множину подібних об'єктів, а екземпляр – конкретний об'єкт з цієї множини. Наприклад, тип об'єкта (тип сегмента) – стіл, тоді екземпляр – той конкретний стіл, за яким ви працюєте, що описується його набором характеристик (колір, розміри, матеріал і т.д.).

Для того, щоб можна було однозначно розрізнити записи, кожен тип запису повинен мати ключ – набір полів, однозначно ідентифікує екземпляр запису. Наприклад, в запису, що описує людину, таким ключем може бути номер паспорта або ідентифікаційний код.

**Зв'язок** – ієрархічне відношення між записами двох типів. Зв'язки при графічному зображенні позначаються дугами орієнтованого графа, типи записів – вершинами. Таке зображення структури БД називається діаграмою Бахмана.

*Тип зв'язку* визначається його ім'ям і задає властивості, загальні для всіх примірників зв'язку даного типу. Примірник зв'язку задається логічно вихідним записом ("власником") і множиною (можливо порожньою) підлеглих записів. Таким чином, кожному підпорядкованому запису в ієрархічній моделі може відповідати тільки один вихідний і одному вихідному запису може відповідати кілька підлеглих.

В ієрархічній моделі сегменти і зв'язки між ними створюють деревовидні структури (дерева). У кожному дереві існує тільки один запис, який не пов'язаний ні з будь-яким вихідним записом, – він називається кореневим. Таким чином, дерево це сукупність кореневого запису і множини підлеглих записів.

Схема ієрархічної БД є сукупністю окремих дерев, кожне дерево в рамках моделі називається *фізичною базою даних*. Кожна фізична БД задовольняє наступним ієрархічним обмеженням:

- в кожній фізичній БД існує один кореневий сегмент, тобто сегмент, у якого немає логічно вихідного (батьківського) типу сегмента;
- кожен логічно вихідний сегмент може бути пов'язаний з довільним числом логічно підлеглих сегментів;
- кожен логічно підлеглий сегмент може бути пов'язаний тільки з одним логічно вихідним (батьківським) сегментом.

Основною одиницею роботи в ієрархічній моделі є запис, над ним можуть проводитися такі операції:

- операція *додати* (INSERT) дозволяє занести в БД новий запис (з урахуванням обмежень, пов'язаних з вимогами щодо унікальності ключа в рамках всієї БД для кореневого запису і серед "близнюків" – для підпорядкованого);
- операція *оновити* (UPDATE) дає можливість змінити значення даних попередньо витягнутого запису. Ключові дані запису не можуть піддаватися оновленню, в подібних випадках старий запис повинен бути видалено, а новий, що містить змінені дані, – збережено;
- операція *видалити* (DELETE) служить для виключення з БД деякого запису і всіх записів, підпорядкованих йому;
- операція *витягти* (GET) має кілька модифікацій – витягти за значенням ключа, витягти наступний запис (в порядку лівостороннього обходу дерева), витягти наступний, що задовольняє додатковій умові і т.д.

Треба відзначити, що в будь-якому випадку обробка починається з кореневого запису, і доступ до некоренових записів здійснюється за ієрархічним шляхом.

До *переваг* ієрархічної моделі даних відносяться ефективне використання пам'яті ЕОМ і непогані показники часу виконання основних операцій над даними. Ієрархічна модель даних зручна для роботи з ієрархічно впорядкованою інформацією.

Крім того, ієрархічні структури відносно просто створюються й підтримуються. Це важливо для ряду додатків, однак безліч даних за своєю природою не зв'язані в деревоподібні структури. У багатьох структурах даних один запис вимагає більше одного подання (тому доводиться розробляти способи об'єднання даних, які по-різному представляються різним користувачам, в одну загальну схему БД. У результаті виходять звичайно більш складні структури в порівнянні з деревоподібними.

**Недоліками** ієрархічної моделі є її громіздкість для обробки інформації з досить складними логічними зв'язками, а також складність розуміння для звичайного користувача.

Також програмне забезпечення, сконструйоване тільки для роботи з деревоподібними структурами, має обмежене застосування й нерідко сильно впливає на можливості збільшення обсягу й розвитку БД.

Принциповим для ієрархічного подання даних є те, що кожен екземпляр запису набуває свого змісту тільки тоді, коли він розглядається у своєму контексті; підлеглий екземпляр запису не може існувати без свого попередника по ієрархії (несиметричність або асиметрія).

Приклад, представлений на рис. 2.4, ілюструє використання ієрархічної моделі бази даних для опису БД "Підприємство".

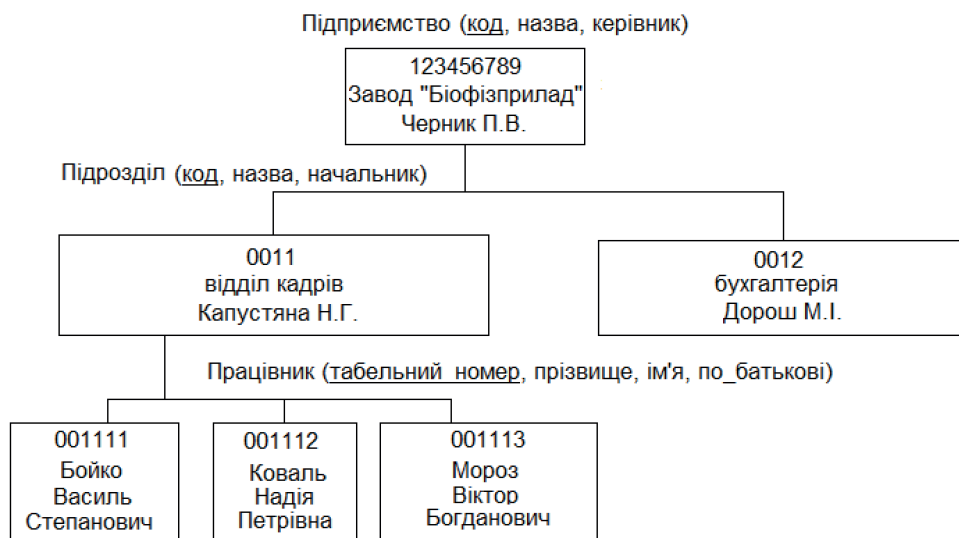


Рис. 2.4. Приклад ієрархічної структури БД

Тут Підприємство є предком Підрозділів, відповідно Підрозділи є нащадками Підприємства і одночасно предками Працівників. Особливістю є те, що кожен Працівник має лише одного предка, тобто відноситься тільки до одного підрозділу.

Також можна розглянути приклад БД "Університет", що також ілюструє ієрархічну структуру. За аналогією до попереднього прикладу, тут відношення предок-нащадок будуть Університет – Інститут – Спеціальність – Студент.

Тут кожен Студент має лише одну Спеціальність, кожна Спеціальність відноситься лише до одного з Інститутів.

Тривалий досвід використання ієрархічних систем показав, що вони досить ефективні лише для досить простих завдань, але вони практично не придатні для використання в складних системах з оперативною обробкою транзакцій і розподіленою архітектурою.

### 2.2.3. Мережна модель

Мережна модель даних дозволяє відображати різноманітні взаємозв'язки елементів даних у вигляді довільного графа, узагальнюючи тим самим ієрархічну модель даних. У мережній структурі при тих же основних поняттях (рівень, вузол, зв'язок) кожен елемент може бути пов'язаний з будь-яким іншим елементом (рис. 2.5).

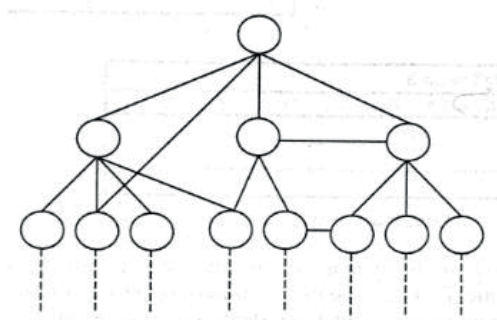


Рис. 2.5. Подання зв'язків в мережній моделі

У **мережній моделі** один і той же об'єкт може одночасно виступати як у ролі головного, так і підлеглого елемента. На формування зв'язків особливих обмежень не накладається. Це означає, що кожний об'єкт може брати участь у довільній кількості зв'язків. Якщо в ієрархічних структурах запис-нащадок міг

мати тільки один запис-предок, то в мережній моделі даних запис-нащадок може мати довільне число записів-предків (зведених батьків).

Фізичне розміщення даних у базах мережного типу може бути організовано практично тими ж методами, що і в ієрархічних базах даних.

Мережна модель визначає наступні операції над записами:

- **запам'ятати** (Store) – дозволяє внести в БД новий запис;
- **включити** в набір (Connect) – пов'язує існуючий підпорядкований запис з записом-власником;
- **оновити** (Modify) – дозволяє змінити значення елементів існуючих в БД записів;
- **знайти** (Find) – знайти запис за значенням первинного ключа. Також можна знайти запис, використовуючи групові відношення, в яких він бере участь;
- **витягти** (Get) – дозволяє витягувати запис (після того, як він знайдений);
- **видалити** (Erase) – видаляє поточний екземпляр запису;
- **виключити** з набору (Disconnect) – виключення поточного запису з поточного екземпляра набору.

Прикладом простої мережної структури БД може бути структура автобусних маршрутів (із будь-якого населеного пункту існують маршрути в інші).

Прикладом складної мережної структури може служити структура бази даних, що містить відомості про студентів, що беруть участь у науково-дослідних роботах (НДРС). Можлива участь одного студента в декількох НДРС, а також участь декількох студентів у розробці однієї НДРС.

Мережна модель даних будується за принципом "головний і підлеглий тип одночасно", тобто будь-який тип даних одночасно може породжувати кілька підлеглих типів (бути власником набору) і бути підлеглим для декількох головних (бути членом набору).

Мережний підхід до організації даних є розширенням ієрархічного. В ієрархічних структурах запис-нащадок повинен мати в точності одного предка; у мережній структурі даних нащадок може мати будь-яке число предків.

**Перевагою** мережної моделі даних є можливість ефективного реалізації за показниками витрат пам'яті та оперативності. У порівнянні з ієрархічною

моделлю мережна модель надає великі можливості в сенсі допустимості освіти довільних зв'язків.

**Недоліками** мережної моделі даних є висока складність і жорсткість схеми БД, побудованої на її основі, а також складність для розуміння і виконання обробки інформації в БД звичайним користувачем. Оскільки логіка процедури вибірки даних залежить від фізичної організації цих даних, то ця модель не є повністю незалежною від програми. Іншими словами, якщо необхідно змінити структуру даних, то потрібно змінити і додаток.

#### **2.2.4. Реляційна модель**

Реляційна модель описує, які дані можуть зберігатися в реляційних базах даних, а також способи маніпулювання такими даними.

Модель реляційної бази даних була вперше розроблена доктором Е.Ф. Коддом на початку 70-х років XX ст. як більш зручний засіб збереження, вибірки й маніпулювання даними, ніж ієрархічні й мережні бази даних.

Реляція – це назва в математиці двовимірної таблиці, що складається з рядків і стовпців даних.

Побудова реляційної моделі даних ґрунтується на тому розумінні, що будь-який набір даних може бути представлений у вигляді відношення, який оформлюється у формі прямокутної таблиці, де кожна клітинка може бути однозначно ідентифікована номером рядка й номером стовпця цієї таблиці.

Кожна з таблиць в реляційній базі даних містить інформацію про об'єкти певного виду. Дані про один об'єкт розміщуються у рядках таблиці (записах).

Кожна реляційна таблиця представляє собою двовимірний масив і має наступні властивості (рис. 2.6):

- кожен елемент таблиці (кожна клітинка) – один елемент даних;
- кожен стовпець в таблиці однорідний, тобто всі елементи в стовпці мають однаковий тип (числовий, символний і т. д.) і довжину;
- кожен стовпець має унікальне ім'я і не повинен повторюватися в тій же таблиці;



- всі рядки однієї таблиці мають однакову структуру – вони складаються з фіксованої кількості стовпців (полів), в яких зберігаються атрибути (характеристики) об'єкта;
- однакові рядки в таблиці відсутні, тобто кожен запис повинен бути унікальним, а це означає, що у даній таблиці такий же запис не повинен більше повторюватися і відрізнятися від інших записів бодай одним зі значень. Зазвичай унікальність запису забезпечує поле первинного ключа (на рис. 2.6 це поле «№ типу м'яча»), яке обов'язково повинно бути у кожній з таблиць реляційної БД;
- порядок проходження рядків і стовпців може бути довільним.

**Стовпці**  
(поля, характеристики, атрибути об'єкта)

| № типу м'яча | Призначення м'яча | Діаметр м'яча, мм | Вага м'яча, грам | Колір м'яча |
|--------------|-------------------|-------------------|------------------|-------------|
| 1            | Футбол            | 216               | 340              | білий       |
| 2            | Футбол            | 222               | 420              | червоний    |
| 3            | Волейбол          | 210               | 270              | білий       |
| 4            | Волейбол          | 213               | 270              | кольоровий  |
| 5            | Теніс великий     | 65                | 57               | жовтий      |

Рядки  
(записи, дані  
про один об'єкт)

Рис. 2.6. Структура таблиці реляційної бази даних

Одним з фундаментальних понять теорії реляційних БД є поняття нормалізації.

**Нормалізація** – покроковий процес розбиття одного відношення (на практиці – таблиці) відповідно до алгоритму нормалізації на декілька відношень на базі функціональних залежностей з метою усунення надмірності даних.

Оскільки зазвичай описати складні логічні структури даних з предметної області в рамках однієї таблиці не вдається, застосовують зв'язування таблиць між собою за допомогою вторинних ключів.

Поле вторинного ключа повинно бути у дочірній таблиці того ж типу, що і поле первинного ключа у батьківській.

Можуть мати місце наступні варіанти зв'язків:

- один до одного (1: 1):

- один до багатьох (1: N);
- багато до багатьох (N: M).

Варіант 1: 1 визначає такий зв'язок між типами записів, коли з кожним типом запису А пов'язаний тільки один тип запису В і навпаки. Прикладом такого зв'язку є зв'язок "автомобіль" – "власник автомобіля".

Варіант 1: N вказує, що з кожним типом запису А може бути пов'язана множина N типів записів В. Прикладом такого зв'язку типів записів є зв'язок "факультет" – "кафедра".

Варіант N: M вказує, що з кожним типом запису А може бути пов'язана множина M типів записів В і, навпаки, з кожним типом запису В може бути пов'язана множина N типів запису А. Прикладом такого зв'язку є зв'язок "студент" – "навчальний курс".

Атрибут називається **зовнішнім ключем**, якщо він є внутрішнім (первинним) ключем будь-якої іншої реляції. Щоб зв'язати дві реляційні таблиці, необхідно первинний ключ батьківської таблиці ввести до складу структури дочірньої таблиці, що називатиметься зовнішнім ключем, і що є точною копією первинного ключа батьківської таблиці.

Зв'язки між таблицями забезпечують можливість об'єднувати дані з декількох таблиць при виконанні запитів, записаних мовою SQL (Structured Query Language).

Приклад реляційної БД "Бібліотека" наведено на рис. 2.7.

|                             |                       |                       |
|-----------------------------|-----------------------|-----------------------|
| Читач                       | Каталог               | Формуляр              |
| № читацького квитка         | шифр книги            | номер запису          |
| прізвище, ім'я, по батькові | автор                 | № читацького квитка   |
| адреса                      | назва                 | шифр книги            |
| дата народження             | видавництво           | дата видачі           |
| місце праці (навчання)      | рік видання           | дата повернення       |
| дата реєстрації читача      | кількість примірників | прізвище бібліотекаря |
|                             | номер тому            |                       |
|                             | ціна                  |                       |
|                             | фото                  |                       |
|                             | шлях до фото          |                       |

Рис. 2.7. Реляційна БД "Бібліотека"

У даному прикладі дочірня таблиця "Формуляр" містить поля "№ читацького квитка" та "шифр книги" відповідно до батьківських таблиць "Читач» та "Каталог», які є вторинними ключами і дозволяють встановити зв'язки між цими таблицями.

Основною відмінністю пошуку даних в ієрархічних, мережних і реляційних базах даних є те, що ієрархічні і мережні моделі даних здійснюють зв'язок і пошук між різними об'єктами за структурою, а реляційні – за значенням атрибутів (наприклад, можна знайти всі записи, значення яких у полі «номер будинку» дорівнює 3, але не можна знайти 3-й рядок).

Фізичне розміщення даних в реляційних базах на зовнішніх носіях легко здійснюється за допомогою звичайних файлів.

Реляційні моделі характеризуються простотою структури даних, зручним для користувача табличним поданням і можливістю використання формального апарату алгебри відношень і реляційного числення для обробки даних. Саме простота і зрозумілість для користувача виявилися основною причиною широкого використання даної моделі даних.

Основними недоліками РМД є наступні: відсутність стандартних засобів ідентифікації окремих записів і складність опису ієрархічних і мережних зв'язків.

### **2.2.5. Постреляційна модель**

Класична реляційна модель передбачає неподільність даних, що зберігаються в полях записів таблиць. Це означає, що інформація в таблиці представляється в першій нормальній формі. Це значить, що дані не можуть бути розгорнуті в процесі подальшої обробки. Таке правило було закладено в основу реляційної алгебри при її розробці як математичної моделі даних. Подальші дослідження показали, що існує ряд випадків, коли обмеження класичної реляційної моделі суттєво заважають ефективній реалізації додатків.

Якщо допустити, що значення даних може саме складатися з підзначень, то в результаті виникає поняття багатозначного поля. Найпростіше розглядати набір багатозначних полів у таблиці як самостійну вкладену (nested) таблицю.

За умови, що вкладена таблиця задовольняє загальним критеріям (наприклад має унікальний ключ), природно відбувається розширення операторів реляційної алгебри. Така модель даних була названа постреляційною.

Багато методологій проектування даних дозволяють визначати багатозначні поля й потім видаляти в процесі нормалізації. При цьому таблиці перетворюються в першу нормальну форму або 1NF. Однак видалення багатозначних полів не завжди сприяє покращенню прикладних програм. У випадках, коли звичайна форма доступу до поля має на увазі звернення до всіх його значень, базі даних з таблицями в 1NF прийдеться проробити операцію з'єднання (Join) щоразу, коли потрібно одержати відповідні значення, що зберігаються в іншій таблиці. Зовсім очевидно, що в подібній ситуації зберігання значень фізично в багатозначних полях може забезпечити більш ефективний доступ до інформації.

На рис.2.8 наочно продемонстровані переваги зберігання даних у базах uniVerse, що відносяться до непершої нормальної форми, перед більш громіздким зберіганням у базах даних першої нормальної форми. Приклад представляє зберігання частини даних такого типового документа, як накладна.

| НАКЛАДНІ        |               |
|-----------------|---------------|
| Номер накладної | Номер покупки |
| 0373            | 8723          |
| 8374            | 8232          |
| 7364            | 8723          |

| НАКЛАДНІ-ТОВАРИ |              |                  |
|-----------------|--------------|------------------|
| Номер накладної | Назва товару | Кількість товару |
| 0373            | Сир          | 3                |
| 0373            | Риба         | 2                |
| 8374            | Лимонад      | 1                |
| 8374            | Сік          | 6                |
| 8374            | Печиво       | 2                |
| 7364            | Йогурт       | 1                |

| НАКЛАДНІ        |               |              |                  |
|-----------------|---------------|--------------|------------------|
| Номер накладної | Номер покупця | Назва товару | Кількість товару |
| 0373            | 8723          | Сир          | 3                |
|                 |               | Риба         | 2                |
| 8374            | 8232          | Лимонад      | 1                |
|                 |               | Сік          | 6                |
|                 |               | Печиво       | 2                |
| 7364            | 8723          | Йогурт       | 1                |

а)

б)

Рисунок 2.8. Структури збереження даних з багатозначними полями в uniVerse і в традиційних системах: а – Подання даних у 1NF та подальша нормалізація (2NF); б – Структура збереження даних в uniVerse NFNF (NF2)

Тоді як в uniVerse можна створювати багатозначні поля змінної довжини, всі замовлення можуть зберігатись в одній таблиці. При цьому для зберігання необхідно менше місця та при обробці не вимагається об'єднувати дані, що підвищує ефективність збереження даних.

Про таблиці, що містять багатозначні поля, говорять, що вони перебувають у **непершій** нормальній формі, або NF2 (Non First Normal Form). Як було помічене

раніше, за умови, що використовувані поля підкоряються певним правилам, що дозволяють звертатися до них, як до таблиці, вбудовані в інші таблиці, форма NF2 не порушує принципи реляційної алгебри. Більше того, така інформація повністю доступна, тому що розширені оператори, які працюють із таблицями NF2, дозволяють витягати вбудовані таблиці й розглядати дані як інформацію, що зробила з таблиць 1NF. І все-таки в багатьох випадках форма 1NF буде скоріше виключенням, а не правилом. У більшості випадків набагато більш ефективно здійснювати доступ до багатозначних полів одночасно з іншими даними, знаючи, що їх завжди можна витягати й розглядати як окрему таблицю в тих випадках, коли це може знадобитися.

Оскільки постреляційна модель допускає зберігання в таблицях ненормалізованих даних, виникає проблема забезпечення цілісності і несуперечності даних. Ця проблема вирішується включенням в СКБД механізмів, подібних збереженням процедур в клієнт-серверних системах.

Перевагою постреляційної моделі є можливість представлення сукупності зв'язаних реляційних таблиць однією постреляційною таблицею. Це забезпечує високу наочність представлення інформації й підвищення ефективності її обробки.

Недоліком постреляційної моделі є складність вирішення проблеми забезпечення цілісності й несуперечності збережених даних.

## **2.2.6. Багатовимірна модель. OLAP технологія**

Багатовимірна модель даних є вузькоспеціалізованою моделлю, яка призначена для оперативної аналітичної обробки інформації. В основі моделі лежить не двовимірна, як в реляційній моделі, а багатовимірна таблиця і багатовимірне логічне представлення структури інформації при описі даних і в операціях маніпулювання даними. У порівнянні з реляційною моделлю, багатовимірна організація даних має більш високу наочність та інформативність.

За визначенням, запропонованим Б. Коддом, *багатовимірне концептуальне представлення (multi-dimensional conceptual view)* – це множинна перспектива, що складається з кількох незалежних вимірювань, уздовж яких можуть бути

проаналізовані визначені сукупності даних. Одночасний аналіз за кількома вимірюваннями визначається як багатовимірний аналіз.

***On-Line Analytical Processing (OLAP)*** – технологія оперативної аналітичної обробки даних, що використовує методи і засоби для збору, зберігання та аналізу багатовимірних даних з метою підтримки процесів прийняття рішень.

Є 12 правил, що визначають OLAP, згідно із концепцією Б. Кодда.

1. ***Багатовимірність*** – OLAP-система на концептуальному рівні має представляти дані у вигляді багатовимірної моделі, що спрощує процеси аналізу і сприйняття інформації.
2. ***Прозорість*** – це спосіб організації даних, джерела, засоби обробки і зберігання.
3. ***Доступність*** – OLAP-система має надавати користувачу єдину, узгоджену і цілісну модель даних, забезпечуючи доступ до даних незалежно від того, як і де вони зберігаються.
4. ***Постійна продуктивність*** при розробленні звітів – продуктивність OLAP-систем не має дуже зменшуватися при збільшені кількості вимірювань, за якими виконується аналіз.
5. ***Клієнт-серверна архітектура*** – OLAP-система має бути здатна функціонувати у клієнт-серверному середовищі, оскільки більшість даних, які потрібно обробляти, зберігається децентралізовано. Серверний компонент інструменту OLAP має бути достатньо інтелектуальним і дозволяти будувати загальну концептуальну схему на основі узагальнення й консолідації різних логічних і фізичних схем корпоративних БД для забезпечення ефекту прозорості.
6. ***Рівноправність вимірювань*** – OLAP-система має підтримувати багатовимірну модель, в якій усі вимірювання рівноправні. За необхідності додаткові характеристики можуть бути надані окремим вимірюванням, але така можливість має бути надана будь-якому вимірюванню.
7. ***Динамічне управління розрідженими матрицями*** – OLAP-система має забезпечувати оптимальну обробку розріджених матриць. Швидкість

доступу повинна зберігатися незалежно від розташування осередків даних і бути постійною величиною для моделей, що мають різну кількість вимірювань і різний ступінь розрідженості даних.

8. **Підтримка розподіленого режиму доступу** – OLAP-система має надавати можливість працювати кільком користувачам спільно з однією аналітичною моделлю або створювати для них різні моделі з єдиних даних. При цьому можливі усі операції.
9. **Підтримка перехресних операцій** – OLAP-система має забезпечувати збереження функціональних відношень, описаних за допомогою певної формальної мови між осередками гіперкуба при виконанні будь-яких операцій зрізу, обертання, консолідації або деталізації. Система має автоматично виконувати перетворення встановлених відношень, не вимагаючи від користувача їх перевизначення.
10. **Інтуїтивна маніпуляція даними** – OLAP-система має надавати спосіб виконання операцій зрізу, обертання, консолідації і деталізації над гіперкубом без необхідності для користувачів здійснювати дії з інтерфейсом. Вимірювання, визначені в аналітичній моделі, мають містити всю необхідну інформацію для виконання операцій.
11. **Гнучкі можливості отримання звітів** – OLAP-система має підтримувати різні способи візуалізації даних, тобто звіти мають подаватися у будь-якому можливому вигляді.
12. **Необмежена розмірність** і кількість рівнів агрегації – одночасно може використовуватися близько 20 вимірювань.

У розвитку концепцій OLAP можна виділити наступні два напрямки:

- системи оперативної (транзакційної) обробки;
- системи аналітичної обробки (системи підтримки прийняття рішень).

Реляційні СКБД призначені для інформаційних систем *оперативної обробки* інформації і в цій області є вельми ефективні. У системах аналітичної обробки вони показали себе дещо неповороткими і недостатньо гнучкими. Більш ефективними тут виявляються багатовимірні СКБД (БСКБД).

БСКБД є вузькоспеціалізованими СКБД, призначеними для інтерактивної аналітичної обробки інформації.

Основні поняття БСКБД: агрегація, історичність і прогнозованість даних.

- **Агрегація** даних означає розгляд інформації на різних рівнях її узагальнення. В інформаційних системах ступінь детальності подання інформації для користувача залежить від його рівня: аналітик, користувач-оператор, керуючий, керівник.
- **Історичність** даних передбачає забезпечення високого рівня статичності (незмінності) власне даних і їх взаємозв'язків, а також обов'язковість прив'язки даних до часу.
- **Прогнозованість** даних передбачає задавання функцій прогнозування і застосування їх до різних часових інтервалів.

Багатомірність моделі даних означає не багатовимірність візуалізації цифрових даних, а багатовимірне логічне представлення структури інформації при описі і в операціях маніпулювання даними.

Якщо мова йде про багатовимірні моделі з розмірністю більше двох, то не обов'язково візуально інформація представляється у вигляді багатовимірних об'єктів (трьох-, чотирьох- і більш мірних гіперкубів). Користувачеві і в цих випадках більш зручно мати справу з двомірними таблицями або графіками. Дані при цьому є "вирізками" (точніше, "зрізами") з багатовимірного сховища даних, виконані з різним ступенем деталізації.

В порівнянні з реляційною моделлю багатовимірна організація даних володіє більш високою інформативністю (рис. 2.9).

До числа основних понять багатовимірної моделі відносяться вимір і комірка.

**Вимір** – це безліч однотипних даних, що утворюють одну з граней багатовимірної таблиці (аналог домену в реляційній моделі). Наприклад, дні, місяці, квартали, роки – це часові виміри, які найчастіше використовуються в аналізі. Прикладами географічних вимірів є міста, райони, регіони і т. ін. У багатовимірній моделі даних виміри відіграють роль індексів, які використовуються для ідентифікації конкретних значень (показників), що знаходяться в комірках гіперкуба.



| Найменування продукції | Квартал | Випуск |
|------------------------|---------|--------|
| Сир                    | I       | 20     |
| Сир                    | II      | 30     |
| Сир                    | IV      | 40     |
| Масло                  | I       | 25     |
| Масло                  | II      | 40     |
| Сметана                | III     | 35     |

а)

| Найменування продукції | Випуск по кварталах |    |     |    |
|------------------------|---------------------|----|-----|----|
|                        | I                   | II | III | IV |
| Сир                    | 20                  | 30 |     | 40 |
| Масло                  | 25                  | 40 |     |    |
| Сметана                |                     |    | 35  |    |

б)

Рис. 2.9. Реляційне та багатовимірне подання даних:

а – реляційне подання даних, б – багатовимірне подання даних

**Комірка** – це поле, значення якого однозначно визначається фіксованим набором вимірів. У багатовимірній СКБД Oracle Express Server показник може бути визначений, як:

- змінна (Variable) – значення таких показників один раз вводяться з будь-якого зовнішнього джерела або формуються програмно, а потім у явному вигляді зберігаються в багатовимірній базі даних;
- формула (Formula) – значення таких показників обчислюються за деякою задалегідь специфікованою формулою. Тобто для показника, який має тип «формула», у БД зберігається не його значення, а формула, за якою це значення може бути обчислено.

В існуючих багатовимірних СКБД використовуються два основні варіанти (схеми) організації даних:

- гіперкубічна;
- полікубічна.

У **гіперкубічній** схемі передбачається, що всі багатовимірні таблиці мають однакову розмірність і співпадаючі виміри (рис. 2.10). Очевидно, в деяких випадках інформація в БД може бути надмірною (якщо вимагати обов’язкове заповнення комірок).

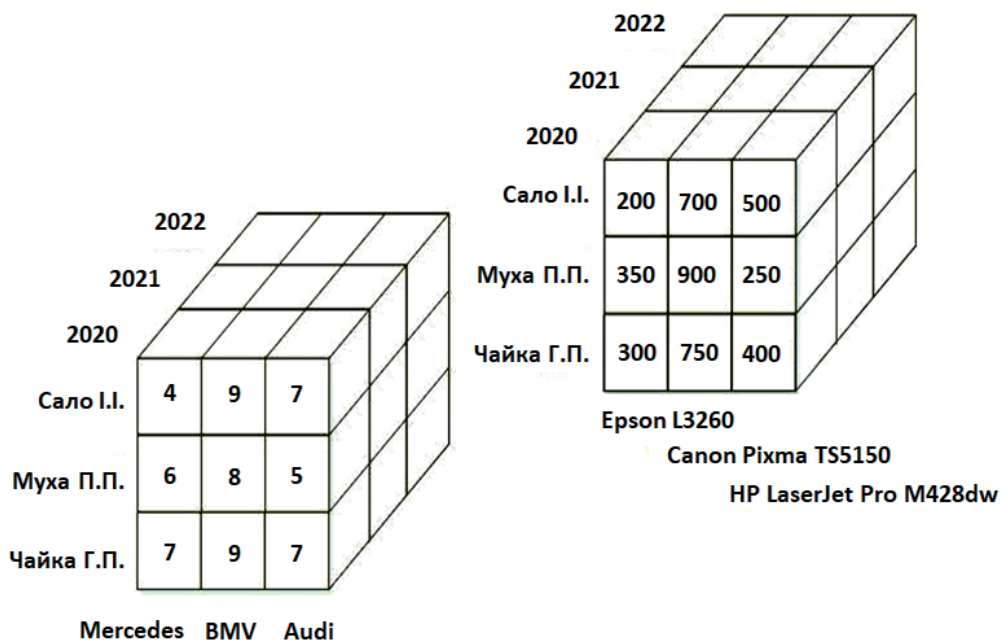


Рис. 2.10. Приклад гіперкубічної моделі

У **полікубічній** схемі може бути визначено декілька таблиць з різною розмірністю і різними вимірами (рис. 2.11). Системи, які підтримують полікубічну модель (прикладом є Oracle Express Server), припускають наявність у багатовимірній БД декількох гіперкубів з різною розмірністю та різними вимірами.

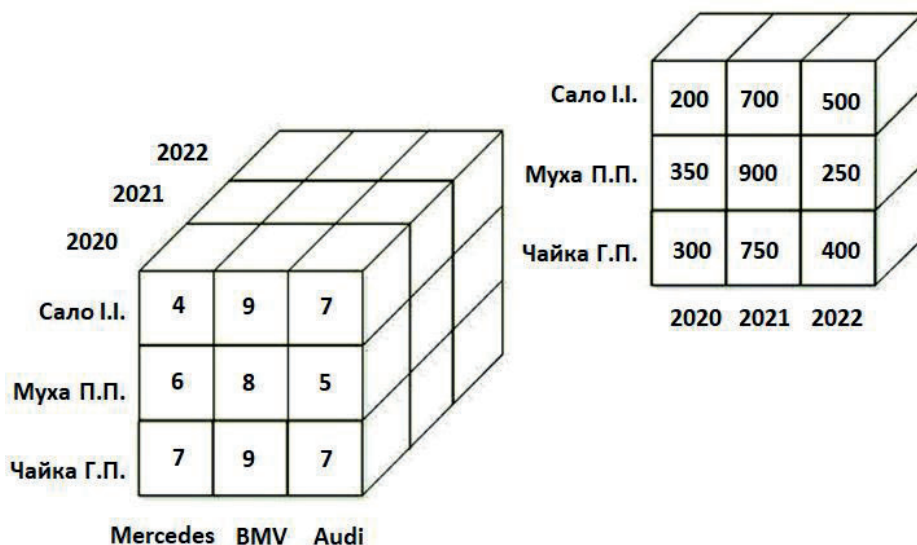


Рис. 2.11. Приклад полікубічної моделі

У разі багатовимірної моделі даних застосовується ряд спеціальних операцій, до яких відносяться наступні:

- формування зрізу;
- обертання;
- агрегація;
- деталізація.

Основною *перевагою* багатовимірної моделі даних є зручність і ефективність аналітичної обробки великих обсягів даних, пов'язаних з часом. При організації обробки аналогічних даних на основі реляційної моделі відбувається нелінійний зростання трудомісткості операцій залежно від розмірності БД та суттєве збільшення витрат оперативної пам'яті на індексацію.

*Недоліком* багатовимірної моделі даних є її громіздкість для найпростіших завдань звичайної оперативної обробки інформації.

Багатовимірні моделі даних мають *три важливі області застосування*, пов'язаних з проблематикою аналізу даних:

1. Сховища даних інтегрують для аналізу інформації з декількох джерел.
2. Системи оперативної аналітичної обробки дозволяють оперативно отримати відповіді на запити, що охоплюють великі обсяги даних у пошуках загальних тенденцій.
3. Додатки видобутку даних служать для виявлення знань за рахунок напівавтоматичного пошуку раніше невідомих шаблонів і зв'язків у базах даних.

З іншого боку, *є істотні обмеження на використання БСКБД*:

1. БСКБД не дозволяють працювати з великими базами даних. На сьогоднішній день їх реальна межа – 10-20 гігабайт.
2. БСКБД порівняно з реляційними малоефективно використовують зовнішню пам'ять. Комірки гіперкуба зберігаються в них у вигляді логічно впорядкованих масивів (блоків фіксованої довжини), причому саме такий блок є мінімальною одиницею, яка індексується. Хоч у багатовимірних СКБД блоки, які не містять жодного певного значення, не зберігаються, це вирішує проблему тільки частково. Оскільки дані

зберігаються у впорядкованому вигляді, невизначені значення не завжди віддаляються повністю, а лише в тому випадку, коли за рахунок вибору порядку сортування дані вдається організувати в максимально великі безперервні групи. Але порядок сортування, який частіше за все використовується в запитах, може не співпадати з порядком, в якому вони повинні бути відсортовані, а бути сформований з метою максимального усунення неіснуючих значень. Таким чином, при проектуванні багатовимірної БД часто доводиться жертвувати або швидкодією (а це одна з перших переваг та головна причина вибору саме багатовимірної СКБД), або зовнішньою пам'яттю.

3. Для багатовимірних СКБД поки відсутні єдині стандарти на інтерфейс, мови опису та маніпулювання даними.
4. Багатовимірні СКБД не підтримують реплікацію даних, яка часто використовується як механізм завантаження даних.

***Використання БСКБД виправдане тільки при умовах, коли:***

1. Рівень агрегації даних досить високий. При цьому об'єм початкових даних для аналізу не дуже великий (не більше кількох гігабайт).
2. Набір інформаційних вимірів стабільний (оскільки будь-яка зміна в їх структурі майже завжди вимагає повної перебудови гіперкуба).

### **2.2.7. Об'єктно-орієнтована модель**

Логічна структура об'єктно-орієнтованої моделі даних зовні схожа на структуру ієрархічної (рис. 2.16). Основна відмінність між ними полягає в методах маніпулювання даними.

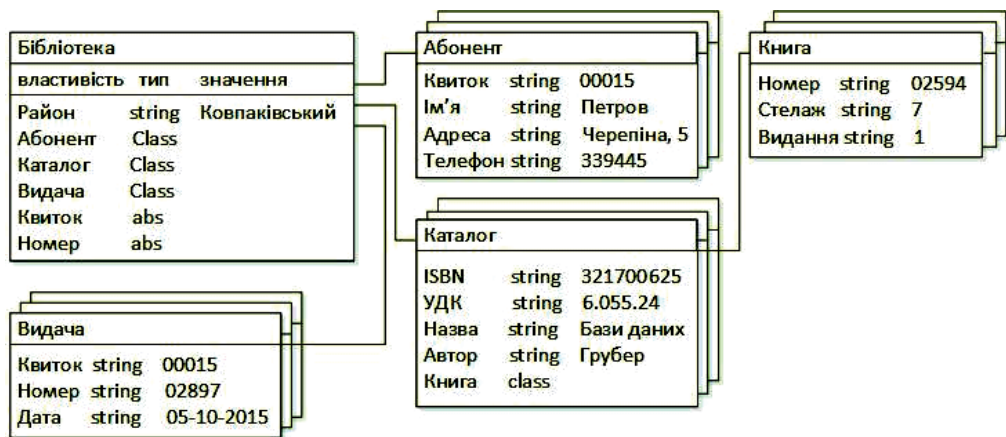


Рисунок 2.16. Об'єктно-орієнтована схема даних

Для виконання дій над даними в об'єктно-орієнтованій моделі застосовуються логічні операції, посилені об'єктно-орієнтованими механізмами:

- інкапсуляції;
- успадкування;
- поліморфізму.

Обмежено можуть застосовуватися операції, подібні командам мови SQL (наприклад, для створення БД).

Створення і модифікація БД супроводжуються автоматичним формуванням і подальшим коректуванням індексів (індексних таблиць), що містять інформацію для швидкого пошуку даних.

Розглянемо коротко поняття інкапсуляції, успадкування та поліморфізму стосовно об'єктно-орієнтованої моделі БД.

**Інкапсуляція** обмежує область видимості імені властивості межами того об'єкта, в якому воно визначене.

**Успадкування**, навпаки, поширює область видимості властивості на всіх нащадків об'єкта.

**Поліморфізм** в об'єктно-орієнтованих мовах програмування означає здатність одного і того ж програмного коду працювати з різнотипними даними. Іншими словами, він означає допустимість в об'єктах різних типів мати методи (процедури або функції) з однаковими іменами. Під час виконання об'єктної

програми одні й ті ж методи оперують з різними об'єктами залежно від типу аргументу.

Стандартизована об'єктно-орієнтована модель описана в рекомендаціях стандарту ODMG-93 (Object Database Management Group – Група управління об'єктно-орієнтованими базами даних).

В стандарті ODMG в якості базового засобу маніпулювання об'єктними базами даних пропонується *мова OQL* (Object Query Language). Це невелика, але досить складна мова запитів. Розробники в цілому характеризують її таким чином:

- OQL спирається на об'єктну модель ODMG (мається на увазі, що в ній підтримуються засоби доступу до всіх можливих структур даних, що допускаються в структурній частині моделі).
- OQL дуже близька до SQL/92. Розширення відносяться до об'єктно-орієнтованих понять, таких як складні об'єкти, об'єктні ідентифікатори, шляхові вирази, поліморфізм, виклик операцій і відкладене зв'язування.
- У OQL забезпечуються високорівневі примітиви для роботи з множинами об'єктів, але, крім того, є настільки ж ефективні примітиви для роботи зі структурами, списками і масивами.
- OQL є функціональною мовою, що допускає необмежену композицію операцій, якщо операнди не виходять за межі системи типів. Це є наслідком того факту, що результат будь-якого запиту володіє типом, що належить до моделі типів ODMG, і тому до результату запиту може бути застосований новий запит.
- OQL не є обчислювально повною мовою. Вона являє собою просту мову запитів.
- Оператори мови OQL можуть викликатися з будь-якої мови програмування, для якого в стандарті ODMG визначені правила зв'язування. І, навпаки, у запитах OQL можуть бути присутніми виклики операцій, запрограмованих на цих мовах.
- У OQL не визначаються явні операції оновлення, а використовуються виклики операцій, визначених в об'єктах для цілей оновлення.

- У OQL забезпечується декларативний доступ до об'єктів. З цієї причини OQL-запити можуть добре оптимізуватися.
- Можна легко визначити формальну семантику OQL.

В рамках цієї теми обмежимося лише одним характерним прикладом виконання запиту на отримання номерів керівників відділів і тих службовців їх відділів, зарплата яких перевищує 2000 грн.

```
SELECT DISTINCT STRUCT (ВДЛ_КЕР: D.ВДЛ_КЕР, СЛУ:
    (SELECT E
      FROM D.CONSISTS_OF AS E
      WHERE E.СЛУ_ЗАРП > 20000.00)
    )
FROM ВІДДІЛИ D
```

Тут передбачається, що для атомарного об'єктного типу ВІДДІЛ визначений екстент типу множини з ім'ям ВІДДІЛИ. У запиті перебираються всі існуючі об'єкти типу ВІДДІЛ, і для кожного такого об'єкта відбувається перехід по зв'язку до літеральної множини об'єктів типу СЛУЖБОВЕЦЬ, відповідних службовцям, які працюють в даному відділі. На основі цієї множини формується «усічена» множина об'єктів типу СЛУЖБОВЕЦЬ, в якій залишаються тільки об'єкти-службовці із зарплатнею, більшою від 2000.00 грн. Результатом запиту є літеральне значення-множина, елементами якого є значення-структури з двома літеральними значеннями, перше з яких є атомарне літеральне значення типу INTEGER, а друге – літеральне значення-множина з елементами-об'єктами типу СЛУЖБОВЕЦЬ.

Більш точно, результат запиту має тип

```
set < struct {integer ВДЛ_КЕР; bag <СЛУЖБОВЕЦЬ> СЛУ}>.
```

У сукупності результатом допустимих в OQL виразів запитів можуть бути:

- колекція об'єктів;
- індивідуальний об'єкт;
- колекція літеральних значень;
- індивідуальне літеральне значення.

Що ж стосується збереження цілісності, то вона підтримується, якщо між двома атомарними об'єктними типами визначається як зв'язок типу «один-до-багатьох». У цьому випадку об'єкти на стороні зв'язку «один» розглядаються як предки, а об'єкти на стороні зв'язку «багато» – як нащадки, і ООСКБД зобов'язана стежити за тим, щоб не утворювалися нащадки без предків.

**Перевагами** об'єктно-орієнтованої моделі є покращення можливості моделювання об'єктів реального світу. Об'єктні типи даних, а також об'єктні таблиці представляють потужний єдиний рівень інтерпретації об'єктів ділової сфери і дозволяють відмовитися від поділу на частини бізнес-даних для зберігання їх в БД при використанні реляційної моделі.

**Недоліками** об'єктно-орієнтованої моделі є висока поняттєва складність, незручність обробки даних, низька швидкість виконання запитів, а також відсутність стандарту об'єктно-орієнтованої моделі через недостатню її теоретичну розробку.

У кінці ХХ століття існували лише експериментальні прототипи об'єктно-орієнтованих систем управління базами даних. В наш час такі системи отримали широке поширення, зокрема, до них належать такі СКБД: POET (POET Software), Jasmine (Computer Associates), Versant (Versant Technologies), 02 (Ardent Software), ODB-Jupiter (науково-виробничий центр «Інтелтек Плюс»), а також Iris, Orion і Postgres.

### **2.2.8. Об'єктно-реляційна модель даних**

У зв'язку з недостатньою розробленістю об'єктно-орієнтованої моделі на практиці застосовується об'єктно-реляційна модель, що є ніби сумішшю реляційної та об'єктно-орієнтованої методологій для представлення даних. Ця модель є розширеною реляційною моделлю, в якій зняті обмеження неподільності даних, які зберігаються в записах таблиць.

Допускаються багатозначні поля – поля, значеннями яких є самостійні таблиці, вбудовані в основну таблицю. Крім цього підтримуються такі концепції об'єктно-орієнтованого програмування, як «абстракція», «клас», «екземпляр», «інкапсуляція», «метод», «перевантаження» і «повідомлення».



Хоча спадкування і є однією з найбільш важливих характеристик об'єктів, але в об'єктно-реляційній моделі воно не підтримується.

В модель вводиться спеціальний об'єктний тип, за допомогою якого можна створити абстрактний тип даних будь-якого ступеня складності.

Використовуючи вкладені об'єктні типи, можна створювати структури, в яких використовуються всі види зв'язків: «один до одного», «один до багатьох» і навіть «багато до багатьох». Хоча це і може призвести до певної надмірності, такий підхід дає переваги в порівнянні з використанням множини нормалізованих таблиць в чисто реляційній моделі.

Покажемо це на прикладі. Розглянемо успадкування таблиць в об'єктно-реляційній СКБД postgresql v. 9.2. Нехай маємо дві таблиці cities (міста) та capitals (столиці). Створимо ці таблиці (тобто задамо їх імена та схеми):

```
CREATE TABLE cities (name text, population real, altitude int );
```

```
CREATE TABLE capitals (state char(2) ) INHERITS (cities);
```

Таблиця capitals успадковує всі атрибути батьківської таблиці cities. Тоді, коли додається запис в таблицю capitals, цей же запис додається в таблицю cities (без атрибута state). Як відомо, в ООП при створенні об'єкта похідного класу цей об'єкт не створюється в батьківському класі.

**Перевагою** об'єктно-реляційної моделі є можливість використання існуючих реляційних баз даних з знову розроблюваними об'єктними додатками.

До **недоліків** моделі можна віднести складність вирішення проблеми забезпечення цілісності і несуперечності збережених даних.

### 2.3. Проблеми маніпулювання даними та обмеження цілісності даних

*Цілісність інформації* (англ. data integrity, information integrity) – властивість інформації, яка полягає в тому, що інформація не може бути модифікована неавторизованим користувачем і/або процесом. Інформація зберігає цілісність, якщо дотримуються встановлені правила її модифікації та видалення.

У кожний момент часу існування бази даних відомості, що містяться в ній, повинні бути повними, несуперечливими і адекватно відображати предметну область. У цьому й полягає її цілісність.

## NULL-значення

Основне призначення баз даних полягає в тому, щоб зберігати і надавати інформацію про реальний світ. Для представлення цієї інформації в базі даних використовуються звичні для програмістів типи даних – рядкові, чисельні, логічні і т.п. Однак у реальному світі часто зустрічається ситуація, коли дані невідомі або не повні. Наприклад, місце проживання або дата народження людини можуть бути невідомі (База даних розшукуваних злочинців). Якщо замість невідомої адреси доречно було б вводити порожній рядок, то що вводити замість невідомої дати? Відповідь – порожню дату – не цілком задовільний, тому що найпростіший запит "видати список людей у порядку зростання дат народження" дасть явно неправильних відповідь. Для того щоб обійти проблему неповних або невідомих даних, в базах даних можуть використовуватися типи даних, поповнені так званим null-значенням.

Null-значення – це, власне, не значення, а якийсь маркер, який показує, що значення невідоме. Таким чином, у ситуації, коли можлива поява невідомих або неповних даних, розробник має на вибір два варіанти.

Перший варіант полягає в тому, щоб обмежитися використанням звичайних типів даних і не використовувати null-значення, а замість невідомих даних вводити або нульові значення, або значення спеціального виду – наприклад, домовитися, що рядок "Адреса невідома" і є ті дані, які потрібно вводити замість невідомої адреси. У будь-якому випадку на користувача (або на розробника) лягає відповідальність на правильне трактування таких даних. Зокрема, може знадобитися написання спеціального програмного коду, що у потрібних випадках "виловлював" би такі дані. Проблеми, що виникають при цьому очевидні – Не всі дані стають рівноправні, потрібен додатковий програмний код, що "відслідковує" цю нерівноправність, в результаті чого ускладнюється розробка і супровід додатків.

Другий варіант полягає у використанні null-значень замість невідомих даних. За здавалося б природністю такого підходу ховаються менш очевидні і більш глибокі проблеми. Найбільшою проблемою є необхідність використання тризначної логіки при оперуванні з даними, які можуть містити null-значення. У

цьому випадку при неакуратному формулюванні запитів, навіть самі природні запити можуть давати неправильні відповіді. Є більш фундаментальні проблеми, пов'язані з теоретичним обґрунтуванням коректності введення null-значень, наприклад, незрозуміло взагалі, чи входять null-значення в домени чи ні.

Практично всі реалізації сучасних реляційних СКБД дозволяють використовувати null значення, незважаючи на їх недостатню теоретичну обґрунтованість.

Тим не менш, приведемо тут опис тризначної логіки, необхідної для роботи з null-значеннями.

### **Тризначна логіка (3VL)**

Оскільки null-значення позначає насправді той факт, що значення невідоме, то будь-які алгебраїчні операції (додавання, множення, конкатенація рядків і т.д.) повинні давати також невідоме значення, тобто null. Дійсно, якщо, наприклад, вага деталі невідома, то невідомо також, скільки важать 10 таких деталей. При порівнянні виразів, що містять null-значення, результат також може бути невідомий, наприклад, значення істинності для виразу є null, якщо один або обидва аргументи є null. Таким чином, визначення істинності логічних виразів базується на *тризначній логіці* (threevalued logic, 3VL), в якій крім значень Т – ІСТИНА і F – ХИБА, введено значення U – НЕВІДОМО. Логічне значення U – це те ж саме, що і null-значення.

### **Контрольні запитання**

1. Що таке модель бази даних?
2. Які є типи моделей даних?
3. Для чого використовується міфілогічне моделювання БД?
4. Що таке даталогічна модель даних у БД?
5. Наведіть недоліки файлової моделі даних.
6. Охарактеризуйте зв'язки в ієрархічній моделі даних.
7. Як відображаються взаємозв'язки елементів даних в мережній моделі?
8. Наведіть переваги і недоліки мережної моделі даних.
9. Які властивості реляційних таблиць?

10. Які варіанти зв'язків за допомогою вторинних ключів між таблицями?
11. Для чого використовується нормалізація?
12. Яка модель даних передбачає поняття багатозначного поля?
13. Що таке OLAP технологія?
14. Наведіть правила, що визначають OLAP, згідно із концепцією Кодда.
15. Що таке БСКБД?
16. Охарактеризуйте основні поняття БСКБД.
17. Наведіть основні схеми організації даних в БСКБД.
18. Які є операції у разі багатовимірної моделі даних?
19. Наведіть переваги та недоліки об'єктно-орієнтованої моделі.
20. Які відмінності між об'єктно-реляційною та об'єктно-орієнтованою моделлю даних?
21. Для чого використовуються null-значення в БД?

## Тема 3. Реляційна модель баз даних

### 3.1. Обмеження цілісності даних

Говорячи про цілісність даних реляційних БД, будемо виходити з таких припущень:

1) в будь-який момент часу база даних містить певний набір значень даних;

2) просто набір значень даних не має сенсу, якщо він не відображає дійсність, тобто не є поданням частини реального світу.

Таким чином, визначення бази даних потребує розширення, що включає правила забезпечення цілісності, призначення яких в тому, щоб інформувати СКБД про обмеження реального світу.

Обмеження цілісності можуть бути специфічними для конкретної БД. Наприклад, група, в яку входить студент, повинна бути з певного списку існуючих груп. Однак в реляційній моделі існують і *загальні правила забезпечення цілісності*, що описуються первинними (потенційними) і зовнішніми ключами.

Нехай  $R$  – реляційне відношення. Тоді потенційним ключем (позначимо його  $K$ ) буде називатися підмножина атрибутів  $R$ , що володіє наступними властивостями:

1) унікальність – у відношенні  $R$  не може бути двох різних кортежів з однаковим значенням  $K$ ;

2) ненадмірність – жодна з відмінних від підмножин  $K$ , не володіє властивістю унікальності.

Потенційний ключ, що складається з одного атрибута, називається простим. Наприклад, простий потенційний ключ може бути Номер студентського квитка (вважаємо, що номер студентського квитка унікальний) або Ідентифікаційний код (він таки унікальний). Потенційний ключ, що складається з декількох атрибутів, називається складовим.

Потенційні ключі важливі через те, що вони забезпечують основний механізм адресації на рівні кортежів. Інакше кажучи, єдиний гарантований

системою спосіб вказати на кортеж – вказати значення його потенційного ключа або одного з таких ключів, якщо їх декілька.

У реляційної моделі прийнято один з потенційних ключів вибирати в якості **первинного ключа**. Інші потенційні ключі будуть альтернативними ключами. З точки зору реляційної моделі, який з ключів вибрати в якості первинного – не принципово. Однак багато СКБД при зберіганні даних впорядковують записи за значенням первинного ключа. В цьому випадку дуже важливо, щоб операція порівняння значень первинного ключа виконувалася максимально швидко. Тому краще буде використовувати простий ключ з цілочисельними значеннями. Якщо серед наявних атрибутів відповідного на таку роль немає, нерідко спеціально для створення первинного ключа вводять атрибут, якому присвоюють значення цілочисельного лічильника. Подібний первинний ключ називають *сурогатним*, на протипагу *природному*, який формується з атрибутів, що характеризують реальний об'єкт.

Відношень у БД може бути досить багато, причому деякі з них можуть бути пов'язані одні з одними. При операціях з ними велике значення мають **зовнішні ключі**.

Нехай  $R_I$  – базове відношення. Підмножина атрибутів  $FK$  відношення  $R_I$  буде називатися зовнішнім ключем, якщо виконуються наступні умови:

1) існує базове відношення  $R$  (причому  $R_I$  і  $R$  не обов'язково різні) з потенційним ключем  $K$ ;

2) кожне значення  $FK$  в поточному значенні відношення  $R_I$  завжди збігається зі значенням ключа  $K$  деякого кортежу в поточному значенні відношення  $R$ .

Зовнішній ключ буде простим тоді і тільки тоді, коли відповідний потенційний ключ простий. Аналогічно для складеного зовнішнього ключа.

Проблема забезпечення того, що БД не включає невірних значень зовнішніх ключів, називається **проблемою забезпечення посилальної цілісності**.

Правило посилальної цілісності полягає в тому, що БД не повинна містити неузгоджених значень зовнішніх ключів. При реалізації даного правила необхідно відповісти як мінімум на два питання:

1. Що має статися при спробі видалити об'єкт посилання зовнішнього ключа?

2. Що має статися при спробі оновити потенційний ключ, на який посилається зовнішній ключ?

У першому випадку можна обмежити дію – заборонити операцію видалення даних до того моменту, поки існують посилання на значення зовнішніх ключів. Інший варіант – "каскадувати" видалення даних, тобто видалити кортеж з відповідним значенням потенційного ключа і всі кортежі, зв'язані зовнішніми ключами. Аналогічно можна вчинити і в другому випадку – обмежити оновлення або каскадувати зміни.

На практиці потрібно дуже акуратно використовувати каскадування видалення записів. Одиначна помилка користувача в цьому випадку може завдати великої шкоди, так як разом з записом, що видаляється, може бути видалено багато інших записів в пов'язаних таблицях.

### 3.2. Реляційна алгебра

При розробленні реляційної моделі Е. Кодд ввів реляційну алгебру, яка складається з набору операторів, які використовують відношення в якості операндів і повертають відношення в якості результату. Вона включає *вісім операцій*:

- традиційні операції над множинами – об'єднання, перетин, віднімання, декартовий добуток;
- спеціальні реляційні операції – вибірку, проекцію, з'єднання, розподіл.

Говорячи про реляційну алгебру, не можна обійти увагою властивість замкнутості. Воно полягає в тому, що результат однієї операції може використовуватися в якості вихідних даних для іншої. Таким чином, можна використовувати вкладені вирази.

Введемо ще одне визначення. Два реляційних відношення сумісні за типом, якщо:

- кожне з них має одну і ту ж множину атрибутів;

- відповідні атрибути (тобто атрибути з однаковими іменами) визначені на одному і тому ж домені.

### 3.2.1. Традиційні операції над множинами

**1) Об'єднання.** Відношення з тим же заголовком, що у вихідних, і з тілом, що складається з множини всіх кортежів, що належать A, або B, або їм обом, називається *об'єднанням* двох сумісних по типу відношень A і B.

Оскільки в результуючому наборі кортежів наявність двох однакових кортежів неможлива, операція об'єднання може супроводжуватися видаленням дублікатів. Це станеться в тому випадку, якщо зустрінуться кортежі, що повністю збігаються у відношеннях A і B, що об'єднуються.

Правила запису реляційних операцій у різних авторів дещо відрізняються. Для запису об'єднання використовують, як правило, одне з двох позначень:

$A \cup B$

A UNION B

Перша з цих форм записи аналогічна прийнятої в теорії множин, друга ближча до форми, яка використовується в мовах запитів до БД.

Приклад:

відношення A

| idSt | ПІБ       | Група |
|------|-----------|-------|
| 123  | Сало І.І. | 382   |
| 124  | Муха П.П. | 382   |

відношення B

| idSt | ПІБ        | Група |
|------|------------|-------|
| 123  | Сало І.І.  | 382   |
| 125  | Чайка Г.П. | 383   |

Об'єднання A U B

| idSt | ПІБ        | Група |
|------|------------|-------|
| 123  | Сало І.І.  | 382   |
| 124  | Муха П.П.  | 382   |
| 125  | Чайка Г.П. | 383   |

**2) Перетин.** Відношення з тим же заголовком, що у вихідних, і тілом, що складається з множини всіх кортежів, що належать обом відношенням одночасно, називається *перетином* двох сумісних по типу відношень A і B. Записується операція так:

$A \cap B$

A INTERSECT B



Приклад:

відношення А

| idSt | ПІБ       | Група |
|------|-----------|-------|
| 123  | Сало І.І. | 382   |
| 124  | Муха П.П. | 382   |

відношення В

| idSt | ПІБ        | Група |
|------|------------|-------|
| 123  | Сало І.І.  | 382   |
| 125  | Чайка Г.П. | 383   |

Перетин А ∩ В

| idSt | ПІБ       | Група |
|------|-----------|-------|
| 123  | Сало І.І. | 382   |

**3) Віднімання.** Відношення з тим же заголовком, що у вихідних, і тілом, що складається з множини всіх кортежів, що належать А і не належать В, називається *відніманням* двох сумісних по типу відношень А і В. Форми запису:

$A - B$

A MINUS B

Приклад:

відношення А

| idSt | ПІБ       | Група |
|------|-----------|-------|
| 123  | Сало М.Ф. | 382   |
| 124  | Муха П.П. | 382   |

відношення В

| idSt | ПІБ        | Група |
|------|------------|-------|
| 123  | Сало М.Ф.  | 382   |
| 125  | Чайка Г.П. | 383   |

Віднімання А – В

| idSt | ПІБ         | Група |
|------|-------------|-------|
| 124  | Коваль В.П. | 382   |

відношення В

| idSt | ПІБ        | Група |
|------|------------|-------|
| 123  | Сало М.Ф.  | 382   |
| 125  | Чайка Г.П. | 383   |

відношення А

| idSt | ПІБ       | Група |
|------|-----------|-------|
| 123  | Сало М.Ф. | 382   |
| 124  | Муха П.П. | 382   |

Віднімання В – А

| idSt | ПІБ        | Група |
|------|------------|-------|
| 125  | Чайка Г.П. | 383   |

Необхідно звернути увагу на наступні моменти. По-перше, успадкування ключів і імен атрибутів дозволило визначити заголовок результуючих відношень. По-друге, в обох відношеннях зустрічається кортеж <123, Сало М.Ф., 382>, а в результаті об'єднання присутній в єдиному екземплярі, в результаті перетину – є єдиним кортежем в тілі відношення, в результаті віднімання – відсутній. Так само як при відніманні дійсних чисел, результат  $A - B$  у загальному випадку не дорівнює результату  $B - A$ .

**4) Декартовий добуток** двох відношень А і В (де А і В не мають спільних імен атрибутів) визначається як відношення з заголовком, який є об'єднанням (

"зчепленням") двох заголовків вихідних відношень A і B, і тілом, що складається з множини усіх можливих впорядкованих пар, у яких перша компонента належить множині X, а друга – множині Y. Форми запису:

$A \times B$  (але не  $A \cdot B$  або  $AB$ )

Приклад:

| відношення A |            | відношення B | Декартів добуток $A \times B$ |            |         |
|--------------|------------|--------------|-------------------------------|------------|---------|
| idSt         | ПІБ        | Предмет      | idSt                          | ПІБ        | Предмет |
| 123          | Сало М.Ф.  | ООП          | 123                           | Сало М.Ф.  | ООП     |
| 124          | Муха П.П.  | ОБД          | 123                           | Сало М.Ф.  | ОБД     |
| 125          | Чайка Г.П. |              | 124                           | Муха П.П.  | ООП     |
|              |            |              | 124                           | Муха П.П.  | ОБД     |
|              |            |              | 125                           | Чайка Г.П. | ООП     |
|              |            |              | 125                           | Чайка Г.П. | ОБД     |

Можна переконатися з наведеного прикладу, що ступінь добутку дорівнює сумі ступенів вихідних відношень ( $2 + 1$ ), а кардинальне число дорівнює добутку кардинальних чисел вихідних відношень ( $3 \times 2$ ). В результуючому відношенні отриманий складовий первинний ключ {StudID, Предмет}, тоді як у вихідних відношеннях він був простим.

3.2.2. Спеціальні реляційні операції

5) *Вибірка* – це скорочена назва  $\theta$ -вибірки (тета-вибірки).

$\theta$ -вибіркою з відношення A за атрибутами X і Y (в цьому порядку) називається відношення, що має той же заголовок, що і відношення A, і тіло, що містить множину T всіх кортежів з відношення A, для яких перевірка умови " $X \theta Y$ " дає значення "істина", а  $\theta$  позначає будь-який логічний оператор порівняння ( $=, >, <, \dots$  і т.д.).

Форми запису:

$A [X \theta Y]$

$A \text{ WHERE } X \theta Y$

Приклад:

відношення STUDENTS

| idSt | ПІБ        | Група |
|------|------------|-------|
| 123  | Сало М.Ф.  | 382   |
| 124  | Муха П.П.  | 382   |
| 125  | Чайка Г.П. | 383   |

Вибірка STUDENTS [Група = 382]

| idSt | ПІБ       | Група |
|------|-----------|-------|
| 123  | Сало М.Ф. | 382   |
| 124  | Муха П.П. | 382   |

Якщо у відношеннях А і В виявилися однакові атрибути, щоб їх розрізнити, імена таких атрибутів формуються у вигляді <Ім'я відношення>.<Ім'я атрибута> Наприклад, у відношенні А є атрибут FIO і в відношенні В є також атрибут FIO, тоді їх необхідно записувати у вигляді A.FIO і B.FIO.

Для операції вибірки необхідно, щоб атрибути Х і Y були визначені на одному домені, а оператор  $\theta$  повинен мати для нього сенс. Константа може бути вказана як замість одного з атрибутів, так і замість обох.

Розширимо визначення операції  $\theta$ -вибірки. Будемо вважати, що умова порівняння для  $\theta$ -вибірки може містити довільне число простих порівнянь, об'єднаних логічними зв'язками. При необхідності можуть використовуватися круглі дужки.

Для логічних зв'язків також використовуються альтернативні форми запису, одна з яких ближче до прийнятої в математичній логіці, інша – до форми записи в мовах запитів до БД.

Побудуємо, наприклад, складений запит на вибірку:

STUDENTS WHERE ((Група = 382) AND (idSt <124))

або, використовуючи інший формат запису:

STUDENTS [Група = 382 & idSt <124]

Результатом стане відношення, що містить кортеж

<123, Сало М.Ф., 382>.

**6) Проекцією** відношення А за атрибутами V, W, ..., Z (де кожен з атрибутів належить відношенню А) називається відношення з заголовком {V, W, ... Z} і тілом, що містить множину всіх кортежів {V: v, W: w, ... Z: z}, для яких відносно

А існував кортеж із значенням атрибута V, рівним V, атрибута W – w, ... і атрибута Z – z. Таким чином, в результаті проєкції частина атрибутів вихідного відношення виключається, після чого може знадобитися видалення повторюваних кортежів.

Іншими словами, операція проєкції є вибіркою з кожного кортежу відношення значень атрибутів, що входять до списку A, і видалення з отриманих відношень повторюваних рядків.

Проекція відношень A за атрибутами V, W, Z позначається як

$A[V, W, Z]$

PROJECT A {x, y, z}

Приклад 1:

відношення STUDENTS

| idSt | ПІБ        | Група |
|------|------------|-------|
| 123  | Сало М.Ф.  | 382   |
| 124  | Муха П.П.  | 382   |
| 125  | Чайка Г.П. | 383   |
| 126  | Сало М.Ф.  | 384   |

Проекція STUDENTS [ПІБ]

| ПІБ        |
|------------|
| Сало М.Ф.  |
| Муха П.П.  |
| Чайка Г.П. |

Приклад 2:

відношення STUDENTS

| Ім'я   | Ріст | Вага |
|--------|------|------|
| Іван   | 182  | 80   |
| Петро  | 177  | 77   |
| Микола | 175  | 81   |
| Семен  | 183  | 80   |
| Данило | 182  | 80   |

PROJECT STUDENTS {Ріст, Вага}

| Ріст | Вага |
|------|------|
| 182  | 80   |
| 177  | 77   |
| 175  | 81   |
| 183  | 80   |

7) **Об'єднання.** Визначено два різновиди операції об'єднання:

- природне об'єднання;

- $\theta$ -об'єднання (тета-об'єднання).

*Природне об'єднання.* Нехай відношення  $A$  і  $B$  мають заголовки  $\{X, Y\}$  і  $\{Y, Z\}$ , де атрибути  $X, Y, Z$  можуть бути складовими. Іншими словами, тільки атрибут  $Y$  (або підмножина атрибутів, що позначається  $Y$ ) присутній як в першому, так і в другому відношенні. Атрибути з однаковими іменами визначені на одних доменах. Тоді природним об'єднанням відношень  $A$  і  $B$  називається відношення з заголовком  $\{X, Y, Z\}$  і тілом, що містить всі кортежі  $\{X: x, Y: y, Z: z\}$ , для яких відносно  $A$  значення атрибута  $X$  дорівнює  $x$ , значення атрибута  $Y$  –  $y$ , а в відношенні  $B$  значення атрибута  $Y$  одно  $y$ , атрибуту  $Z$  –  $z$ .

Позначається природне об'єднання  $A$  і  $B$  як:

$A \text{ JOIN } B$

Приклад:

відношення STUDENTS

| idSt | ПІБ        | Група |
|------|------------|-------|
| 123  | Сало М.Ф.  | 382   |
| 124  | Муха П.П.  | 382   |
| 125  | Чайка Г.П. | 383   |

відношення GROUPS

| Староста   | Група |
|------------|-------|
| Муха П.П.  | 382   |
| Бекас Б.О. | 384   |

Природне об'єднання STUDENTS JOIN GROUPS

| idSt | ПІБ       | Група | Староста  |
|------|-----------|-------|-----------|
| 123  | Сало М.Ф. | 382   | Муха П.П. |
| 124  | Муха П.П. | 382   | Муха П.П. |

У наведеному прикладі через те, що в першому відношенні немає кортежів із значенням атрибута "Група", рівним "384", а в другому відношенні немає кортежів із значенням того ж атрибута "383", в результуючому відношенні залишилися тільки кортежі, що містять дані про студентів групи "382".

З вимог унікальності і ненадлишкових для первинного ключа у результуючому відношенні первинний ключ буде idSt.

Якщо відношення  $A$  і  $B$  не мають спільних імен атрибутів, то їх природне об'єднання еквівалентно декартовому добутку.

$\theta$ -об'єднання. Операція тета-об'єднання визначає відношення, що містять кортежі з декартового добутку відношень  $R$  і  $S$ , що задовольняють предикату  $F$ . Предикат  $F$  має вигляд  $R.ai \theta S.bi$ , де замість  $\theta$  може бути зазначений один з операторів порівняння ( $=$ ,  $>$ ,  $<$ , тощо).

Можуть використовуватися два варіанти запису цієї операції:

$A [X \theta Y] B$

$(A \text{ TIMES } B) \text{ WHERE } X \theta Y$

Приклад. Нехай відношення  $A$  містить інформацію про ціну приладів, а відношення  $B$  – про ціну підставок для приладів. Необхідно отримати ті комбінації приладів, підставок і цін на них, де підставка не дорожче приладу. Це можна зробити за допомогою операції  $\theta$ -об'єднання за умовою "Ціна\_приладу  $>$  Ціна\_підставки".

Так як в обох відношеннях  $A$  і  $B$  є однакові атрибути "Ціна", тоді необхідно або їх перейменувати, або використовувати складену форму запису  $\langle \text{Ім'я відношення} \rangle. \langle \text{Ім'я атрибута} \rangle$ . Використаємо складену форму:

відношення  $A$

| Прилад   | Ціна |
|----------|------|
| Прилад 1 | 100  |
| Прилад 2 | 200  |

відношення  $B$

| Підставка   | Ціна |
|-------------|------|
| Підставка 1 | 50   |
| Підставка 2 | 150  |

$\theta$  - об'єднання  $A [A.Ціна \geq B.Ціна] B$

| Прилад   | A.Ціна | Підставка   | B.Ціна |
|----------|--------|-------------|--------|
| Прилад 1 | 100    | Підставка 1 | 50     |
| Прилад 2 | 200    | Підставка 1 | 50     |
| Прилад 2 | 200    | Підставка 2 | 150    |

**8) Розподіл (ділення).** Нехай відношення  $A$  і  $B$  мають заголовки  $\{X, Y\}$  і  $\{Y\}$  відповідно, атрибути  $X$  і  $Y$  можуть бути складовими. Однаково названі атрибути двох відношень визначені на одних і тих же доменах.

Результатом розподілу відношень A на B буде відношення з заголовком {X} і тілом, що містить множину всіх кортежів {X: x}, таких, що в A існує кортеж {X: x, Y: y}, для всіх кортежів {Y: y} з B. Позначається ця операція як

$A / B$

A DIVIDEBY B

Приклад:

| відношення A |         | відношення B | Розподіл A / B |
|--------------|---------|--------------|----------------|
| idSt         | Предмет | Предмет      | idSt           |
| 123          | ООП     | ООП          | 123            |
| 123          | ОБДЗ    | ОБДЗ         |                |
| 124          | ОБДЗ    |              |                |
| 125          | ООП     |              |                |

Тобто, відношення A і B мають однакові атрибути "Предмет", а результатом будуть значення атрибута "idSt" такі, що є всі у відношенні B та ті самі в A з однаковим "idSt".

Підсумовуючи розгляд операцій над множинами варто наголосити, що на практиці здебільшого використовуються комбінації цих операцій, тобто результат однієї операції є атрибутом іншої.

Для прикладу розглянемо використання реляційних операцій об'єднання (A JOIN B) та проекції (A [V, W, Z]).

Потрібно написати вираз, в результаті виконання якого з відношень STUDENTS і GROUPS буде отримано відношення, що містить дані про прізвища і ініціали студента та старосту його групи. Шуканий вираз:

(STUDENTS JOIN GROUPS) [ПІБ, Староста]

### 3.3. Нормалізація реляційних баз даних

На етапі логічного проектування реляційних БД розробляється логічна структура: визначаються реляційні відношення, їх атрибути, потенційні і зовнішні

ключі. Це завдання дуже відповідальне, оскільки помилки, зроблені при розробленні структури бази даних, можуть привести до *аномалій модифікації*.

Під *аномаліями модифікації* розуміються небажані ефекти в реляційних базах даних при збереженні, видаленні і заміні даних.

*Аномалія видалення* проявляється в тому, що видаляючи факти, що відносяться до однієї сутності (об'єкта, явища і т.п.), мимоволі видаляються факти, які стосуються іншої сутності.

*Аномалія вставки* призводить до того, що не можна помістити в БД деякий факт про одну сутність, не вказавши додатково деякий факт про іншу сутність.

*Аномалія поновлення* полягає в тому, що для внесення зміни в дані про один факт (подію, об'єкті) необхідно виконати множинні зміни в БД.

Однією з істотних переваг реляційної моделі є наявність формального механізму оцінки якості логічної структури БД засобами теорії нормалізації.

**Нормалізація** – це процес приведення структури реляційних відношень до форми, яка має кращі властивості при додаванні, зміні і видаленні даних. Остаточна мета нормалізації зводиться до отримання такого проекту БД, в якому кожен факт з'являється лише в одному місці, тобто виключена надмірність інформації. Крім завдання більш ефективного використання пам'яті, нормалізація дозволяє знизити загрозу порушення цілісності БД через появу в ній внутрішніх суперечностей.

Вводиться поняття **нормальних форм** (НФ) відношень, для кожної з яких є певний набір обмежень. Відношення знаходиться в даній нормальній формі, якщо задовольняє зазначеним обмеженням. Кожна наступна НФ включає в себе вимоги всіх попередніх, тобто є більш суворим обмеженням. Перш ніж перейти до розгляду конкретних НФ, необхідно ввести ряд визначень.

Нехай  $R$  – реляційне відношення, а  $X$  і  $Y$  – деякі підмножини атрибутів цього відношення.  $Y$  функціонально залежно від  $X$  тоді і тільки тоді, коли для кожного значення множини  $X$  існує тільки одне значення множини  $Y$ . Інакше кажучи, якщо два кортежі відношень збігаються за значенням  $X$ , то вони обов'язково будуть збігатися і за значенням  $Y$ . Записується функціональна залежність (ФЗ) як  $X \rightarrow Y$ ,



читається як " $X$  функціонально визначає  $Y$ ". Якщо існує ФЗ  $X \rightarrow Y$ , то  $X$  називають детермінантою, а  $Y$  – залежною частиною.

З визначення ФЗ, зокрема, випливає, що будь-яка підмножина атрибутів відношень функціонально залежна від будь-якого з потенційних ключів.

Твердження про те, що дана ФЗ виконується завжди, по суті, є обмеженням цілісності для відношень і накладає певні обмеження на множину допустимих значень кортежів цього реляційного відношення.

Розглянемо приклад. Нехай відношення STUDENTS має наступну структуру:

| idSt | ПІБ         | Група  |
|------|-------------|--------|
| 121  | Сало М.Ф.   | КН-21  |
| 122  | Коваль В.П. | КН-22  |
| 123  | Чайка Г.П.  | ІСТ-21 |

і атрибут idSt є первинним ключем.

У представленому відношенні є наступні функціональні залежності:

- $\{idSt\} \rightarrow \{ПІБ\}$ ;
- $\{idSt\} \rightarrow \{Група\}$ ;
- $\{idSt, ПІБ\} \rightarrow \{Група\}$  і т.д.

У той же час підмножина атрибутів {Група} в загальному випадку не є функціонально залежною від {ПІБ}, так як в списку студентів можуть виявитися однофамільці з однаковими ініціалами, але навчатися в різних групах.

Як видно з прикладу, множина функціональних залежностей, навіть для відношень з невеликим ступенем, може бути досить великою. У той же час ФЗ є обмеженням цілісності, і їх виконання повинне перевірятися СКБД при таких операціях, як додавання, видалення або зміна даних. Тому бажано виключити з розгляду такі ФЗ, які можна вивести з інших ФЗ.

### 3.3.1. Перша нормальна форма

Відношення знаходиться в *першій нормальній формі* (1НФ) тоді і тільки тоді, коли воно містить тільки скалярні значення атрибутів і жоден з ключових атрибутів не має значення NULL. Ключовим є атрибут, що входить в будь-який з потенційних ключів.

Щодо NULL потрібно пояснити, що це спеціальне значення, що показує, що значення атрибута не визначено. Нехай, наприклад, щодо "Студенти" є атрибут "Оцінка за іспит". Якщо оцінка якогось студента за якийсь іспит невідома, то відповідному атрибуту кортежу буде встановлено значення NULL. Важливо не плутати NULL з числом 0.

Для прикладу, розглянемо асоційовану таблицю, що порушує вимоги 1НФ. У ній значення атрибутів "Предмет" і "Оцінка" не є атомарними. Відповідно, ця структура не є відношенням в 1НФ.

**Асоційоване відношення STUDENTS**

| idSt | ПІБ         | Група  | Предмет | Оцінка |
|------|-------------|--------|---------|--------|
| 121  | Сало М.Ф.   | КН-21  | ООП     | 5      |
|      |             |        | ОБДЗ    | 4      |
| 122  | Коваль В.П. | КН-22  | ООП     | 5      |
|      |             |        | ОБДЗ    | 5      |
| 123  | Чайка Г.П.  | ІСТ-21 | ГІС     | 5      |
|      |             |        | ОБДЗ    | 4      |

Щоб привести дану таблицю до вимог 1НФ, треба виконати деякі перетворення, зокрема заповнити атрибути idSt, ПІБ та Група для Предмету ОБДЗ кожного студента. Відповідно, атрибут idSt в цьому випадку вже не може бути первинним ключем, адже він перестає бути унікальним. Тому у якості первинного ключа доцільно використовувати складений ключ з атрибутів {idSt, Предмет}, що забезпечить його унікальність (неповторюваність).

В результаті цих перетворень спостерігається покращення структури відношення, але міститься надлишкова інформація – повторювані прізвища студентів і назви їх груп. Крім того, для цього відношення властиві аномалії модифікації. Зокрема, не можна внести інформацію про студента, не вказавши хоча б один предмет (аномалія вставки) і неможливо видалити інформацію про всі здані студентом навчальні дисципліни, не видаливши інформацію про нього самого (аномалія видалення).

### Відношення STUDENTS у 1НФ

| idSt | ПІБ         | Група  | Предмет | Оцінка |
|------|-------------|--------|---------|--------|
| 121  | Сало М.Ф.   | КН-21  | ООП     | 4      |
| 121  | Сало М.Ф.   | КН-21  | ОБДЗ    | 5      |
| 122  | Коваль В.П. | КН-22  | ООП     | 5      |
| 122  | Коваль В.П. | КН-22  | ОБДЗ    | 5      |
| 123  | Чайка Г.П.  | ІСТ-21 | ГІС     | 5      |
| 123  | Чайка Г.П.  | ІСТ-21 | ОБДЗ    | 4      |

Відповідь на запитання, як покращити структуру бази даних, дається в термінах старших нормальних форм.

#### 3.3.2. Друга нормальна форма

Реляційне відношення знаходиться у другій нормальній формі (2НФ), якщо воно задовольняє визначенню 1НФ і всі його атрибути, що не входять в первинний ключ, повністю залежні від цього первинного ключа.

При описі 2НФ і 3НФ всюди, крім випадків, де це зазначено явно, передбачається, що реляційні відношення мають тільки один потенційний ключ, який і є первинним.

Якщо потенційний ключ є простим, тобто складається з єдиного атрибута, то будь-яка функціональна залежність від нього є повною.

Друга нормальна форма за визначенням забороняє наявність неключових атрибутів, які взагалі не залежать від потенційного ключа. Таким чином, друга нормальна форма в тому числі забороняє створювати відношення як незв'язані (хаотичні, випадкові) набори атрибутів.

Повернемося до відношення STUDENTS у 1НФ і розглянемо, чи знаходиться воно в другій нормальній формі.

Якщо первинний ключ складений з атрибутів {idSt, Предмет}, тоді згідно визначення, не всі атрибути, що не входять в первинний ключ, повністю залежать від нього. Зокрема, атрибут "ПІБ" залежить від частини ключа – {idSt} і не

залежить від Предмет}, атрибут "Група" також залежить лише від частини ключа – {idSt}.

Отже, відношення STUDENTS не знаходиться у 2НФ.

Для покращення структури відношення STUDENTS можливі декілька варіантів.

Перший – **додати** ще один атрибут, який стане одним потенційним ключовим полем, наприклад id (номер за порядком). Але тоді і далі спостерігається надлишковість даних, зокрема дублювання атрибутів "ПІБ", "Група".

Інший варіант – **розбити** відношення STUDENTS на два, що знаходяться у 2НФ. Тут виникає проблема декомпозиції без втрат, тобто такого розбиття відношення, щоб в результаті цієї процедури не відбулася втрата інформації.

Процес розбиття відношення  $R\{A, B, C\}$  на відношення  $R1\{A, B\}$ ,  $R2\{A, C\}$  називається *проектуванням*, а відношення  $R1$  і  $R2$  – *проекціями*. Тут  $A, B$  і  $C$  – це деякі непересічні підмножини атрибутів вхідного відношення, об'єднання яких дасть всі множини атрибутів. Якщо була проведена декомпозиція без втрат, то об'єднання проєкцій  $R1$  і  $R2$  має дати початкове відношення  $R$ .

При проведенні декомпозиції часто опираються на теорему Хеза (IJ Heath): нехай  $R\{A, B, C\}$  є реляційним відношенням, де  $A, B, C$  – атрибути (можливо, складові) цього відношення. Якщо є функціональна залежність  $A \rightarrow B$ , то проєкції  $R1\{A, B\}$  і  $R2\{A, C\}$  утворюють декомпозицію без втрат.

Для нормалізації відношення STUDENTS використаємо ФЗ {idSt}  $\rightarrow$  {ПІБ, Група} та {idSt}  $\rightarrow$  {Предмет, Оцінка}. Це дозволить відповідно до теореми Хеза виконати декомпозицію без втрат, отримавши два нових відношення STUD та RESULT.

Необхідно відзначити, що якщо в реляційному відношенні тільки один потенційний ключ, і він простий, наявність 1НФ автоматично робить відношення відповідним 2НФ.

Таким чином, відношення STUD вимогам 2НФ задовольняє. Що стосується відношення RESULT, то в ньому ФЗ {idSt, Предмет}  $\rightarrow$  {Оцінка} є однозначною, що доводить відповідність вимогам 2НФ.

**Відношення STUD**

| idSt | ПІБ         | Група  |
|------|-------------|--------|
| 121  | Сало М.Ф.   | КН-21  |
| 122  | Коваль В.П. | КН-22  |
| 123  | Чайка Г.П.  | ІСТ-21 |

**Відношення RESULT**

| idSt | Предмет | Оцінка |
|------|---------|--------|
| 121  | ООП     | 4      |
| 121  | ОБДЗ    | 5      |
| 122  | ООП     | 5      |
| 122  | ОБДЗ    | 5      |
| 123  | ГІС     | 5      |
| 123  | ОБДЗ    | 4      |

### 3.3.3. Третя нормальна форма

Відношення знаходиться в третій нормальній формі (3НФ), якщо воно:

- задовольняє визначенню 2НФ;
- жоден з його неключових атрибутів не залежить функціонально від будь-якого іншого неключового атрибута.

Простіше кажучи, друге правило вимагає виносити всі неключові поля, вміст яких може стосуватися кількох записів таблиці, в окремі таблиці.

Розглянемо приклад. Реляційне відношення SUBJECT містить список предметів із зазначенням викладачів, що їх ведуть та посад викладачів.

Первинним ключем є атрибут ID, альтернативних ключів немає, відношення знаходиться у 2НФ.

**Відношення SUBJECT**

| id | Предмет                 | Викладач          | Посада   |
|----|-------------------------|-------------------|----------|
| 1  | ООП                     | Соколовський Я.І. | професор |
| 2  | ОБДЗ                    | Дендюк М.В.       | доцент   |
| 3  | ГІС                     | Дендюк М.В.       | доцент   |
| 4  | ТРПО                    | Соколовський Я.І. | професор |
| 5  | Геометричне моделювання | Крошний І.М.      | доцент   |

При цьому відомо, що "Посада" цілковито визначається "Викладачем". Іншими словами, існує ФЗ {Викладач} → {Посада}, що робить відношення SUBJECT таким, що не відповідає вимогам ЗНФ, і в ньому є аномалії модифікації.

Скористаємося теоремою Хеза і виконаємо декомпозицію, використовуючи наведену вище ФЗ. Отримані реляційні відношення позначимо SUBJ і LECT, в SUBJ повинен бути визначений зовнішній ключ {Викладач}, який посилається на первинний ключ відношення LECT.

**Відношення SUBJ**

| id | Предмет                 | Викладач          |
|----|-------------------------|-------------------|
| 1  | ООП                     | Соколовський Я.І. |
| 2  | ОБДЗ                    | Дендюк М.В.       |
| 3  | ГІС                     | Дендюк М.В.       |
| 4  | ТРПО                    | Соколовський Я.І. |
| 5  | Геометричне моделювання | Крошній І.М.      |

**Відношення LECT**

| Викладач          | Посада   |
|-------------------|----------|
| Соколовський Я.І. | професор |
| Дендюк М.В.       | доцент   |
| Крошній І.М.      | доцент   |

Легко показати, що вимоги ЗНФ тут будуть виконуватися: щодо SUBJ між атрибутами "Предмет" і "Викладач" функціональних залежностей немає, а щодо LECT, неключовий атрибут – єдиний.

### 3.3.4. Нормальна форма Бойса-Кодда

Нормальну форму Бойса-Кодда (НФБК) часто називають посиленою третьою нормальною формою (або посиленою ЗНФ).

За визначенням, відношення знаходиться в *нормальній формі Бойса-Кодда*, якщо воно знаходиться у ЗНФ і в ньому відсутні залежності атрибутів складеного ключа від неключових атрибутів.

Звідси випливає, що вимоги нормальної форми Бойса-Кодда пред'являються тільки до таблиць, у яких первинний ключ є складеним. Відношення, у яких первинний ключ простий, і вони знаходяться в третій нормальній формі, автоматично знаходяться і в нормальній формі Бойса-Кодда, наприклад, отримані раніше SUBJ і LECT.

Головне правило НФБК звучить наступним чином: частина складеного первинного ключа не повинна залежати від неключових атрибутів.

Таким чином, вимога про фактичну залежність неключових атрибутів від усього ключа цілком і ні від чого іншого, крім як від ключа, поширюється і на ключові атрибути.

Приклад. Розглянемо отримане раніше відношення RESULT, у якого ключ є складеним {idSt, Предмет}. Згідно означення, воно знаходяться в 3НФ, адже жоден з його неключових атрибутів не залежить функціонально від будь-якого іншого неключового атрибута, бо неключовий атрибут – єдиний.

**Відношення RESULT**

| idSt | Предмет | Оцінка |
|------|---------|--------|
| 121  | ООП     | 4      |
| 121  | ОБДЗ    | 5      |
| 122  | ООП     | 5      |
| 122  | ОБДЗ    | 5      |
| 123  | ГІС     | 5      |
| 123  | ОБДЗ    | 4      |

Щоб відношення RESULT перебувало в НФБК, необхідно, щоб частина складеного первинного ключа не залежала від неключових атрибутів. У нашому випадку частини складеного ключа {idSt} або {Предмет} не залежать від атрибута {Оцінка}, тобто за Оцінкою однозначно не можна визначити ні студента, ні предмет, що і треба було довести.

Але бувають випадки, коли за неключовим атрибутом можна однозначно визначити частину ключа.

Для прикладу розглянемо організацію, яка реалізує декілька різних проектів. При цьому в кожному проекті робота ведеться за кількома функціональними напрямками, в кожному з яких є свій куратор. Співробітник може бути куратором тільки того напрямку, на якому він спеціалізується, тобто якщо співробітник програміст, він не може займатися в проекті напрямком, що пов'язаний з бухгалтерією.

### Відношення проектів і кураторів

| Проект | Напрямок    | Куратор     |
|--------|-------------|-------------|
| 1      | Розробка    | Сало М.Ф.   |
| 1      | Бухгалтерія | Бекас Б.О.  |
| 2      | Розробка    | Сало М.Ф.   |
| 2      | Бухгалтерія | Коваль В.П. |
| 2      | Реалізація  | Пірко І.Б.  |
| 3      | Розробка    | Крамар А.А. |

Наведене відношення знаходиться в третій нормальній формі, так як має складений первинний ключ і неключовий атрибут залежить від усього ключа, а не від якоїсь його частини.

Але в даному випадку відношення не знаходиться в нормальній формі Бойса-Кодда, справа в тому, що знаючи куратора, можна чітко визначити, який напрямок він курує, іншими словами, частина складеного ключа, тобто "Напрямок", залежить від неключового атрибута, тобто "Куратор".

Щоб привести це відношення до НФБК, необхідно зробити декомпозицію даного відношення, тобто розбити на декілька:

#### Відношення кураторів

| Id куратора | ПІБ         | Напрямок    |
|-------------|-------------|-------------|
| 1           | Сало М.Ф.   | Розробка    |
| 2           | Бекас Б.О.  | Бухгалтерія |
| 3           | Коваль В.П. | Бухгалтерія |
| 4           | Пірко І.Б.  | Реалізація  |
| 5           | Крамар А.А. | Розробка    |

#### Відношення зв'язку кураторів і проектів

| id  | Проект | Id куратора |
|-----|--------|-------------|
| 101 | 1      | 1           |
| 102 | 1      | 2           |
| 103 | 2      | 1           |
| 104 | 2      | 3           |
| 105 | 2      | 4           |
| 106 | 3      | 5           |

Таким чином, в таблиці кураторів у нас зберігається список кураторів і їх спеціалізація, тобто напрям, який вони можуть курирувати, а в таблиці зв'язку кураторів і проектів відбивається зв'язок проектів і кураторів.



### 3.3.5. Четверта нормальна форма

На практиці процес нормалізації часто закінчується приведенням відношень БД у відповідність вимогам НФБК, але в деяких випадках потрібно привести відношення до більш старших НФ – четвертої і п'ятої форми для усунення таких типів функціональних залежностей, як залежності ключів від неключових атрибутів та незалежних багатозначних залежностей.

Відношення перебуває у *четвертій нормальній формі*, коли воно перебуває у третій нормальній формі і не має багатозначних залежностей між атрибутами.

Для прикладу, розглянемо відношення Викладач-Предмет-Група.

**Відношення Викладач-Предмет-Група**

| Викладач          | Предмет                 | Група  |
|-------------------|-------------------------|--------|
| Соколовський Я.І. | ООП                     | КН-21  |
| Соколовський Я.І. | ООП                     | КН-22  |
| Соколовський Я.І. | ООП                     | ІСТ-21 |
| Дендюк М.В.       | ОБДЗ                    | КН-21  |
| Дендюк М.В.       | ОБДЗ                    | КН-22  |
| Дендюк М.В.       | ОБДЗ                    | ІСТ-21 |
| Крошній І.М.      | Геометричне моделювання | КН-21  |
| Крошній І.М.      | Геометричне моделювання | КН-22  |
| Крошній І.М.      | Геометричне моделювання | ІСТ-21 |

У даному відношенні відсутні неключові атрибути, бо єдиний ключ це всі три атрибути, що входять в складовий первинний ключ {Викладач, Предмет, Група} і вимоги НФБК в даному випадку виконуються, але існує багатозначна залежність між атрибутами предмет та група, тому що багато груп вивчають багато предметів, які може читати один викладач.

Тому дане відношення не задовольняє вимогам 4НФ, але і використовувати теорему Хеза для проведення декомпозиції не вдасться через відсутність відповідних ФЗ: всі атрибути входять в складового ключа і будь-які ФЗ будуть тривіальними.

У подібних випадках використовується теорема Фейгіна (Ron Fagin): нехай  $A$ ,  $B$  і  $C$  є підмножинами атрибутів відношення  $R \{A, B, C\}$ . Відношення  $R$  дорівнюватиме об'єднанню своїх проєкцій  $\{A, B\}$  і  $\{A, C\}$  тоді і тільки тоді, коли для  $R$  виконується багатозначна залежність  $A \twoheadrightarrow BC$ .

Скориставшись цією теоремою, проведемо нормалізацію і отримаємо два відношення: Викладач-Предмет та Викладач-Група, які не зв'язані між собою.

**Відношення Викладач-Предмет**

| Викладач          | Предмет                 |
|-------------------|-------------------------|
| Соколовський Я.І. | ООП                     |
| Дендюк М.В.       | ОБДЗ                    |
| Крошний І.М.      | Геометричне моделювання |

**Відношення Викладач-Група**

| Викладач          | Група  |
|-------------------|--------|
| Соколовський Я.І. | КН-21  |
| Соколовський Я.І. | КН-22  |
| Соколовський Я.І. | ІСТ-21 |
| Дендюк М.В.       | КН-21  |
| Дендюк М.В.       | КН-22  |
| Дендюк М.В.       | ІСТ-21 |
| Крошний І.М.      | КН-21  |
| Крошний І.М.      | КН-22  |
| Крошний І.М.      | ІСТ-21 |

Цілісність даних в таких відношеннях підтримують за допомогою тригерів, які зберігаються на сервері бази даних і які будуть розглянуті пізніше.

Отримані відношення Викладач-Предмет та Викладач-Група задовольняють 4НФ та об'єднання яких дасть початкове відношення Викладач-Предмет-Група.

### 3.3.6. П'ята нормальна форма

Інколи зустрічається ситуація, коли таблиця знаходиться в 4НФ, але не в 5НФ. Якщо така таблиця не нормалізована до 5НФ, тягар підтримання логічних зв'язків покладається наприклад, на тригери або безпосередньо на застосунок користувача, який займається редагуванням таблиці. В цьому випадку існує підвищений ризик порушення логічної цілісності і 5НФ усуває можливість такої нецілісності.

П'ята нормальна форма (5НФ), також відома як проєкційно-об'єднувана нормальна форма (ПЗ/НФ) – ступінь нормалізації баз даних, утворений для усунення надлишковості даних в реляційних базах даних, які містять багатозначні факти, шляхом відокремлення семантично пов'язаних кратних взаємозв'язків. Відношення знаходиться в 5НФ тоді і тільки тоді, коли кожна нетривіальна залежність об'єднання визначається потенційним ключем. Іншими словами, кожна проєкція такого відношення містить не менше одного можливого ключа і не менше одного неключового атрибута.

Залежність об'єднання  $\{A, B, \dots Z\}$  на  $R$  визначається потенційним ключем  $R$  тоді і тільки тоді, коли кожен з  $A, B, \dots, Z$  є суперключем для  $R$ .

До цього часу ми припускали, що єдиною операцією, необхідною для усунення надмірності у відношенні, була декомпозиція його на дві проєкції. Однак, існують відношення, для яких не можна виконати декомпозицію без втрат на дві проєкції, але які можна піддати декомпозиції без втрат на три (або більше) проєкцій. Цей факт отримав назву *залежності за об'єднанням*, а такі відношення називають 3-декомполітними відношеннями (ясно, що будь-яке відношення можна назвати "n-декомполітним", де  $n \geq 2$ ).

Для пояснення візьмемо приклад з Вікіпедії: на деякому етапі проєктування БД отримано відношення Продавець-Фірма-Товар, у якому зберігаються дані про асортимент кількох продавців, які торгують продукцією кількох фірм (номенклатура товарів фірм може перетинатися).

Якщо додаткових умов немає, то дане відношення, яке знаходиться в 4-й нормальній формі, є коректним і відображає всі необхідні обмеження.

Тепер припустимо, що потрібно врахувати наступне обмеження: *кожен продавець має в своєму асортименті обмежений список фірм і обмежений список типів товарів і пропонує товари зі списку товарів, що виробляються фірмами зі списку фірм.*

Тобто продавець не має право торгувати якими завгодно товарами яких завгодно фірм. Якщо продавець  $P$  має право торгувати товарами фірми  $F$ , і якщо продавець  $P$  має право торгувати товарами типу  $T$ , то в цьому випадку в

асортимент продавця П входять товари типу Т фірми Ф за умови, що фірма Ф виробляє товари типу Т.

**Відношення Продавець-Фірма-Товар**

| Продавець | Фірма    | Товар      |
|-----------|----------|------------|
| Іваненко  | Кардинал | Порохотяг  |
| Іваненко  | Кардинал | Хлібниця   |
| Дацко     | Адаптер  | Сучкоріз   |
| Дацко     | Адаптер  | Порохотяг  |
| Дацко     | Адаптер  | Хлібниця   |
| Дацко     | Адаптер  | Парасолька |
| Олійник   | Адаптер  | Порохотяг  |
| Олійник   | Адаптер  | Телескоп   |
| Олійник   | Кардинал | Порохотяг  |
| Олійник   | Кардинал | Лампа      |
| Олійник   | Антика   | Вішалка    |

Таке обмеження може бути викликано, наприклад, тим, що список типів товарів продавця обмежений наявними у нього ліцензіями, або знаннями і кваліфікацією, необхідними для їх продажу, а список фірм кожного продавця визначений партнерськими угодами.

У розглянутому прикладі, зокрема, передбачається, що продавець Іваненко має право торгувати товарами тільки фірми «Кардинал», продавець Дацко – товарами тільки фірми «Адаптер», зате продавець Олійник не має права торгувати хлібницею і сучкорізом тощо.

В даному випадку для приведення до 5NF початкове відношення має бути розбите на три: {Продавець, Фірма}, {Фірма, Товар}, {Продавець, Товар}.

Варто зазначити, що п'ята нормальна форма є остаточною нормальною формою по відношенню до операцій розбиття таблиць на проєкції і їх об'єднання, саме тому її альтернативна назва – проєкційно-об'єднувана нормальна форма.

**Продавець-Товар****Продавець-Фірма****Фірма-Товар**

| Продавець | Товар      |
|-----------|------------|
| Іваненко  | Порохотяг  |
| Іваненко  | Хлібниця   |
| Дацко     | Сучкорез   |
| Дацко     | Порохотяг  |
| Дацко     | Хлібниця   |
| Дацко     | Парасолька |
| Олійник   | Телескоп   |
| Олійник   | Порохотяг  |
| Олійник   | Лампа      |
| Олійник   | Вішалка    |

| Продавець | Фірма    |
|-----------|----------|
| Іваненко  | Кардинал |
| Дацко     | Адаптер  |
| Олійник   | Адаптер  |
| Олійник   | Кардинал |
| Олійник   | Антика   |

| Фірма    | Товар      |
|----------|------------|
| Кардинал | Порохотяг  |
| Кардинал | Хлібниця   |
| Кардинал | Лампа      |
| Адаптер  | Сучкоріз   |
| Адаптер  | Порохотяг  |
| Адаптер  | Хлібниця   |
| Адаптер  | Парасолька |
| Адаптер  | Телескоп   |
| Антика   | Вішалка    |

Таким чином, якщо таблиця знаходиться в 5NF, то гарантується, що вона не містить аномалій, які можуть бути виключені за допомогою її розбиття на проєкції.

Також варто відзначити, що таблиці, які необхідно нормалізувати до п'ятої нормальної форми, зустрічаються вкрай рідко, тобто це дуже окремий випадок. Більш того, такі таблиці є не зовсім вдалими з точки зору проєктування. Крім усього іншого, щоб привести таблицю до п'ятої нормальної форми, необхідно дуже добре розбиратися в предметній області, щоб визначити залежності об'єднання, адже це дійсно дуже складно. Іншими словами, якщо вдасться визначити ці залежності об'єднання, то тільки в цьому випадку можливо привести таблицю до п'ятої нормальної форми.

### 3.3.7. Доменно-ключова нормальна форма. Денормалізація

Подальші дослідження в області нормалізації йшли по шляху пошуку ситуацій, в яких можуть проявитися приховані аномалії, пов'язані з 5НФ. У 1981 р Р. Фейгін опублікував статтю, в якій визначив доменно-ключову НФ і показав, що відношення в ДКНФ не має аномалій модифікації. Більш того, будь-яке

відношення, яке не має аномалій модифікації, має перебувати в ДКНФ. Після цього відпала необхідність в НФ більш високих порядків, зокрема, для усунення аномалій модифікації.

Відношення знаходиться в *доменно-ключовій НФ*, якщо кожне обмеження, що накладаються на це відношення, є логічним наслідком визначення доменів і ключів.

Обмеження домену – обмеження, що пропонує використовувати для певного атрибута значення тільки з деякого заданого домену. Обмеження по своїй суті є задаванням переліку (або логічного еквівалента переліку) допустимих значень типу і оголошенням про те, що вказаний атрибут має даний тип.

Обмеження ключа – обмеження, яке стверджує, що деякий атрибут або комбінація атрибутів є потенційним ключем.

Будь-яке відношення, що знаходиться в ДКНФ, обов'язково знаходиться в 5НФ. Однак не будь-яке відношення можна привести до ДКНФ.

Закінчуючи розмову про нормалізацію, необхідно згадати і про *денормалізацію*, яка визначається як навмисне приведення структури реляційних відношень до форми, що порушує вимоги НФ. У процесі нормалізації реляційні відношення зазвичай піддаються декомпозиції, поділяються на кілька. При цьому для читання даних з декількох відношень доводиться проводити операції об'єднання, які вимагають додаткового часу.

У більшості випадків втрати часу на об'єднання таблиць відносно невеликі, і важливіше спроектувати таку структуру БД, в якій не буде аномалій модифікації. Але в ряді випадків турбота про швидкодію системи змушує розробників погодитися з деякою надмірністю даних, що зберігаються і відсутністю у окремих реляційних відношень (таблиць реляційної БД) старших нормальних форм. Це можна розглядати як крайній захід, і рішення про денормалізацію треба приймати дуже обережно.

### **3.3.8. Переваги та недоліки нормалізації**

Метою нормалізації відношень БД є усунення надлишкової інформації. Як видно з наведених вище прикладів, нормалізовані відношення БД містять тільки один елемент надлишкових даних – це атрибути зв'язку, які присутні одночасно в

батьківському та дочірньому відношеннях. Оскільки надлишкові дані у відношеннях не зберігаються, то заощаджується місце на носіях інформації. Однак у нормалізованій БД є й недоліки, насамперед, практичного характеру.

Чим більше суб'єктів (сутностей) охоплюється предметною областю, тим з більшого числа відношень буде складатися нормалізована БД. Бази даних у складі великих систем, які задіяні у життєдіяльності великих організацій та підприємств, можуть містити сотні зв'язаних між собою відношень. Оскільки поріг людського сприйняття не дозволяє одночасно охопити велику кількість об'єктів з урахуванням їх взаємозв'язків, то можна стверджувати, що зі збільшенням числа нормалізованих відношень цілісне сприйняття БД як системи взаємозалежних даних зменшується. Тому при розробці й експлуатації великих систем існують ситуації, коли кожний співробітник уявляє собі процеси, що відбуваються тільки в частині системи. Відомі випадки еволюційного створення таких систем, коли принципи їх функціонування згодом виходили за межі розуміння.

Іншим недоліком нормалізованої БД є необхідність зчитування з відношень зв'язані дані при виконанні складних запитів, які надають інформацію про взаємодію сутностей технологічного процесу між собою. При великих обсягах це призводить до збільшення часу доступу до даних.

Третя нормальна форма та нормальна форма Бойса-Кодда є теоретичними конструкціями, у той час як більшість розроблювачів БД працюють у реальному світі. Тому доречно зробити кілька зауважень про недоліки, що властиві відношенням, які зведені до 3НФ. Існують варіанти, коли має сенс поділити відношення на декілька дрібних, якщо частина наведених у ньому даних непостійна й часто оновлюється (оперативна інформація), а інші дані пасивні та змінюються не так часто (довідкова інформація). Також є сенс у разі необхідності об'єднати відношення, щоб забезпечити високу швидкість реакції на запит. Можна навіть піти на дублювання даних у відношеннях, якщо це дозволить зменшити витрати на обробку запитів, хоча формально не варто було б цього робити.

Необхідно також відзначити, що відношення між собою доцільно об'єднувати за атрибутами, які зовсім звільнені від семантичної залежності, що породжується особливостями реалізації дійсних процесів у БД. Для цього використовують спеціально виділені інкрементні атрибути однозначної ідентифікації кортежів

усередині відношень. Такий підхід дозволяє позбавитися необхідності процесу перевизначення зв'язків між відношеннями, що виникає в реальному житті при зміні правил обліку різних суб'єктів діяльності організації.

Наведені вище міркування не слід сприймати як заклик зовсім не нормалізувати дані. Вони лише мають довести, що при роботі з даними великого обсягу доводиться шукати компроміс між вимогами нормалізації (тобто "логічності" даних і економії місця на носіях інформації) та необхідністю збільшення швидкодії. При цьому треба звернути увагу на вимоги користувачів, щоб уникнути зайвої деталізації суб'єктів реальних процесів, які відбуваються на підприємстві.

### **Контрольні запитання**

1. Для чого потрібен первинний ключ?
2. Назвіть основні операції реляційної алгебри.
3. Чим відрізняється вибірка від проекції?
4. Що забезпечує та гарантує процедура нормалізації?
5. Які нормальні форми відношень БД Вам відомі?
6. Коли відношення зведено до першої нормальної формі?
7. Коли відношення зведено до другої нормальної формі?
8. Коли відношення зведено до третьої нормальної формі?
9. Які нормальні форми відношень використовують на практиці?
10. Які нормальні форми відношень мало використовують на практиці?
11. За яких умов у БД можна залишити відношення, що не є в 1НФ?
12. Як позбутися семантичної залежності в зв'язках між відношеннями?
13. Які існують способи реалізації зв'язків 1:1 та 1:М у БД?
14. Назвіть переваги та недоліки нормалізації.
15. В яких випадках доцільно використовувати денормалізацію?



## Тема 4. Етапи проектування баз даних

Проектування баз даних – це ітераційний, багатоетапний процес прийняття обґрунтованих рішень у процесі аналізу інформаційної моделі предметної області, вимог до даних з боку прикладних програмістів і користувачів, синтезу логічних і фізичних структур даних, аналізу та обґрунтування вибору програмних і апаратних засобів.

Проектування баз даних відбувається в чотири етапи:

1. Системний аналіз і словесний опис інформаційних об'єктів;
2. Проектування інфологічної моделі або Концептуальне проектування;
3. Логічне проектування Баз Даних;
4. Фізичне проектування Баз Даних.

Від того, як спроектована БД, залежить подальша робота з нею.

### 4.1. Етап системного аналізу і словесного опису

Системний аналіз передбачає мовний опис реальних об'єктів предметної області, визначення зв'язків між об'єктами, дослідження характеристик об'єктів і зв'язків. Результати дослідження використовуються при наступному, концептуальному проектуванні БД.

Для визначення складу і структури предметної області застосовуються або функціональний, або предметний підходи.

Функціональний підхід застосовує рух "від задач" і використовується у тих випадках, коли заздалегідь відомі функції майбутніх користувачів БД, а також відомі всі задачі, для інформаційних потреб яких створюються БД. В цьому випадку на основі виробничих документів, опитувань замовників можна чітко визначити мінімальний набір об'єктів предметної області та їх взаємозв'язок.

Предметний підхід застосовується у тому випадку, коли інформаційні потреби майбутніх користувачів чітко не визначені. В цьому випадку не можна чітко визначити мінімальний набір об'єктів предметної області. В опис предметної області включаються об'єкти та зв'язки, які є найбільш характерними

та найбільш суттєвими для неї. БД називається предметною і може використовуватися для розв'язання задач, які заздалегідь не визначені.

У практичній діяльності використовується комплексний підхід, який з одного боку дозволяє розв'язувати конкретні інформаційні та функціональні задачі, а з іншого боку – враховує можливість додавання нових застосувань. У загальному випадку існує два підходи до проектування БД: низхідне проектування і висхідне проектування.

Низхідне проектування починається з визначення наборів даних, потім визначаються елементи даних для кожного з таких наборів. Цей процес включає в себе ідентифікацію різних типів сутностей і визначення атрибутів кожної сутності.

*Низхідне проектування* включає операції декомпозиції, що передбачає заміну вихідної множини відношень, що входять в схему БД, іншою множиною відношень, які є проєкціями вихідних відношень. Цей підхід рекомендується застосовувати у тих випадках, коли кількість, різноманітність та складність сутностей, зв'язків і транзакцій значна за розмірами. Найбільш поширеними моделями для цього проектування є моделі "сутність – зв'язок" (ER-моделі, Entity-Relationship model).

*Висхідне проектування* починається з виявлення елементів даних, які потім групуються в набори даних. Спочатку визначаються атрибути, які потім об'єднуються в сутності. Висхідне проектування включає операції синтезу, що передбачає виконання компоновки із заданої множини функціональних залежностей між об'єктами предметної області вихідних відношень схеми БД.

Цей підхід рекомендується застосовувати у тому випадку, якщо розробляється невелика БД з незначною кількістю об'єктів, атрибутів і транзакцій.

#### **4.2. Етап концептуального (інфологічного) проектування. ER-діаграми**

Результатом цього етапу є подання предметної області у вигляді інформаційних об'єктів і зв'язків з одним чи декількома відношеннями (таблицями).

Концептуальна модель представляє загальний погляд на дані. Розрізняють два головних підходи до моделювання даних при концептуальному проектуванні:

- семантичні моделі;
- об'єктні моделі.

*Семантичні моделі* головну увагу приділяють структурі даних. Найбільш поширеною семантичною моделлю є модель "сутність–зв'язок" (Entity Relationship model, ER-модель). ER-модель складається із сутностей, зв'язків, атрибутів, доменів атрибутів, ключів. Моделювання даних відображає логічну структуру даних, так само, як блок-схеми алгоритмів відображають логічну структуру програми.

*Об'єктні моделі* головну увагу приділяють поведінці об'єктів даних і засобам маніпуляції даними. Головне поняття таких моделей – об'єкт, тобто сутність, яка має стан і поведінку. Стан об'єкта визначається сукупністю його атрибутів, а поведінка об'єкта визначається сукупністю операцій специфікованих для нього.

Опис предметної області в міфологічній моделі зазвичай проводиться в термінах "об'єкт–відношення", хоча на практиці використовуються частіше терміни "сутність–зв'язок".

**Сутність** (об'єкт) – множина екземплярів реальних або абстрактних однотипних об'єктів, присутніх в предметній області. Інформація про сутність повинна зберігатися в базі даних і бути доступною в будь-який момент для користувача. Називати сутність прийнято іменником в однині. Наприклад, студент, викладач – це об'єкт або сутність, а Гнатів, Іваненко – конкретні прізвища студента та викладача, є елементами (екземплярами) сутності. Сутності зв'язані між собою **зв'язками**.

Сутність має набір властивостей, які ще називають **атрибутами**. У таблицях БД атрибути відповідають полям (стовпцям) сутностей.

Для того щоб одну сутність можна було відрізнити від іншої, вибирається один або кілька атрибутів з унікальним значенням (унікальною комбінацією значень), який називається **ідентифікатором сутності**. Часто подібні атрибути називають ключовими.

Атрибути можуть бути:

- 1) простими і складеними;
- 2) ключовими і неключовими;
- 3) обов'язковими і необов'язковими;
- 4) однозначними і багатозначними.

Щодо поділу атрибутів на прості і складові можна зробити наступне зауваження: складовий атрибут, наприклад такий як адреса, може включати в себе набір значень – індекс, країна, місто, вулиця, будинок. При побудові інфологічної моделі такі атрибути допустимі. При переході до даталогічної моделі все залежить від обраної моделі даних і можливостей СКБД. Наприклад, в реляційній моделі їх треба буде представляти як множину простих.

Значення обов'язкових атрибутів завжди має бути задано. Необов'язкові атрибути можуть бути відсутніми (бути не задані).

Атрибути також можуть бути однозначними і багатозначними. Для прикладу, на кафедрі є лише один завідувач, у якого може бути декілька номерів телефонів – внутрішній, міський, мобільні від декількох операторів тощо. При побудові міфологічної моделі предметної області допустимо вказувати і багатозначні атрибути, але при переході на більш низький рівень проектування, коли треба враховувати особливості обраної моделі даних і СКБД, від подібних атрибутів, можливо, доведеться позбавлятися.

Атрибути повинні бути задані на доменах (домен в даному випадку – це множина допустимих значень атрибута). На ER-діаграмах, які будуть розглядатися нижче, зазвичай домени не вказуються.

**Зв'язок** (відношення) при інфологічному проектуванні описує наявний в предметній області логічний зв'язок між двома або більше сутностями або сутністю сама з собою (рекурсивний зв'язок).

В будь-якому зв'язку виділяють два кінці (відповідно до існуючої пари сутностей, що зв'язуються). Причому зв'язок повинен мати ім'я, ступінь, обов'язковість). Існує поняття ступеня зв'язку між сутностями, що входять у зв'язок. *Ступінь зв'язку* визначає, яка кількість екземплярів однієї сутності може бути зв'язана з екземплярами іншої сутності, що входить в цей зв'язок.

Виділяють такі типи зв'язків:

- 1) "один-до-одного";
- 2) "один-до-багатьох";
- 3) "багато-до-багатьох".

Позначаються ступені зв'язку "один" – 1 та "багато" латинською літерою N або M, наприклад: 1:N, M:1, N:M, 1:1.

Наведемо формальні визначення для даних типів зв'язків. Нехай є дві сутності (A і B), що беруть участь в зв'язку R. Якщо кожен екземпляр сутності A пов'язаний не більше ніж з одним екземпляром сутності B, а кожен екземпляр сутності B пов'язаний не більше ніж з одним екземпляром сутності A, то зв'язок R має тип "один-до-одного" (рис. 4.1).

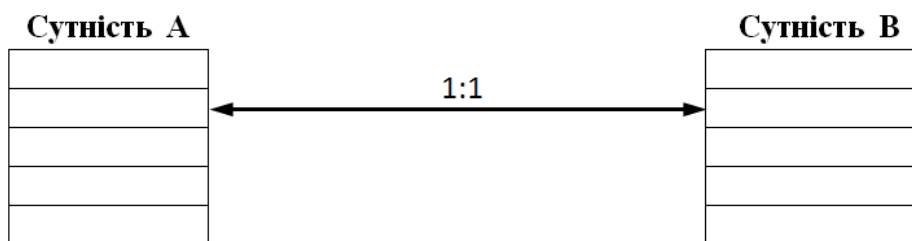


Рис. 4.1. Графічне представлення відношення "один-до-одного"

На практиці даний тип зв'язку зустрічається досить рідко.

Зв'язок R має тип "один-до-багатьох", якщо екземпляр сутності A може бути пов'язаний з нулем, одним або декількома екземплярами сутності B, а екземпляр сутності B може бути пов'язаний не більше ніж з одним екземпляром сутності A (рис. 4.2а).

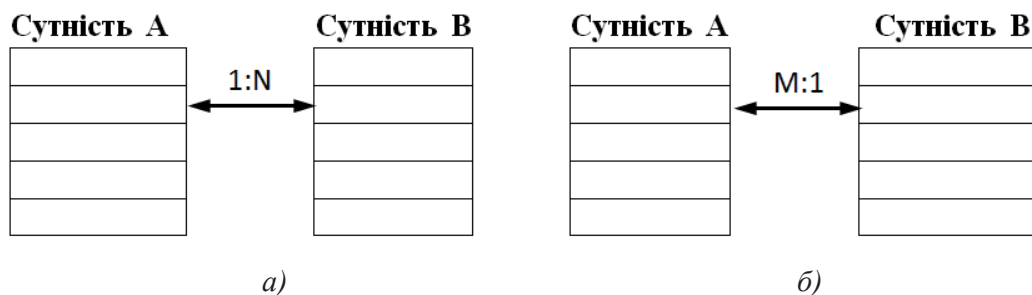


Рис. 4.2. Графічне представлення відношення:

а) "один-до-багатьох"; б) "багато-до-одного"

Зворотним по відношенню до типу "один-до-багатьох" є тип зв'язку "багато-до-одного" (рис. 4.2 б).

Зв'язки типу "один-до-багатьох" найбільш часто зустрічаються при розробці БД. Наприклад, якщо один абонент може взяти в бібліотеці декілька книг, а кожну книгу може взяти тільки один абонент, тоді зв'язок між абонентами і книгами матиме тип "один-до-багатьох".

Зв'язок між сутностями А і В матиме тип "багато-до-багатьох", якщо екземпляр сутності А може бути пов'язаний з нулем, одним або декількома екземплярами сутності В і екземпляр сутності В також може бути пов'язаний з нулем, одним або декількома екземплярами сутності А (рис. 4.3).

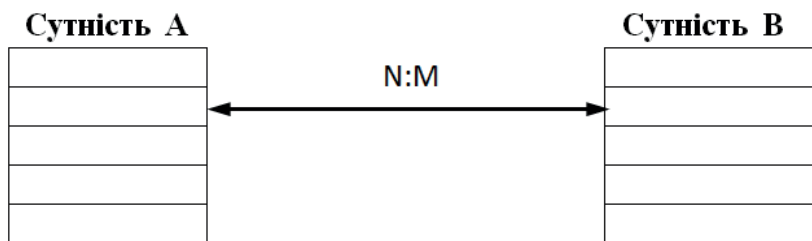


Рис. 4.3. Графічне представлення відношення "багато-до-багатьох"

Наприклад, якщо один курс можуть читати кілька викладачів, а один викладач може читати декілька курсів, то між сутностями "Викладач" і "Курс" є зв'язок типу "багато-до-багатьох".

У більшості випадків в процесі подальшого проектування БД зв'язки типу "багато-до-багатьох" видаляються шляхом введення складових (асоційованих) сутностей і визначення двох зв'язків типу "один-до-багатьох".

*Складова (асоційована) сутність* представляє дані, які асоціюються зі зв'язком між двома або більше сутностями. Введення даного типу сутностей пов'язано з тим, що зв'язок, за своєю суттю, не повинен мати власні властивостями. Але часто трапляється, що це не так. Наприклад, зв'язок між пасажиром і авіарейсом, яким він летить, може бути охарактеризована власними параметрами: номер місця, клас, ціна, дата придбання квитка, тощо. Щоб уявити всі ці дані, можна ввести складову сутність "Квиток".

Кожен об'єкт характеризується *поведінкою*, що вказує, які дії і коли можна виконати з даним об'єктом.

Існує декілька різних нотацій (мов) або правил зображення діаграм "сутність-зв'язок" (ER-діаграм). Історично першою є нотація Чена. В стандартах IDEF модель «сутність-зв'язок» реалізується нотацією IDEF1X. Використовуються також нотації Мартіна і Баркера, в тому числі і графічні елементи UML.

Під *нотацією* розуміється позначення типів зв'язків між двома сутностями.

Вибір тієї чи іншої нотації може визначатися як корпоративними стандартами компанії-розробника, так і вимогами замовника. Тому розробнику БД корисно знати декілька нотацій.

#### **4.2.1. ER-діаграма в нотації Чена**

При графічному відображенні концептуальної моделі "сутність-зв'язок" в нотації Чена сутність зображується у вигляді прямокутника, що містить її ім'я, яке виражається іменником. Ім'я сутності повинно бути унікальним в рамках однієї моделі.

Зв'язок зображується у вигляді ромба. Якщо сутність бере участь у зв'язку, вони пов'язуються лінією. Якщо зв'язок не є обов'язковим – тоді лінія пунктирна.

Ім'я зв'язку зазвичай виражається дієсловом в теперішньому часі: «має», «належить», тощо або дієсловом з пояснювальними словами: «включає в себе», і т. д. Ім'я може бути одне для всього зв'язку або окремо для кожного з кінців зв'язку. У другому випадку, назва лівого кінця зв'язку вказується над лінією зв'язку, а правого – під нею. Кожна з назв розташовуються поруч з сутністю, до якої вона відноситься.

*Тип зв'язку* вказується індексами «1», «N» або «M» над відповідною лінією.

Атрибути зображуються у вигляді овалів і зв'язуються лінією з відповідною сутністю. Імена атрибутів записуються всередині овалу і виражаються іменником в однині (можливо, з уточнюючими словами). Серед атрибутів виділяється ключ сутності – ненадлишковий набір атрибутів, значення яких в сукупності є унікальними для кожного екземпляра сутності. Зазвичай ключ сутності підкреслюється.

Отже, концептуальна модель "сутність-зв'язок" в нотації Чена зображується схемою, для побудови якої використовуються такі елементи (рис. 4.4):

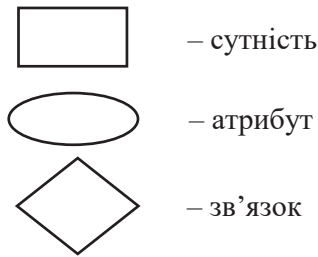


Рис. 4.4. Елементи для побудови концептуальної моделі БД в нотації Чена

Під час побудови ER-діаграми в нотації Чена використовують наступні правила.

- Кожна сутність концептуальної моделі подається у вигляді відношення (таблиці). Найменування сутності – назва таблиці.
- Кожний екземпляр сутності – рядок таблиці, який зазвичай, називають записом або кортежем.
- Кожний атрибут сутності – назва стовпця таблиці (поле). Сукупність полів утворюють структуру таблиці. Оскільки у відношенні не повинно бути повторюваних записів, то в будь-якому відношенні обов'язково має бути поле, що забезпечує унікальність (неповторювальність) кожного запису (кортежу). Таке поле називають **полем первинного ключа**.

Таким чином, будь-яка БД є складовою двох елементів:

- структури;
- даних.

На рис. 4.5 зображені дві сутності "Працівник" та "Керівник" і заданий зв'язок "Підпорядковується" типу "один-до-багатьох" (одному керівнику можуть підпорядковуватися кілька людей, кожен працівник має "над собою" тільки одного прямого керівника). Для сутності "Працівник" також позначені її атрибути. Тут атрибут "id\_Працівника" є ключовим, атрибут "Адреса\_Працівника" – складовим.



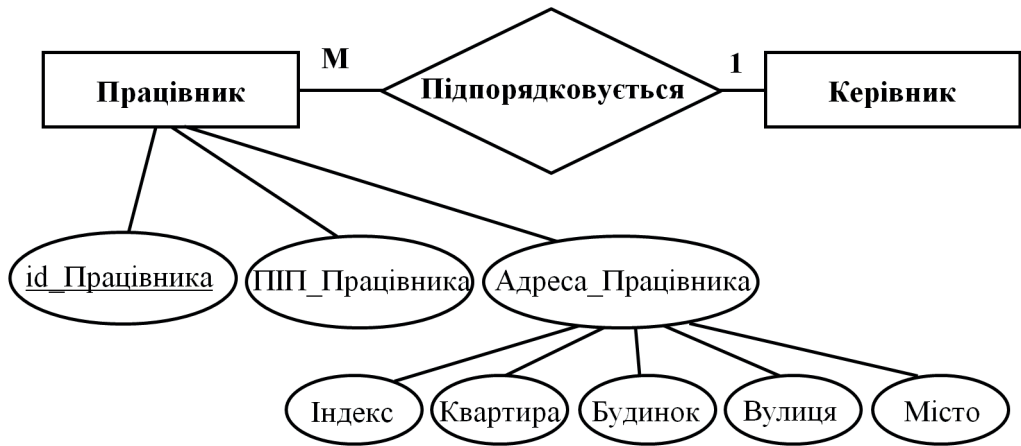


Рис. 4.5. ER-діаграма в нотації Чена

Під час перетворення таблиць, які відображають зв'язки, необхідно враховувати властивості зв'язку, а саме його *ступінь* та *обов'язковість*.

- Якщо ступінь зв'язку між двома сутностями є 1:1 і участь обох сутностей у цьому зв'язку є обов'язковою, тобто всі записи обох таблиць об'єднані зв'язком, то необхідно створити одне відношення (таблицю), в яке ввійдуть всі атрибути обох сутностей.
- Якщо ступінь зв'язку між двома сутностями є 1:1, записи однієї з них обов'язково входять у зв'язок, а записи іншої ні, тоді для кожної сутності створюється відношення (таблиця), причому для тієї таблиці, для якої всі записи входять у зв'язок, додається додатковий атрибут – первинний ключ відношення з необов'язковим зв'язком.
- Якщо ступінь зв'язку між двома сутностями є 1:1 і записи обох таблиць необов'язково входять у цей зв'язок, тоді необхідно для зв'язку створити додаткове відношення (таблицю), у яке включити первинні ключі відношень.
- Якщо ступінь зв'язку між двома сутностями є 1:M і участь сутності зі ступенем зв'язку M є обов'язковою, тоді необхідно створити два відношення. Сутність зі ступенем зв'язку 1 перетворюється у самостійне відношення. Сутність зі ступенем зв'язку M також перетворюється у самостійне відношення, проте сукупність її полів доповнюється ключовим атрибутом сутності зі ступенем зв'язку 1.

- Якщо ступінь зв'язку між двома сутностями є 1:M і участь сутності зі ступенем зв'язку M не обов'язкова, тоді необхідно створити три відношення. Вихідні сутності перетворюються у самостійні відношення, крім того створюється третє відношення з іменем назви зв'язку, яке складається з ключових атрибутів вихідних сутностей.
- Якщо ступінь зв'язку між двома сутностями є N:M, то необхідно створити три відношення. Вихідні сутності перетворюються у самостійні відношення, крім того створюється третє відношення з іменем назви зв'язку, яке складається з ключових атрибутів вихідних сутностей.

Для пояснення викладеного матеріалу розглянемо приклад бази даних "Бібліотека". Діаграма сутність-зв'язок виглядатиме: читач – замовляє – книгу (рис. 4.6).

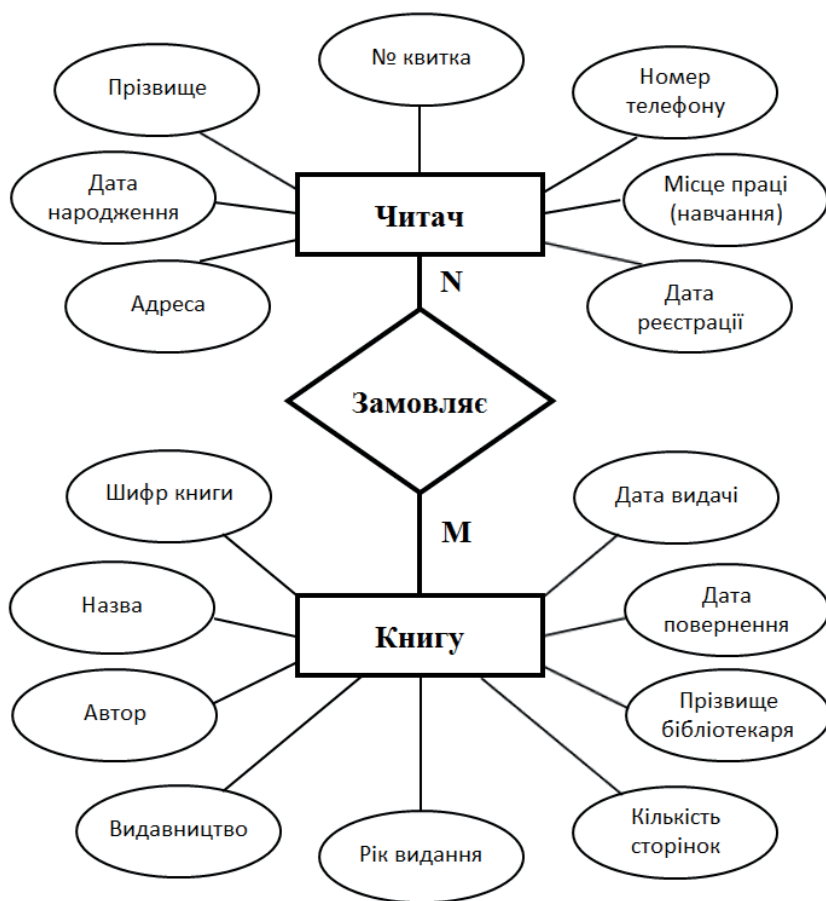


Рис. 4.6. Діаграма сутність-зв'язок

*Предметна область* міститиме інформацію про всі книжки у бібліотеці (назва, автор, рік видання, видавництво, кількість примірників тощо) та про всіх абонентів бібліотеки (прізвище, рік народження, адреса, місце проживання, місце навчання чи праці тощо).

Оскільки один читач може замовити багато книжок, а з другої сторони одну книгу можуть замовити багато читачів (є декілька примірників), тому між сутностями читач і книжки є зв'язок типу "багато-до-багатьох" N:M, що вимагає перетворення цих відношень шляхом створення третьої сутності, наприклад, "Формуляр", куди увійдуть поля первинних ключів обох сутностей і крім того додаткові атрибути, що затверджують факт отримання книжки.

Таким чином, створена додаткова сутність дозволяє отримати зв'язки між сутностями типу "один-до-багатьох": один читач може замовити декілька книг (1:N), що буде відзначено у сутності (таблиці) "Формуляр" і одну книгу можуть замовити декілька читачів (1:M), що також буде відзначено у сутності "Формуляр" (рис. 4.7).

#### **4.2.2. ER-діаграми в нотації Баркера і Мартіна (Information Engineering)**

Нотації Р. Баркера (Richard Barker) та нотації Information Engineering (IE) Дж. Мартіна (James Martin) і К. Фінкештейна (Clive Finkelstein) більшою мірою орієнтовані на БД реляційного типу.

У запропонованих нотаціях, на відміну від нотації Пітера Чена, сутності зображуються у вигляді прямокутника, що містить в заголовку ім'я сутності, і далі йде перелік атрибутів. Ключові атрибути виділяються на діаграмі певним шрифтом, спеціальними символами або відокремлюються рискою від інших.

Передбачається, що всі зв'язки є бінарними (тобто тільки з двома учасниками) і зображуються лінією, що об'єднує дві сутності.

На рис. 4.8 представлені правила зображення зв'язків в нотаціях Баркера і Мартіна.

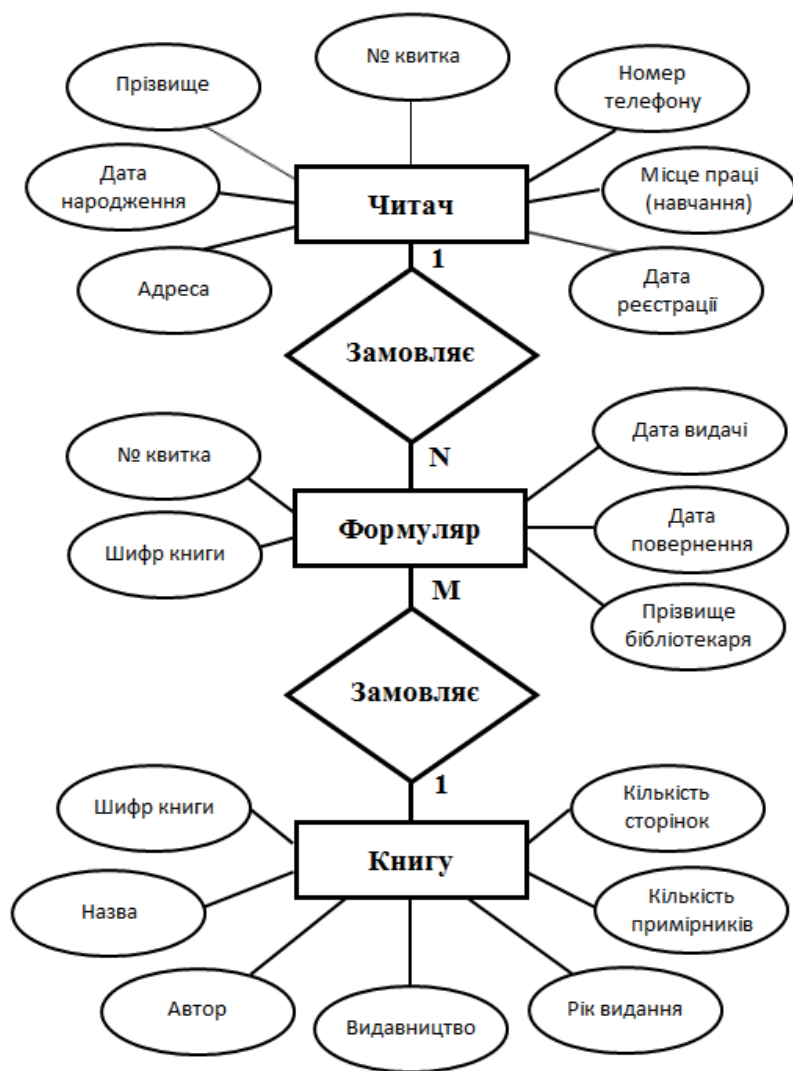


Рис. 4.7. Діаграма сутність-зв'язок БД "Бібліотека"

| нотація Баркера |             |
|-----------------|-------------|
| Позначення      | Тип зв'язку |
| -----           | 0,1         |
| ————            | 1,1         |
| ----->          | 0,N         |
| ————>           | 1,N         |

| нотація Мартіна (ІЕ) |             |
|----------------------|-------------|
| Позначення           | Тип зв'язку |
| ————                 | немає       |
| ———o—                | 0,1         |
| ———+—                | 1,1         |
| ———o>                | 0,N         |
| ———+>                | 1,N         |
| ———>                 | M,N         |

Рис. 4.8. Правила зображення зв'язків в нотації Баркера та Мартіна (ІЕ)

Слід зазначити, що через особливості зображення зв'язків нотації Баркера і Мартіна в літературі іноді називають "crow's foot notation" (дослівно – "нотація воронячої лапки") (рис. 4.9).

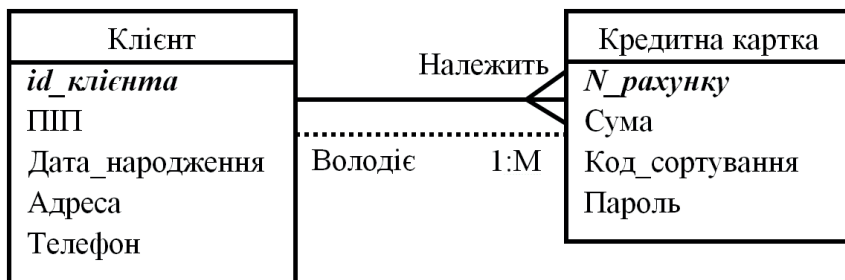


Рис. 4.9. Модель нотації Баркера

На рис. 4.10 наведено фрагмент діаграми в нотації Мартіна, що зображає дві сутності "Клієнт" і "Замовлення" і зв'язок між ними. Первинні ключі на рисунку виділяються символом "#". Передбачається, що:

- клієнт може здійснити одне, декілька замовлень або ж жодного;
- одне замовлення може бути здійснено одним і тільки одним клієнтом.

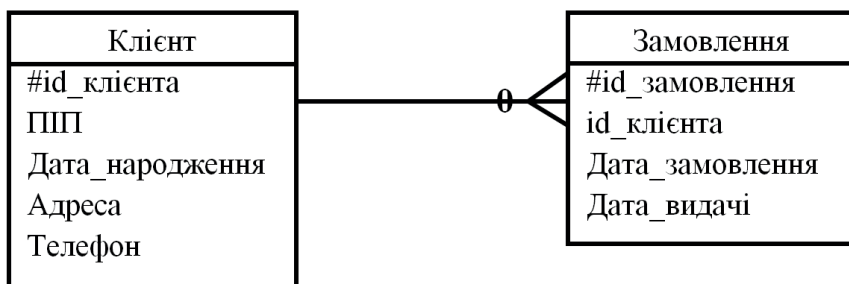


Рис. 4.10. ER-діаграма в нотації Мартіна

#### 4.2.3. Проектування баз даних з використанням методології IDEF1X

Методологія IDEF1X призначена для побудови концептуальної схеми реляційної бази даних, яка була б незалежною від програмної платформи її кінцевої реалізації.

Для проектування концептуальної моделі бази даних з використанням нотації IDEF1X можна також скористатись інструментальними CASE-засобами (CaseStudio, ERWinDataModeler, SyBasePowerDesigner, тощо), що надає змогу реалізувати практично повністю етап проектування інформаційної системи від

опису бізнес-функцій до структурування даних і перекладу моделі в фізичну базу даних. Це означає, що на основі розробленої моделі в зазначених CASE-засобах подалі може бути згенерований скрипт, виконання якого в середовищі обраної СКБД призведе до створення необхідних структур БД.

Такий підхід зобов'язує розробника БД враховувати особливості обраної СКБД, наприклад конкретні типи даних, в той час як в попередніх нотаціях достатньо було зазначити лише узагальнений тип даних, наприклад, числовий, рядковий тощо.

IDEF1X використовує поняття сутностей, атрибутів, відношень і ключів аналогічно, як і попередньо розглянутих нотаціях, хоча і є деякі особливості.

Для позначення сутностей використовується прямокутник, але сутності можуть бути як незалежні, так і залежні (рис. 4.11).



Рис. 4.11. Позначення сутностей в нотації IDEF1X

*Незалежна сутність* містить незалежні дані, які завжди присутні в системі. При цьому відношення з іншими сутностями можуть як існувати, так і бути відсутніми.

У свою чергу, *залежна сутність* містить дані, що залежать від інших сутностей в системі. Тому вона повинна завжди мати відношення з іншими сутностями.

Для позначення первинного ключа до найменування атрибута додається аббревіатура "PK" (Primary Key). При позначенні зовнішнього ключа до атрибуту додається аббревіатура "FK" (Foreign Key). Таким чином, опис сутності містить необхідний набір відомостей про її структуру (рис. 4.12).

## Особа

|                                 |
|---------------------------------|
| Id_клієнта: INTEGER (PK)        |
| ПІП: CHARACTER(45) NOT NULL     |
| Дата_народження: DATE NOT NULL  |
| Телефон: CHARACTER(13) NOT NULL |

Рис. 4.12. Приклад детального опису сутності в нотації IDEF1X

Як і в інших нотаціях, в нотації IDEF1X можна вказувати змістове наповнення зв'язку і його потужність (кардинальність).

Для позначення множинності примірників сутності, що пов'язуються з одним екземпляром іншої сутності, на кінці лінії ставиться жирна крапка.

Зв'язки можуть бути ідентифікуючими, неідентифікуючими, повної або неповної категорії, а також неспецифічними.

Зв'язок називається **ідентифікуючим**, якщо екземпляр дочірньої сутності ідентифікується через її зв'язки з батьківською сутністю, тобто дочірня сутність при ідентифікуючому зв'язку завжди є залежною. При цьому атрибути, що становлять первинний ключ батьківської сутності, будуть входити у первинний ключ дочірньої сутності. Батьківська сутність в ідентифікуючому зв'язку, в залежності від її зв'язків з іншими сутностями, може бути як незалежною, так і залежною.

Зв'язок називається **неідентифікуючим**, якщо екземпляр дочірньої сутності ідентифікується інакше, ніж через зв'язок з батьківською сутністю. При цьому атрибути, що становлять первинний ключ батьківської сутності, будуть входити до складу неключових атрибутів дочірньої сутності.

Однак, щоб вказати потужність (кардинальність) зв'язку, недостатньо використовувати позначення крапкою – потрібно ще вказати відповідне літерне позначення:

- Z – екземпляр першої сутності може бути пов'язаний не більше ніж з одним екземпляром другої сутності;
- Р – екземпляр першої сутності повинен бути обов'язково пов'язаний з екземплярами другої сутності, в припущенні можливості зв'язку з довільною кількістю екземплярів;

- N – екземпляр першої сутності повинен бути пов'язаний точно з N примірниками другої сутності.

Позначення кардинальності ідентифікуючих зв'язків наведено на рис. 4.13.

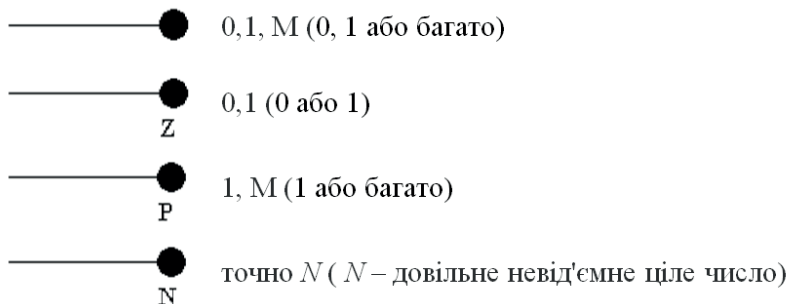


Рис. 4.13. Позначення кардинальності зв'язків в нотації IDEF1X

Якщо ж зв'язок неідентифікуючий, тоді лінія буде пунктирною.

Допустимість порожніх Null-значень у неідентифікуючих зв'язках зображується порожнім ромбиком на лінії зв'язку з боку батьківської сутності.

Якщо на кінці зв'язку літера не вказується, то такий зв'язок позначає можливість зв'язку екземпляра першої сутності з будь-якою кількістю екземплярів другої сутності, допускаючи можливість відсутності пов'язаного примірника другої сутності.

Так, для позначення зв'язку "багато-до-багатьох" (М.М) жирна крапка буде розміщуватися по обидва боки зв'язку.

Розглянемо приклад, який ілюструє використання різних типів сутностей і зв'язків, що визначаються нотацією IDEF1X. Нехай потрібно розробити БД для зберігання інформації про локальну мережу комп'ютерів кафедри.

Комп'ютери мають унікальні інвентарні номери і імена, причому для одного комп'ютера може бути використано декілька додаткових імен (псевдонімів). Наприклад, комп'ютер С3876495201 може мати псевдоніми а52\_01 і а52\_dendiuk. Комп'ютери розміщені в різних аудиторіях кафедри, на них встановлено ПО.

Фрагмент логічної моделі БД представлений на рис. 4.14.



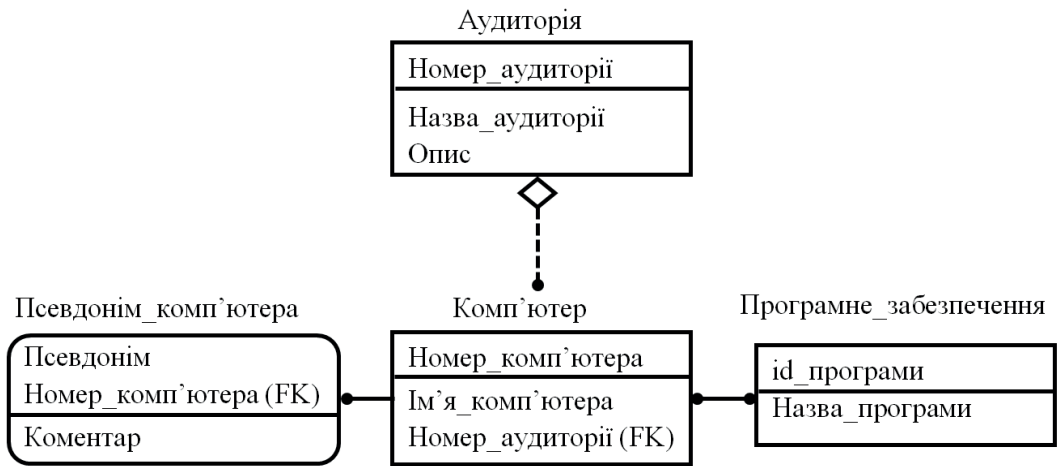


Рис. 4.14. Фрагмент логічної моделі бази даних в нотації IDEF1X

Модель бази даних, заснована на ключах (англ. Key-based), є більш докладною, оскільки включає всі сутності, їх ключі, а також зв'язки.

Звичайно ж, зв'язок "багато-до-багатьох" Комп'ютер–Програмне\_забезпечення необхідно подалі перетворити, ввівши додаткову проміжну сутність і таким сином знизивши потужність зв'язків, наприклад, на зв'язки "один-до-багатьох".

#### 4.2.4. Нотація бази даних UML

Багато сучасних інформаційних систем будується на основі об'єктно-орієнтованого підходу, і бази даних, що використовуються в цих системах, найчастіше реалізуються також за цими ж принципами.

Сучасні технології розробки та програмування дають досить великий набір інструментів для реалізації об'єктно-орієнтованих баз даних, зокрема і UML (Unified Modeling Language) – уніфіковану мову моделювання.

Unified Modeling Language (UML) – уніфікована мова моделювання і фактично є галузевим стандартом у сфері розробки програмного забезпечення, IT-інфраструктури і бізнес-систем.

UML описує об'єкт в єдиному заданому синтаксисі, тому її правила будуть зрозумілими для всіх, хто знайомий з цією графічною мовою.

Опис об'єктів предметної області, які подаються в логічній моделі бази даних у вигляді **сутностей**, в нотації UML виконується за допомогою класів.

Кожен клас представляється у вигляді прямокутника, поділеного на блоки.

Ім'я сутності вказується у верхній частині, а атрибутивний склад розміщується в другому блоці позначення класу (рис. 4.15).

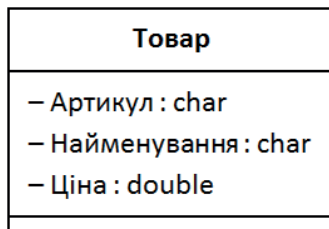


Рис. 4.15. Приклад подання класу в нотації UML

Як ім'я сутності зазвичай використовується іменник в однині (наприклад, ВІДДІЛ, ПРОЕКТ, СПІВРОБІТНИК).

Поряд з описом атрибутів в класі вказуються типи даних, які будуть регламентувати правила подання та обробки.

Оскільки при моделюванні бази даних немає необхідності описувати дії (процедури), які виконуються над даними, то третя (нижня) частина позначення не заповнюється.

Бінарні зв'язки позначаються лініями із зазначенням імені зв'язку та потужності зв'язку з боку кожної сутності. Потужність зв'язку визначає кількість об'єктів, що об'єднуються з кожним об'єктом на іншому кінці зв'язку:

- 1 .. \* – один або більше;
- 0 .. \* – нуль або більше;
- 1 – рівно один;
- 0..1 – може бути один.

Може бути заданий також діапазон (наприклад, 1..10) або точна кількість, відмінна від 1.

В нотації UML немає спеціального позначення для подання зв'язку «багато-до-багатьох», але суть даного типу зв'язку фактично є зв'язуванням рівноправних елементів, що в UML може бути представлено асоціативним зв'язком між рівноправними класами (рис. 4.16).

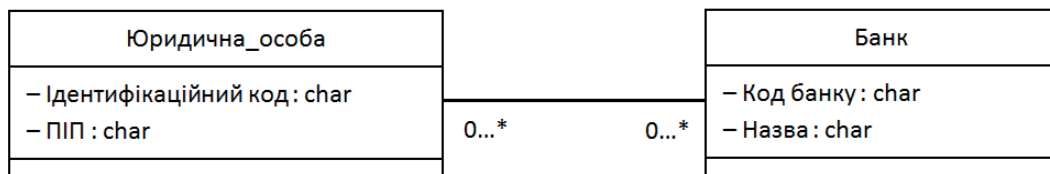


Рис. 4.16. Асоціативний зв'язок між рівноправними класами

Оскільки вказані класи є рівноправними, то зазвичай нижня межа у зв'язку з обох сторін буде "0", показуючи, що кожен клас може не мати пов'язаних примірників з іншим класом.

Особливість цього позначення полягає в тому, що на кінцях лінії відсутні будь-які кінцеві елементи, а кардинальність, що вказується на асоціативних зв'язках, має верхньою межею значення - "\*", що позначає можливість подання безлічі екземплярів відповідного класу.

Іншим варіантом асоціативного зв'язку є зв'язок, що показує взаємодію екземплярів класів шляхом вказання спрямованості навігації або агрегації (рис. 4.17).

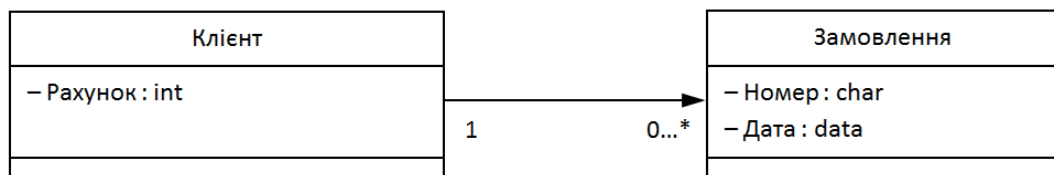


Рис. 4.17. Приклад асоціативного зв'язку з напрямком навігації

Спрямованість навігації в асоціативному зв'язку показує необхідність використання примірника першого класу в складі примірника другого класу. Тим самим, асоціативний зв'язок з напрямком навігації дає можливість уявити простий зв'язок «один-до-багатьох».

Для позначення n-арного зв'язку використовується ромб, всередині якого вказується ім'я зв'язку. Кратність n-арного зв'язку для кожної сутності показує, яким може бути кількість її об'єктів у разі, якщо з боку інших сутностей-учасниць в зв'язку бере участь рівно по одному об'єкту.

Наприклад, тристоронній зв'язок на рис. 4.18 визначає, що поставляти партії будь-яких деталей для будь-якого проекту може будь-який постачальник.

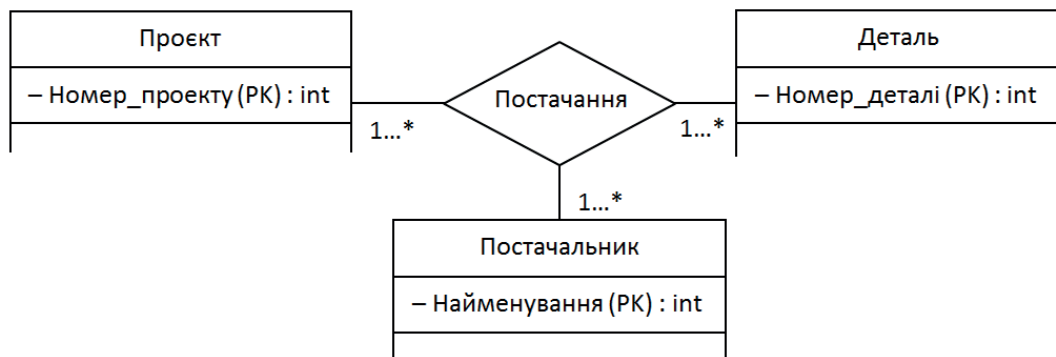


Рис. 4.18. Приклад тристороннього зв'язку

Мова UML належить об'єктному світу і цей світ набагато складніший за реляційний світ. Оскільки UML може використовуватися для уніфікованого об'єктно-орієнтованого моделювання всього, що завгодно, в ньому міститься маса різних понять, термінів і варіантів використання, абсолютно надлишкових з точки зору проектування реляційних БД. Якщо виокремити з загального механізму діаграм класів те, що дійсно потрібно для проектування реляційних БД, то ми отримаємо в точності ER-діаграми з іншою нотацією і термінологією.

Хочу ще раз підкреслити, що в контексті проектування реляційних БД структурні методи проектування, засновані на використанні ER-діаграм, і об'єктно-орієнтовані методи, засновані на використанні мови UML, розрізняються, головним чином, лише термінологією. ER-модель концептуально простіше UML, в ній менше понять, термінів, варіантів використання.

#### 4.2.5. CASE-засоби

Завдання проектування БД для сучасної інформаційної системи корпоративного рівня може бути досить трудомісткою і вимагати спільної роботи великої групи фахівців: фахівців в предметній області, для якої розробляється БД, аналітиків, розробників БД, розробників прикладного ПО тощо. Для автоматизації цього процесу широко використовуються CASE-засоби.

CASE (англ. Computer-Aided Software Engineering) – набір інструментів і методів програмної інженерії для проектування програмного забезпечення, що допомагає забезпечити високу якість програм, відсутність помилок і простоту в обслуговуванні програмних продуктів.

Як приклад можна назвати програмні продукти:

- ERwin Data Modeler (розробник – компанія CA Technologies);
- ER / Studio (розробник – Embarcadero Technologies);
- PowerDesigner (розробник – компанія Sybase, в даний час придбана SAP).

Більшість же реляційних баз даних, за винятком MS Access, складаються з двох окремих компонентів: «back-end», де зберігаються дані і «front-end» – призначений для користувача інтерфейс для взаємодії з даними. Цей тип конструкції досить розумний, так як він розпаралелює дворівневу модель програмування, яка відділяє шар даних від інтерфейсу користувача і дозволяє сконцентрувати ринок ПЗ безпосередньо на покращення своїх продуктів.

На етапі розроблення БД ми говоримо про «back-end» інформаційних систем, і тому розглянемо CASE-засоби суцільно для БД типу MySQL.

В Інтернеті можна знайти багато продуктів для розробки і адміністрування баз даних MySQL.

**1. *Перше місце***, по праву належить інструменту MySQL Workbench (розробка компанії Sun Systems / Oracle), який може працювати на платформах Microsoft Windows, Mac OS X і Linux. Workbench об'єднує в собі розробку й адміністрування баз даних і є наступником DBDesigner4. MySQL Workbench поширюється під вільною ліцензією – Community Edition і з щорічної оплачуваної підпискою – Standard Edition. Остання включає в себе додаткові можливості, які здатні істотно покращити продуктивність, як розробників, так і адміністраторів баз даних.

Популярним Workbench робить:

- можливість представити модель БД в графічному вигляді, а також редагування даних в таблиці;
- наявність простого і функціонального механізму по створенню зв'язків між полями таблиць, серед яких реалізована зв'язок «багато-до-багатьох» з можливістю створення таблиці зв'язків;
- функція Reverse Engineering дозволяє відновлювати структуру таблиць і зв'язків з тієї, яка була реалізована раніше і зберігається на сервері БД;

– наявність редактора SQL-запитів, який дає можливість при відправці на сервер отримувати відповідь в табличному вигляді і інші можливості.

**2. Друге місце** займає Navicat (розробка компанії PremiumSoft CyberTech Ltd) – інструмент для розробки та адміністрування баз даних, який працює на будь-якому сервері MySQL, починаючи з версії 3.21. Для MySQL, Navicat доступний для роботи на платформах Microsoft Windows, Mac OS X і Linux.

Вартість продукту варіюється від 199 до 379 дол. США.

Популярним Navicat робить:

- наявність візуального конструктора запитів;
- можливість імпорту, експорту та резервного копіювання даних;
- можливість створювати звіти;
- SSH і HTTP туннелінг;
- міграція та синхронізація даних і структури;
- інструмент для планування завдань і інші можливості.

**3. Третє місце** займає phpMyAdmin – безкоштовний додаток з відкритим кодом, призначений для адміністрування СКБД MySQL. В phpMyAdmin є веб-інтерфейс за допомогою якого можна адмініструвати сервер MySQL, запускати команди і переглядати вміст таблиць і БД через браузер.

Популярним phpMyAdmin робить:

- можливість управляти СКБД MySQL без безпосереднього введення SQL команд;
- панель управління phpMyAdmin надає можливість адміністрування виділених БД;
- інтенсивний розвиток;
- можливість інтегрувати phpMyAdmin у власні розробки завдяки ліцензії GNU General Public License та інші можливості.

**4. dbForge Studio for MySQL** – інструмент, який представляє інтерес як для користувачів MySQL, так і для розробників БД. З його допомогою можна легко автоматизувати рутинну роботу і заощадити час. Сьогодні dbForge Studio for MySQL представлений в трьох редакціях: Express, Standard і Professional, що дозволяє вибрати необхідний інструмент саме для вас. Користуватися dbForge

Studio for MySQL можна як безкоштовною, так і комерційною версією, ціна останньої становить 49,95 дол. США (стандартне видання) і 99,99 дол. США (фахове видання).

Можливості dbForge Studio:

- наявність засобів для централізованого адміністрування;
- інструменти для порівняння БД;
- візуальний профілювальник запитів;
- можливість управляти привілеями користувачів;
- наявність Дизайнера БД, який дозволяє будувати візуальні діаграми;
- поліпшена робота з проектами БД і інші можливості.

5. HeidiSQL – безкоштовний інструмент для управління базами даних. Гідна альтернатива phpMyAdmin, яка дозволяє створювати і редагувати таблиці, уявлення, тригери, процедури, а також переглядати і редагувати дані. Також HeidiSQL надає можливість експорту даних як в SQL файл, так і в буфер обміну на інших серверах.

Можливості HeidiSQL:

- можливість підключатися до сервера за допомогою командного рядка;
- можливість пакетної оптимізації і відновлення таблиць;
- можливість редагування стовпців, індексів і зовнішніх ключів таблиць, редагування тіла і параметрів SQL процедур, тригерів і ін .;
- просте форматування невпорядкованих SQL;
- синхронізація таблиці між різними базами даних та інші можливості.

6. SQL Maestro для MySQL – інструмент для адміністрування, розробки та управління найбільш використовуваних СКБД. Зручний графічний інтерфейс дає можливість виконувати SQL запити і скрипти, керувати привілеями користувачів, експортувати і створювати резервні копії даних. Залежно від обраної ліцензії і варіанти використання, вартість даного інструменту варіюється від 99 до 1949 дол. США.

Можливості SQL Maestro для MySQL:

- підтримка версій MySQL сервера з версії 3.23;
- наявність конструктора баз даних;

- можливість редагування, угруповання, сортування та фільтрації даних;
- візуальний конструктор запитів;
- SSH і HTTP туннелінг;
- BLOB-редактор і інші можливості.

7. EMS SQL Manager for MySQL – інструмент для розробки та адміністрування баз даних, який підтримує різні функції MySQL і працює з усіма версіями MySQL старше 3.23. Дозволяє візуально редагувати, імпортувати і експортувати БД, виконувати сценарії SQL, управляти привілеями користувачів, візуально проектувати бази даних MySQL. Існує платна і безкоштовна версії додатка. Остання має ряд функціональних обмежень. Вартість платної версії варіюється в межах 95 - 245 дол. США.

Можливості EMS SQL Manager for MySQL:

- підтримка даних UTF8;
- просте управління різними об'єктами MySQL;
- сумісність з усіма версіями з 3.23 по 6.0 включно;
- наявність графічних і текстових інструментів для формування запитів;
- SSH і HTTP туннелінг;
- зручний конструктор звітів і інші можливості.

8. SQLyog – один з найбільш потужних інструментів, який поєднує в собі можливості MySQL Administrator, phpMyAdmin і деякі інші інструменти для адміністрування і розробки баз даних. SQLyog працює на платформах Windows і Linux за допомогою Wine. Доступна як безкоштовна, так і платна версія SQLyog. Вартість платної версії - від 99 до 1499 дол. США (варіюється в залежності від кількості користувачів і ліцензії, з підтримкою або без неї).

Можливості SQLyog:

- зручний конструктор запитів;
- можливість синхронізації даних;
- підтримка Unicode;
- SSH і HTTP, HTTPS туннелінг\$
- «розумне» автозавершення роботи;
- інтелектуальне доповнення коду та інші можливості.



9. DBTools Manager – додаток для управління даними, з вбудованою підтримкою MySQL, PostgreSQL, MS Access, MSSQL Server, Oracle та інших БД. DBTools Manager представлений в безкоштовному (Standard) і платному варіанті (Enterprise). Вартість – 69.90 дол. США за одну ліцензію, при покупці декількох ліцензій передбачені знижки.

Можливості DBTools Manager:

- управління базами даних, таблицями;
- наявність редактора запитів;
- наявність майстра створення форм і звітів;
- можливість імпорту та експорту даних з різних джерел, серед яких MSAccess, MS Excel, Paradox, FoxPro, DBF, ODBC таблиці, текстові і XML файли;
- конструктор діаграм і інші можливості.

10. MyDB Studio – безкоштовний інструмент для адміністрування БД MySQL, який дозволяє створювати, редагувати і видаляти записи, таблиці і бази даних. Працює виключно на платформі Windows.

Можливості MyDB Studio:

- можливість підключатися до необмеженої кількості баз даних;
- можливість підключення по SSH каналах;
- створення відкатів і експорт БД в різні формати;
- можливість перенесення, резервного копіювання, також відновлення БД і інші можливості.

### **4.3. Основи роботи з MySQL Workbench**

Для проектування складної бази даних з великою кількістю таблиць і зв'язків, можливостей phpMyAdmin катастрофічно не вистачає. Тому зазвичай використовується продукт MySQL Workbench – чудова безкоштовна програма для роботи з БД MySQL.

MySQL Workbench – інструмент для візуального проектування баз даних, що інтегрує проектування, моделювання, створення та експлуатацію БД в єдине безшовне оточення для системи баз даних MySQL.

Особливості MySQL Workbench дозволяють користувачеві створювати та керувати підключеннями до сервера баз даних та запускати SQL-запити на цих підключеннях до бази даних за допомогою вбудованого редактора SQL.

MySQL Workbench базується на п'яти важливих моментах:

- **Розробка SQL:** дозволяє створювати та керувати підключеннями до серверів баз даних. На додаток до того, що дозволяє користувачеві налаштовувати параметри підключення, MySQL Workbench надає можливість запускати запити SQL на підключеннях до бази даних за допомогою вбудованого редактора SQL.
- **Моделювання даних (дизайн):** дозволяє графічно моделювати схему бази даних, здійснювати зворотне та пряме проектування між схемою та активною базою даних та редагувати всі аспекти бази даних за допомогою всеосяжного редактора таблиць. Редактор таблиць надає зручні засоби для редагування таблиць, стовпців, індексів, тригерів, розділів, параметрів, вставок та привілеїв, процедур та подань.
- **Адміністрування сервера:** Дозволяє керувати екземплярами сервера MySQL, керуючи користувачами, виконуючи резервне копіювання та відновлення, перевіряючи дані аудиту, переглядаючи стан бази даних та контролюючи продуктивність сервера MySQL.
- **Міграція даних:** дозволяє переходити з Microsoft SQL Server, Microsoft Access, Sybase ASE, SQLite, SQL Anywhere, PostgreSQL та інших таблиць, об'єктів та даних RDBMS до MySQL. Міграція також підтримує перехід зі старих версій MySQL на останні версії.
- **Підтримка MySQL Enterprise:** підтримка корпоративних продуктів, таких як MySQL Enterprise Backup, MySQL Firewall та MySQL Audit.

Все це дозволяє архітекторам даних візуалізувати вимоги, спілкуватися із зацікавленими сторонами та вирішувати проблеми проектування до того, як будуть зроблені великі інвестиції часу та ресурсів. Це дозволяє моделювати базу даних на основі моделі, яка є найефективнішою методологією для створення дійсних і ефективних баз даних, забезпечуючи при цьому гнучкість відповідати зростаючим вимогам бізнесу.

Утиліти перевірки моделей та схем дотримуються стандартів найкращої практики моделювання даних, а також застосовують специфічні для MySQL стандарти фізичного проектування, щоб не було помилок під час створення нових ER діаграм або створення фізичних баз даних MySQL.

Стартовий екран програми відображає основні напрямки її функціональності – проектування моделей баз даних та їх адміністрування (рис. 4.19).



Рис. 4.19. Стартове вікно MySQL Workbench

MySQL Workbench надає можливість роботи як з локальними БД, як з БД на SQL-сервері, так і хмарними серверами SQL, розміщених в кластерах.

Оскільки клієнт-серверні БД передбачають розміщення власне БД на віддаленому SQL-сервері, тому роботу з локальними БД опустимо.

Якщо БД на SQL-сервері створена, тоді для роботи необхідно встановити зв'язок з нею. Якщо ж БД на сервері немає, тоді необхідно, щоб її створив користувач з правами системного адміністратора зазвичай за допомогою phpMyAdmin – безкоштовного додатка, який призначений для адміністрування СКБД MySQL.

### 4.3.1. Проектування моделі бази даних

Після того, як логічна модель БД розроблена і нормалізована, MySQL Workbench надає декілька сервісів для фізичної реалізації таблиць БД на SQL-сервері:

- при відкритому з'єднанні безпосередньо на сервері за допомогою конструктора (рис. 4.20);

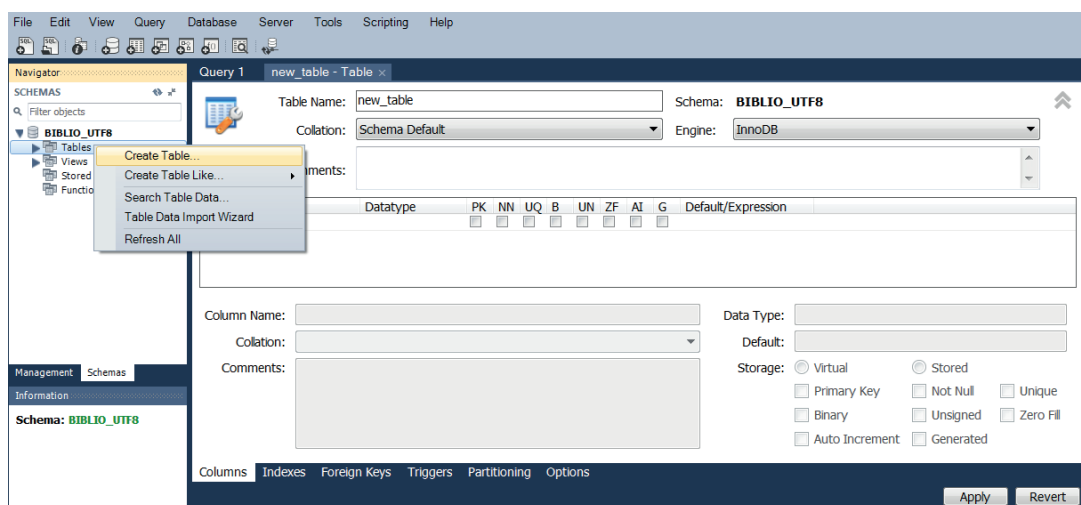


Рис. 4.20. Фізична реалізація БД при відкритому з'єднанні безпосередньо на сервері за допомогою конструктора

- при відкритому з'єднанні безпосередньо на сервері у вікні Query за допомогою SQL-команди CREATE TABLE... (рис. 4.21);

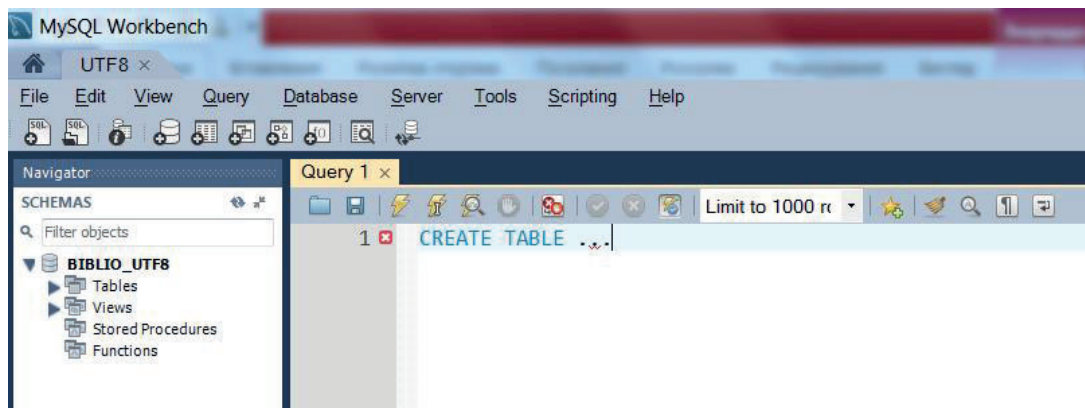


Рис. 4.21. Фізична реалізація БД при відкритому з'єднанні безпосередньо на сервері за допомогою виконання SQL-команд CREATE TABLE...

- при закритому з'єднанні у конструкторі моделі БД (рис 4.22) з наступним імпортом цієї схеми даних з на SQL-сервер.

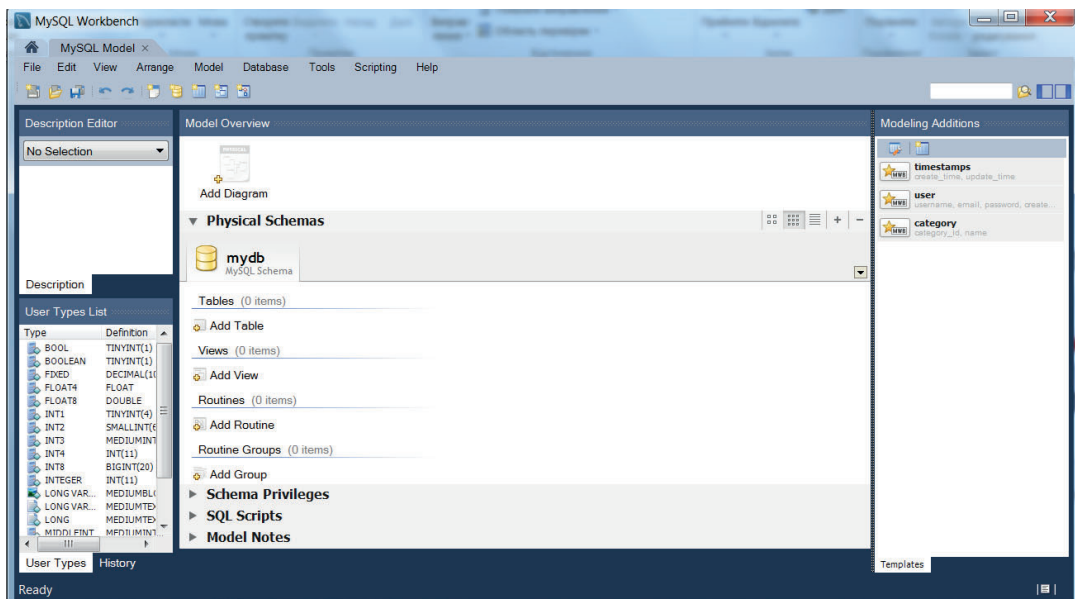


Рис. 4.22. Фізична реалізація БД при закритому з'єднанні з SQL-сервером

Реалізовану таким чином БД можна імпортувати на SQL-сервер за допомогою таких сервісів:

- синхронізувати створену модель з базою даних на сервері;
- використати майстер побудови бази даних;
- почергово скопіювати код створення таблиць у буфер обміну та запустити його на виконання у вікні SQL Editor, тощо.

Таким чином, можливе поєднання способів фізичної реалізації БД, зокрема, можна створити модель БД при закритому з'єднанні та запустити на виконання скопійовані (та відредаговані) команди створення таблиць CREATE TABLE... вже при відкритому з'єднанні з SQL-сервером.

Варто зазначити, що таке поєднання способів фізичної реалізації БД надає додаткові можливості:

- засоби для візуального конструювання таблиць, стовпців, індексів, тригерів, привілеїв, процедур та переглядів;
- візуального конструювання зв'язків між таблицями;

- створення ER діаграми з створеної моделі даних, що дозволяє наочно уявити модель бази даних у графічному вигляді.

Як вже зазначалося вище, все це дозволяє візуалізувати вимоги, спілкуватися із зацікавленими сторонами та вирішувати проблеми проектування до того, як будуть зроблені великі інвестиції часу та ресурсів.

Адже від того, як спроектована база даних, значно залежить в подальшому робота з нею.

#### 4.3.3.1. Створення нової моделі БД

Першим кроком роботи у MySQL Workbench є створення нової моделі БД за командою File – New Model або комбінацією клавіш Ctrl + N. При цьому відкриється конструктор нової моделі mydb (рис. 4.23).

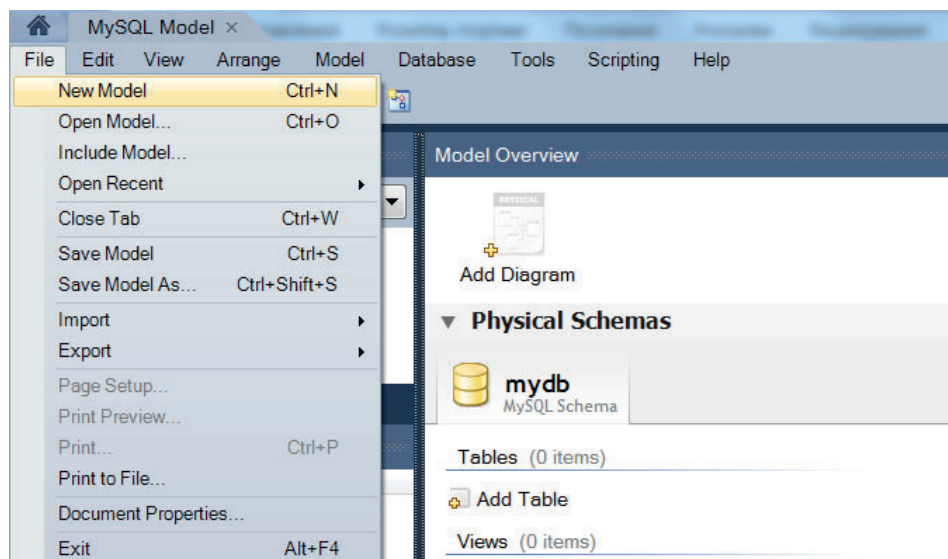


Рис. 4.23. Створення нової моделі бази даних у MySQL Workbench

У зв'язку з тим, що код створення таблиць буде генеруватися, необхідно, щоб ім'я моделі співпадало з іменем бази даних на SQL-сервері. У нашому випадку ім'я БД та ім'я користувача user співпадатимуть.

Тому необхідно для нової схеми mydb відкрити вікно редагування параметрів схеми подвійним кліком або за допомогою контекстного меню Edit Schema..., у якому задати необхідне ім'я БД (рис. 4.24).

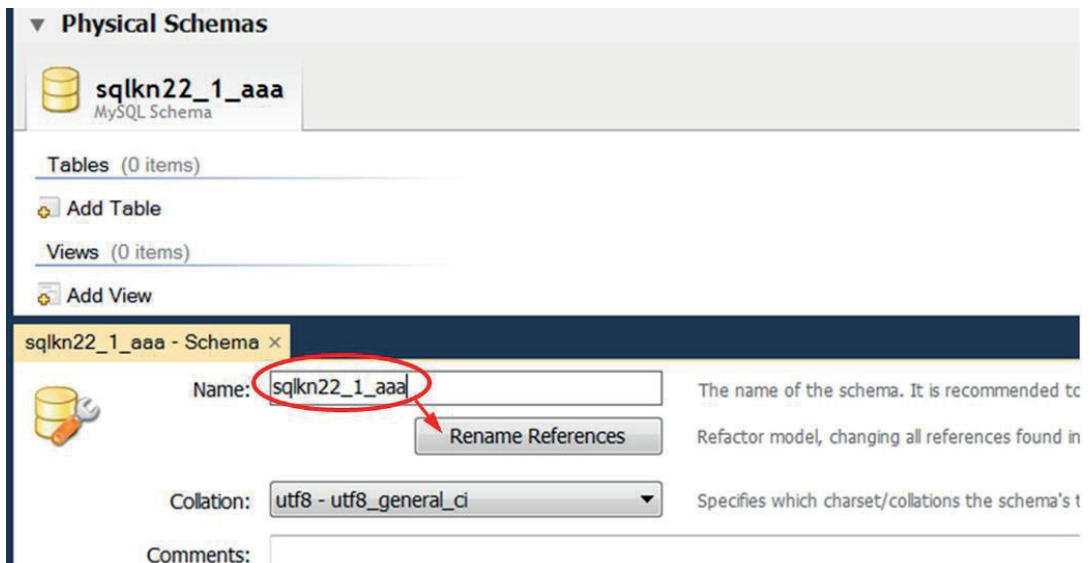


Рис. 4.24. Перейменування моделі бази даних

Як видно з рис. 4.24, кодова таблиця символів, що підтримує кирилицю, за замовчуванням utf8. Забігаючи наперед, варто зазначити, що при створенні таблиць необхідно вказувати ту ж кодову таблицю символів.

#### 4.3.3.2. Додавання таблиць

Список баз даних проекту і список таблиць в межах бази даних буде розташовуватися у вкладці Physical Schema.

Імена таблиць можуть мати до 128 символів. Вони повинні бути унікальними по відношенню до користувача – тобто дві таблиці з однаковими іменами не можуть належати одній БД, проте в іншій БД може бути таблиця з тим же іменем.

Ім'я таблиці повинно записуватися згідно синтаксису запису імен змінних у будь-якій мові програмування:

- починатися з латинської букви, далі іде послідовність латинських букв, цифр і знаку підкреслення;
- не містити розділових і спеціальних символів, наприклад, %, (, /, «, !, ?, тощо;
- не містити пробілів, які можна замінити знаком підкреслення;
- не дублювати команди мови SQL, наприклад, SELECT, CREATE, DROP, ALTER, INSERT, UPDATE, DELETE і т.д.;

- не дублювати імена системних змінних, таких як USER, DATABASE, PASSWORD, FLUSH, PORT, SOCKET, VERSION і т.д.;
- не дублювати назви типів даних – INT, DECIMAL, REAL, FLOAT, DATE, TIME і т.д.;
- не дублювати назви функцій – MIN, SUM, COUNT, SIN, YEAR, LENGTH, TRIM, UPPER, тощо.

Приклади імен таблиць: Katalog001, TypeWare, Class\_Animal тощо.

Варто також зазначити, що на імена полів поширюються ці ж обмеження!

Кожне поле визначається набором властивостей:

- Column Name – ім'я поля;
- Datatype – тип поля (тип даних);
- PK – PRIMARY KEY (поле первинного ключа);
- NN – NOT NULL (поле не може бути порожнім);
- UQ – UNIQUE INDEX (поле повинно містити унікальні значення);
- BIN – BINARY (дані у вигляді двійкових рядків (binary string). Сортування і порівняння засноване на числових значеннях байтів);
- UN – UNSIGNED (цілі без знакові значення);
- ZF – ZEROFILL (дозаповнення нулями – додає до значення нулі ліворуч, якщо довжина значення менше довжини поля. Наприклад, якщо довжина поля INT(5), тоді кожне значення дозаповнюється до 5 символів, 12 = 00012, 400 = 00400 і т.д.);
- AI – AUTO INCRIMENT (лічильник++);
- Default – значення за замовчуванням.

Для створення нової таблиці необхідно вибрати команду з панелі інструментів Add new Table (рис. 4.25 а) або двічі клацнути кнопкою мишки на команді "+Add Table", розміщеної в області фізичної схеми (рис. 4.25 б).



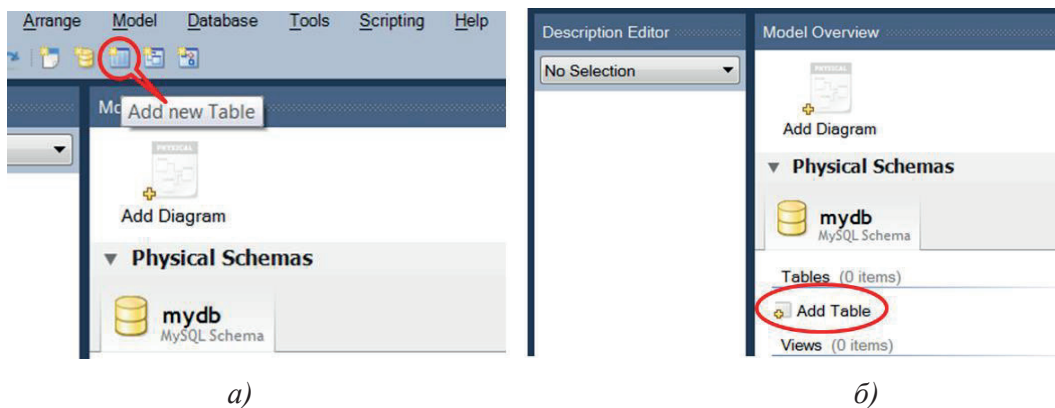


Рис. 4.25. Створення нової таблиці у схемі БД:

а) кнопкою з панелі інструментів; б) кнопкою в області фізичної схеми

У відкритій закладці конструктора необхідно ввести ім'я таблиці, а також назви полів (стовпців, атрибутів) та задати їх властивості (рис. 4.26).

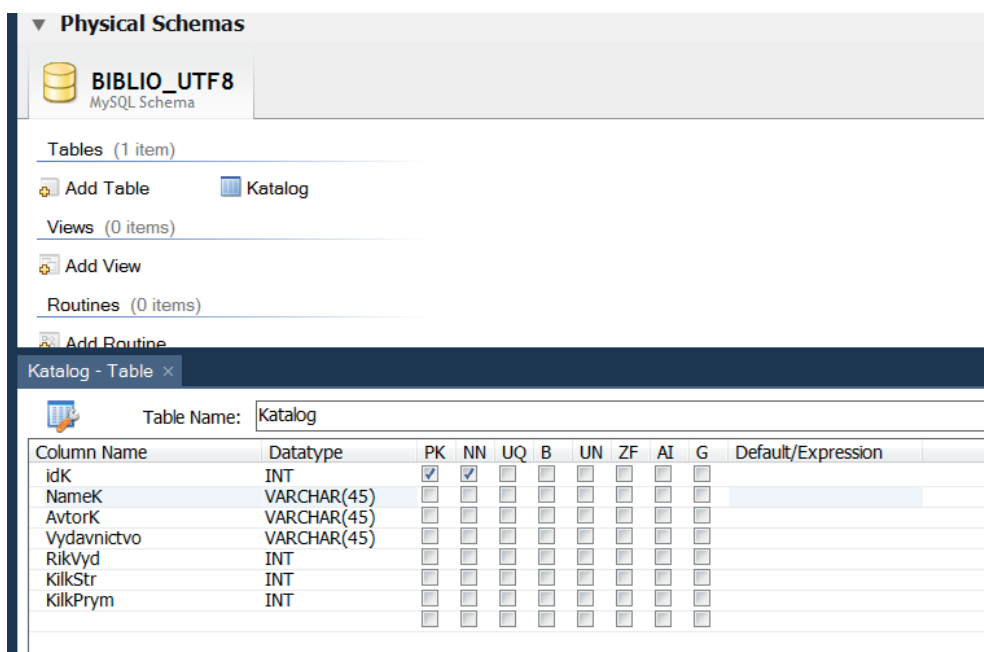


Рис. 4.26. Створення нової таблиці у конструкторі MySQL Workbench

Аналогічно створюються всі решту таблиць БД.

### 4.3.3.3. Управління індексами

У СКБД MySQL індекси є основним засобом прискорення доступу до вмісту таблиць, особливо це стосується запитів, що включають об'єднання декількох таблиць.

СКБД MySQL використовує індекси в декількох аспектах:

- для пошуку рядків, що відповідають певним умовам або рядків, що мають відповідності в інших таблицях при виконанні об'єднання;
- для прискорення пошуку максимального або мінімального значення індексованого стовпця при роботі з функціями MIN() або MAX();
- для прискорення сортування.

Додавати, видаляти і редагувати індекси таблиць можна у вкладці "Indexes" інтерфейсу управління таблицею (рис. 4.27): вводимо назву індексу, вибираємо його тип, потім помічаємо в потрібному порядку список полів, що беруть участь в даному індексі. Порядок полів буде відповідати порядку, в якому були проставлені галочки.

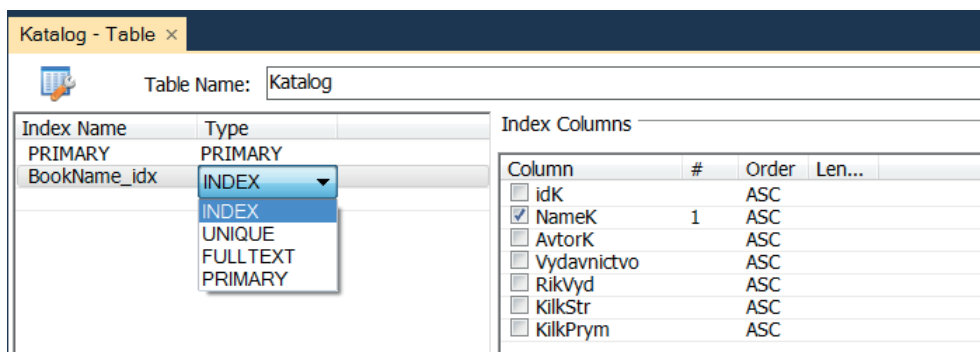


Рис. 4.27. Створення індексів у конструкторі MySQL Workbench

В даному прикладі додано унікальний індекс userIndex до поля nameuser.

Індекс з назвою PRIMARY і типом PRIMARY був автоматично створений за ключовим полем idK, оскільки під час створення структури таблиці поле idK було помічено як поле первинного ключа даної таблиці (рис. 4.26).

Існує декілька типів індексів:

- INDEX або KEY;
- UNIQUE;

- FULLTEXT;
- SPATIAL;
- PRIMARY.

Індекси типу PRIMARY, INDEX і UNIQUE зберігаються у вигляді B-дерев. Рядки автоматично стискаються з видаленням пропусків в префіксах і кінцевих пробілів.

Тип INDEX або KEY відноситься до нормального неоднозначного індексу. Індекс може містити рядки з однаковими значеннями. Цей тип індексів використовується тільки для того, щоб деякі запити працювали швидко.

UNIQUE відноситься до типу, де всі рядки індексу повинні бути унікальними. А також використовується для прискорення запитів. Тип UNIQUE можна використовувати до полів, в яких дозволено значення NULL (вважається, що NULL не дорівнює самому собі), в цьому випадку два рядки можуть бути ідентичними, якщо вони обидва містять значення NULL. Щоб запобігти цьому, необхідно при створенні таблиці встановити для поля атрибут NOT NULL.

PRIMARY діє точно так же, як UNIQUE індекс, за винятком того, що він завжди називається «PRIMARY» і може бути тільки одним (і завжди повинен бути один). Первинний індекс призначений в якості основного засобу, щоб однозначно ідентифікувати будь-який рядок в таблиці. Деякі СКБД (наприклад, InnoDB в MySQL) будуть зберігати записи на диску в тому порядку, в якому вони знаходяться в первинному індексі.

Спеціальні індекси FULLTEXT застосовуються для повнотекстного пошуку. Ці індекси підтримуються тільки таблицями типу MyISAM і вони можуть бути створені тільки з стовпців типу CHAR, VARCHAR і TEXT. Індексування завжди виконується для всього стовпця цілком, часткова індексація не підтримується.

Індекси типу SPATIAL можуть використовуватися тільки в таблицях типу MyISAM і тільки для індексування просторових даних, тобто для індексації багатовимірної інформації, такої, наприклад, як географічні дані – з двовимірними координатами широтою та довготою.

#### 4.3.3.4. Зв'язки між таблицями

Установка зовнішніх ключів і зв'язування таблиць можливо тільки для таблиць InnoDB (ця система зберігання даних вибирається за замовчуванням). Для управління зв'язками в кожній таблиці знаходиться вкладка Foreign Keys.

Для додавання зв'язку між таблицями необхідно відкрити вкладку Foreign Keys дочірньої таблиці, ввести у поле Foreign Key Name ім'я зовнішнього ключа та вибрати у полі Referenced Table таблицю-батька і відповідне поле.

Крім того, у полі Foreign Key Options необхідно налаштувати поведінку зовнішнього ключа при зміні відповідного поля (ON UPDATE) і видаленні (ON DELETE) батьківського запису:

- RESTRICT – видавати помилку при зміні / видаленні батьківського запису;
- CASCADE – оновлювати зовнішній ключ при зміні батьківського запису, видаляти дочірній запис при видаленні батька;
- SET NULL – встановлювати значення зовнішнього ключа NULL при зміні / видаленні батька (неприйнятно для полів, у яких встановлено прапор NOT NULL!);
- NO ACTION – не робити нічого, однак за фактом ефект аналогічний RESTRICT.

Для прикладу, встановимо зв'язок між таблицями і Katalog та Formuljar (рис. 4. 28). Таблиця Formuljar є дочірньою по відношенню до Katalog, оскільки містить поле idK, яке не є первинним ключем, в той час як поле idK в таблиці Katalog є первинним ключем і, відповідно, таблиця Katalog є батьківською.

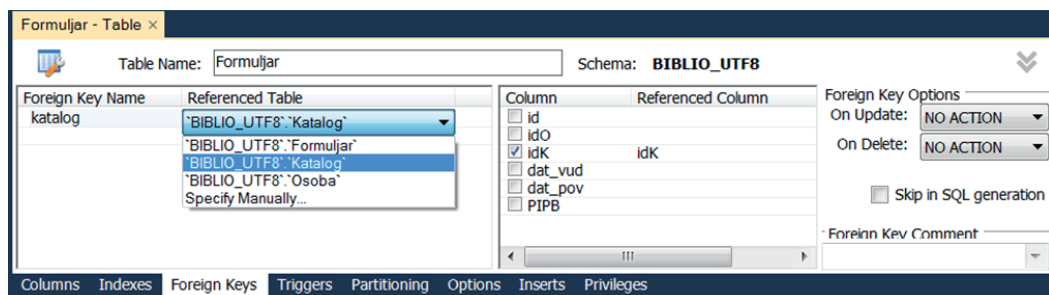


Рис. 4.28. Створення зв'язків між таблицями у конструкторі MySQL Workbench

При створенні зовнішніх ключів в дочірній таблиці автоматично створюються відповідні індекси.

#### 4.3.3.5. Створення ER-діаграми

Модель сутність-зв'язок (ER-модель) (англ. Entity Relationship model, ERM) – модель даних, що дозволяє описувати концептуальні схеми предметної області.

ER-модель використовується при високорівневому (концептуальному) проектуванні баз даних. З її допомогою можна виділити ключові сутності (таблиці) і позначити зв'язки, які можуть встановлюватися між цими таблицями.

Зв'язки між сутностями відповідають логічним відношенням між сутностями, які встановлюють суттєвий зв'язок у даній предметній області. Наприклад, читач замовляє книгу, водій керує тролейбусом, виріб виготовлений з матеріалу.

Зв'язки мають такі характеристики:

- ступінь зв'язку (деколи називають потужність зв'язку);
- клас належності (обов'язковий або необов'язковий);
- тип зв'язку (ідентифікуючий або неідентифікуючий).

**Ступінь зв'язку** (потужність зв'язку) може бути один до одного (1:1), один до багатьох (1:M), багато до одного (M:1) чи багато до багатьох (M:N).

Наприклад, студент читає книжку – такий зв'язок має ступінь 1:1. У студента може бути декілька книжок, але кожна з цих книг є тільки в користуванні цього студента. Такий зв'язок має ступінь (1: M). Можлива інша ситуація, коли декілька студентів мають різні книжки, причому кожен з цих студентів може користуватися книжками друзів. Такий зв'язок матиме ступінь (M:N).

**Клас належності** визначає, обов'язковий є зв'язок або необов'язковий. Обов'язковий клас належності відповідає такому зв'язку, при якому кожен екземпляр сутності обов'язково повинен брати участь у зв'язку, необов'язковий клас належності – деякі екземпляри сутності можуть не брати участі в зв'язку. Наприклад, обов'язковий клас належності: кожен студент має прізвище. І приклад необов'язкового класу належності – студент має iPhone, але не кожен студент має iPhone. Студент може мати інший вид телефону.

**Тип зв'язку** між залежними сутностями може бути ідентифікуючим (identifying relationship) або неідентифікуючим (non-identifying relationship).

Якщо книга не може існувати без автора, а вона не може, тоді тип зв'язку "книга – автор" буде ідентифікуючим. І якщо книга може існувати без власника, а вона може, тоді тип зв'язку "книга – власник" буде неідентифікуючим.

Для представлення схеми даних, сутностей і їх зв'язків в графічному вигляді в MySQL Workbench існує редактор ER-діаграм.

Зв'язки між сутностями зображують лініями. Ідентифікуючий зв'язок показують суцільною, неідентифікуючий – пунктирною лінією.

Створюється ER-діаграма за допомогою команди головного меню Model – Create Diagram From Catalog Object. При цьому відкриється нова вкладка із зображеними таблицями і зв'язками між ними (рис. 4.29).

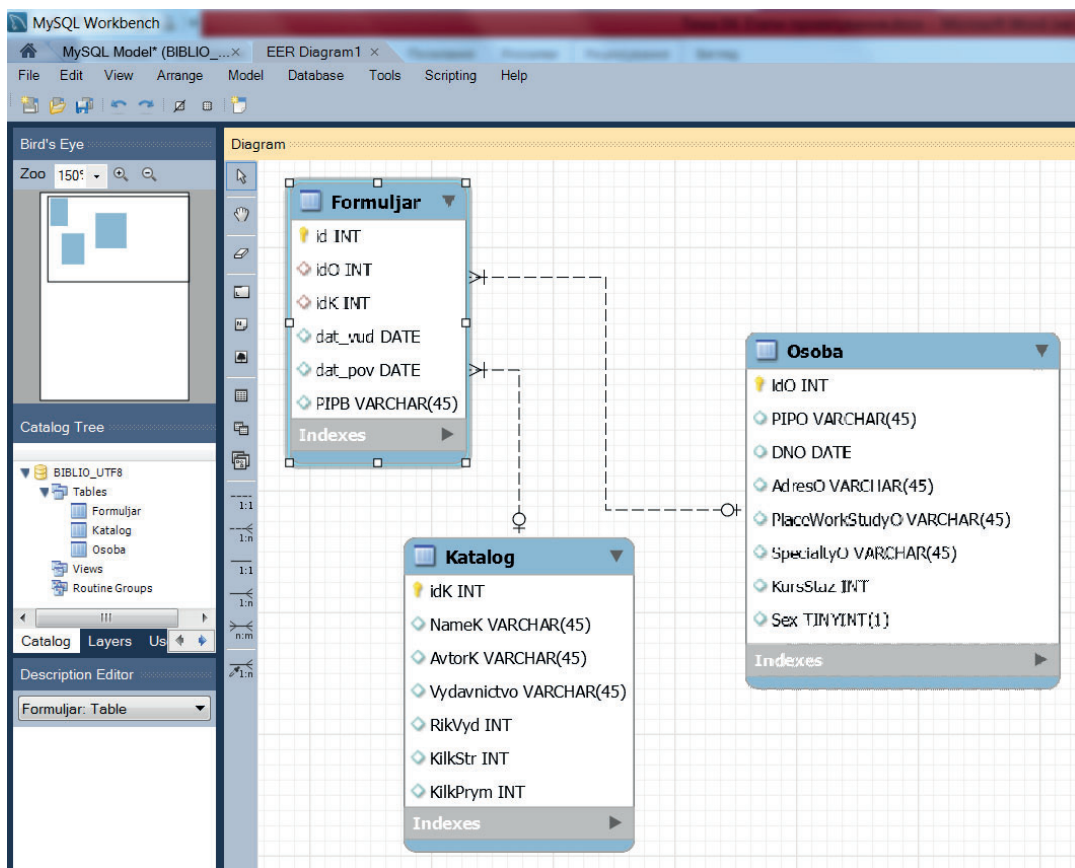


Рис. 4.29. Створення ER-діаграми у Workbench

Якщо зв'язки між таблицями встановлено на етапі створення таблиць у конструкторі (вкладка Foreign Keys, рис. 4.28), тоді MySQL Workbench при створенні ER-діаграми для цих зв'язків автоматично визначить ступінь зв'язку,

клас належності та тип і, відповідно, відобразить на діаграмі, як показано на рис. 4.29.

Щоб додати на діаграму новостворену таблицю (в область Diagram), необхідно просто перетягнути її з вікна Catalog Tree.

Для експорту схеми даних в графічний файл необхідно вибрати "File → Export", а потім один з варіантів (PNG, SVG, PDF, PostScript File).

#### **4.3.2. Підключення MySQL Workbench до MySQL-сервера**

MySQL Workbench – інструмент для візуального проектування баз даних, що інтегрує проектування, моделювання, створення та експлуатацію БД у єдине безшовне оточення для системи баз даних MySQL.

Власне, для фізичного створення БД на MySQL-сервері, або для перенесення схеми БД з MySQL Workbench на MySQL-сервер, першим кроком є встановлення зв'язку з цим сервером, будь від локальний, віддалений чи у хмарі.

Основними параметрами для з'єднання з MySQL-сервером є:

- Ім'я з'єднання;
- Метод з'єднання;
- IP сервера;
- Порт;
- Ім'я користувача;
- Ім'я бази даних;
- Пароль користувача.

Залежно від вибору методу (протоколу) з'єднання – SSH чи SSL, кількість параметрів з'єднання може бути ще більшою.

Протоколи SSH та SSL – це криптографічні протоколи безпечного передавання даних мережею, що використовують більшість одних і тих же криптографічних примітивів, але різняться аутентифікацією за ключами.

У більшості випадків роботи з клієнт-серверними БД цілком достатньо використання стандартного протоколу передачі даних TPC/IP.

Задаються параметри з'єднання у вікні, яке викликається командою New Connection (рис. 4.30) або кнопкою  $\oplus$  панелі MySQL Connection залежно від версії Workbench.

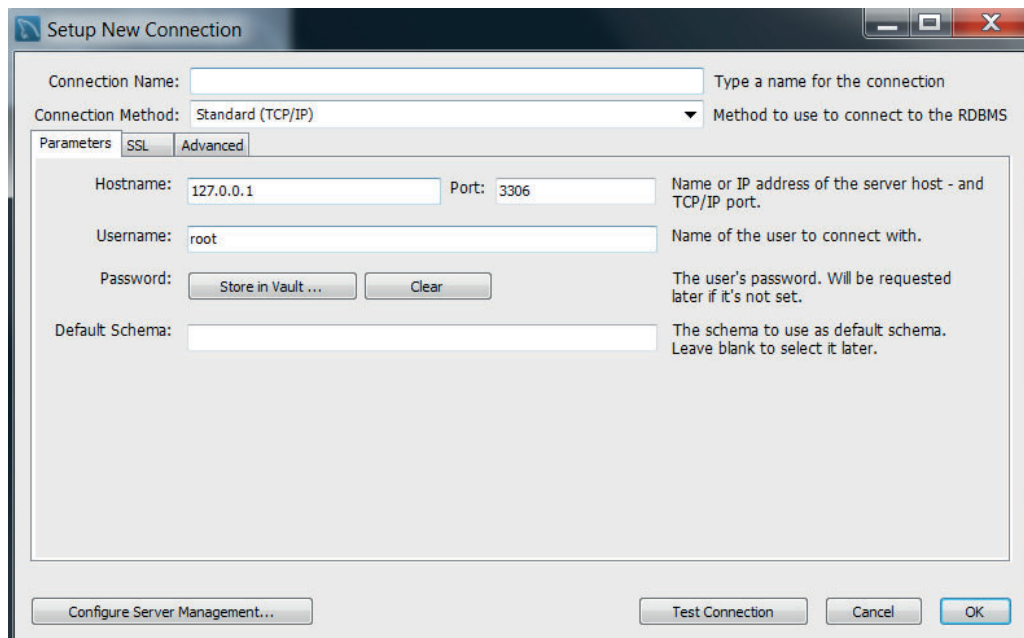


Рис. 4.30. Задавання параметрів з'єднання з сервером з MySQL Workbench

Connection Name є обов'язковим параметром та дозволяє розрізнити з'єднання при роботі з кількома серверами і має нести змістове навантаження про тип з'єднання, користувача або до якої БД здійснюється підключення.

Connection Method визначає способи передачі даних мережею. За замовчуванням пропонується стандартний протокол Standard (TCP/IP).

Hostname визначає IP адресу сервера. За замовчуванням пропонується підключення до локального сервера. Якщо ж підключення здійснюється до віддаленого MySQL сервера, тоді необхідно знати його IP адресу.

Port для MySQL сервера (локального або віддаленого) – 3306. Для хмарної СКБД MySQL порт буде іншим.

Username та Password вказує логін та пароль зареєстрованого користувача на сервері.

Default Schema є необов'язковим параметром, задає використання за замовчуванням однієї з БД при відкритому з'єднанні.



Для роботи MySQL Workbench на стороні сервера необхідно відкрити підключення до SQL-сервера командою Open Connection to Start Querying (рис. 3.19) та у полі Stored Connection вибирати необхідне під'єднання, наприклад UTF8. Для ідентифікації користувача також необхідно ввести пароль користувача на SQL-сервері.

MySQL Workbench відкриє вікно провідника (Object Browser) та вікно SQL-редактора (SQL Editor) (рис. 4.31).

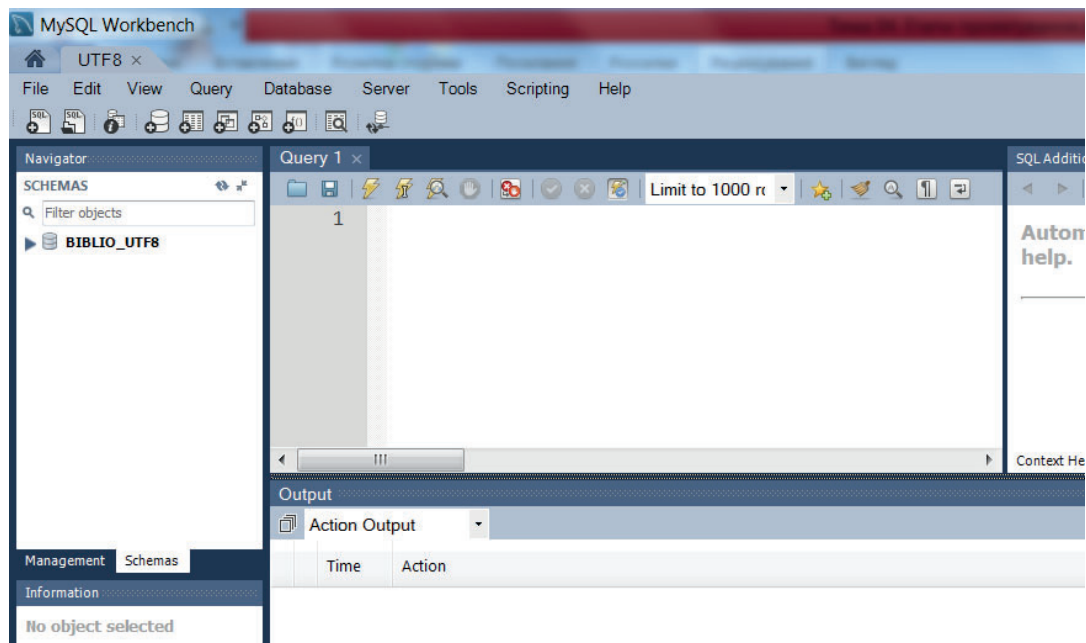


Рис. 4.31. Відкрите з'єднання MySQL Workbench з MySQL-сервером

Тепер у відкритому з'єднанні можна безпосередньо працювати з сервером:

- додавати, змінювати та видаляти бази даних, таблиці, перегляди, збережені процедури (в тому числі транзакції), тригери, тощо;
- додавати, редагувати, видаляти дані та здійснювати вибірки даних з таблиць;
- додавати нових користувачів, змінювати привілеї та паролі наявних користувачів, видаляти користувачів, здійснювати копіювання та відновлення БД, тощо.

Всі ці дії можна реалізовувати як за допомогою меню та відповідних піктограм, так і за допомогою SQL команд у вікні Query.

Основи мови SQL розглянемо в наступних темах, а зараз – етапи створення нових таблиць та зв'язків між ними засобами моделювання Workbench.

#### **4.3.3. Імпорт схеми даних з MySQL WorkBench на MySQL-сервер**

Створити БД на віддаленому SQL-сервері на основі розробленої схеми БД в MySQL Workbench можна декількома способами:

- синхронізувати створену модель з базою даних на сервері;
- використати майстер побудови бази даних;
- почергово скопіювати код створення таблиць у буфер обміну та запустити його на виконання у вікні SQL Editor, тощо.

##### ***4.3.3.1. Синхронізація моделі з базою даних на сервері***

При здійсненні синхронізації MySQL Workbench надає можливість не тільки створення на сервері нових таблиць, переглядів та процедур, але і можливість модифікувати БД на сервері шляхом зміни графічної схеми БД.

Додаючи нові або змінюючи наявні об'єкти (таблиці, перегляди, процедури, а також окремі стовпці таблиць і т.д.) у графічному редакторі, дуже просто змінити так само БД на сервері шляхом синхронізації.

Синхронізація моделі та БД на сервері реалізується командою Database – Synchronize Model... з вікна MySQL Model або EER Diagram (рис. 4.32) та складається з декількох кроків:

- встановлення параметрів з'єднання з SQL сервером шляхом вибору наявного збереженого з'єднання в полі Stored Connection;
- задавання опцій генерації SQL команд зміни БД згідно змін в моделі;
- перевірка коректності підключення до SQL-сервера;
- вибір БД на сервері для синхронізації з моделлю;
- узгодження схеми даних і структури БД на сервері;
- визначення відмінностей між графічною схемою даних і структурою БД на сервері;
- генерація SQL коду приведення БД на сервері до графічної схеми даних;
- виконання згенерованого коду (Execute).

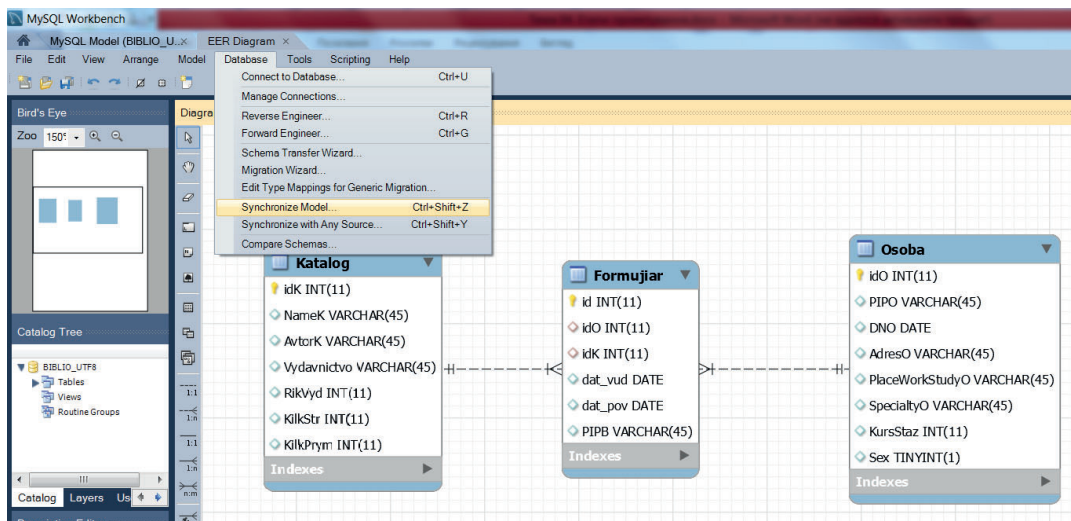


Рис. 4.32. Синхронізація моделі даних та БД на MySQL сервері

Згенерований код за цим способом можна також редагувати і потім виконувати, можна зберегти у файл (Save to File) або скопіювати в буфер обміну (Copy to Clipboard).

Синхронізацію проводиться кожен раз після будь-яких змін у моделі даних, якщо ці зміни необхідно відобразити на сервері.

#### 4.3.3.2. Використання майстра побудови бази даних

Для фізичної реалізації БД на стороні сервера також можна викликати майстер побудови бази даних командою Database – Forward Engineer (рис. 4.33) з вікна MySQL Model або EER Diagram.

Робота майстра здійснюється декілька кроків:

- встановлення параметрів з'єднання з SQL сервером (аналогічно першому кроку синхронізації моделі та БД);
- задавання опцій створення БД. При повторному створенні бази даних для видалення старих таблиць необхідно включити прапорці DROP Objects Before Each CREATE Object та Generate DROP SCHEMA;
- вибір об'єктів (таблиць, переглядів, процедур, тригерів та користувачів) з моделі для створення на їх сервері;
- генерація SQL команд з попереднім переглядом. Як і при синхронізації, так і тут можна редагувати згенеровані SQL команди ще до їх виконання.

Згенеровані SQL команди можна також зберегти у файл (Save to File) або скопіювати в буфер обміну (Copy to Clipboard);

- з'єднання з сервером та виконання SQL команд.

Якщо помилки відсутні, тоді отримаємо повідомлення про успішне завершення роботи майстра побудови бази даних (Forward Engineer Finished Successfully). В іншому випадку, щоб переглянути протокол помилок, необхідно скористатися командою Show Logs.

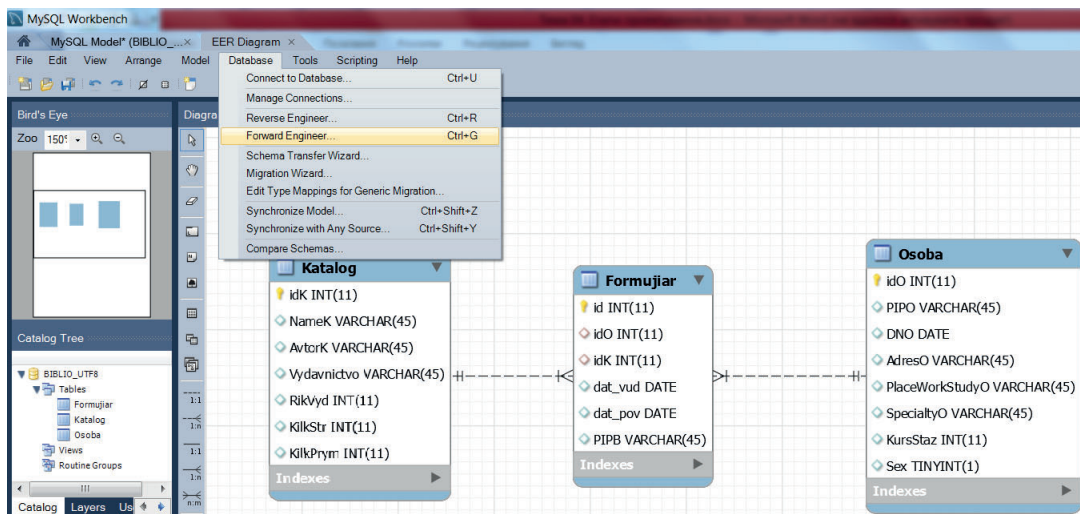


Рис. 4.33. Виклик майстра побудови бази даних

#### 4.3.3.3. Виконання на стороні сервера SQL команд (з моделі даних)

Ще один з варіантів дозволяє створювати таблиці не всі відразу, а по чергово. Для даного алгоритму необхідно на кожній з таблиць у вікні EER Diagram за допомогою ПКМ вибрати команду Copy SQL to Clipboard (копіювати в буфер обміну, рис. 4.34), перейти у вікно SQL редактора (Query) Workbench, вставити їх з буфера обміну та запустити на виконання.

Послідовність створення таблиць має значення, тому спочатку створюються батьківські таблиці, а потім – дочірні.

Якщо ж під час створення таблиці у конструкторі для кожного текстового поля не задано collation кодової таблиці, яка підтримує кирилицю, тоді необхідно явно дописати у кінці SQL команди назву кодової таблиці, яка підтримує кирилицю, наприклад, utf8 або cp1251 (рис. 4.35).

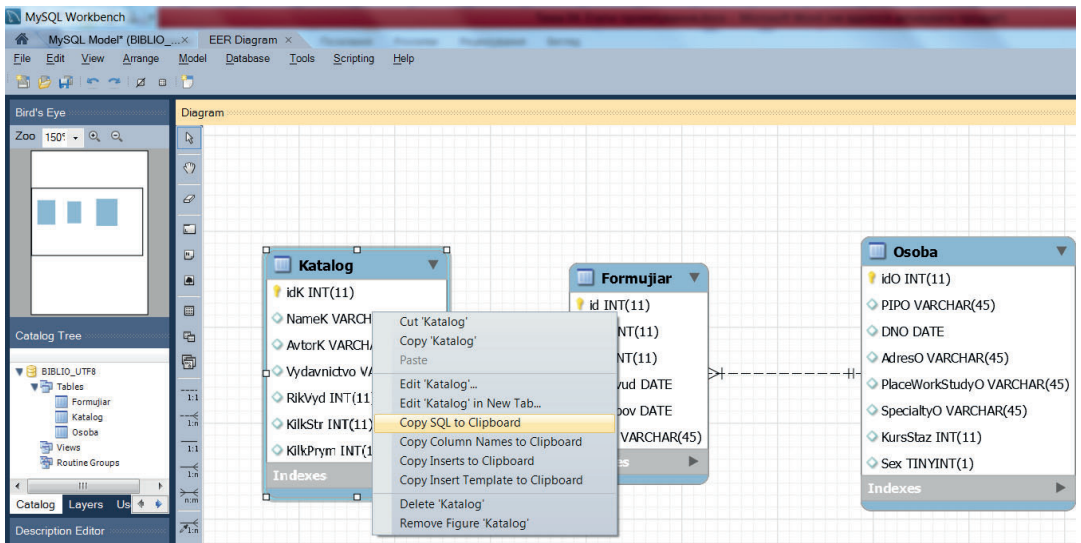


Рис. 4.34. Копіювання SQL коду створення таблиці з моделі даних у буфер обміну

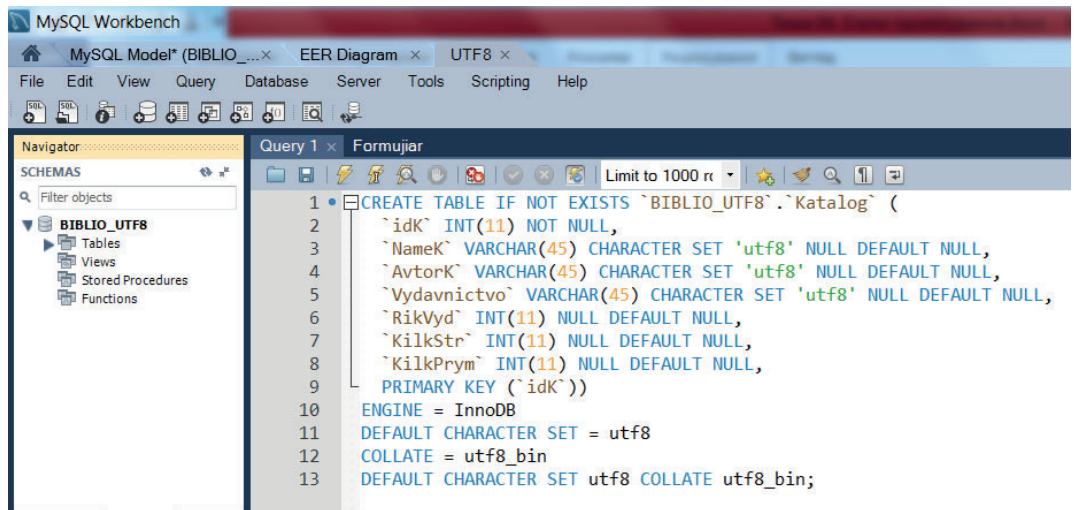


Рис. 4.35. Редагування згенерованого коду створення таблиці

Характерною помилкою при генерації коду створення дочірніх таблиць БД зазвичай є назви зв'язків між таблицями, яка виникає, якщо ключові поля у декількох (або у всіх) таблицях мають однакову назву, наприклад – id. Щоб не модифікувати структури таблиць, в багатьох випадках достатньо у скриптах створення таблиць відредагувати назви зв'язків, наприклад:

...

CONSTRAINT `id1`

...

CONSTRAINT `id2`

...

Але вирішальне значення має етап логічного проектування БД, на якому необхідно детально розробити та нормалізувати схему БД.

### **Контрольні запитання**

1. Які є 4 етапи проектування БД?
2. Які операції включає низхідне проектування?
3. Коли рекомендується застосовувати висхідне проектування?
4. Що таке ER-модель?
5. Що називають сутністю?
6. Якими можуть бути атрибути сутності?
7. Які типи зв'язків виділяють між сутностями?
8. Які правила використовують для побудови ER-діаграми в нотації Чена?
9. Як відрізняються правила зображення зв'язків в нотації Баркера та Мартіна від нотації Чена?
10. Для чого призначена методологія IDEF1X?
11. Яка нотація використовується на основі об'єктно-орієнтованого підходу?
12. Які CASE-засоби використовуються для автоматизації процесу проектування БД?
13. Наведіть популярні продукти для розробки і адміністрування баз даних MySQL. Які з них з вільним доступом?
14. Для чого призначений інструмент MySQL Workbench?
15. Наведіть три сервіси для фізичної реалізації таблиць БД на MySQL-сервері.
16. Вкажіть правила синтаксису запису імен змінних у будь-якій мові програмування.
17. Розшифруйте набір властивостей поля при створенні нової таблиці.
18. Для чого СКБД MySQL використовує індекси?
19. Які бувають індекси?
20. Як в моделі MySQL Workbench створюються зв'язки між таблицями?

21. Як можна налаштувати поведінку зовнішнього ключа при зміні і/або видаленні відповідного поля батьківського запису?
22. Наведіть алгоритм створення ER-діаграми з моделі БД.
23. Які основні параметри вказуються у рядку з'єднання з MySQL-сервером?
24. Які способи створення БД на SQL-сервері на основі розробленої схеми БД Ви знаєте?
25. Яка послідовність створення таблиць на SQL-сервері?



# Розділ 2. РЕЛЯЦІЙНІ БАЗИ ДАНИХ MySQL

## Тема 5. Основи мови SQL

### 5.1. Роль і місце мови SQL

Сьогодні існує багато різноманітних СКБД: Microsoft Access, Microsoft Visual FoxPro, Microsoft SQL Server, Oracle, DB2, MySQL, Informix, і це далеко не повний список. Крім того, зазвичай для користувачів БД створюються застосунки, написані різноманітними мовами програмування. Зважаючи на таке різноманіття і різнобій СКБД і мов програмування для уніфікації керування даними в будь-якій СКБД було розроблено уніфіковану декларативну мову SQL (від англ. Structured Query Language – мова структурованих запитів), яка була затверджена Американським національним інститутом стандартів (ANSI – American National Standards Institute) у 1986 році, а в 1987 році – Міжнародною організацією зі стандартизації (ISO – International Organization for Standardization).

Мова SQL є мовою програмування для взаємодії користувачів з базами даних. Вона застосовується для формування запитів, оновлення і керування реляційними БД, створення схеми бази даних і її модифікації, системи контролю за доступом до бази даних (рис. 5.1).



*Рис. 5.1. Місце мови SQL у роботі користувача з БД*

Незважаючи на наявність міжнародного стандарту ANSI SQL-92, багато компаній, наприклад, Oracle, Sybase, Microsoft, MySQL, що займаються розробкою СКБД, вносять зміни до мови SQL, яка використовується в розроблених ними СКБД, тим самим відступаючи від стандарту. Таким чином з'являються специфічні для кожної конкретної СКБД діалекти мови SQL.



Оскільки SQL не є мовою програмування (тобто не надає засобів для автоматизації операцій з даними), введені різними виробниками розширення стосуються в першу чергу процедурних розширень. Це збережені процедури (англ. stored procedures) і процедурні мови-«надбудови». Практично в кожній СКБД застосовується своя модифікація SQL, кожна з яких може мати декілька версій.

## 5.2. Основні поняття мови SQL

### 5.7.1. Склад мови SQL

Мова SQL об'єднала мови роботи з БД, які використовувалися раніше окремо. У її склад входить декілька сімейств мов (рис. 5.2).

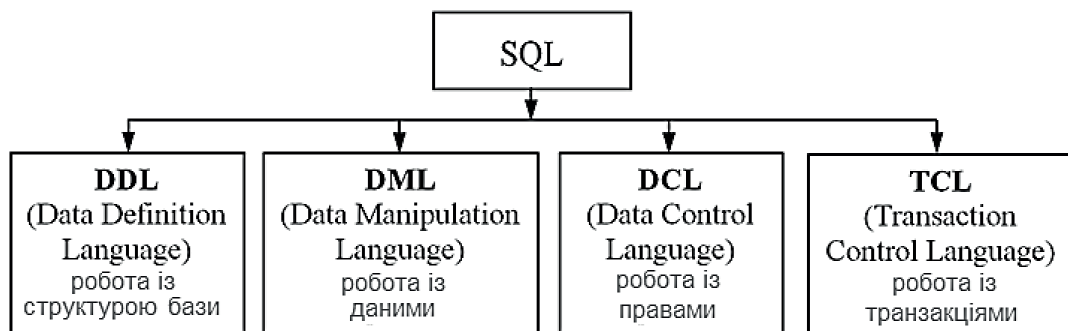


Рис. 5.2. Склад мови SQL

Кожне сімейство має своє функціональне призначення та використовує початкове слово в заяві (запиті), яке майже завжди є дієсловом.

1. Сімейство DDL є частиною SQL та використовується для опису структури даних і має такі команди:

- CREATE (Створити);
- ALTER (Змінити);
- DROP (Видалити).

2. Сімейство DML є частиною SQL та використовується для отримання, вставки, видалення або зміни даних в базах даних і має такі команди:

- SELECT (Вибрати);
- INSERT (Вставити);
- UPDATE (Оновити/Модифікувати);

- DELETE (Видалити).

3. Сімейство DCL це також частина SQL, що використовуються в комп'ютерних програмах або користувачами баз даних для контролю доступу до даних в базах даних. DCL командами є:

- GRANT (Надати) – дозволити визначеним користувачам виконувати визначені маніпуляції;
- DENY (Відмовити) – явна заборона для користувача (має пріоритет над дозволом);
- REVOKE (Скасувати) – скасувати раніше надані права.

4. Сімейство TCL є також частиною SQL і використовуються для роботи з транзакціями. TCL командами є:

- BEGIN TRANSACTION (Почати транзакцію);
- SAVEPOINT (точка збереження) – ділить транзакцію на більш дрібні частини.
- ROLLBACK (Відкат) – відкатує всі зміни, зроблені в контенті поточної транзакції.

### 5.7.2. Оператори мови SQL

Основні оператори, які використовуються у написанні команд, наведено у табл. 5.1.

**Таблиця 5.1. Основні оператори**

| Оператор  | Опис              | Приклад                  |
|-----------|-------------------|--------------------------|
| 1         | 2                 | 3                        |
| =         | Дорівнює          | Author = 'Alcott'        |
| <> або != | Не дорівнює       | Dept <> 'Sales'          |
| >         | Більше            | Hire_Date > '2012-01-31' |
| <         | Менше             | Bonus < 50000.00         |
| >=        | Більше рівно      | Dependents >= 2          |
| <=        | Менше рівно       | Rate <= 0.05             |
| OR        | Логічне додавання |                          |
| AND       | Логічне множення  |                          |

| 1             | 2                                 | 3                              |
|---------------|-----------------------------------|--------------------------------|
| BETWEEN       | Між діапазоном, включно           | Cost BETWEEN 100.00 AND 500.00 |
| LIKE          | Збіг з шаблоном символів          | First_Name LIKE 'Will%'        |
| IN            | Рівно одному з множини вказаних   | DeptCode IN (101, 103, 209)    |
| IS або IS NOT | Перевірка на наявність (не пусте) | Address IS NOT NULL            |

### 5.7.3. Типи даних мови SQL

Основні типи даних мови SQL наведено у табл. 5.2.

**Таблиця 5.2. Основні типи даних, які підтримуються MySQL базах даних**

| Назва типу          | Збережені дані                                                                                                                                                                                                                                                                                    |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1                   | 2                                                                                                                                                                                                                                                                                                 |
| <b>РЯДКОВІ ТИПИ</b> |                                                                                                                                                                                                                                                                                                   |
| CHAR(32)            | Рядок фіксованої довжини, що складається з n символів довжини від 1 до 32767 байт.                                                                                                                                                                                                                |
| VARCHAR(32)         | Рядок змінної довжини. В пам'яті буде збережено рівно стільки символів, скільки містить поточне значення. Довжина від 1 до 32765 байт.                                                                                                                                                            |
| TINYTEXT            | Текст довжиною до 255 байт.                                                                                                                                                                                                                                                                       |
| TEXT                | Текст довжиною до 65 КБ.                                                                                                                                                                                                                                                                          |
| MEDIUMTEXT          | Текст довжиною до 16 МБ.                                                                                                                                                                                                                                                                          |
| LARGETEXT           | Текст довжиною до 4 ГБ.                                                                                                                                                                                                                                                                           |
| <b>ЧИСЛОВІ ТИПИ</b> |                                                                                                                                                                                                                                                                                                   |
| TINYINT             | Цілі числа від -127 до 128, займає 1 байт.                                                                                                                                                                                                                                                        |
| BOOL                | Фактично не представляє окремий тип, а є лише псевдонімом для типу TINYINT(1) і може зберігати два значення – 0 або 1. Однак даний тип може також в якості значення приймати вбудовані константи <b>TRUE</b> (представляє число 1) і <b>FALSE</b> (число 0). Також має псевдонім <b>BOOLEAN</b> . |

| 1                   | 2                                                                                                                                                                                                                                                                                                                                              |
|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| TINYINT UNSIGNED    | Цілі числа від 0 до 255, займає 1 байт.                                                                                                                                                                                                                                                                                                        |
| SMALLINT            | Цілі зі знаком. Займають в пам'яті 16 біт. Діапазон представлених значень від -32768 до 32767.                                                                                                                                                                                                                                                 |
| SMALLINT UNSIGNED   | Цілі числа від 0 до 65535, займає 2 байти.                                                                                                                                                                                                                                                                                                     |
| MEDIUMINT           | Цілі числа від -8388608 до 8388607, займає 3 байти.                                                                                                                                                                                                                                                                                            |
| MEDIUMINT UNSIGNED  | Цілі числа від 0 до 16777215, займає 3 байти.                                                                                                                                                                                                                                                                                                  |
| INT                 | Ціле числа зі знаком. Розмір пам'яті 32 біти. Діапазон значень – від -2 147 483 648 до 2 147 483 647.                                                                                                                                                                                                                                          |
| INT UNSIGNED        | Цілі числа від 0 до 4294967295, займає 4 байти.                                                                                                                                                                                                                                                                                                |
| BIGINT              | Цілі числа від -9 223 372 036 854 775 808 до 9 223 372 036 854 775 807, займає 8 байт.                                                                                                                                                                                                                                                         |
| BIGINT UNSIGNED     | Цілі числа від 0 до 18 446 744 073 709 551 615, займає 8 байт.                                                                                                                                                                                                                                                                                 |
| <b>DECIMAL(4,2)</b> | Десяткове число, де m – кількість всіх цифр в числі без врахування десяткової крапки, n – кількість цифр після крапки. Займає пам'яті 16, 32 або 64 біти. Якщо дане містить меншу кількість знаків, ніж вказано в описі, то резервується пам'ять, що відповідає конкретному даному. Даний тип має також псевдоніми <b>DEC</b> , <b>FIXED</b> . |
| NUMERIC(m,n)        | Відповідає типові DECIMAL, проте для конкретного даного резервується кількість пам'яті, яка відповідає заданим m та n.                                                                                                                                                                                                                         |
| <b>FLOAT</b>        | Число з плаваючою крапкою подвійної точності, яке складається з 7 цифр. Розмір – 32 біти. Діапазон значень від $1,175 \cdot 10^{-38}$ до $2147483647 \cdot 10^{-38}$ . Може приймати форму <b>FLOAT(m,n)</b> , де m – загальна кількість цифр, а n – кількість цифр після коми.                                                                |
| <b>DOUBLE</b>       | Дробові числа з плаваючою крапкою подвійної точності. Може зберігати числа від $-1.7976 \cdot 10^{308}$ до $1.7976 \cdot 10^{308}$ , займає 8 байт. Може приймати форму <b>DOUBLE(m,n)</b> і має псевдонім <b>REAL</b> .                                                                                                                       |

| 1                                             | 2                                                                                                                                                                                                         |
|-----------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| DOUBLE<br>PRECISION                           | Число з плаваючою крапкою подвійної точності, яке складається з 15 цифр. Розмір – 64 біти. Діапазон значень – $2,225 \cdot 10^{-308}$ до $1,797 \cdot 10^{308}$ .                                         |
| <b>ТИПИ ДАНИХ ДЛЯ РОБОТИ З ДАТОЮ ТА ЧАСОМ</b> |                                                                                                                                                                                                           |
| DATE                                          | Значення дати. Розмір 32 біти. Зберігає дати з 1 січня 1000 року до 31 грудня 9999 року (з "1000-01-01" до "9999-12-31"). За замовчуванням використовується формат уууу-mm-dd.                            |
| TIME                                          | Значення цього типу представляє час від 00:00:00.000 до 23:59:59.999. За замовчуванням – формат "hh:mm:ss". Займає 3 байти.                                                                               |
| DATETIME                                      | Об'єднує дату і час, діапазон дат і часу – з 1 січня 1000 року до 31 грудня 9999 року (з "1000-01-01 00:00:00" до "9999-12-31 23:59:59"). За замовчуванням – формат "уууу-mm-dd hh:mm:ss". Займає 8 байт. |
| TIMESTAMP                                     | Об'єднує також дату і час, однак діапазон – від "1970-01-01 00:00:01" UTC до "2038-01-19 03:14:07" UTC. Займає в пам'яті до 4 байти.                                                                      |
| YEAR                                          | Рік у вигляді 4 цифр. Діапазон доступних значень від 1901 до 2155. Займає 1 байт.                                                                                                                         |
| TINYBLOB                                      | Двійковий об'єкт (бінарні дані) у вигляді рядка довжиною до 255 байт.                                                                                                                                     |
| <b>БІНАРНІ ТИПИ</b>                           |                                                                                                                                                                                                           |
| BLOB                                          | Великий двійковий об'єкт. Цей тип використовується для зберігання великих об'ємів (до 65 КБ) текстової інформації, графічних даних, відеокліпів, звуку в цифровому форматі.                               |
| MEDIUMBLOB                                    | Бінарні дані у вигляді рядка довжиною до 16 МБ.                                                                                                                                                           |
| LARGEBLOB                                     | Бінарні дані у вигляді рядка довжиною до 4 ГБ.                                                                                                                                                            |

Мова SQL – мова структурованих запитів, яка дозволяє формувати досить складні запити до бази даних.

Запит – це запитання до бази даних, що повертає запис чи множину записів у відповідь на задане питання.

Використання запитів дає *такі переваги*:

- можливість об'єднувати декілька таблиць зв'язками, які не передбачені під час проектування бази даних;
- можливість поєднувати в одному вивідному наборі інформацію не тільки з різних таблиць однієї бази даних, але й з різних баз;
- задавати фільтри будь-якого рівня складності;
- здійснювати багаторівневе сортування, створювати обчислювальні та підсумкові поля.

*Основні вимоги.*

1. Стандарт SQL вимагає, щоб усі ключові слова були написані великими літерами. Проте BDE автоматично здійснює переведення до верхнього регістру.
2. Всі коментарі починаються та закінчуються /\*.
3. Константи дати та часу беруться в подвійні лапки і мають формат “уу.мм.дд” або ж “hh:mm:ss”.
4. Логічні константи задаються ключовими словами TRUE і FALSE або їх початковими літерами T і F.

### 5.3. Робота з доменами (Domain)

Для опису полів бази, які мають однакове означення, використовуються домени. Домен (Domain) – це глобальне в межах даної бази визначення поля, яке може використовуватись в операторах створення та модифікації таблиць або іншими словами – тип даних, визначений користувачем.

Для створення домену використовують оператор

```
CREATE DOMAIN <ім'я домену> [AS] <тип даних>
    [DEFAULT { константа | NULL | USER}]
    [NOT NULL]
    [CHECK (<умова >)]
    [COLLATE порядок сортування];
```

де тип даних – це один із допустимих типів.

Якщо поле таблиці задає масив, тоді в квадратних дужках задається розмір масиву, верхній та нижній індекс задається через двокрапку. Наприклад, поле

таблиці містить масив із трьох рядків змінної довжини, максимальна довжина яких 20.

```
CREATE DOMAIN MAS AS VARCHAR(20) [1:3];
```

DEFAULT – задає початкове значення полю, якщо в нього не введено значення користувачем, це може бути константа, порожнє, ім'я користувача. Якщо вказано NOT NULL (не порожнє), тоді обов'язково повинно бути введено значення.

CHECK (<умова>) – вказано обмеження на значення поля, що вводиться, тобто здійснюється елементарна перевірка вводу. Умова задається таким чином:

ім'я поля <логічний вираз> , який містить логічні операції >, <, >=, <=, !=, <>, !>, !<;

ім'я поля [NOT] BETWEEN <вираз> AND <вираз> ;

ім'я поля [NOT] LIKE <вираз>;

ім'я поля [NOT] IN <вираз>, <вираз>;

ім'я поля IS [NOT] NULL;

<умова> OR <умова>;

<умова> AND <умова>.

Наприклад, розробити домен, що описує номери місяців від 1 до 12, присвоїти початкове значення змінні 1.

```
CREATE DOMAIN NOM AS INTEGER
```

```
DEFAULT 1
```

```
CHECK (NOT BETWEEN 1 AND 12).
```

Для редагування вже існуючого домену використовується оператор ALTER DOMAIN, що має таку структуру:

```
ALTER DOMAIN {ім'я TO нове ім'я}
```

```
{[SET DEFAULT {константа | NULL |  
USE}]}
```

## 5.4. Команда створення таблиць

Перед створенням таблиці необхідно визначити всі домени, які будуть використовуватись для оголошення її полів.

Імена таблиць можуть мати до 128 символів. Вони повинні бути унікальними по відношенню до користувача – тобто дві таблиці з однаковими іменами не можуть належати одному користувачу, тобто одній БД, проте в іншого користувача може бути таблиця з тим же іменем.

Таблиця може мати до 1024 полів (стовпців). Для кожного стовпця визначаються як мінімум ім'я і тип даних. Імена стовпців можуть мати довжину до 128 символів і є унікальними в межах таблиці. Імена стовпців відповідають прийнятим правилам вибору імен ідентифікаторів і складаються з букв, цифр і службових символів ( \_ , @ и # ).

Перед створенням таблиці необхідно визначити всі домени, які будуть використовуватись для оголошення її полів (стовпців). Оператор створення таблиці має таку структуру:

#### **CREATE TABLE**

```
[IF NOT EXISTS] <ім'я_таблиці >
(<визначення поля 1> ... [, <визначення поля N>] ).
PRIMARY KEY (ім'я_поля) ,
[INDEX 'ім'я_індексу' (ім'я_поля` DECS) ,]
[CONSTRAINT ...]
[TABLE_OPTIONS];
```

- 1) Ключові слова **CREATE TABLE** є *обов'язковими*, оскільки задають тип оператора.
- 2) Якщо вказано параметр **IF NOT EXISTS** і є спроба створити таблицю з дублюючим іменем (тобто таблиця з таким іменем в поточній БД уже є), тоді таблиця створена не буде і повідомлення про помилку не появиться. В протилежному випадку таблиця також створена не буде, зате команда викличе помилку. Необхідно зазначити, що при створенні таблиць порівнюються тільки їх імена. Внутрішні структури не порівнюються.
- 3) Ім'я\_таблиці повинно бути унікальним для активної бази даних, а сама таблиця повинна містити хоча б одне поле. Ім'я може бути складним – ім'я\_бази.ім'я\_таблиці, тобто вказувати у якій БД створюється таблиця, наприклад: 'BIBLIO'.`formyljar`.



4) Визначення поля задається наступним чином

**Ім'я поля {тип даних | COMPUTER [BY] (<вираз>) | домен}**  
**[DEFAULT {константа | NULL | USER}]**  
**[NOT NULL] [обмеження для поля]**  
**[COLLATE порядок сортування]**

Після імені вказується **тип поля** або домен.

COMPUTER BY (<вираз>) задає обчислювальне поле, яке обраховується завжди, коли відбувається звернення до цього запису. У виразі можна використовувати поля інших таблиць, проте ці таблиці повинні бути попередньо створені оператором CREATE TABLE. Усі обмеження на поля та його початкові значення задаються як і доменах.

Обмеження для поля задається [CONSTRAINT ім'я обмеження] і може складатися з таких елементів:

- PRIMARY KEY (поле1,...,поле N) – одне поле чи сукупність полів, які вказано в круглих дужках, задають первинний ключ таблиці. Всі значення стовпця, вказаного як первинний ключ, не повинні містити значення NULL.
- UNIQUE(поле1,..., поле N) – вміст полів, імена яких вказано в круглих дужках, містить значення, що не повторюється.

5) Ключове слово **INDEX** задає поля, які будуть проіндексовані за стовпцем ім'я\_поля. Ім'я\_індексу зазвичай задають як «ім'я\_поля» + «\_idx». Індексція полів корисна для прискорення роботи команди SELECT (причому прискорення іноді буває досить добре).

6) На етапі створення таблиці можна задати зв'язок між нею і іншими, уже створеними таблицями. Для цього використовується конструкція

**CONSTRAINT <ім'я зв'язку>**  
**FOREIGN KEY (поле1, ..., поле N)**  
**REFERENCES <ім'я зв'язаної таблиці> (поле1, ..., поле N).**

Поля, які вказуються в круглих дужках, після ключового слова FOREIGN KEY, входять до складу зовнішнього ключа, тобто це поля, якими доповнилась таблиця для реалізації зв'язку і який посилається на поле

первинного ключа другої таблиці, які задаються в круглих дужках після конструкції REFERENCES <ім'я зв'язаної таблиці>. Крім того, тут можна задати дії, які виконуються, якщо значення первинного ключа змінюється (ON UPDATE) або вибуває (ON DELETE):

- дії відсутні (NO ACTION);
- в ситуації ON UPDATE значення полів, які утворюють зовнішній ключ, будуть змінені відповідно з новим значенням первинного ключа (CASCADE), що і гарантує збереження цілісності даних;
- у випадку ON DELETE будуть видалені всі записи, зовнішні ключі яких співпадають з видаленим первинним ключем;
- поля отримують свої значення за замовчуванням (DEFAULT);
- поля, що утворюють зовнішній ключ, отримують значення NULL.

7) ENGINE задає додаткові параметри таблиці.

[ENGINE = InnoDB | BDB | EXTERNAL FILE ... ];

BDB – Таблиці з підтримкою транзакцій і блокуванням сторінок.

InnoDB – Таблиці з підтримкою транзакцій і блокуванням рядків.

Створену таблицю можна також зберегти в базі даних або у зовнішньому текстовому файлі, шлях до якого вказується конструкцією EXTERNAL FILE 'шлях до файлу'.

**Приклад 5.1.** Записати код створення таблиці Katalog, який відповідатиме відповідній таблиці з раніше розробленої схеми БД у Workbench:

```
CREATE TABLE
    IF NOT EXISTS `BIBLIO_UTF8`.`Katalog` (
        `idK` INT(11) NOT NULL,
        `NameK` VARCHAR(45) NULL,
        `AvtorK` VARCHAR(45) NULL,
        `Vydavnictvo` VARCHAR(45) NULL,
        `RikVyd` INT(11) NULL,
        `KilkStr` INT(11) NULL,
        `KilkPrym` INT(11) NULL,
        PRIMARY KEY (`idK`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8 COLLATE = utf8_bin;
```

В даному прикладі в оголошенні для текстових полів не вказана явно кодова таблиця символів utf8, що підтримує кирилицю, тому у кінці команди необхідно дописати рядок:

```
DEFAULT CHARACTER SET = utf8 COLLATE = utf8_bin;
```

Під час синхронізації схеми даних та БД на сервері або використання майстра побудови таблиць як в оголошенні кожного текстового поля є задана кодова таблиця символів utf8, так і загалом для всієї таблиці:

```
CREATE TABLE IF NOT EXISTS `BIBLIO_UTF8`.`Katalog` (  
    `idK` INT(11) NOT NULL,  
    `NameK` VARCHAR(45) CHARACTER SET 'utf8' NULL DEFAULT NULL,  
    `AvtorK` VARCHAR(45) CHARACTER SET 'utf8' NULL DEFAULT NULL,  
    `Vydavnictvo` VARCHAR(45) CHARACTER SET 'utf8' NULL DEFAULT NULL,  
    `RikVyd` INT(11) NULL DEFAULT NULL,  
    `KilkStr` INT(11) NULL DEFAULT NULL,  
    `KilkPrym` INT(11) NULL DEFAULT NULL,  
    PRIMARY KEY (`idK`))  
ENGINE = InnoDB  
DEFAULT CHARACTER SET = utf8 COLLATE = utf8_bin;
```

**Приклад 5.2.** Записати код створення дочірньої таблиці Formujiar, що пов'язує таблиці Osoba та Katalog, який відповідатиме відповідній таблиці з раніше розробленої схеми БД у Workbench:

```
CREATE TABLE IF NOT EXISTS `BIBLIO_UTF8`.`Formujiar` (  
    `id` INT(11) NOT NULL,  
    `idO` INT(11) NULL DEFAULT NULL,  
    `idK` INT(11) NULL DEFAULT NULL,  
    `dat_vud` DATE NULL DEFAULT NULL,  
    `dat_pov` DATE NULL DEFAULT NULL,  
    `PIPB` VARCHAR(45) NULL DEFAULT NULL,  
    PRIMARY KEY (`id`), /* Первинний ключ */  
    INDEX `osoba_idx` (`idO` ASC), /* індекс за полем idO */  
    INDEX `katalog_idx` (`idK` ASC), /* індекс за полем idK */  
    CONSTRAINT `osoba_rel` /* Зв'язок з таблицею Osoba*/  
        FOREIGN KEY (`idO`)  
        REFERENCES `BIBLIO_UTF8`.`Osoba` (`idO`)  
    ON DELETE CASCADE
```

```

        ON UPDATE CASCADE,
CONSTRAINT `katalog_rel`          /* Зв'язок з таблицею Katalog*/
    FOREIGN KEY (`idK`)
    REFERENCES `BIBLIO_UTF8`.`Katalog` (`idK`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8 COLLATE = utf8_bin;

```

Якщо в схемі є зв'язки різних дочірніх таблиць з однією і тою ж батьківською, тоді особливу увагу необхідно звернути на імена зв'язків, які в межах однієї бази даних повинні бути унікальними.

## 5.5. Команди модифікації структури таблиці

Часто в процесі роботи з БД виникає необхідність до створеної таблиці додати або вилучити поле. Іноді розробники на етапі проектування БД роблять невірні припущення щодо типу даних, які вони збираються зберігати в таблиці, назв стовпців, порядку їх розташування, накладання різноманітних обмежень на етапі створення таблиць тощо. Невірний вибір призводить до неефективного зберігання даних або ж дані можуть виявитися занадто великими і не помістяться у стовпцях. В таких ситуаціях слід змінити визначення стовпців командою ALTER TABLE. Формат запису цього оператора:

```

USE <ім'я_БД>;
ALTER TABLE ім'я_таблиці
    <оператор 1>,..., <оператор N>.

```

Операторами можуть бути:

- ADD – додається елемент на рівні структури таблиці. Додаватися можуть:
  - нові поля:
 

```
ADD <нове_ім'я_поля> <новий_тип_поля> [FIRST | AFTER <ім'я_поля>];
```
  - нові індекси:

ADD INDEX <ім'я\_індексу> (<поле, за яким створюється індекс>  
[ASC | DESC]), де ASC – сортування за зростанням, DESC – за  
спаданням);

- нові первинні ключі:

ADD PRIMARY KEY (<ім'я\_поля>);

- нові обмеження:

CONSTRAINT <нове\_ім'я\_обмеження> <умова\_обмеження>;

- нові зв'язки:

ADD CONSTRAINT <ім'я\_зв'язку> FOREIGN KEY (<ім'я\_поля>  
REFERENCES <ім'я\_таблиці> (<ім'я\_поля>  
[ON DELETE CASCADE, ON UPDATE CASCADE]);

- CHANGE – зміна структури таблиці. Формат запису:

CHANGE <ім'я\_поля><Нове визначення поля>,

яке задається як

{<нове\_ім'я\_поля> <новий тип даних> [FIRST | AFTER ім'я\_поля]};

- MODIFY – зміна структури таблиці. Формат запису:

MODIFY <ім'я\_поля><новий тип даних>;

- RENAME <нове\_ім'я\_таблиці>;

- DROP – вилучає вказаний елемент із структури таблиці. Вилучати можна поля, первинні ключі, індекси, обмеження та зв'язки:

- DROP <ім'я\_поля>;
- DROP PRIMARY KEY;
- DROP INDEX <ім'я\_індексу>.

Оператор ADD <визначення поля> використовується для доповнення структури таблиці новим елементом структури. В операторі ALTER TABLE можна одночасно вилучати і додавати те саме поле, декілька полів, тоді вони перераховуються через кому, або ж виконувати одну із операцій доповнення чи вилучення.

Оператор DROP <ім'я\_поля> використовується для вилучення поля із структури таблиці, проте це не може бути поле первинного ключа або поле, за яким є зв'язок з іншою таблицею.

Приклади використання команди ALTER TABLE наведено нижче.

**Приклад 5.3.** Додавання нового поля стать (sex) типу VARCHAR(1) в таблицю osoba3:

```
ALTER TABLE BIBLIO.osoba3  
ADD sex VARCHAR(1);
```

Нове поле буде додано в кінець структури таблиці (як останній стовпець).

**Приклад 5.4.** Додавання нового поля номер телефону ntelo типу VARCHAR(12) після поля datanaro у таблиці osoba3:

```
ALTER TABLE BIBLIO.osoba3  
ADD ntelo VARCHAR(12)  
AFTER datanaro;
```

**Приклад 5.5.** Додавання нового індексу за полем PIPo:

```
ALTER TABLE BIBLIO.osoba3  
ADD INDEX idx_PIPo (PIPo ASC);
```

**Приклад 5.6.** Створення ключового поля у таблиці osoba2 за полем id1:

```
ALTER TABLE BIBLIO.osoba3  
ADD PRIMARY KEY (id1);
```

**Приклад 5.7.** Додавання обмеження на поле stat у таблиці osoba3:

```
ALTER TABLE BIBLIO.osoba3  
ADD CONSTRAINT stat_ck CHECK (stat LIKE "[A-Z]");
```

**Приклад 5.8.** Додавання у таблицю osoba3 нового поля rik типу INT з обмеженням на нього (можна вводити тільки числа):

```
ALTER TABLE BIBLIO.osoba3  
ADD rik INT,  
ADD CONSTRAINT rik_ck CHECK (rik LIKE "[%#%]");
```

**Приклад 5.9.** Додавання зв'язку між таблицею osoba3 за полем idknigy (дочірня таблиця) та таблицею katalog за полем Idk (батьківська таблиця). Зв'язок назвати newconnect:

```
ALTER TABLE BIBLIO.osoba3  
ADD CONSTRAINT newconnect FOREIGN KEY (idknigy)  
REFERENCES katalog (Idk);
```

**Приклад 5.10.** Перейменування назви поля (стовпця) sex на stat у таблиці osoba3:

```
ALTER TABLE BIBLIO.osoba3  
CHANGE sex stat VARCHAR(1);
```

Без типу нової назви поля цей оператор не працює.

**Приклад 5.11.** Перемістити поле stat типу VARCHAR(1) після поля pipo у таблиці osoba3:

```
ALTER TABLE BIBLIO.osoba3  
CHANGE stat stat VARCHAR(1) AFTER pipo;
```

**Приклад 5.12.** Зміна розміру поля pipo із 40 символів на 30 у таблиці osoba3:

```
ALTER TABLE BIBLIO.osoba3  
CHANGE pipo pipo111 VARCHAR(30);
```

або

```
ALTER TABLE BIBLIO.osoba3  
MODIFY pipo VARCHAR(30);
```

**Приклад 5.13.** Зміна типу поля datareestro із типу DATE на тип INT у таблиці osoba3:

```
ALTER TABLE BIBLIO.osoba3  
CHANGE datareestro datareestro INT;
```

або

```
ALTER TABLE BIBLIO.osoba3  
MODIFY datareestro INT;
```

**Приклад 5.14.** Перейменувати таблицю osoba на osoba3:

```
ALTER TABLE BIBLIO.osoba  
RENAME BIBLIO.osoba3;
```

**Приклад 5.15.** Витерти з таблиці osoba3 поле з назвою ppp3:

```
ALTER TABLE BIBLIO.osoba3  
DROP COLUMN ppp3;
```

**Приклад 5.16.** Видалити з таблиці osoba3 первинний ключ :

```
ALTER TABLE BIBLIO.osoba3  
DROP PRIMARY KEY;
```

**Приклад 5.17.** Видалити з таблиці osoba3 індекс idx\_PIPo:

```
ALTER TABLE BIBLIO.osoba3  
DROP INDEX idx_PIPo;
```

При видаленні елементів зі структури таблиці необхідно пам'ятати, що виконання команди можуть завадити деякі залежності. Наприклад, якщо поле є первинним ключем, видалити не вдасться, поки не буде знято відповідне обмеження. Якщо в іншій таблиці існує посилання на стовпець, тоді його також не дозволено видаляти до зняття обмеження.

**Приклад 5.18.** Видалити зв'язок `nom\_sh` між таблицями osoba3 та Formuljar. Таблиця Formuljar є дочірньою таблицею таблиці osoba3, тобто зв'язок створено у таблиці Formuljar:

```
ALTER TABLE BIBLIO. Formuljar  
drop FOREIGN KEY nom_sh;
```

## 5.6. Команда вилучення таблиці

Для видалення таблиці використовують команду

```
DROP TABLE <ім'я таблиці>.
```

В цьому випадку з бази даних вилучається опис таблиці, її дані, індекси та тригери.

***Вилучені таблиці відновленню не підлягають!***

**Приклад 5.19.** Видалити таблицю osoba3:

```
DROP TABLE BIBLIO.osoba3;
```

## 5.7. Команди модифікації вмісту таблиць

### 5.7.1. Команда додавання записів

Команда INSERT має чотири основні синтаксичні форми. У перших двох формах задаються значення стовпців одного запису. Дві інші форми дозволяють додати декілька записів, оснований на результатах команди SELECT або збереженої процедури. Синтаксис всіх чотирьох форм INSERT виглядає наступним чином образом:



1. INSERT [INTO] {таблиця} {список\_стовпців}  
VALUES ({DEFAULT | вираз\_константа }[,...n]);
2. INSERT [INTO] {таблиця} {список\_стовпців}  
DEFAULT VALUES;
3. INSERT [INTO] {таблиця}  
{список\_стовпців} команда\_select;
4. INSERT [INTO] {таблиця}  
{список\_стовпців} команда\_execute;

Команда INSERT найчастіше використовується для вставлення в таблицю одного окремого запису. Зазвичай при цьому використовується ключове слово VALUES, за яким розташовується необов'язковий список значень. Він містить значення, які присвоюються відповідним стовпцям у списку стовпців.

В наступних прикладах продемонстровано різні варіанти вставлення запису в таблицю з іменем osoba3, що містить два стовпці (a і b).

**Приклад 5.20.** Команда вставляє новий запис в таблицю osoba3. Стовпцю a присвоюється значення "a", а стовпцю b – значення 1234.

```
INSERT BIBLIO.osoba3 VALUES ('a', 1234);
```

**Приклад 5.21.** В команді використано необов'язкове ключове слово INTO, яке не впливає на виконання команди INSERT.

```
INSERT INTO BIBLIO.osoba3 (a) VALUES ('ccc');
```

**Приклад 5.22.** При наявності ключових слів DEFAULT VALUES список стовпців не допускається. Кожному стовпцю osoba3 присвоюється значення за замовчуванням.

```
INSERT BIBLIO.osoba3 DEFAULT VALUES;
```

**Приклад 5.23.** В цьому прикладі список стовпців теж не використовується. Стовпцю a присвоюється значення за замовчуванням, а стовпцю b – значення 5.

```
INSERT BIBLIO.osoba3 VALUES (DEFAULT,5);
```

Іноді при вставленні даних в таблицю потрібно, щоб SQL Server присвоїв стовпцю значення за замовчуванням. Для цього використовується ключове слово DEFAULT. В команді INSERT існує два можливих застосування ключового слова DEFAULT:

1) Якщо DEFAULT використовується у списку значень, воно відноситься до відповідного стовпцю списку.

2) Якщо ключове слово DEFAULT знаходиться перед початком списку, воно відноситься до всіх непохідних стовпців таблиці, в цьому випадку список стовпців повинен бути порожнім.

Ключове слово DEFAULT обробляється наступним чином:

- Якщо у стовпці є значення за замовчуванням, воно заноситься в стовпець.
- Якщо у стовпці немає значення за замовчуванням, присвоюється значення типу NULL.

- Якщо стовпець не допускає значень типу NULL, SQL Server видає повідомлення про помилку.

- Якщо стовпець має тип TIMESTAMP, стовпцю присвоюється наступне значення часу.

- SQL Server не дозволяє використовувати DEFAULT в списку значень для стовпця лічильника, однак використання ключових слів DEFAULT VALUES допускається.

Застосування ключового слова DEFAULT для стовпців лічильника показано на прикладах 5.24-5.25. Створена таблиця:

```
CREATE TABLE Test (  
    Col1 NUMERIC(10) IDENTITY,  
    Col2 VARCHAR(32) DEFAULT 'Test' NOT NULL);
```

**Приклад 5.24.** Команда INSERT вставляє запис з наступним значенням в стовпці Col1 і значенням 'Test' в стовпці Col2:

```
INSERT Test DEFAULT VALUES;
```

**Приклад 5.25.** Команда INSERT завершиться невдачею, навіть якщо в таблиці встановлений параметр, який дозволяє виконання команди INSERT для стовпців лічильника:

```
INSERT Test VALUES ( DEFAULT, DEFAULT);
```

### 5.7.2. Команда редагування записів

Оператор UPDATE використовується для редагування записів таблиці БД значень вказаних полів для одного або декількох записів.

В більшості випадків необхідно використовувати оператор WHERE для обмеження записів, які будуть змінені. **Якщо ж опустити оператор WHERE, будуть змінені всі записи в таблиці.**

Для того, щоб запит UPDATE не спрацьовував без умови WHERE, необхідно включити опцію SQL\_SAFE\_UPDATES = 1, якщо ж необхідно її виключити, тоді SQL\_SAFE\_UPDATES = 0.

Формат запису команди UPDATE:

```
UPDATE <ім'я_таблиці>  
SET <ім'я_поля>=<значення>,[ідентифікатор_поля=значення]  
WHERE <умова>.
```

Редагуючи запис, необхідно вказати ім'я таблиці, поля, які будуть змінюватися, і нові значення цих полів.

Розглянемо декілька прикладів модифікації вмісту таблиць.

**Приклад 5.26.** Збільшити кількість всіх книжок, у назві яких зустрічається словосполучення «ОБД», на 10 одиниць.

```
UPDATE katalog SET KilK=KilK+10  
WHERE NameK LIKE '%ОБД%';
```

**Приклад 5.27.** У таблиці katalog збільшити у двічі кількість примірників книжок, які в позичались протягом минулого року.

```
UPDATE katalog SET katalog.KilK = katalog.KilK *2  
WHERE id_kniga  
IN (SELECT id_kniga FROM formyljar  
WHERE formyljar.dat_vud>='01.01.2015'  
AND formyljar.dat_vud>='31.12.2015')
```

Можна використовувати запит UPDATE також для модифікації декількох полів.

**Приклад 5.28.** Змінити у таблиці osoba поля Adress та Grupa для запису 12345:

UPDATE osoba

SET Adress = 'Львів, Чупринки 105', Grupa = 'KH-3'

WHERE id\_Osoba = '12345';

### 5.7.3. Команда видалення записів

Для видалення записів (рядків) у таблиці БД в MySQL використовується два оператори: DELETE та TRUNCATE.

Оператор TRUNCATE фактично є частковим випадком оператора DELETE без оператора умови, коли необхідно витерти всі записи з таблиці, хоча між ними все таки є відмінності:

- операція TRUNCATE виконується швидше та вимагає менших ресурсів системи;
- операція TRUNCATE не повертає якогось значення (зазвичай повертає 0) на відміну від DELETE, що повертає кількість видалених рядків;
- операція TRUNCATE обнулює значення лічильників (для полів з AUTOINCREMENT) на відміну від DELETE, яка продовжує нумерація навіть після повного очищення вмісту таблиці.

Формат запису команди TRUNCATE:

**TRUNCATE TABLE <ім'я\_таблиці>.**

Наслідком виконання такої команди є видалення усіх записів (рядків) таблиці. Причому сама таблиця (її структура) – залишається (не слід плутати з DROP TABLE).

Формат запису команди DELETE:

**DELETE [LOW\_PRIORITY | QUICK] FROM <ім'я\_таблиці>  
[WHERE <умова>] [LIMIT <кількість\_рядків>]**

Оператор DELETE видаляє з таблиці рядки, що задовольняють заданим у WHERE умовам, і повертає число видалених записів.

Якщо оператор DELETE виконується без умови WHERE, тоді витираються всі записи (аналогічно використанню оператора TRUNCATE):

**DELETE [LOW\_PRIORITY | QUICK] FROM <ім'я\_таблиці>.**

Якщо вказано ключове слово `LOW_PRIORITY`, виконання даної команди `DELETE` буде затримано до того часу, поки інші клієнти не завершать читання цієї таблиці.

Якщо заданий параметр `QUICK`, тоді при виконанні команди `DELETE` не об'єднуюватимуться індекси, адже в деяких випадках це може прискорити час виконання операції.

Специфічна для MySQL опція `LIMIT` для команди `DELETE` вказує серверу максимальну кількість рядків, які слід видалити до повернення управління клієнту. Ця опція може використовуватися для гарантії того, що дана команда `DELETE` не вимагатиме занадто багато часу для виконання. Можна просто повторювати команду `DELETE` до того часу, поки кількість вилучених рядків менше, ніж величина `LIMIT`.

**Приклад 5.29.** Вилучити з таблиці `catalog` всі книги, де зустрічається у назві книг фрагмент 'Borland 6':

```
DELETE FROM catalog
WHERE NameK LIKE '%Borland 6%'
```

## 5.8. Використання індексів

Індекси дозволяють значно прискорити отримання даних із таблиці, сортування та переміщення набором даних. Із іншої сторони, велика кількість індексів, які визначені для таблиці, призводять до надлишкових втрат часу під час додавання, коригування та вилучення записів із таблиці.

Для таблиці можна створити до 64 індексів. Слід мати на увазі, що сервер автоматично створює індекси для полів (чи груп полів), які оголошені полями первинних або зовнішніх ключів, а також для полів, які оголошено унікальними (`UNIQUE`).

Індекси створюються як командою `CREATE TABLE`, так і окремо оператором

```
CREATE [UNIQUE] [ASC[ENDING] | DESC[ENDING]]
INDEX <Ім'я_Індексу> ON <Ім'я_таблиці> (поле, [, поле ...]).
```

У клієнт-серверних базах даних індекси є об'єктами бази, а не таблиці. За допомогою наведеного вище оператора створюється індекс Ім'я\_Індекса для таблиці Ім'я\_таблиці, якщо використано ключове слово UNIQUE, тоді індекс повинен містити унікальні (не співпадаючі) значення. Для індексу можна задати порядок сортування – ASC – за зростанням (приймається за замовчуванням), DESC – за спаданням.

За допомогою оператора

**ALTER INDEX Ім'я\_індекса {ACTIVE | INACTIVE}**

можна активувати і деактивувати уже створений індекс. Якщо оновлюємо чи модифікуємо таблицю, що має велику кількість записів, необхідно деактивувати індекси, а після завершення роботи знову зробити індекси активними.

Знищити індекс можна за допомогою оператора

**DROP INDEX Ім'я\_індекс.**

Проте індекси, які автоматично створюються сервером на основі первинних або зовнішніх ключів, видалити за допомогою DROP INDEX не можна.

### Контрольні запитання

1. Що таке SQL та MySQL?
2. Які сімейства мов входять в склад SQL?
3. Які основні команди сімейства DDL мови SQL?
4. Наведіть основні команди сімейства DML мови SQL.
5. Для чого застосовуються команди сімейства DCL мови SQL?
6. Яке сімейство використовується для роботи з транзакціями? Які основні команди цього сімейства?
7. Наведіть основні типи даних мови SQL.
8. Що таке запит?
9. Для чого в мові SQL використовуються домени?
10. Наведіть структуру команди CREATE TABLE.
11. Для чого служить параметр IF NOT EXISTS команди CREATE TABLE?
12. Проаналізуйте структуру визначення поля команди CREATE TABLE.

13. Якою конструкцією створюються зв'язки з іншими таблицями в команді CREATE TABLE?
14. Як явно вказати кодову таблицю ОС для даних типу рядок в команді створення таблиці?
15. Яка команда застосовується для модифікації структури таблиці?
16. Наведіть та охарактеризуйте основні оператори команди ALTER TABLE.
17. Як вилучити таблицю зі структури БД?
18. Запишіть основні синтаксичні форми команди INSERT.
19. В яких випадках в команді INSERT можна опустити список полів?
20. Як працює команда редагування записів без умови WHERE?
21. Чим відрізняються команди DELETE та TRUNCATE? Чим вони схожі?
22. Для чого призначено індексування таблиць?
23. Як створити свій індекс, його змінити, вилучити?

## Тема 6. Команда вибірки SELECT

### 6.1. Формат запису команди SELECT

Запит типу SELECT використовується для відбору інформації з таблиці бази даних за різноманітними критеріями і має такий синтаксис:

**SELECT**

**[DISTINCT | ALL]**

**<список полів>**

**FROM <список таблиць>**

**[WHERE <умова вибору>]**

**[GROUP BY <список полів групи>]**

**[ORDER BY <список полів сортування>].**

Окремі фрагменти запиту, що починаються з ключового слова і є логічно завершеними, називаються реченнями.

При написанні запиту необхідно точно зберігати порядок ключових слів, заданий синтаксисом. Наприклад, вираз **WHERE** повинен розміщуватися після всіх **FROM** і перед всіма **GROUP BY** або **ORDER BY**.

- 5) Ключове слово **SELECT** є *обов'язковим*, оскільки задає тип оператора.
- 6) Якщо оператор вибірки повинен повертати лише унікальні записи (які не співпадають), тоді задається ключове слово **DISTINCT**. Якщо ж необхідно повернути всі записи, які відповідають умовам запиту, можна задавати **ALL** або нічого не задавати.
- 7) Список полів також є *обов'язковим* і повинен *містити хоча б одне поле*. Декілька полів відокремлюються комою. Список полів може містити обчислювальні поля. Якщо у списку виводяться поля з різних таблиць, що мають однакові імена, тоді використовується складне ім'я: ім'я\_таблиці.ім'я\_поля. Наприклад, Table1.Kodm.
- 8) Речення **FROM** є *обов'язковою* складовою оператора **SELECT**. Після **FROM** йде список назв таблиць, поля яких використано в **SELECT**.



- 9) Речення **WHERE** використовується для формулювання умови для вибору записів із таблиці.
- 10) Речення **GROUP BY** використовується для об'єднання декількох записів в результуючий за допомогою групових операцій.
- 11) Речення **ORDER BY** використовується для сортування, якщо таке не вказано вище у реченні GROUP BY.

## 6.2. Використання запиту SELECT для вибору записів з однієї таблиці

Запит SELECT використовується для вибірки записів з таблиць. У цього запиту є багато форм, і найпростіша з них дозволяє вказувати перелік стовбців та ім'я таблиці.

*Приклад 6.1.* Вивести з таблиці Osoba стовбці IdO, PIPO, AdressO та TelO за допомогою SELECT-запиту.

```
SELECT IdO, PIPO, AdressO, TelO FROM Osoba;
```

Стовбці, які будуть отримані з таблиці, перераховані після ключового слова SELECT, а ім'я таблиці задається після ключового слова FROM.

Можна отримати дані із всіх стовбців таблиці, поставивши відразу після слова SELECT зірочку (\*).

*Приклад 6.2.* Вивести з таблиці Osoba всі стовбці:

```
SELECT * FROM Osoba;
```

Цей запит виведе всі записи з таблиці Osoba.

Та використання зірочки \* в SELECT-запитах необхідно уникати, оскільки можна отримати значно більше інформації, чим очікується. Тому бажано перераховувати тільки ті поля, які дійсно необхідні.

### 6.2.1. Використання оператора AS

Оператор AS використовується для задавання псевдонімів імен полів та таблиць, тобто таких імен, під якими вони будуть відображатися на виході запиту SELECT. Це може пригодитися, наприклад, якщо користувачу зручніше аналізувати отриману інформацію за назвами стовпців, а реальні назви йому не подобаються.

**Приклад 6.3.** Відобразити у таблиці Osoba поле PIP під назвою `Name osoby`, а поле NTel – `Nomer telef`:

```
SELECT PIP AS `Name osoby`,  
       NTel AS `Nomer telef`  
FROM Osoba;
```

Оператор AS також можна використовувати для задавання псевдонімів таблиць. Це доцільно при довгих іменах таблиць, а також при роботі з декількома таблицями.

**Приклад 6.4.** Відобразити назву таблиці Formuljar під псевдонімом Frm:

```
SELECT F.id, F.idBook, F.idOsoba, F.Bibliotekar, F.DataIn  
FROM Formuljar F
```

### 6.2.2. Задавання умови WHERE

Ключове слово **WHERE** використовується для формулювання умови для вибору записів із таблиці. Ця умова задається логічними виразами, які формуються із полів таблиці або констант і приймають значення TRUE або FALSE. Логічними виразами можуть бути логічні операції (=, <> або !=, >, <, >=, <=) та логічні функції (OR, AND, NOT).

Оператор NOT в умові WHERE можна використовувати тільки з іншим оператором для інвертування значення цього другого оператора, наприклад, NOT LIKE, NOT IN або NOT BETWEEN.

**Приклад 6.5.** Вивести назви всіх книг у бібліотеці, кількість примірників яких рівна 100:

```
SELECT NameK AS `Name book`, KilK AS `Kilkist`  
FROM Katalog  
WHERE KilK=100;
```

**Приклад 6.6.** Вивести назви всіх книг у бібліотеці, кількість примірників яких більша за 100:

```
SELECT NameK AS `Name book`, KilK AS `Kilkist`  
FROM Katalog  
WHERE KilK > 100;
```

**Приклад 6.7.** Вивести назви всіх книг у бібліотеці, кількість примірників яких знаходиться в межах від 100 до 200:

```
SELECT NameK AS `Name book`, KilK AS `Kilkist`  
FROM Katalog  
WHERE KilK >= 100 AND KilK <= 200;
```

В умові WHERE також можна задавати складні умови та умови для декількох полів за допомогою логічних функцій.

**Приклад 6.8.** Вивести назви всіх книг у бібліотеці, кількість примірників яких більша за 100 та випущених у 2020 році:

```
SELECT NameK AS `Name book`, KilK AS `Kilkist`  
FROM Katalog  
WHERE KilK >= 100 AND YEAR(DataVyd) = 2020;
```

**Приклад 6.9.** Вивести назви всіх книг у бібліотеці, які не випущені у 2020 році:

```
SELECT NameK AS `Name book`, KilK AS `Kilkist`  
FROM Katalog  
WHERE NOT (YEAR(DataVyd) = 2020);
```

### 6.2.3. Порівняння за шаблоном LIKE та REGEXP

В MySQL є два типи порівняння за шаблоном:

- LIKE;
- REGEXP або RLIKE.

Оператор LIKE використовується в умові WHERE для порівняння значення поля з шаблоном та має таку структуру:

**<поле> [NOT] LIKE <шаблон>.**

Елемент **шаблон** – це текстовий рядок, який використовується для порівняння із “значенням”. У шаблоні можна використовувати один або декілька символів підстановки, які наведені в табл. 6.1.

Крім того, під час порівняння враховується регістр.

Розглянемо декілька прикладів.

**Приклад 6.10.** Вивести назви всіх книг у бібліотеці, у назві яких є всього чотири символи, а перші три з яких – ОБД. Шаблон LIKE відповідатиме `ОБД\_`:

```
SELECT NameK AS `Name book`, KilkK AS `Kilkist`
FROM Katalog
WHERE NameK LIKE `ОБД_`;
```

**Табл. 6.1. Символи підстановки**

| Символ | Опис                                                                                                                                                             |
|--------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| –      | Відповідає одному будь-якому символу. Наприклад, K_т відповідає Кіт та Кут                                                                                       |
| %      | Відповідає будь-якій кількості довільних символів. Наприклад: %вання відповідає Програмування та Моделювання; %fire% відповідає starfire, firestarter та fireman |

**Приклад 6.11.** Вивести назви всіх книг у бібліотеці, у назві яких є довільна кількість символів, а перші три з яких – ОБД. Шаблон LIKE відповідатиме `ОБД%`:

```
SELECT NameK AS `Name book`, KilkK AS `Kilkist`
FROM Katalog
WHERE NameK LIKE `ОБД%`;
```

У другому типі шаблонів, передбачених MySQL, використовуються розширені регулярні вирази, що задаються оператором

**<поле> [NOT] REGEXP <шаблон>.**

або

**<поле> [NOT] RLIKE <шаблон>.**

У шаблоні можна використовувати один або декілька символів підстановки, які наведені в табл. 6.2.

**Табл. 6.2. Символи підстановки**

| Символ  | Опис                                                            |
|---------|-----------------------------------------------------------------|
| 1       | 2                                                               |
| ^       | відповідає позиції на початку шуканого рядка                    |
| \$      | відповідає позиції наприкінці шуканого рядка                    |
| .       | відповідає будь-якому окремому символу                          |
| [...]   | відповідає будь-якому символу, вказаному у квадратних дужках    |
| [^ ...] | відповідає будь-якому символу, не вказаному у квадратних дужках |

| 1       | 2                                                                |
|---------|------------------------------------------------------------------|
| p1   p2 | відповідає будь-якому з шаблонів p1 або p2                       |
| *       | відповідає попередньому символу нуль або більше разів            |
| +       | відповідає попередньому символу один чи кілька разів             |
| {n}     | відповідає n кількостям екземплярів попереднього символу         |
| {m,n}   | відповідає від m до n кількості екземплярів попереднього символу |

Щоб продемонструвати роботу регулярних виразів, розглянемо приклади.

**Приклад 6.12.** Вивести назви всіх книг у бібліотеці, у назві яких є довільна кількість символів та починаються на букву К, Т або У. Шаблон REGEXP відповідатиме K|T|Y, тобто, або К, або Т, або У, а щоб вказати, що значення починаються на ці букви, попереду поставити ^:

```
SELECT NameK AS `Name book`, KilK AS `Kilkist`
FROM Katalog
WHERE NameK REGEXP `^(K|T|Y)`;
```

**Приклад 6.13.** Вивести назви всіх книг у бібліотеці, назви яких не починаються на букву К, Т або У. Шаблон REGEXP аналогічний попередньому, але із запереченням:

```
SELECT NameK AS `Name book`, KilK AS `Kilkist`
FROM Katalog
WHERE NameK NOT REGEXP `^(K|T|Y)`;
```

**Приклад 6.14.** Вивести назви всіх книг у бібліотеці, прізвища авторів у яких закінчуються на букву 'к'. Шаблон REGEXP відповідатиме `к\$`:

```
SELECT * FROM Katalog
WHERE AvtorK REGEXP `к$`;
```

**Приклад 6.15.** Вивести назви всіх книг у бібліотеці автора Аркаса. Шаблон REGEXP відповідатиме 'Аркас':

```
SELECT * FROM Katalog
WHERE AvtorK REGEXP `Аркас`;
```

**Приклад 6.16.** Вивести назви всіх книг у бібліотеці, назва яких починається з цифри. Шаблон REGEXP відповідатиме `^[0-9]`:

```
SELECT * FROM Katalog
      WHERE NameK REGEXP '^[0-9]';
```

**Приклад 6.17.** Вивести назви всіх книг у бібліотеці, назва яких містить тільки 10 символів. Шаблон REGEXP відповідатиме від початку '^' до кінця '\$' будь-які символи '.', що повторюються 10 раз:

```
SELECT * FROM Katalog
      WHERE NameK REGEXP '^.{10}$';
```

#### **6.2.4. Використання оператора IN для задавання списку можливих значень**

В умові WHERE можна також використовувати оператор IN для виведення значень, які задаються деяким списком. Операція IN має таку структуру:

**<і'мя поля> [NOT] IN (список значень).**

Елементами цього списку можуть бути константи, перераховані через кому, або підлеглий запит.

**Приклад 6.18.** Вивести дані про такі книги: ОС, ООП, ОБД:

```
SELECT NameK AS `Name book`, KilK AS `Kilkist`
      FROM Katalog
      WHERE NameK IN ('ОС', 'ООП', 'ОБД');
```

**Приклад 6.19.** Вивести назви книг, які було взято абонентами у 2020 році:

```
SELECT NameK AS `Name book`, KilK AS `Kilkist`
      FROM Katalog
      WHERE idK IN (
          SELECT idK FROM Formuljar
          WHERE YEAR(dat_pov) = 2020 );
```

#### **6.2.5. Задавання діапазону значень оператором BETWEEN**

Операція BETWEEN використовується для перевірки входження значення в заданий діапазон і має таку структуру:

**ім'я\_поля [NOT] BETWEEN значення1 AND значення2.**

**Приклад 6.20.** Вивести назви всіх книг у бібліотеці, кількість примірників яких знаходиться в межах від 100 до 200:

```
SELECT NameK AS `Name book`, Kilkk AS `Kilkist`  
FROM Katalog  
WHERE Kilkk BETWEEN 100 AND 200;
```

або :

```
WHERE Kilkk >= 100 AND Kilkk <=200;
```

**Приклад 6.21.** Вивести прізвища бібліотекарів, які видали абонентам книги на протязі 2018-2020 років:

```
SELECT Bibliotekar  
FROM Formuljar  
WHERE YEAR(DataIN) BETWEEN 2018 AND 2020;
```

### 6.2.6. Використання оператора IS NULL

Якщо під час створення таблиць для деякого поля було задано, що воно може приймати значення NULL, тобто воно є необов'язковим для заповнення, тоді при роботі з даною таблицею можуть бути записи із значенням NULL. Значення NULL відрізняється від порожнього поля або нуля – це значення, яке або ще не задано, або невідоме.

Для визначення записів із значеннями типу NULL в умові WHERE використовується оператор IS NULL і має такий формат:

**ім'я\_поля IS [NOT] NULL.**

**Приклад 6.22.** Вивести прізвища бібліотекарів, які видали абонентам книги та не заповнили дату видачі книги:

```
SELECT PIPB FROM Formuljar  
WHERE dat_vud IS NULL;
```

### 6.2.7. Об'єднання декількох записів в результуючий за допомогою групових операцій

Для об'єднання декількох записів в результуючий за допомогою групових операцій в запиті SELECT використовується речення GROUP BY. Поля, за якими йде групування, задаються після GROUP BY через кому. Всі вони повинні бути

вказані в реченні **SELECT**. Всі записи, які мають для цих полів однакові значення, об'єднуються в один запис. Для отримання підсумкових значень для певних полів використовують агрегативні функції. В мові SQL-запитів використовують такі агрегативні функції.

- a. **AVG** – відповідне поле згрупованого запису буде містити середнє арифметичне всіх значень поля для записів, що входять в групування;
- b. **COUNT** – відповідне поле буде містити кількість записів, що увійшли в групу;
- c. **MIN, MAX** – в результаті групування поле буде містити мінімальне (максимальне) значення поля, розраховане для всіх записів, які увійшли в групу;
- d. **SUM** – просумувати значення полів, які увійшли в групу.

Слід пам'ятати, що під час використання групових операцій в **SELECT** містяться тільки поля, за якими йде групування та поля, які використано в агрегативних операціях.

**Приклад 6.23.** Вивести назву та автора найстарішої в бібліотеці книги:

```
SELECT NameK, AvtorK, MAX(DataVyd)
FROM Katalog
GROUP BY NameK
```

**Приклад 6.24.** Вивести сумарну кількість книг у бібліотеці з ООП:

```
SELECT SUM (KilkK)
FROM Katalog
WHERE NameK LIKE '%ООП%'
GROUP BY NameK;
```

**Приклад 6.25.** Вивести сумарну кількість примірників у бібліотеці за роками видання.

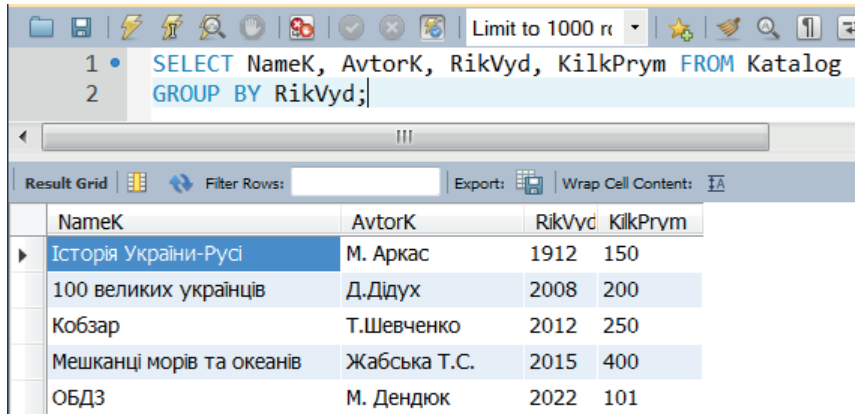
Маємо таблицю, наведену на рис. 6.1.



| idK | NameK                      | AvtorK        | Vydavnictvo                      | RikVyd | KilkStr | KilkPrym |
|-----|----------------------------|---------------|----------------------------------|--------|---------|----------|
| 100 | Кобзар                     | Т.Шевченко    | Київ, АРІЙ                       | 2012   | 608     | 250      |
| 101 | Тільки у Львові            | Олесь Ганущак | Львів, АФІША                     | 2012   | 125     | 100      |
| 102 | 100 великих українців      | Д.Дідух       | Київ, АРІЙ                       | 2008   | 496     | 200      |
| 103 | Українська народна писанка | Віра Манько   | Львів, СВІЧАДО                   | 2012   | 80      | 60       |
| 104 | Мешканці морів та океанів  | Жабська Т.С.  | Харків, ВЕСТА                    | 2015   | 245     | 400      |
| 105 | Історія України-Русі       | М. Аркас      | Краків: Накладом Ольги Аркасової | 1912   | 424     | 150      |
| 106 | ОБДЗ                       | М. Дендюк     | Львів, НЛТУУ                     | 2022   | 800     | 101      |

Рис. 6.1. Таблиця Katalog

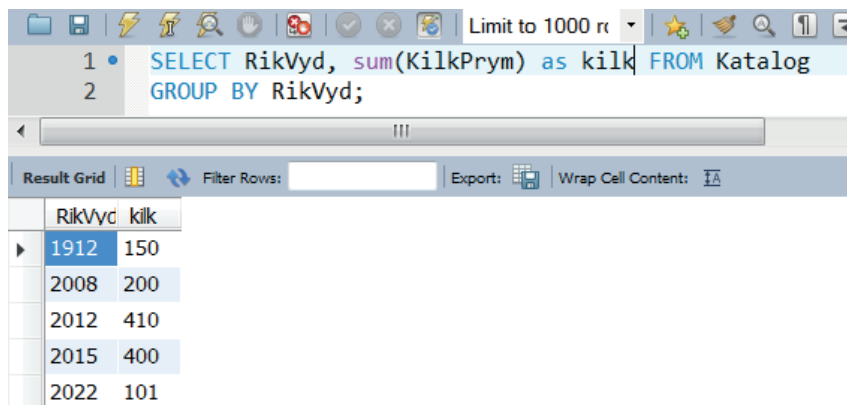
Спочатку згрупуємо записи за роками видання (рис. 6.2):



| NameK                     | AvtorK       | RikVyd | KilkPrym |
|---------------------------|--------------|--------|----------|
| Історія України-Русі      | М. Аркас     | 1912   | 150      |
| 100 великих українців     | Д.Дідух      | 2008   | 200      |
| Кобзар                    | Т.Шевченко   | 2012   | 250      |
| Мешканці морів та океанів | Жабська Т.С. | 2015   | 400      |
| ОБДЗ                      | М. Дендюк    | 2022   | 101      |

Рис. 6.2. Таблиця Katalog згрупована за роком видання RikVyd

Але в цьому випадку кількість примірників за роками не сумується, а береться перше значення відповідного запису. Щоб зробити підсумок, необхідно вказати агрегативну функцію SUM за полем кількості примірників KPRMk (рис. 6.3):



| RikVyd | kilk |
|--------|------|
| 1912   | 150  |
| 2008   | 200  |
| 2012   | 410  |
| 2015   | 400  |
| 2022   | 101  |

Рис. 6.3. Сумарна кількість примірників у бібліотеці за роками видання

Дійсно, випущених у 2012 році наявна кількість книг  $250 + 100 + 60 = 410$ .

Аналогічно можна знайти максимальну чи мінімальну кількість книг у бібліотеці, випущених у відповідному році.

### **6.2.8. Сорткування записів оператором ORDER BY**

Речення ORDER BY використовується для сорткування, якщо таке не вказано вище у реченні GROUP BY. Після ORDER BY вказуються поля, за якими йде сорткування та порядок сорткування.

За замовчуванням використовується ASC – сорткування за зростанням і його можна не вказувати. Для сорткування за спаданням використовується ключове слово DESC.

Всі поля, вказані після ORDER BY, повинні бути вказані також на початку SELECT.

**Приклад 6.26.** Відсортувати таблицю Osoba за датами народження (поле DNO):

```
SELECT * FROM Osoba  
ORDER BY DNO.
```

При такому сортуванні всі абоненти будуть відсортовані від найстарішого до наймолодшого.

**Приклад 6.27.** Відсортувати таблицю Osoba за датами народження абонентів від наймолодшого до найстарішого:

```
SELECT * FROM Osoba  
ORDER BY DNO DESC.
```

В операторі ORDER BY для сорткування можна також задавати декілька полів.

**Приклад 6.28.** Відсортувати таблицю Osoba за Прізвищами (за зростанням) та датами народження абонентів (за спаданням):

```
SELECT * FROM Osoba  
ORDER BY PIPO ASC, DNO DESC;
```

### **6.2.9. Отримання діапазону записів оператором LIMIT**

Для того, щоб не витягувати весь набір результатів і відкидати надлишкові дані, в запиті SELECT використовується оператор LIMIT.

LIMIT приймає один або два аргументи. Значення обох аргументів мають бути нульовими або додатними цілими числами. Формат запису:

```
SELECT select_list
      FROM table_name
      LIMIT [offset,] row_count;
```

У цьому синтаксисі:

- offset вказує зміщення першого рядка;
- row\_count визначає кількість рядків, що повертаються оператором SELECT.

На рис. 6.4 наведено ілюстрацію вибірки з LIMIT.

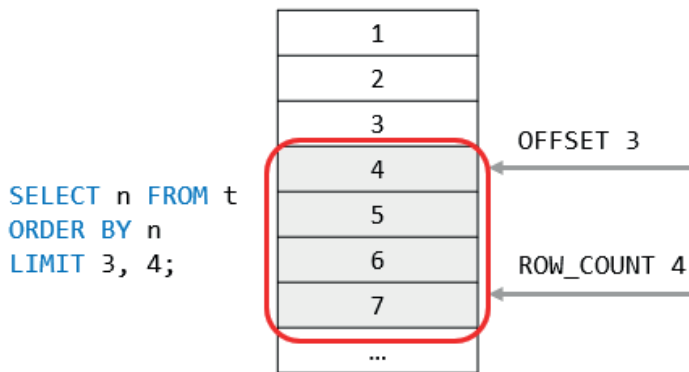


Рис. 6.4. Вибірka SELECT з оператором LIMIT

Якщо ж в операторі LIMIT задати лише один аргумент row\_count, результатом буде вибірка з першого елемента, тобто записи:

```
LIMIT row_count;
```

і

```
LIMIT 0 , row_count;
```

є ідентичними.

Варто зазначити, що сортування ORDER BY є одним з факторів, які змінюють результат виконання LIMIT і кінцевий результат вибірки SELECT може бути іншим.

### 6.3. Використання SELECT для вибору записів з декількох таблиць

Для виведення в одному запиті зв'язаної інформації, що зберігається в декількох таблицях, в мові SQL використовується операція об'єднання. У цьому проявляється одна з найбільш важливих особливостей запитів SQL – здатність визначати зв'язки між численними таблицями і виводити інформацію з них в рамках цих зв'язків. Саме ця операція надає гнучкість і легкість мови SQL, адже взаємозв'язки, що існують між даними, розташованими в різних таблицях, проявляються шляхом їх об'єднання.

Синтаксис об'єднання за предикатом має вигляд:

```
SELECT <список полів для виведення>  
FROM <таблиця 1>  
[INNER]  
{LEFT | RIGHT | CROSS} JOIN <таблиця 2>  
[ON <предикат>]  
[WHERE <умова>] ...
```

В операторі FROM може бути вказана явна операція об'єднання двох і більше таблиць. Серед ряду операцій об'єднання, описаних у стандарті MySQL, багатьма серверами баз даних підтримується тільки операція об'єднання за предикатом.

Об'єднання може бути або внутрішнім (INNER), або одним з зовнішніх (OUTER). Службові слова INNER і OUTER можна опускати, оскільки *зовнішнє об'єднання* однозначно визначається його типом – LEFT (ліве) або RIGHT (праве), а просто JOIN означатиме внутрішнє об'єднання.

Помилково вважається, що при вибірці даних типу SELECT з двох таблиць, розділених комою, результат буде такий же, як при використанні JOIN. Але результатом такого запиту буде декартовий добуток CROSS JOIN (кожен запис з першої таблиці буде продубльовано для кожного запису з другої таблиці).

Предикат визначає умову об'єднання рядків з різних таблиць, при цьому в результуючий набір потраплять тільки ті рядки з двох таблиць, для яких значення предиката – TRUE. Зазвичай, предикат визначає з'єднання за зовнішнім та

первинним ключами таблиць, які з'єднуються, хоча це не обов'язково, та має формат запису:

$$\langle \text{таблиця1} \rangle . \langle \text{поле} \rangle = \langle \text{таблиця2} \rangle . \langle \text{поле} \rangle$$

тобто у предикаті порівнюються поле однієї таблиці з полем іншої таблиці. Крім порівняння, в предикаті можна використовувати будь-який логічний оператор, наприклад, «більше» (>), «менше» (<), «не рівно» (<>) тощо. До недоліків використання нерівностей варто віднести дублювання записів.

Об'єднання даних з декількох таблиць дійсно важливо при роботі з СКБД. Оператори об'єднання надають потужний, швидкий і гнучкий спосіб вибірки потрібних даних. Однак їх треба використовувати з обережністю, оскільки неправильне їх використання може призвести до хибного сприйняття даних з БД.

### 6.3.1. Об'єднання внутрішнє

Об'єднання таблиць, яке найчастіше використовується в SQL – INNER JOIN та повертає перетин двох множин. У термінах таблиць, воно повертає тільки записи з обох таблиць, які відповідають вказаному критерію. На рис. 6.5 показана діаграма, що ілюструє перетин двох таблиць. Результатом операції є зафарбована область.

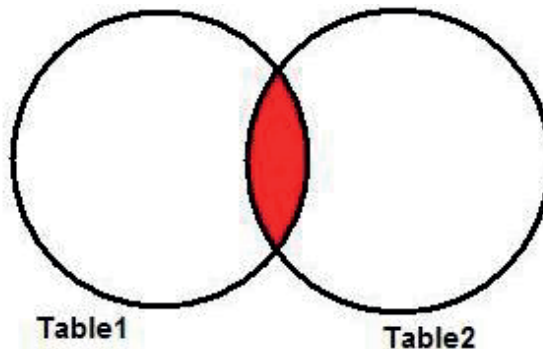


Рис. 6.5. Результат об'єднання двох таблиць за допомогою INNER JOIN

Записується цей оператор після слова FROM та має такий формат запису:

$$\begin{aligned} &\langle \text{ім'я\_таблиці1} \rangle \text{ INNER JOIN } \langle \text{ім'я\_таблиці2} \rangle \\ &\text{ON} \quad \quad \quad (\langle \text{ім'я\_таблиці1} \rangle . \langle \text{ім'я\_поля1} \rangle \\ &\quad \quad \quad \langle \text{ім'я\_таблиці2} \rangle . \langle \text{ім'я\_поля2} \rangle). \end{aligned} =$$

де ім'я\_таблиці1, ім'я\_таблиці2 задають імена таблиць, що зв'язуються. Для того,

щоб таблиці можна було зв'язати, необхідно на етапі створення структури таблиці передбачити поля, за якими здійсниться цей зв'язок.

Поля, за якими здійснюється зв'язок, записуються після ключового слова ON і між ними ставиться операція відношення (не обов'язково рівність).

**Приклад 6.29.** Вивести прізвища абонентів, які мають на руках книжки з бібліотеки:

```
SELECT O.NameO
      FROM Osoba O
     INNER JOIN Formuljar F ON O.idOs = F.IdO
    WHERE F.DataOut IS NULL
           OR F.DataOut > NOW();
```

За допомогою INNER JOIN можна задати об'єднання не тільки двох, але і декількох таблиць. Існує ще й можливість зовнішнього об'єднання таблиць, під час якого в результуючий набір попадуть записи, що відповідають внутрішньому об'єднанню і всі решта записів однієї із таблиць.

### 6.3.2. Об'єднання зовнішнє

Існує два типи зовнішнього об'єднання OUTER JOIN:

- ліве LEFT JOIN;
- праве RIGHT JOIN.

Записується цей оператор після слова FROM та має такий формат запису:

```
<ім'я_таблиці1> {LEFT | RIGHT} JOIN <ім'я_таблиці2>
ON          (<ім'я_таблиці1>.<ім'я_поля1>          =
<ім'я_таблиці2>.<ім'я_поля2>).
```

Повне зовнішнє об'єднання (FULL JOIN) не підтримуються в MySQL. Таке об'єднання представити за допомогою об'єднання UNION лівого і правого зовнішніх об'єднань.

#### 6.1.1.1. Ліве зовнішнє об'єднання LEFT JOIN

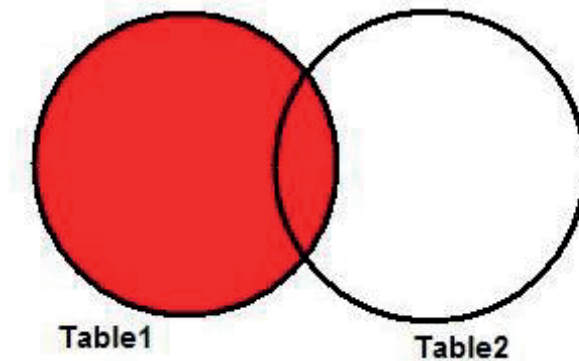
*Ліве зовнішнє об'єднання* LEFT JOIN повертає дані з таблиці в лівій частині JOIN та тільки ті дані з правої таблиці, що пересікаються з першою таблицею. Графічно LEFT JOIN показано на рис. 6.6.

**Приклад 6.30.** Вивести всі записи з таблиці Formular та ті записи із таблиці Osoba, для яких співпадають значення полів id особи:

```
SELECT * FROM Formular F
      LEFT JOIN Osoba O ON F.IdO = O.IdO;
```

**Приклад 6.31.** Якщо необхідно вказати перелік полів, які необхідно вивести, доцільно скористатися псевдонімами таблиць:

```
SELECT f.Idf, o.PIPO, f.dat_vud, f.dat_pov
      FROM Formular f
      LEFT JOIN Osoba o  ON f.IdO = o.IdO;
```



*Рис. 6.6. Результат об'єднання двох таблиць за допомогою LEFT JOIN*

**Приклад 6.32.** За необхідності виводу записів за умовою, наприклад, тих читачів, у яких не заповнено поле дата повернення, запит виглядатиме:

```
SELECT f.Idf, o.PIPO, f.dat_vud, f.dat_pov
      FROM Formular f
      LEFT JOIN Osoba o
      ON f.IdO = o.IdO
      WHERE f.dat_pov IS NULL;
```

#### **6.1.1.2. Праве зовнішнє об'єднання RIGHT JOIN**

*Праве зовнішнє об'єднання RIGHT JOIN повертає дані з таблиці, що стоїть в правій частині оператора JOIN та тільки ті дані з лівої таблиці, що пересікаються з таблицею праворуч. Графічно RIGHT JOIN показано на рис. 6.7.*

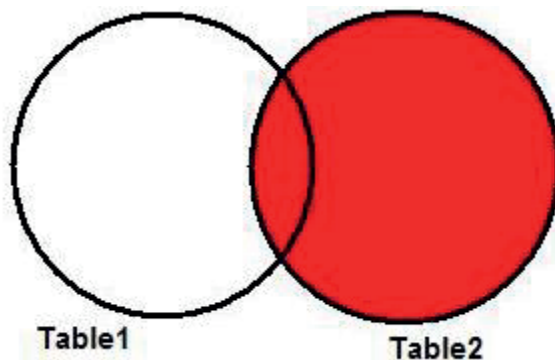


Рис. 6.7. Результат об'єднання двох таблиць за допомогою RIGHT JOIN

**Приклад 6.33.** Вивести всі записи з таблиці Katalog та ті записи із таблиці Formular, для яких співпадають значення полів id книги:

```
SELECT *
FROM Formular f
RIGHT JOIN Katalog k ON f.Id_kn=k.Idk;
```

### 6.3.3. Декартовий добуток CROSS JOIN

Декартовий добуток CROSS JOIN або використання коми як розділювача між переліком таблиць, повертає величезний набір результатів, де кожен запис з Table1 буде дублюватися для кожного запису з Table2. Наприклад, якщо в таблиці Table1 міститься  $N1$  записів, а в таблиці Table2 –  $N2$  записів, тоді результатом буде таблиця розміром  $N1 \times N2$  записів (рис. 6.8).

На практиці декартовий добуток CROSS JOIN майже не використовується.

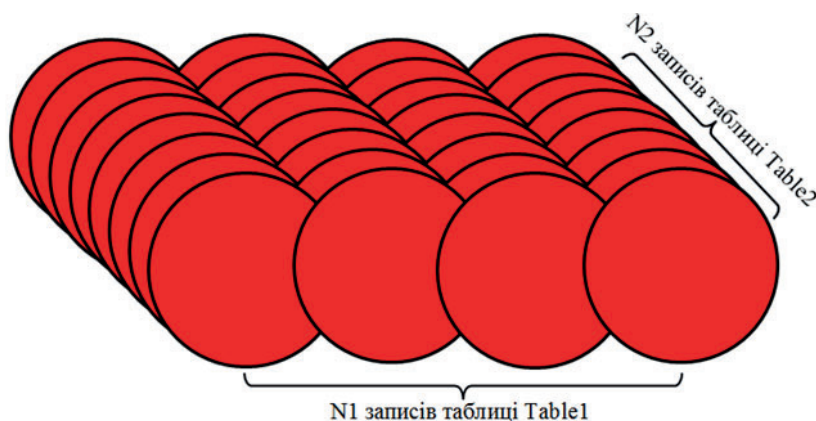


Рис. 6.8. Результат об'єднання двох таблиць за допомогою CROSS JOIN



### 6.3.4. Віднімання таблиць

Результатом віднімання двох таблиць будуть всі записи з таблиці Table1, за винятком тих, які є в таблиці Table2 (рис. 6.9).

Для віднімання множин не потрібно особливого оператора JOIN. Такі результати можна отримати, використовуючи умову WHERE.

**Приклад 6.34.** Вивести всіх читачів з таблиці Osoba, які не фігурують в таблиці Formular (zareestrowanih abonentiv, yakі nikoli ne brali knizhki v biblioteci):

```
SELECT *  
FROM BIBLIO.Osoba o  
LEFT JOIN BIBLIO.Formular f ON f.Id_ch = o.Ido  
WHERE f.Id_ch IS NULL;
```

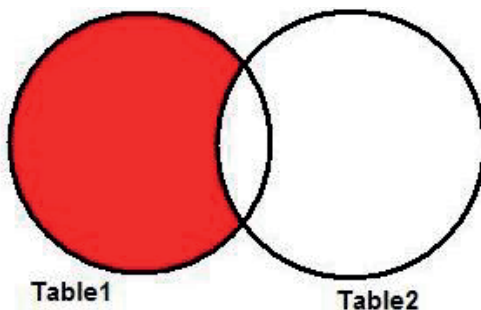


Рис. 6.9. Виключення перетину множин

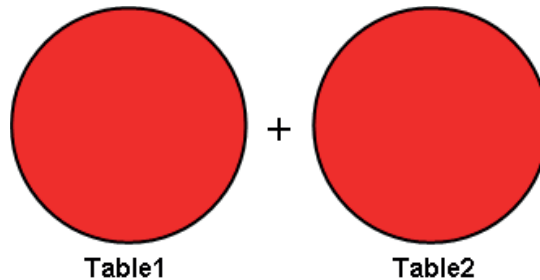
При виконанні таких запитів необхідно правильно вибрати поле для умови WHERE, зокрема, яке не допускає значень NULL. В іншому випадку набір результатів може містити непотрібні записи.

Оскільки такі абоненти відсутні в таблиці Formular, можна використати об'єднання LEFT JOIN, яке повертає дані з таблиці Osoba в лівій частині JOIN та тільки тих читачів з правої таблиці Formular, записів про яких немає – IS NULL.

В даному прикладі в умові WHERE використовується ключове поле таблиці Formular. Оскільки первинні ключі не можуть містити значення NULL, це гарантує, що набір результатів буде список абонентів, відсутніх у таблиці Formular, як і потрібно.

### 6.3.5. Додавання таблиць

Якщо результати виконання декількох запитів типу **SELECT** необхідно об'єднати один під одним (додавати записи в результуючий набір даних), використовується об'єднання типу **UNION** (рис. 6.10).



*Рис. 6.10. Додавання множин*

Оператор **UNION** вказується між запитами **SELECT**. Формат запису оператора **UNION**:

```
SELECT ...  
UNION [ALL]  
SELECT ...  
[UNION [ALL]  
SELECT ...]
```

Щоб всі рядки в результуючій таблиці були унікальними, за замовчуванням встановлено **DISTINCT** для всього результуючого набору даних. Якщо ж вказати ключове слово **ALL**, тоді результат буде містити всі знайдені рядки з усіх об'єднаних команд **SELECT**.

Для того, щоб операція об'єднання за допомогою оператора **UNION** була успішною, необхідно:

1. Кількість та порядок стовпців повинні збігатися в усіх запитах, що об'єднуються;
2. Типи даних у відповідних стовпцях повинні бути сумісні. Типи даних **DATA** і **BINARY** повинні обов'язково збігатися з іншими стовпцями такого ж типу.
3. У випадку сортування **ORDER BY** вказується у кінці запиту.

Конструкція UNION або UNION ALL значно збільшує час виконання запиту, оскільки запити при об'єднанні через UNION виходять досить великими та можуть містити дуже багато даних, і тому зловживати нею не потрібно.

**Приклад 6.35.** Необхідно вивести всі прізвища, що знаходяться у БД Бібліотека та розпочинаються на префікс «Де». Прізвища читачів знаходяться у таблицях Osoba, а прізвища авторів книг – у таблиці Katalog.

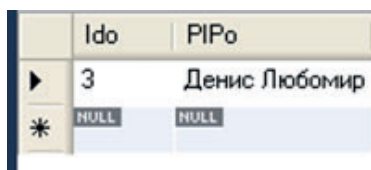
Спочатку запишемо запити до кожної з таблиць:

```
SELECT Ido, PIPo
FROM BIBLIO.Osoba
WHERE PIPo LIKE 'Де%';
```

та

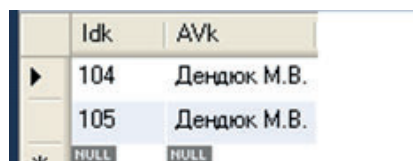
```
SELECT Idk, AVk
FROM BIBLIO.Katalog
WHERE AVk LIKE 'Де%';
```

Результатами виконання цих запитів будуть дійсно прізвища, що розпочинаються на «Де» та знаходяться у різних таблицях (рис. 6.11).



|   | Ido  | PIPo          |
|---|------|---------------|
| ▶ | 3    | Денис Любомир |
| * | NULL | NULL          |

а)



|   | Idk  | AVk         |
|---|------|-------------|
| ▶ | 104  | Дендюк М.В. |
|   | 105  | Дендюк М.В. |
| * | NULL | NULL        |

б)

Рис. 6.11. Результати виконання окремих запитів до таблиць:

а) Osoba; б) Katalog

Для об'єднання (додавання) цих результатів в один можна використати оператор UNION, оскільки кількість стовпців однакова, а типи даних у відповідних стовпцях є сумісними (1 – цілочисельний, 2 – символічний).

Таким чином, отримаємо запит:

```
SELECT Ido AS N, PIPo AS Surname
FROM BIBLIO.Osoba
WHERE PIPo LIKE 'Де%'

UNION
```

```
SELECT Idk, AVk
FROM BIBLIO.Katalog
WHERE AVk LIKE 'Де%';
```

Для наглядності перше поле у результаті виконання цього запиту назвемо «N», а друге – «Surname». Результат виконання показано на рис. 6.12.

|   | N    | Surname       |
|---|------|---------------|
| ▶ | 3    | Денис Любомир |
|   | 104  | Дендюк М.В.   |
|   | 105  | Дендюк М.В.   |
| * | NULL | NULL          |

Рис. 6.12. Результат виконання запиту з конструкцією UNION

Як бачимо з прикладу 6.35, запити на об'єднання таблиць з допомогою конструкції UNION виходять досить великими. Якщо такі запити необхідно часто використовувати, тоді доцільно їх створювати за допомогою переглядів VIEW або збережених процедур.

### 6.3.6. Самооб'єднання таблиць

Самооб'єднання або рефлексивне об'єднання (self-join) передбачає об'єднання таблиці самої з собою та використовується у випадках, коли необхідно отримати інформацію, вибрану особливим чином тільки з однієї таблиці.

Це не окремий вид об'єднання, а просто об'єднання таблиці з собою за допомогою псевдоніми (аліасів). Самооб'єднання корисні у випадках, коли потрібно отримати пари аналогічних елементів з однієї і тієї ж таблиці.

Формат запису самооб'єднання таблиць можна представити у вигляді:

```
SELECT DISTINCT <список полів>
FROM <ім'я_таблиці> <аліас1>
JOIN <ім'я_таблиці> <аліас2>
ON <аліас1>.<ім'я_поля> = <аліас2>.<ім'я_поля>
[WHERE <умова>]
[ORDER BY <аліас1>.<ім'я_поля> [<тип_сортування>]]
```

Щоб вказати таблицю два рази в одному і тому ж запиті, необхідно задавати для імен таблиць псевдоніми, які допомагатимуть оброблювачу запитів визначити, які дані повинні бути представлені в стовпці (з правої або з лівої версії таблиці).

Так само необхідно задавати і для назв полів їх псевдоніми.

**Приклад 6.36.** Нехай у таблиці *Osoba* зустрічаються абоненти, які зареєстровані в один і той же день. Необхідно вивести попарно прізвища цих абонентів, щоб записи не повторювалися.

У цьому випадку можна використати декартовий добуток таблиць (CROSS JOIN або кома) з обмеженням на неповторювані записи DISTINCT.

Таблиця *osoba* містить такі записи, як на рис. 6.13.

|   | Ido  | PIPo             | ADRo  | RNo  | MPo  | DRo        | FotoO |
|---|------|------------------|-------|------|------|------------|-------|
| ▶ | 1    | Війтович Іван    | Львів | 1995 | НПТУ | 2015-05-15 | BLOR  |
|   | 2    | Гавришев Михайло | Львів | 1995 | НПТУ | 2016-05-15 | HULL  |
|   | 3    | Денис Любомир    | Львів | 1996 | НПТУ | 2015-09-15 | HULL  |
|   | 4    | Ляшенко Євген    | Львів | 1994 | НПТУ | 2015-09-15 | HULL  |
|   | 5    | Мендзяк Андрій   | Львів | 1996 | НПТУ | 2017-09-10 | HULL  |
|   | 7    | Нечай Богдан     | Львів | 1993 | НПТУ | 2017-09-10 | HULL  |
| * | HULL | HULL             | HULL  | HULL | HULL | HULL       | HULL  |

Рис. 6.13. Вміст таблиці *Osoba*

Текст запиту виглядатиме так:

```
SELECT DISTINCT t1.PIPo, t2.PIPo, t1.DRo
FROM Osoba t1, Osoba t2
WHERE t1.DRo = t2.DRo
AND t1.Ido < t2.Ido;
```

А результатом його виконання будуть записи, показані на рис. 6.14.

|   | PIPo           | PIPo          | DRo        |
|---|----------------|---------------|------------|
| ▶ | Денис Любомир  | Ляшенко Євген | 2015-09-15 |
|   | Мендзяк Андрій | Нечай Богдан  | 2017-09-10 |

Рис. 6.14. Результат запиту за допомогою самооб'єднання таблиці *Osoba*

Щоб у результаті записи не дублювалися на зразок:

доцільно використовувати DISTINCT.

#### 6.4. Використання запитів SELECT ... INTO

Команда SELECT ... INTO копіює дані, отримані з таблиці оператором вибору, у:

- змінні;
- файл;
- дамп.

1. Синтаксис команди SELECT ... INTO при копіюванні даних з таблиці у змінні:

```
SELECT col_name [, col_name] ...  
      INTO var_name [, var_name] ...  
      FROM ...
```

Імена змінних користувача var\_name не чутливі до регістру.

Запит повинен повернути один рядок. Якщо запит не повертає рядків, з'являється попередження з кодом помилки 1329 (Немає даних), а значення змінних залишаються незмінними. Якщо запит повертає декілька рядків, виникає помилка 1172 (Результат містить більше одного рядка). Якщо є ймовірність, що оператор може отримати декілька рядків, набір результатів можна обмежити оператором LIMIT 1, що гарантовано отримати одним рядком.

**Приклад 6.41.** Записати у змінну @x середню кількість книг в таблиці Katalog:

```
SELECT AVG(KilkPrym)  
      INTO @x  
      FROM Katalog;
```

Також можна одночасно записати значення у декілька змінних.

**Приклад 6.42.** Скопіювати у змінну @x середню кількість книг в таблиці Katalog, а у змінну @S – сумарну кількість книг у тій же таблиці:

```
SELECT avg(KilkPrym), sum(KilkPrym)
```

```
INTO @x, @S
FROM Katalog;
```

Речення INTO також можна вказати в кінці оператора.

**Приклад 6.43.** Скопіювати у змінну @n рік видання книги Т.Шевченка.

Оскільки є ймовірність, що таких книг декілька, треба обмежити набір результатів:

```
SELECT RikVyd
FROM Katalog
LIMIT 1
INTO @n;
```

2. Синтаксис команди SELECT ... INTO при запису результату запиту у файл:

```
SELECT ... INTO OUTFILE 'file_name'
[CHARACTER SET charset_name]
[export_options]
```

де export\_options визначає певний формат виведення стовпців FIELDS і рядків LINES:

```
[{FIELDS | COLUMNS}
[TERMINATED BY 'string']
[[OPTIONALLY] ENCLOSED BY 'char']
[ESCAPED BY 'char']
]
[LINES
[STARTING BY 'string']
[TERMINATED BY 'string']
]
```

Файл не повинен існувати. Його неможливо перезаписати.

Користувачеві потрібні права FILE для виконання цього оператора. Однак з міркувань безпеки звичайним користувачам не надається право FILE, необхідне для виконання операторів SELECT INTO OUTFILE. Ці права мають лише адміністратори.

У клієнт-серверних БД файл `file_name` завжди зберігається на стороні сервера за вказаним шляхом.

Синтаксис `export_options` складається з тих самих частини оператора, що і `LOAD DATA – FIELDS і LINES` і дозволяє використовувати термінатори стовпців і рядків для визначення певного формату виведення. Це зумовлено тим, щоб потім коректно прочитати збережений файл.

За замовчуванням поля `FIELDS` завершуються символами табуляції (`\t`), а рядки `LINES` – символами нового рядка (`\n`). Для прикладу, якщо опустити частини `FIELDS` та `LINES`, значення за замовчуванням будуть такими ж, як якщо б було прописано:

```
FIELDS TERMINATED BY '\t' ENCLOSED BY " ESCAPED BY \"  
LINES TERMINATED BY '\n' ESCAPED BY "
```

Таким чином, термінатор `TERMINATED BY` вказує, як завершуються поля `FIELDS` чи рядки `LINES` відповідно.

Опція `FIELDS [OPTIONALLY] ENCLOSED BY` вказує, яким чином поле екранується.

Якщо опустити слово `OPTIONALLY`, усі поля беруться у символи, вказані в `ENCLOSED BY`.

Приклад такого виводу (з використанням коми як роздільника поля):

```
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY ""
```

В результаті у файл запишеться наступний рядок:

```
"1","a string","100.20"
```

Якщо ж вказати `OPTIONALLY`, опція `ENCLOSED BY` екрануватиме вказаним символом лише значення із стовпців, які мають рядковий тип даних (наприклад `CHAR`, `BINARY`, `TEXT`, або `ENUM`):

```
FIELDS TERMINATED BY ',' ENCLOSED BY ""
```

В результаті у файл запишеться наступний рядок:

```
1,"a string",100.20
```

Опція `FIELDS ESCAPED BY` вказує, як інтерпретувати спеціальні символи, такі як пробіл, табуляція, новий рядок тощо. Якщо ж опція `FIELDS ESCAPED BY`



відсутня або містить порожній символ, інтерпретація вихідної послідовності не відбувається.

Опція STARTING BY відноситься лише до рядків LINES і вказує, який загальний префікс матиме кожен рядок у файлі.

**Приклад 6.44.** Створити файл mydata у форматі значень, розділених комами (CSV), у який записати вміст таблиці Katalog:

```
SELECT * INTO OUTFILE "mydata.csv"
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n'
FROM Katalog;
```

3. Синтаксис команди SELECT ... INTO при копіюванні даних з таблиці у даім:

```
SELECT ... INTO DUMPFIL 'file_name'
```

Дана команда пише лише один рядок у файл без будь-якого стовпця або завершення рядка і не виконує обробки escape. Це корисно для зберігання BLOB значень у файлі.

## 6.5. Використання запитів SELECT в інших запитах

### 6.5.1. Створення таблиці на основі існуючої

Створюється копія існуючої таблиці (структура та дані). Формат запису:

```
CREATE TABLE <ім'я нової таблиці>
SELECT * FROM <ім'я існуючої таблиці>
```

Варто відзначити, що імена як нової, так і існуючої таблиць необхідно вказувати:

```
<ім'я_БД>.<ім'я_нової_таблиці>.
```

**Приклад 6.45.** З таблиці formyljar створити копію з назвою formyljar1 та скопіювати в неї всі дані.

```
CREATE TABLE formyljar1
SELECT * FROM formyljar
```

**При такому створенні таблиці поле первинного ключа не копіюється!**

Якщо ж необхідно створити тільки шаблон таблиці без даних, тоді замість SELECT використовується LIKE:

**Приклад 6.46.** Створити на основі таблиці Formujjar нову таблицю Formujjar2 без даних (тільки схему):

```
CREATE TABLE Formujjar2  
LIKE Formujjar
```

### 6.5.2. Створення таблиці на основі існуючої за умовою

Створюється копія існуючої таблиці тільки з даними, що відповідають певній умові. Формат запису:

```
CREATE TABLE <ім'я нової таблиці>  
SELECT * FROM <ім'я існуючої таблиці>  
WHERE <умова>;
```

**Приклад 6.47.** Створити нову таблицю Katalog1 з таблиці Katalog, яка міститиме тільки назви книг, що включають слово “програмування”.

```
CREATE TABLE Katalog1  
SELECT * FROM Katalog  
WHERE Nazva LIKE “%програмування %”
```

При такому створенні таблиці нова таблиця матиме структуру таблиці formyljar, всі записи, що відповідають вказаній умові, будуть перекопійовані у неї, *а поле первинного ключа не буде вказано!*

### 6.5.3. Створення гібридних таблиць

Гібридна таблиця створюється на основі існуючої з доповненням порожніх полів. Формат запису:

```
CREATE TABLE <ім'я нової таблиці>  
(<визначення поля 1> ... [, <визначення поля N>])  
SELECT <список полів>  
FROM <ім'я існуючої таблиці>
```

**Приклад 6.48.** Створити таблицю formyljar2 на основі таблиці formyljar та доповнити її полями POS типу VARCHAR(30) та ZVAN типу VARCHAR(30):

```
CREATE TABLE formyljar2
```

```
(POS VARCHAR(30),  
ZVAN VARCHAR(30))  
SELECT * FROM biblio. formyljar
```

*При такому створенні таблиці поле первинного ключа не копіюється!*

#### **6.5.4. Створення тимчасових таблиць**

Тимчасові таблиці використовуються для зберігання проміжних результатів і відразу після їх використання вони повинні бути видалені. Створюється тимчасова таблиця за аналогією до звичайних таблиць. Формат запису:

```
CREATE TEMPORARY TABLE <ім'я тимчасової таблиці>  
(<визначення поля 1> ... [, <визначення поля N>])
```

*Приклад 6.49.* Створити тимчасову таблицю TEM1 із полями N1 та N2, відповідно типів SMALLINT та CHAR(20), поле N1 – первинного ключа:

```
CREATE TEMPORARY TABLE Biblio.TEM1  
(N1 SMALLINT NOT NULL PRIMARY KEY,  
N2 CHAR(20));
```

#### **6.5.5. Створення тимчасових таблиць на основі існуючих**

Формат запису:

```
CREATE TEMPORARY TABLE <ім'я тимчасової таблиці>  
SELECT * FROM <ім'я існуючої таблиці>
```

*Приклад 6.50.* Створити тимчасову таблицю TEM2 на основі таблиці Osoba:

```
CREATE TEMPORARY TABLE Biblio.TEM2  
SELECT * FROM Biblio.Osoba
```

#### **6.5.6. Задавання первинних ключів у створених таблицях на основі існуючих**

Створення первинних ключів у таблицях здійснюється за допомогою команди модифікації таблиці ALTER TABLE. Формат запису:

```
ALTER TABLE <ім'я таблиці>  
ADD PRIMARY KEY (<ім'я поля>)
```

**Приклад 6.51.** Задати поле первинного ключа у гібридній таблиці formyljar2:

```
ALTER TABLE biblio. formyljar2  
ADD PRIMARY KEY ('Id')
```

### **6.5.7. Додавання записів з іншої таблиці**

Додавання записів у будь-яку створену таблицю здійснюється командою INSERT. Якщо ж необхідно додавати записи із наявних таблиць, тоді формат запису команди INSERT дещо буде відмінним згаданого вище:

```
INSERT INTO <ім'я таблиці, у яку додаємо записи> (<список полів>)  
SELECT <список полів>  
FROM <ім'я таблиці, із якої додаємо записи>  
WHERE <умова>
```

**Приклад 6.52.** Необхідно доповнити у таблиці Osoba1 поля Nom\_Osoba та Prizv записами з таблиці Osoba:

```
INSERT INTO Biblio.osoba1 (Nom_Osoba, Prizv)  
SELECT Nom_Osoba, Prizv  
FROM Biblio.Osoba
```

Після виконання цього запиту записи із таблиці Osoba перекопійовуються в поля Nom\_Osoba та Prizv таблиці Osoba1, а решта полів отримують значення NULL.

**Приклад 6.53.** Якщо ж необхідно доповнити таблицю Osoba2 для всіх полів записами із таблиці Osoba з умовою, що читачі проживають у Львові, тоді в команді INSERT поля не вказуються, а в команді SELECT – ставимо зірочку \*:

```
INSERT INTO Biblio.osoba2  
SELECT * FROM Biblio.Osoba  
WHERE Adresa LIKE "%Львів%"
```

Після виконання цього запиту тільки записи для читачів, що проживають у Львові, із таблиці Osoba перекопійовуються в таблицю Osoba2.

**Приклад 6.54.** Якщо ж необхідно доповнити таблицю Osoba3 записами із таблиці Osoba з умовою, що адреса проживання не заповнена, тоді:

```
INSERT INTO Biblio.Osoba3
SELECT * FROM Biblio.Osoba
WHERE Adresa IS NULL
```

В таблицю Biblio.Osoba3 перекопійовуються дані з незаповненими адресами.

### Контрольні запитання

1. Яке призначення команди SELECT?
2. Наведіть синтаксис запиту типу SELECT. Чи можна змінювати послідовність виразів?
3. Чим можна замінити перелік всіх полів у запиті SELECT?
4. Яким оператором задаються псевдоніми імен полів?
5. Для чого використовується конструкція WHERE?
6. Які є типи порівняння за шаблоном?
7. Наведіть конструкцію в умові WHERE для виведення значень, які задаються деяким списком. В які дужки береться список?
8. Як задається діапазон значень в умові WHERE?
9. Який оператор використовується для визначення записів із значеннями типу NULL в умові WHERE? Наведіть приклади.
10. Які групові операції підтримує мова SQL?
11. Чи можна використовувати поля таблиці в групуванні, які не використовуються в агрегативних операціях?
12. Яким оператором задається сортування та як вказується порядок сортування?
13. Як задати вибірку з двох таблиць у запиті SELECT? Наведіть загальний синтаксис.
14. Який оператор використовується для внутрішнього об'єднання таблиць в запиті SELECT?
15. Проаналізуйте результати виконання об'єднання LEFT JOIN та RIGHT JOIN. В чому їх відмінності?
16. Який результат повертає запит SELECT при декартовому добутку?

17. Як можна над таблицями здійснити операцію віднімання?
18. Для чого призначений оператор UNION?
19. Які вимоги пред'являються до таблиць, над якими виконуються оператори UNION?
20. В яких випадках застосовується самооб'єднання таблиць?
21. Як створити нову таблицю на основі наявної?
22. Що таке гібридна таблиця?
23. Де знаходиться створена тимчасова таблиця?
24. Як додати в таблицю записи з іншої таблиці?

## Тема 7. SQL функції

### 7.1. Оператори загального призначення

#### 7.1.1. Змінні в MySQL

Для конкретного процесу користувач може задавати *локальні змінні*, які в MySQL позначаються як:

**@<ім'я змінної>.**

Ім'я такої змінної починається з символу «@», далі йде безпосередньо її ім'я.

Ім'я локальної змінної повинно записуватися згідно вимог запису імен змінних: починатися з букви, дозволено використовувати цифри, букви (підтримувані сервером), символи «\_», «\$», «.» та не містити розділових знаків і спеціальних символів, в т.ч. пробіли.

Локальні змінні не вимагають ініціалізації, оскільки за замовчуванням приймають значення NULL. В них можуть зберігатися цілі числа, дійсні числа або рядкові величини. При запуску конкретного процесу всі оголошені в ньому локальні змінні автоматично активуються.

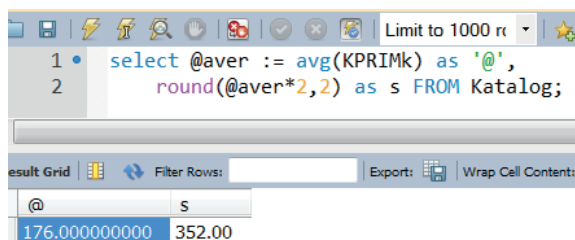
Оголошується локальна змінна командою SET:

```
SET @a = 5;
```

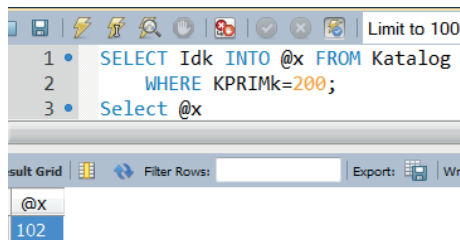
```
SET @b=" MySQL ";
```

Введені користувачем змінні можуть застосовуватися тільки в складі виразів і там, де ці вирази допустимі.

Локальну змінну можна також оголосити і ініціалізувати безпосередньо оператором SELECT:



a)



б)

Рис. 7.1. Створення локальних змінних оператором SELECT

Однозначно, оператор SELECT повинен повертати єдине значення.

Сервер MySQL підтримує велику кількість *системних змінних*, за допомогою яких можна зробити підлаштування сервера без зупинки сервера.

Системні змінні діляться на два типи:

- глобальні;
- сеансові.

Отримати повний список системних змінних в MySQL можна за допомогою команди:

### **SHOW VARIABLES;**

Частина системних змінних можна перевизначити динамічно, тобто під час роботи MySQL сервера, за допомогою команди SET.

Глобальні змінні (GLOBAL) ініціалізуються значеннями за замовчуванням при старті MySQL сервера або приймають значення, встановлені у файлі конфігурації.

Сеансові змінні (SESSION) – створюються для кожного з'єднання клієнта з сервером і отримують значення, встановлені для глобальних змінних.

Для встановлення нового значення глобальної змінної використовується команда SET, а щоб переглянути її значення – SHOW.

#### **7.1.2. Значення NULL та операції над ним**

В MySQL відсутність значення у полі таблиці позначається значенням NULL. Значення NULL – це не нуль і не FALSE, воно відповідає поняттю «порожнє поле», тобто «поле, яке не містить ніякого значення» і введене для того, щоб розрізняти незаповнені поля або відсутні значення у таких полях.

Значення NULL не є значенням в повному розумінні слова: за визначенням воно означає відсутність значення і не належить жодному з типів даних. Значення NULL у полі первинного ключа таблиці не дозволяється.

Перевіряти значення на рівність NULL за допомогою звичайних арифметичних операторів порівняння (=, <, > і т.д.) не можна.



Якщо в MySQL в операції порівняння один або обидва аргументи – NULL, тоді і результат порівняння буде NULL. Це справедливо для всіх операторів, за винятком  $\leq$ . Більш того, NULL не дорівнює NULL!

Щоб визначити, чи величина дорівнює NULL, використовуються функції:

- **<значення> IS [NOT] NULL;**
- **ISNULL(<значення>).**

Якщо <значення> дорівнює NULL, тоді результатом буде істина (1), в решти випадках – хиба (0).

Результати роботи IS [NOT] NULL показано на рис. 7.2 а, а ISNULL(<значення>) – на рис. 7.2 б.

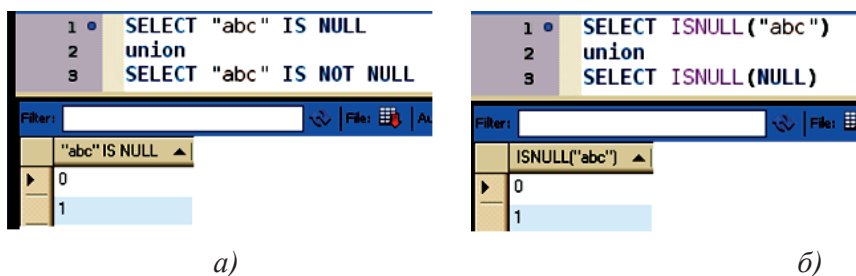


Рис. 7.2. Результати перевірки значення на NULL за допомогою:  
а) IS [NOT] NULL, б) ISNULL

Для перевірки результату на значення NULL використовуються функції:

- **IFNULL(<аргумент1>,<аргумент2>);**

Якщо <аргумент1> не NULL, тоді функція IFNULL повертає значення <аргумент1>, в іншому випадку – <аргумент2> (рис. 7.3).

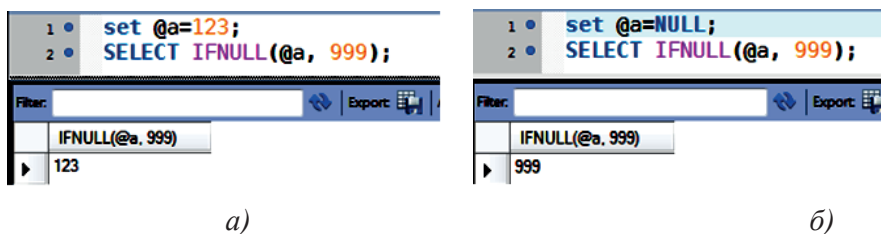


Рис. 7.3. Результати перевірки результату на значення на NULL:  
а) <аргумент1> не NULL, б) <аргумент1> NULL

Залежно від контексту функція IFNULL може повертати або числове, або вільне значення. У наведеному прикладі <аргумент1> є результатом ділення (рис. 7.4).

[illegible]

a)

```
1 • set @a=123/0, @b="mysql";
2 • SELECT @a, IFNULL(@a, @b);
```

Filter:  Export Autosave

|   | @a   | IFNULL(@a, @b) |
|---|------|----------------|
| ▶ | NULL | mysql          |

6)

Рис. 7.4. Значення результату функції IFNULL: а) числове, б) символічне

- **NULLIF(<аргумент1>,<аргумент2>).**

Функція NULLIF еквівалентна оператору:

IF (x = y) THEN NULL ELSE x,

тобто, якщо `<аргумент1> = <аргумент2>` істина, тоді функція `NULLIF` повертає `NULL`, в іншому випадку – `<аргумент1>` (рис. 7.5).

```
1 • set @a=123, @b=123;  
2 • SELECT NULLIF(@a, @b);
```

a)



The screenshot shows a SQL editor with the following code:

```
1 • set @a=123, @b="mysql";
2 • SELECT NULLIF(@a, @b);
```

Below the editor, there is a table view with a filter bar. The filter bar contains the text "Filter:" followed by a search input field and an "Export" button. The table view shows a single row with the value "123" in the first column. The second column is labeled "NULLIF(@a, @b)" and has a dropdown arrow.

6)

Рис. 7.5. Значення результату функції `NULLIF`:

а)  $\langle \text{аргумент1} \rangle = \langle \text{аргумент2} \rangle$ , б)  $\langle \text{аргумент1} \rangle \neq \langle \text{аргумент2} \rangle$

**Приклад 7.1.** Необхідно вивести назви книг, занесених в каталог. У випадку, коли назву не заповнено, тоді вивести – ‘НЕ ВІРНО ЗАПОВНЕНО’ :

```
SELECT IFNULL(NAMk,'НЕ ВІРНО ЗАПОВНЕНО')
FROM BIBLIO.Katalog;
```

### 7.1.3. Оператори керування в запитах

У запитах типу SELECT у блоці переліку полів в якості обчислювальних полів можна використовувати такі оператори керування:

- IF;

- CASE.

Оператор IF має такий формат запису:

**IF(<аргумент1>,<аргумент2>,<аргумент3>)**

Результатом <аргументу1> можуть бути такі значення:

- істина (TRUE);
- хиба (FALSE);
- 0 (нуль);
- NULL.

Якщо <аргумент1> дорівнює значенню ІСТИНА або деякому числу (не нуль), тоді функція IF повертає <аргумент2>, в іншому випадку – <аргумент3> (рис. 7.6-7.7).



Рис. 7.6. Значення результату функції IF, якщо <аргумент1> – істина або число

Аргументи <аргумент2> та <аргумент3> можуть бути виразами. Тоді результатом функції IF буде результат відповідного виразу.



Рис. 7.7. Значення результату функції IF, якщо <аргумент1> – хиба або 0

Залежно від контексту функція IF може повертати або числове, або символічне значення.

**Приклад 7.2.** Необхідно вивести кількість примірників для кожної з книг. У випадку, якщо їх кількість менше 50, тоді вивести – ‘НЕДОСТАТНЯ КІЛЬКІСТЬ’:

```
SELECT Idk, AVk,
       IF(KPRIMk>50, KPRIMk, 'НЕДОСТАТНЯ КІЛЬКІСТЬ') AS Kilk
FROM BIBLIO.Katalog;
```

Оператор CASE має дві форми запису:

- **CASE WHEN** <значення-умова> **THEN** <значення-результат>  
[**WHEN** <значення-умова> **THEN** <значення-результат> ...]  
[**ELSE** <значення за замовчуванням> ]  
**END;**
- **CASE** <перемикач>  
**WHEN** <значення перемикача> **THEN** <значення-результат>  
[**WHEN** <значення перемикача> **THEN** <значення-результат>...]  
[**ELSE** <значення за замовчуванням> ]  
**END.**

Перший оператор CASE працює аналогічно оператору IF та повертає значення-результат, якщо значення-умова є істинною, в протилежному випадку повертає значення за замовчуванням (рис. 7.8). Якщо ж в цьому операторі відсутній блок ELSE, а значення-умова є хибною, тоді результатом буде NULL.



Рис. 7.8. Значення результату функції CASE першої форми

Друга форма запису оператора CASE повертає значення-результат для того блоку WHEN, у якому значення перемикача співпадає, в протилежному випадку повертає значення за замовчуванням (рис. 7.9).

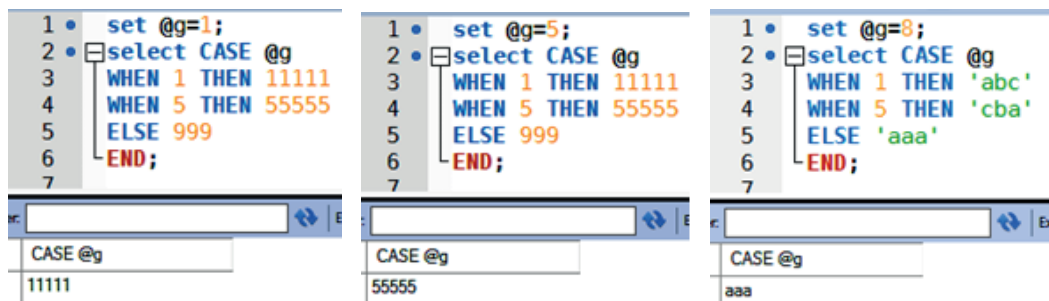


Рис. 7.9. Значення результату функції CASE другої форми

**Приклад 7.3.** Необхідно вивести авторів та назви книг, занесених в каталог. У випадку, коли у полі видавництво (VYDk) є ‘СВІТ ДИТИНИ’, тоді замінити на ‘АБАБАГАЛАМАГА’, коли ж ‘Вільямс’ – на ‘Наукова думка’, в решти випадках видавництво залишити без змін:

```
SELECT AVk, NAMk,  
       CASE VYDk  
         WHEN 'СВІТ ДИТИНИ' THEN 'АБАБАГАЛАМАГА'  
         WHEN 'Вільямс' THEN 'Наукова думка'  
         ELSE VYDk  
       END  
FROM BIBLIO.Katalog;
```

## 7.2. Агрегатні функції

Для проведення статистичних обчислень над таблицею в SQL використовуються агрегатні (групові або статистичні) функції. У СКБД MySQL є шість агрегатних функцій, які дозволяють отримувати різні види підсумкової інформації:

- **AVG()** – середнє арифметичне значення зі всіх вибраних не NULL значень даного поля. Функція AVG підраховує суму відомих значень і ділить її на кількість цих значень, а не на загальну кількість значень, серед яких можуть бути NULL-значення. Якщо стовпець складається тільки з порожніх значень, тоді функція AVG також поверне NULL;
- **COUNT()** – кількість записів зі всіх вибраних значень даного поля (за винятком NULL значень);
- **COUNT(\*)** – загальна кількість записів у таблиці і не залежить від наявності або відсутності значень NULL. Якщо рядків у таблиці немає, ця функція повертає нуль;
- **MAX()** – найбільше зі всіх вибраних значень даного поля;
- **MIN()** – найменше зі всіх вибраних значень даного поля;
- **SUM()** – арифметична сума всіх вибраних значень даного поля.

В операторі SELECT ці функції записуються у блоці переліку полів як дія над деяким полем, наприклад: MAX(Katalog.Kprimk), AVG(Osoba.RNo).

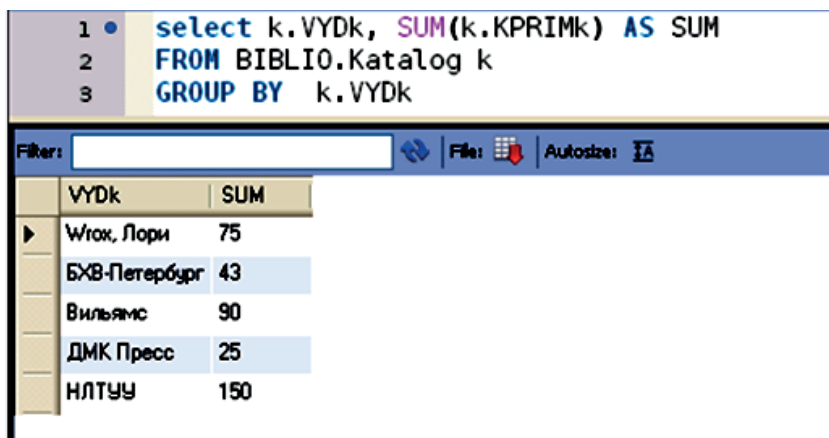
Щоб здійснити деяку агрегатну функцію над необхідними записами таблиці, необхідно використовувати умову WHERE.

Операція GROUP BY використовується в операторі SELECT для того, щоб об'єднувати повторювані значення в групи. Ця операція GROUP BY повинно бути розміщена за виразом WHERE і перед операцією ORDER BY.

За стандартом SQL в запиті з групуванням у блоці переліку полів можна вказувати тільки ті поля, за якими йде групування, або які використовуються з агрегатними функціями. Якщо ж вказати інші поля, тоді при виконанні такого запиту ці поля не будуть мати ніякого змісту.

З огляду на сказане вище, у запитах, що об'єднують повторювані значення в групи, вказувати всі поля, наприклад \*, недоцільно.

**Приклад 7.4.** Вивести сумарну кількість книг кожного з видавництв, наявних в бібліотеці (рис .7.10).



The screenshot shows a database query interface. At the top, there is a query editor with the following SQL code:

```
1 • select k.VYDk, SUM(k.KPRIMk) AS SUM
2 FROM BIBLIO.Katalog k
3 GROUP BY k.VYDk
```

Below the query editor, there is a table with the following data:

| VYDk          | SUM |
|---------------|-----|
| Wrox, Лори    | 75  |
| БХВ-Петербург | 43  |
| Вильямс       | 90  |
| ДМК Пресс     | 25  |
| ИЛТУЧ         | 150 |

Рис. 7.10. Операція сумування за групами

Отримані результати можна сортувати за допомогою операції ORDER BY. У цьому випадку сортується не вміст таблиці, а результати запиту.

## 7.3. Числові функції

### 7.3.1. Математичні функції

СКБД MySQL надає широкий набір вбудованих математичних функцій, які можна використовувати для перетворення даних та різноманітних математичних розрахунків.

Відмінною рисою MySQL є те, що при використанні функцій пробіли між ім'ям функції і круглими дужками неприпустимі. Так, написання NOW() коректно, NOW\_() – ні.

У табл. 7.1 представлений список основних математичних функцій, доступних в СКБД MySQL.

**Таблиця 7.1. Математичні функції в MySQL**

| Функція          | Пояснення                                     | Приклад                                              | Результат         |
|------------------|-----------------------------------------------|------------------------------------------------------|-------------------|
| 1                | 2                                             | 3                                                    | 4                 |
| • ABS(x)         | абсолютне значення числа x                    | ABS(-3.124)                                          | 3.125             |
| • SIGN(x)        | знак числа x                                  | SIGN(-3.124)<br>SIGN(3.124)                          | -1<br>1           |
| • PI()           | значення числа $\pi$                          | PI()                                                 | 3.141593          |
| • RAND()         | випадкове значення з діапазону від 0 до 1     | RAND()                                               | випадкове число   |
| • MOD(x, y)      | модуль (залишок) від ділення x/y              | MOD(5, 3)<br>MOD(4, 2)<br>MOD(5, 2)                  | 2<br>0<br>1       |
| • ROUND(x, y)    | заокруглене значення x з y цифрами після коми | ROUND(PI(), 3)<br>ROUND(3.468, 0)<br>ROUND(3.468, 1) | 3.142<br>3<br>3.5 |
| • TRUNCATE(x, y) | «відрізане» значення x з y цифрами після коми | TRUNCATE(PI(),3)                                     | 3.141             |
| • CEILING(x)     | найменше ціле більше числа x                  | CEILING(-3.125)<br>CEILING(3.125)                    | -3<br>4           |
| • FLOOR(x)       | найбільше ціле менше числа x                  | FLOOR(-3.125)<br>FLOOR(3.125)                        | -4<br>3           |
| • SQRT(x)        | корінь квадратний числа x                     | SQRT(2)                                              | 1.41421356        |
| • EXP(x)         | значення $e^x$                                | EXP(1)                                               | 2.71828183        |

| 1                                  | 2                                              | 3                        | 4                        |
|------------------------------------|------------------------------------------------|--------------------------|--------------------------|
| • <b>POW(x, y) або POWER(x, y)</b> | $x^y$                                          | POW(2, 3)<br>POWER(3, 2) | 8<br>9                   |
| • <b>DEGREES(x)</b>                | результат перетворення x<br>радіан в градуси   | DEGREES(PI())            | 180                      |
| • <b>RADIANS(x)</b>                | результат перетворення x<br>градусів у радіани | RADIANS(180)             | 3.141593                 |
| • <b>SIN(x)</b>                    | синус числа x (в радіанах)                     | SIN(PI()/6)              | 0.4999999                |
| • <b>COS(x)</b>                    | косинус числа x (в радіанах)                   | COS(PI()/3)              | 0.5000001                |
| • <b>TAN(x)</b>                    | тангенс числа x (в радіанах)                   | TAN(PI()/4)              | 0.9999999                |
| • <b>COT(x)</b>                    | котангенс числа x (в радіанах)                 | COT(PI()/4)              | 1.0000002                |
| • <b>ASIN(x)</b>                   | арксинус числа x (в радіанах)                  | ASIN(0.4999999)          | 0.5235986                |
| • <b>ACOS(x)</b>                   | арккосинус числа x (в<br>радіанах)             | ACOS(0.4999999)          | 1.04719766               |
| • <b>ATAN(x)</b>                   | арктангенс числа x (в<br>радіанах)             | ATAN(0.4999999)          | 0.46364752               |
| • <b>LN(x) або LOG(x)</b>          | $\log_e x$                                     | LN(2)<br>LOG(2)          | 0.69314718<br>0.69314718 |
| • <b>LOG(x,a)</b>                  | $\log_a x$                                     | LOG(15, 2)               | 0.25595802               |

Для представлення результатів роботи математичних операцій і функцій доцільно використовувати функцію форматування:

- **FORMAT(<число>, N)** – форматує аргумент <число> у формат виду ‘#,###,###.##’ з заокругленням до N десяткових знаків.

Якщо N рівне 0, тоді результат буде без цифр після десяткової точки (рис. 7.11).

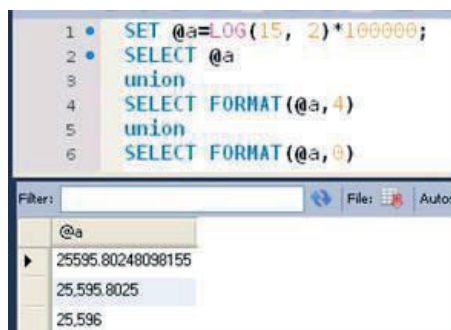


Рис. 7.11. Функція форматування чисел



### 7.3.2. Функції дати і часу

СКБД MySQL надає користувачу велику кількість можливостей для роботи з даними типу дати і часу. Розглянемо деякі з них.

Для отримання поточного значення дати і часу використовуються функції:

- **NOW();**
- **SYSDATE();**
- **CURRENT\_TIMESTAMP();**

які записуються без аргументів та повертають поточну дату і час як величину у форматі YYYY-MM-DD HH:MM:SS (табл. 7.2).

**Таблиця 7.2. Функції визначення поточної дати і часу**

| Функція                       | Результат             |
|-------------------------------|-----------------------|
| • <b>NOW();</b>               | ‘2017-02-05 14:50:26’ |
| • <b>SYSDATE();</b>           |                       |
| • <b>CURRENT_TIMESTAMP();</b> |                       |

Для отримання поточної дати застосовуються функції:

- **CURRENT\_DATE();**
- **CURDATE();**

без аргументів. Результатом буде сьогоднішня дата у форматі YYYY-MM-DD, наприклад, ‘2017-02-24’.

Рядок дати складається з декількох аргументів і для отримання цих аргументів з календарних значень дати використовуються функції перетворення:

**Таблиця 7.3. Функції дати**

| Функція                | Пояснення                                              | Приклад                | Результат |
|------------------------|--------------------------------------------------------|------------------------|-----------|
| 1                      | 2                                                      | 3                      | 4         |
| • <b>YEAR(дата)</b>    | Порядковий номер року                                  | YEAR(‘2017-02-05’);    | 2017      |
| • <b>MONTH(дата)</b>   | Порядковий номер місяця в році в діапазоні від 1 до 12 | MONTH(‘2017-02-05’);   | 2         |
| • <b>QUARTER(дата)</b> | Номер кварталу року в діапазоні від 1 до 4             | QUARTER(‘2017-02-05’); | 1         |

| 1                                               | 2                                                             | 3                         | 4        |
|-------------------------------------------------|---------------------------------------------------------------|---------------------------|----------|
| • <b>WEEK(дата)</b>                             | Номер тижня в році в діапазоні від 1 до 53                    | WEEK('2017-02-05');       | 6        |
| • <b>WEEKDAY(дата)</b>                          | Номер дня тижня (0 – понеділок, 1 – вівторок, ... 6 – неділя) | WEEKDAY('2017-02-05');    | 6        |
| • <b>DAYOFWEEK(дата)</b>                        | Номер дня тижня (1 – неділя, 2 – понеділок, ... 7 – субота)   | DAYOFWEEK ('2017-02-05'); | 1        |
| • <b>DAY(дата)</b><br>• <b>DAYOFMONTH(дата)</b> | Порядковий номер дня діапазоні від 1 до 31                    | DAY('2017-02-05');        | 5        |
| • <b>DAYOFYEAR(дата)</b>                        | Порядковий номер дня року в діапазоні від 1 до 366            | DAYOFYEAR('2017-02-05');  | 36       |
| • <b>MONTHNAME(дата)</b>                        | Назва місяця в році                                           | MONTHNAME('2017-02-05');  | February |
| • <b>DAYNAME(дата)</b>                          | Назва дня тижня                                               | DAYNAME('2017-02-05');    | Sunday   |

Формат запису дати за замовчуванням згідно синтаксису MySQL:

**"рік-місяць-день " ("2017-02-25").**

Якщо необхідно отримати або записати дату в іншому форматі, використовується функція:

- **DATE\_FORMAT("date", 'format'),**

де date – власне дата, а format – рядок, який складається із кодів, записаних через крапку і задає формат відображення дати, наприклад, запит:

```
SELECT DATE_FORMAT("2017-02-05", '%d.%m.%Y');
```

поверне значення 05.02.2017.

Аргумент format може складатися з кодів, які наводнено в табл. 7.4.

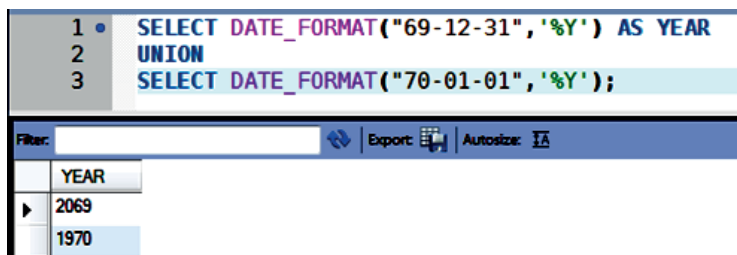
Послідовність і кількість кодів у рядку format можуть бути довільними.

**Таблиця 7.4. Функції кодів у рядку *format* аргументу *date***

| Код | Пояснення                           | Приклад:<br>date="2017-02-05" | Результат<br>DATE_FORMAT |
|-----|-------------------------------------|-------------------------------|--------------------------|
| %Y  | Рік, число, 4 розряди               | '%d.%m.%Y'                    | 05.02.2017               |
| %y  | Рік, число, 2 розряди               | '%d.%m.%y'                    | 05.02.17                 |
| %m  | Місяць, число (00..12)              | '%d.%m.%Y'                    | 05.02.2017               |
| %c  | Місяць, число (0..12)               | '%d.%c.%Y'                    | 05.2.2017                |
| %M  | Назва місяця                        | '%d.%M.%Y'                    | 05 February 2017         |
| %b  | Скорочена назва місяця              | '%d.%b.%Y'                    | 05 Feb 2017              |
| %d  | День місяця, число (00..31)         | '%d.%m.%Y'                    | 05.02.2017               |
| %e  | День місяця, число (0..31)          | '%e.%b.%Y'                    | 5 Feb 2017               |
| %W  | Назва дня тижня                     | '%W.%e.%b.%Y'                 | Sunday 5 Feb 2017        |
| %a  | Скорочена назва дня тижня           | '%a.%e.%b.%Y'                 | Sun 5 Feb 2017           |
| %j  | День року (001..366)                | '%j.%a.%b'                    | 036 Sun Feb              |
| %w  | День тижня (0 - неділя, 6 - субота) | '%d.%a.%w'                    | 05 Sun 0                 |

Проте **необхідно бути уважним** при задаванні скороченої форми запису року (без століття). За замовчуванням зі скороченим відображенням року дата приймає значення від 0 до 69 – XXI століття, від 70 до 99 – XX століття.

Приклад відображення століття показано на рис. 7.12.



*Рис. 7.12. Відображення століття у скороченому форматі року*

Для отримання поточного часу призначені функції:

- **CURRENT\_TIME();**
- **CURTIME();**

без аргументів. Результатом буде поточний час у форматі HH-MM-SS, наприклад, '14:50:26'.

Рядок часу складається з декількох аргументів і для отримання цих аргументів з календарних значень використовуються такі функції (табл. 7.5).

**Таблиця 7.5. Функції часу**

| Функція               | Пояснення                                | Приклад                                               | Результат |
|-----------------------|------------------------------------------|-------------------------------------------------------|-----------|
| • <b>hour(time)</b>   | час в діапазоні від 0 до 23              | hour('14:50:26');<br>hour('2017-02-05 14:50:26');     | 14        |
| • <b>minute(time)</b> | кількість хвилин в діапазоні від 0 до 59 | minute('14:50:26');<br>minute('2017-02-05 14:50:26'); | 50        |
| • <b>second(time)</b> | кількість секунд в діапазоні від 0 до 59 | second('14:50:26');<br>second('2017-02-05 14:50:26'); | 26        |

Якщо необхідно отримати або записати час в іншому форматі, використовується та ж функція, що і для форматування дати:

- **TIME\_FORMAT('time', 'format'),**

де time – власне час, а format – рядок, який складається із кодів, записаних через крапку і задає формат відображення часу, наприклад, запит:

SELECT TIME\_FORMAT("14:50:26", '%l.%i.%s');

поверне значення 2:50:26.

Аргумент format може складатися з наступних кодів (табл. 7.6).

**Таблиця 7.6. Функції кодів у рядку format аргументу time**

| Код              | Пояснення                                     | time                     | Приклад                  | Результат                  |
|------------------|-----------------------------------------------|--------------------------|--------------------------|----------------------------|
| <b>%H</b>        | Година, 00..23                                | "08:06:09"               | '%H.%i.%s'               | 08:06:09                   |
| <b>%k</b>        | Година, 0..23                                 | "08:06:09"               | '%k.%i.%s'               | 8:06:09                    |
| <b>%h або %I</b> | Година, 01..12                                | "08:06:09"<br>"16:06:09" | '%h.%i.%s'<br>'%I.%i.%s' | 08:06:09<br>04:06:09       |
| <b>%l</b>        | Година, 1..12                                 | "08:06:09"               | '%l.%i.%s'               | 08:06:09                   |
| <b>%i</b>        | Хвилини, 00..59                               | "08:06:09"               | '%h.%i.%s'               | 08:06:09                   |
| <b>%S або %s</b> | Секунди, 00..59                               | "08:06:09"<br>"08:06:09" | '%h.%i.%s'               | 08:06:09                   |
| <b>%r</b>        | Час, 12- годинний формат (hh:mm:ss [AP   AM]) | "08:06:09"<br>"14:05:06" | '%r'                     | 08:06:09 AM<br>02:05:06 PM |
| <b>%T</b>        | Час, 24-годинний формат (hh:mm:ss)            | "08:06:09"<br>"14:05:06" | '%T'                     | 08:06:09<br>14:05:06       |
| <b>%p</b>        | AM або PM                                     | "08:06:09"<br>"14:05:06" | '%p'                     | AM<br>PM                   |

Якщо в аргументі date або time є більше значень, чим в рядку format, тоді результатом будуть тільки ті значення, що задіяні в перетворенні формату:

```
SELECT DATE_FORMAT("2017-02-06 14:05:06", '%H.%i.%s'); → 14:05:06
```

```
SELECT DATE_FORMAT("2017-02-06 14:05:06", '%e.%b.%Y'); → 6 Feb 2017
```

```
SELECT DATE_FORMAT("2017-02-06 14:05:06", '%Y.%H'); → 2017.14
```

СКБД MySQL надає також можливості користувачу здійснювати арифметичні дії над датою і часом:

- **DATE\_ADD(date, INTERVAL <інтервал> <TYPE>)**

або

```
ADDDATE(date, INTERVAL <інтервал> <TYPE>);
```

- **DATE\_SUB(date, INTERVAL <інтервал> <TYPE>)**

або

- **SUBDATE(date, INTERVAL <інтервал> <TYPE>);**

де аргумент data – значення типу date або time, до якого додається або віднімається значення <інтервал> <TYPE> одиниць.

Ключове слово **INTERVAL** є обов'язковим.

Дії можна здійснювати над секундами (SECOND), хвилинами (MINUTE), годинами (HOUR), днями (DAY), місяцями (MONTH), роками (YEAR), а також над комбінаціями перерахованих типів.

*Приклад 7.5.* Додати до деякого часу 1 секунду:

```
SELECT "2017-02-06 14:05:06" + INTERVAL 1 SECOND;
```

```
→ 2017-02-06 14:05:07
```

```
SELECT "2017-02-06 23:59:59" + INTERVAL 1 SECOND;
```

```
→ 2017-02-07 00:00:00
```

*Приклад 7.6.* Додати до деякої дати 1 день:

```
SELECT INTERVAL 1 DAY + "2017-02-06 14:05:06"
```

```
→ 2017-02-07 14:05:06
```

```
SELECT INTERVAL 1 DAY + "2017-02-28 14:05:06"
```

```
→ 2017-03-01 14:05:06
```

При використанні некоректних значень дат результатом буде значення NULL. Якщо ж при підсумовуванні MONTH, YEAR\_MONTH або YEAR номер

дня в результуючій даті перевищує максимальну кількість днів у новому місяці, тоді номер дня результуючої дати приймається рівним останнього дня нового місяця. Наприклад, якщо до 30 січня додати один місяць, повинно було б вийти 30 лютого, однак результатом буде останній день лютого (рис. 7.13).



Рис. 7.13. Результати додавання 1 місяця до різних дат

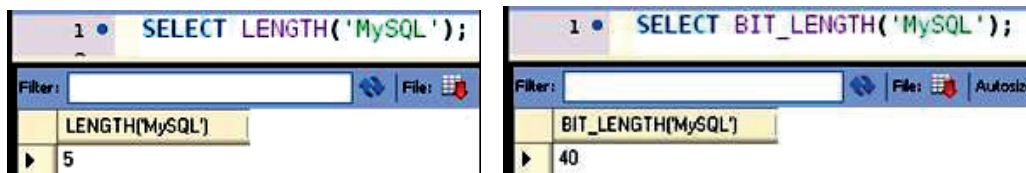
Крім перерахованих, СКБД MySQL має багато інших функцій для роботи з датою і часом, які можна подивитися за посиланням: Довідник з MySQL – Функції дати і часу, наприклад, <https://www.tutorialspoint.com/mysql/mysql-date-time-functions.htm>.

## 7.4. Функції роботи з рядками

### 7.4.1. Функції визначення властивостей рядків

Нумерація позицій для функцій, які працюють з рядками, починається з 1.

Для визначення довжини рядка у символах та бітах використовуються відповідно функції LENGTH(str) та BIT\_LENGTH(str) (рис. 7.14).



а)

б)

Рис. 7.14. Визначення довжини рядка у: а) символах; б) бітах

Щоб отримати значення ASCII-коду деякого символу, використовується одноіменна функція ASCII({символ | число | рядок}) (рис. 7.15).

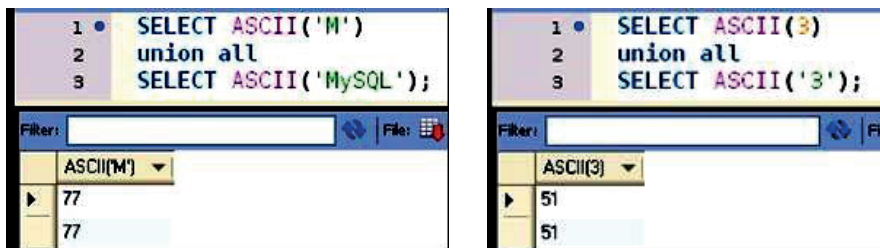


Рис. 7.15. Перетворення символу в ASCII-код

Аргументом функції ASCII можуть бути як символ, так і число або рядок символів. Якщо аргумент – символ або число від 0 до 9, тоді функція повертає його ASCII-код. Якщо аргумент – від’ємне число, тоді функція повертає ASCII-код знаку «-» (мінус). Якщо аргумент – декілька символів (рядок) або число більше 9, тоді повертає ASCII-код першого символу.

Щоб отримати зворотнє перетворення з ASCII у символ, число або рядок, використовується функція CHAR( $x_1, x_2, \dots, x_n$ ) (рис. 7.16).

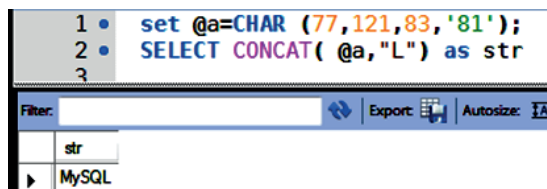


Рис. 7.16. Перетворення ASCII кодів у символи

СКБД MySQL також надає можливості конвертування чисел з однієї системи числення в іншу (рис. 7.17). Формат запису функції:

**CONV(<число>,<з якої системи>,<в яку систему>)**

Функція CONV повертає представлення перетвореного <числа> із вказаних систем. Якщо хоча б один з аргументів рівний NULL, тоді і результат буде NULL.

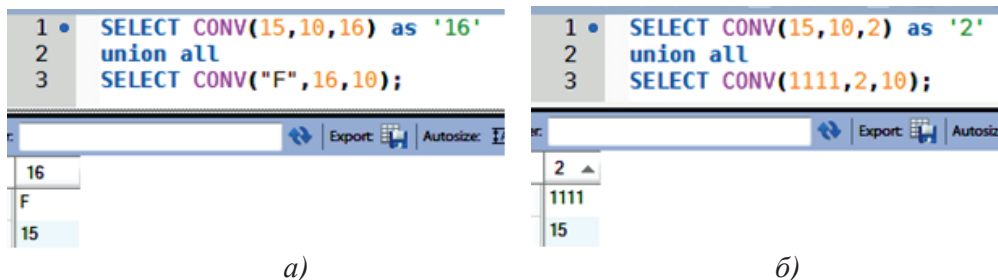


Рис. 7.17. Перетворення числа: а) з десятичної системи числення в шістнадцяткову та навпаки; б) з десятичної системи числення в двійкову та навпаки;

Аргумент <число> інтерпретується як ціле число, але може бути заданий як рядок. Мінімальне значення основи системи числення рівне 2, а максимальне – 36.

Якщо аргумент <число> може приймати від'ємні значення, тоді і систему числення, в яку перетворюється <число>, необхідно вказати зі знаком мінус (рис. 7.18).

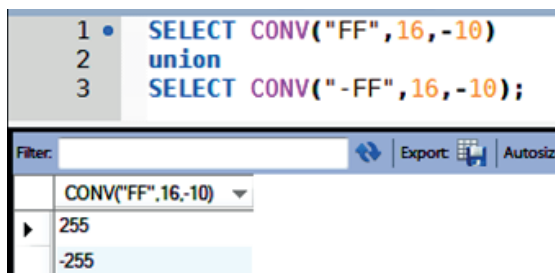


Рис. 7.18. Перетворення від'ємних чисел

Функція CONV має функції-аналоги для перетворення з десятикової системи числення у двійкову, вісімкову та шістнадцяткову систему числення:

- **BIN(<число>)** – перетворює <число> в двійкову систему, еквівалентна `CONV(<число>, 10, 2);`
- **OCT(<число>)** – перетворює <число> у вісімкову систему, еквівалентна `CONV(<число>, 10, 8);`
- **HEX(<число>)** – перетворює <число> в шістнадцяткову систему, еквівалентна `CONV(<число>, 10, 16).`

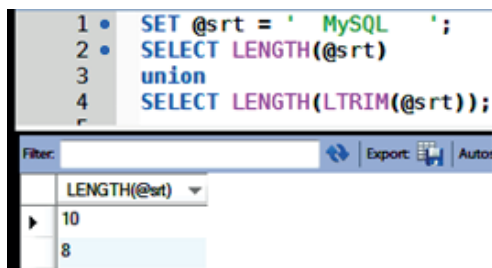
#### 7.4.2. Функції перетворення у рядках

Пробіли в кінці рядка або на його початку спотворюють його характеристики. Для усунення пробілів в MySQL використовуються такі функції:

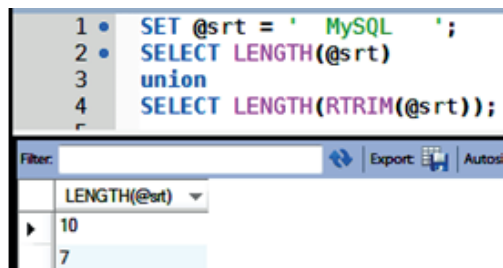
- **LTRIM(<рядок>)** – відсікає в рядку всі пробіли зліва;
- **RTRIM(<рядок>)** – відсікає в рядку всі пробіли справа;
- **TRIM(<рядок>)** – відсікає в рядку всі пробіли справа і зліва.

На рис. 7.19 показано використання перших двох функцій LTRIM та RTRIM.





а)



б)

Рис. 7.19. Використання функцій а) LTRIM та б) RTRIM

У рядку @str ліворуч від тексту MySQL є 2 пробіли, а праворуч – 3, а загальна кількість символів – 10. Після застосування функції LTRIM зліва від тексту MySQL усунено 2 пробіли та кількість символів у рядку стала 8, відповідно, при застосуванні функції RTRIM – усунено праворуч 3 пробіли та кількість символів у рядку стала 7.

Функція TRIM має дещо складніший формат запису:

**TRIM([BOTH | LEADING | TRAILING]  
[<фрагмент>] FROM] <рядок>)**

та усуває не тільки пробіли, але і задані в аргументі <фрагмент> символи.

Якщо аргументи [[BOTH | LEADING | TRAILING][<фрагмент>] FROM] не задані, тоді видаляються всі пробіли ліворуч та праворуч у <рядку>, тобто функція працює як сума функцій LTRIM та RTRIM (рис. 7.20).

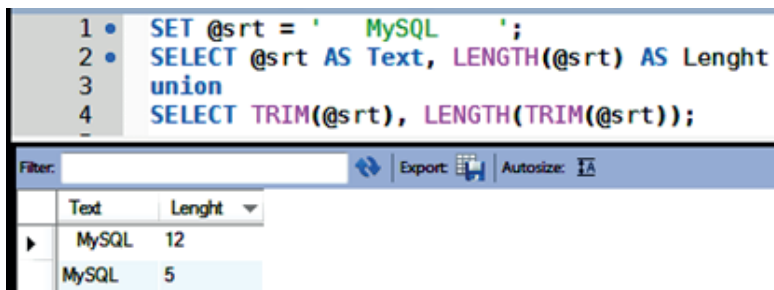


Рис. 7.20. Використання функції TRIM без аргументів

У рядку @str ліворуч від тексту MySQL є 3 пробіли, а праворуч – 4, а загальна кількість символів – 12. Після застосування функції TRIM без аргументів пробіли зліва та праворуч від тексту MySQL усунено та кількість символів у рядку стала 5.

Якщо ж аргумент [<фрагмент>] FROM] задано, а аргументи [BOTH | LEADING | TRAILING] не задані, а тоді за замовчуванням спрацьовує аргумент BOTH – усунення заданих символів <фрагмент> з двох сторін рядка (рис. 7.21).

У рядку @str ліворуч від тексту MySQL є 3 символи x, а праворуч – 4, а загальна кількість символів – 12. Після застосування функції TRIM без аргументів [BOTH | LEADING | TRAILING] символи 'x' та 'xx' зліва та праворуч від тексту MySQL усунено.

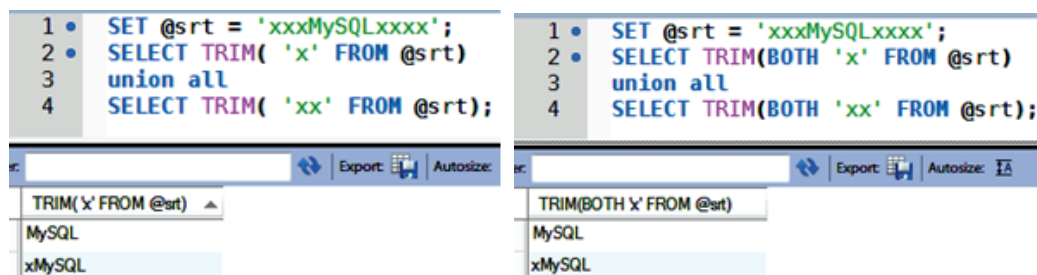


Рис. 7.21. Використання функції TRIM без аргументів [BOTH | LEADING | TRAILING]

Для видалення вказаних символів в аргументі <фрагмент> зліва в аргументі <рядок> необхідно вказати LEADING, а праворуч – TRAILING (рис. 7.22).

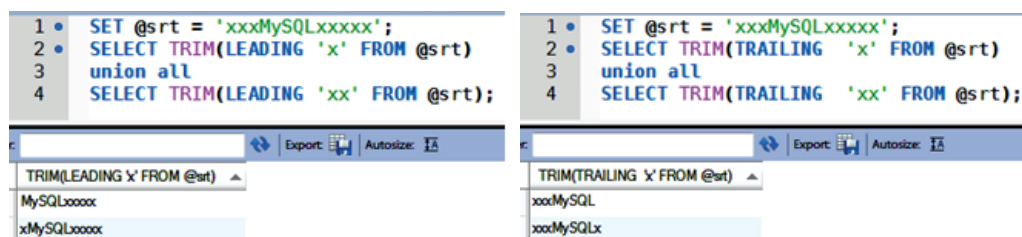


Рис. 7.22. Використання функції TRIM з аргументами LEADING або TRAILING

Якщо в аргументі <фрагмент> вказана деяка кількість символів, тоді усувається їх кратна кількість.

На відміну від видалення символів, ряд функцій MySQL доповнюють рядок заданою кількістю символів або фрагментів.

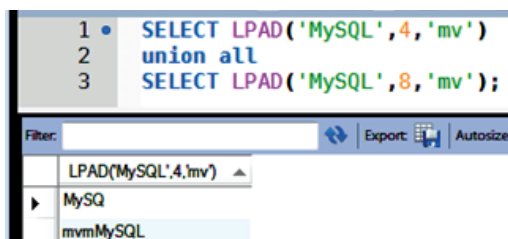
Формат запису функцій доповнення рядків:

- **SPACE(<len>)** – заповнює рядок з <len> пробілів;

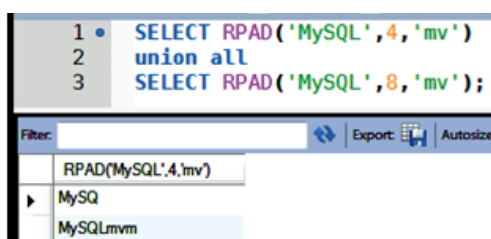
- **LPAD(<рядок>,<len>,<рядок доповнення>)** – дозаповнює <рядок> зліва значеннями <рядок доповнення> до вказаної кількості символів <len>;
- **RPAD(<рядок>,<len>,<рядок доповнення>)** – дозаповнює <рядок> праворуч значеннями <рядок доповнення> до вказаної кількості символів <len>;
- **REPEAT(<рядок>,<кількість повторень>)** – дозаповнює вихідний рядок аргументами <рядок> вказаною <кількістю повторень>.

Якщо будь-який з аргументів цих функцій рівний NULL, тоді результатом також буде NULL.

Якщо для функцій LPAD та RPAD аргумент <рядок> довший, чим <len>, тоді <рядок> буде вкорочено до довжини <len>. Якщо ж аргумент <рядок доповнення> не поміщається кратно кількість раз для отримання довжини <len> дозаповненого рядка, тоді останнє дозаповнення вкорочується так, щоб отримати необхідну кількість символів у вихідному рядку (рис. 7.23).



а)



б)

Рис. 7.23. Використання функцій: а) LPAD; б) RPAD

Якщо для функції REPEAT аргумент <кількість повторень> рівний нулю, тоді результатом буде порожній рядок, інакше – рядок з <кількістю повторень> аргументу <рядок> (рис. 7.24).

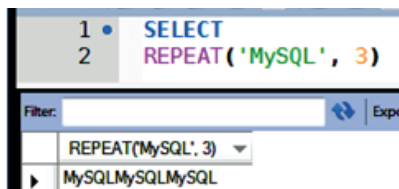
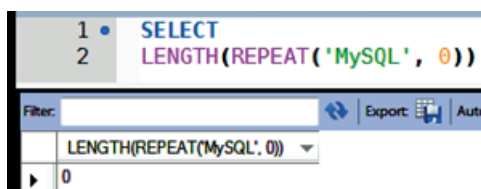


Рис. 7.24. Використання функції REPEAT

Крім видалення і доповнення рядків символами, СКБД MySQL надає змогу замінювати частину рядка іншими символами.

Формат запису функцій заміни фрагментів рядків:

- **INSERT(<рядок>, <номер позиції>, <len>, <рядок заміни>)** – замінює символи у <рядку> з позиції <номер позиції> значеннями <рядок заміни> вказаною кількістю символів <len>;
- **REPLACE(<рядок>, <що>, <рядок заміни>)** – замінює у <рядку> символи <що> на <рядок заміни>.

Якщо будь-який з аргументів цих функцій рівний NULL, тоді результатом також буде NULL.

Якщо для функції INSERT в аргументі <len> вказано менше символів, чим є в <рядку заміни>, тоді у вихідному рядку підлягає заміні тільки вказана кількість символів на <рядок заміни>. В цьому разі довжина вихідного рядка буде більшою за довжину вхідного на різницю між довжиною <рядка заміни> та значенням <len>. Якщо ж в аргументі <len> вказано більше символів, чим в <рядку заміни>, тоді у вихідному рядку місця для символів, яких немає у <рядку заміни>, вилучаються. В цьому випадку довжина вихідного рядка буде менша за довжину вхідного на різницю між значенням <len> та довжиною <рядка заміни> (рис. 7.25).

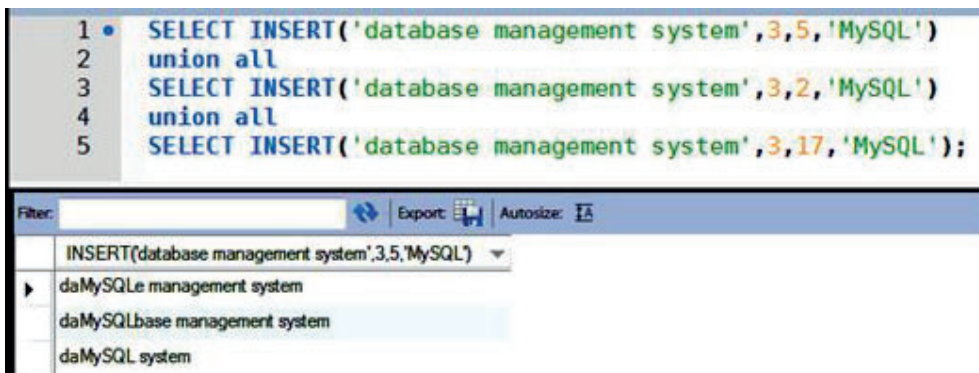


Рис. 7.25. Використання функції INSERT

Використання функції REPLACE наведено на рис. 7.26.

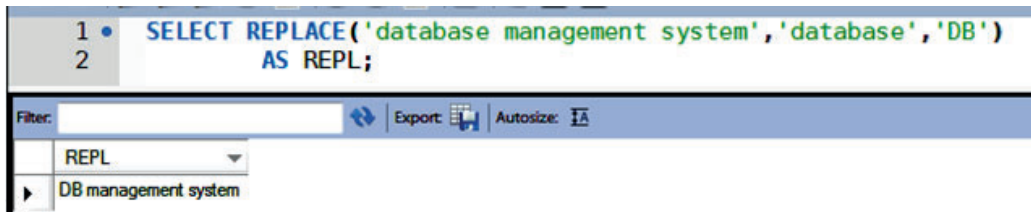


Рис. 7.26. Використання функції REPLACE

Перевертає рядок у зворотному порядку функція REVERSE (рис. 7.27):

- **REVERSE (<рядок>).**

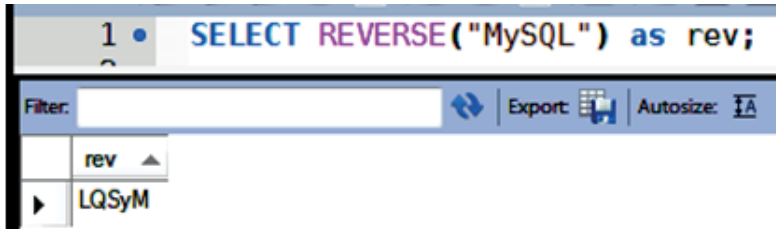


Рис. 7.27. Використання функції REVERSE

Для об'єднання декількох рядків в один в MySQL використовуються функції CONCAT та CONCAT\_WS, які мають формат запису:

- **CONCAT(<рядок1>, <рядок2>,...);**
- **CONCAT\_WS(<separator>, <рядок1>, <рядок2>,...).**

Результатом використання функції CONCAT буде рядок, «склеєний» з аргументів <рядок1>, <рядок2>,... Результатом CONCAT\_WS – також рядок, «склеєний» з аргументів <рядок1>, <рядок2>,... за допомогою розділювача <separator> (рис. 7.28).

Розділювачом може бути як один символ, записаний як рядок у подвійних лапках, так і рядок символів.

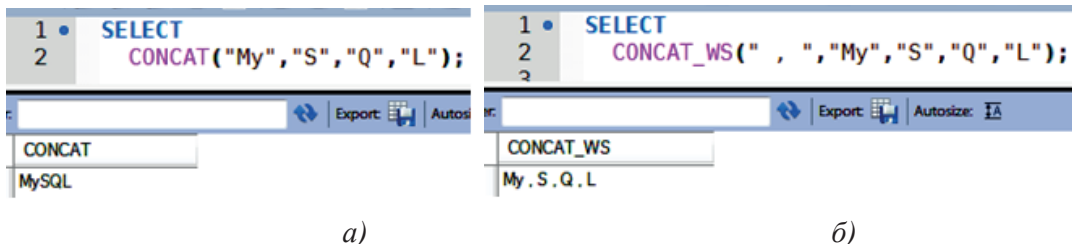


Рис. 7.28. Використання функцій: а) CONCAT; б) CONCAT\_WS

### 7.4.3. Пошук та видобування частини рядка

Функції пошуку значень у рядках в MySQL можна розділити на дві групи:

- які повертають позицію входження;
- які повертають частину рядка (підрядок).

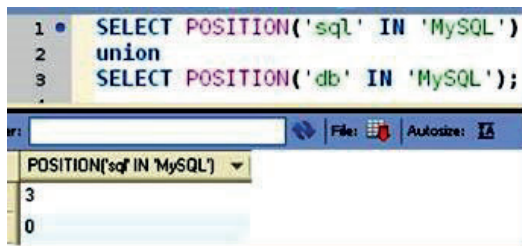
Формат запису функцій пошуку позиції входження:

- **POSITION(<рядок пошуку> IN <рядок>)** – повертає позицію першого підрядка <рядок пошуку> в аргументі <рядок>;
- **INSTR(<рядок>, <рядок пошуку>)** – повертає позицію першого входження підрядка <рядок пошуку> в аргументі <рядок> аналогічно функції POSITION з двома аргументами, за винятком зміни місць аргументів;
- **LOCATE(<рядок пошуку>, <рядок>)** – повертає позицію першого входження підрядка <рядок пошуку> в аргументі <рядок>;
- **LOCATE(<рядок пошуку>, <рядок>, <позиція>)** – повертає позицію першого входження підрядка <рядок пошуку> в аргументі <рядок>, але починаючи зі вказаної <позиції>;
- **FIND\_IN\_SET(<рядок пошуку>, <список>)** – шукає в аргументі <список>, який містить розділені комами рядки, аргумент <рядок пошуку> і повертає положення індексу відповідності;
- **FIELD(<рядок пошуку>, <рядок1>, <рядок2>,...<рядокN>)** – шукає в аргументах <рядок1>, <рядок2>,...<рядокN> рядок <рядок пошуку> і повертає індекс елемента, який відповідає критерію пошуку.

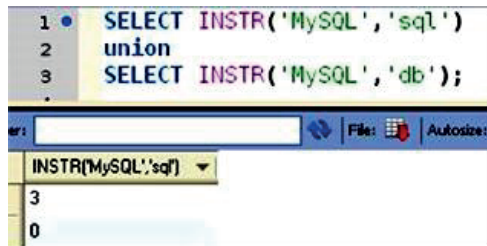
Якщо будь-який з аргументів цих функцій рівний NULL, тоді результатом також буде NULL.

Для того, щоб визначити, з якої позиції деякий фрагмент <рядок пошуку> знаходиться у <рядку>, застосовуються функції POSITION, INSTR або LOCATE.

Функції POSITION та INSTR відрізняються послідовністю розміщення аргументів (рис. 7.29).



а)



б)

Рис. 7.29. Використання функцій: а) POSITION; б) INSTR

Якщо фрагмент <рядок пошуку> відсутній у <рядку>, функції POSITION та INSTR повертають 0 позицію, якщо ж аргумент <рядок> розпочинається з фрагменту <рядок пошуку>, тоді – 1.

На відміну від POSITION та INSTR, функція LOCATE надає змогу здійснювати пошук позиції розміщення фрагменту <рядок пошуку> в аргументі <рядок> зі вказаної позиції в аргументі <позиція>. Якщо опустити аргумент <позиція>, тоді функція LOCATE працює аналогічно функціям POSITION та INSTR (рис. 7.30).

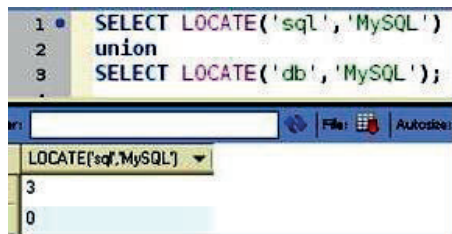
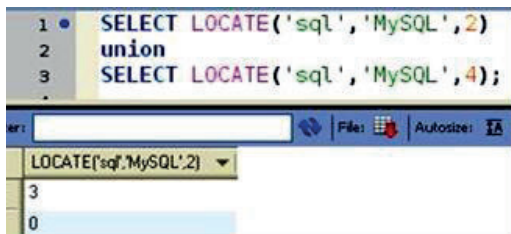


Рис. 7.30. Використання функції LOCATE

Якщо необхідно здійснити пошук позиції входження деякого рядка у списку, розділеного комами, використовується функція FIND\_IN\_SET (рис. 7.31).

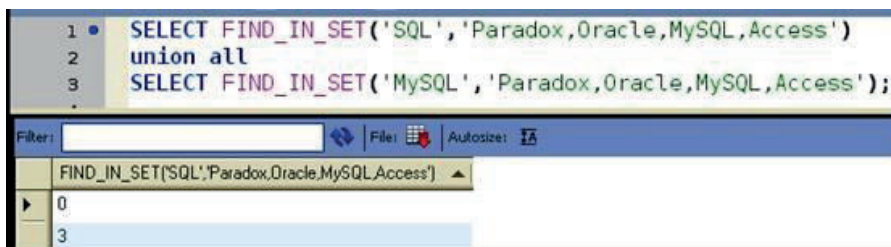


Рис. 7.31. Використання функції FIND\_IN\_SET

Пошук буде вдалим, якщо аргумент <рядок пошуку> повністю співпадає з елементом списку, виділеного комами. Якщо в аргументі <список> після ком



проставлено пробіли, тоді для коректного пошуку необхідно поставити пробіл на початку в аргументі <рядок пошуку>.

Здійснити пошук розташування деякого аргументу <рядок пошуку> в декількох рядках надає змогу функція FIELD (рис. 7.32).

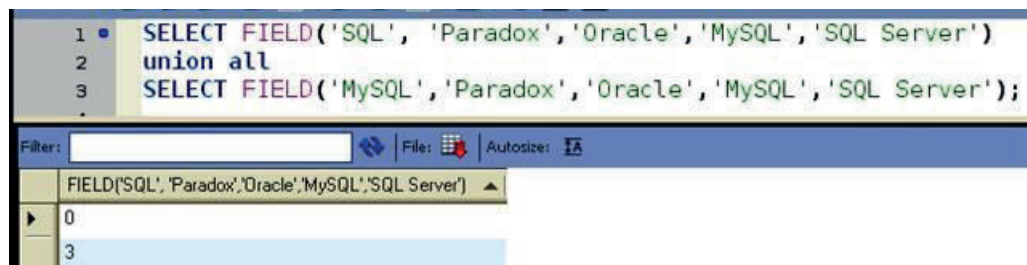


Рис. 7.32. Використання функції FIELD

Результатом пошуку буде значення позиції у переліку аргументів <рядок1>, <рядок2>, ... <рядокN> тільки в тому випадку, якщо аргумент <рядок пошуку> повністю співпадає з одним із цих аргументів.

Формат запису функцій, які повертають частину рядка (підрядок):

- **LEFT(<рядок>, <len>)** – повертає <len> лівих символів аргументу <рядок>;
- **RIGHT(<рядок>, <len>)** – повертає <len> правих символів аргументу <рядок>;
- **MID(<рядок>, <позиція>, <len>)** – повертає частину рядка (підрядок) з аргументу <рядок> довжиною <len> символів із вказаної <позиції>;
- **SUBSTRING(<рядок>, <позиція>, <len>)** – повертає частину рядка (підрядок) аргументу <рядок> довжиною <len> символів із вказаної <позиції>;
- **SUBSTRING\_INDEX(<рядок>, <separator>, <count>)** – повертає частину рядка (підрядок) з аргументу <рядок> перед появленням <count> входжень розділювача <separator>.

Якщо будь-який з аргументів цих функцій рівний NULL, тоді результатом також буде NULL.

Для виокремлення частини рядка спочатку або з кінця на вказану кількість символів використовуються функції LEFT та RIGHT (рис. 7.33).





Рис. 7.33. Використання функцій: а) LEFT; б) RIGHT

Для виокремлення довільної частини рядка на вказану кількість символів необхідно використовувати функції MID або SUBSTRING. Функція MID є тільки синонімом для SUBSTRING. Основна різниця полягає в тому, як вони індексують рядки символів.

У записі цих функцій можна опускати аргумент <len>, тоді результатом буде підрядок, починаючи із вказаної <позиції> і до кінця аргументу <рядок> (рис. 7.34).

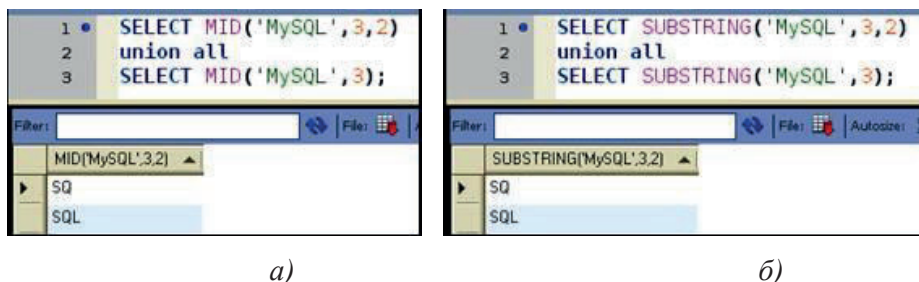


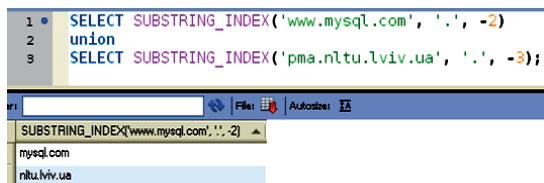
Рис. 7.34. Використання функцій: а) MID; б) SUBSTRING

Якщо <рядок> є списком елементів, між якими є розділювачі <separator>, тоді для пошуку частини рядка між цими розділювачами використовується функція SUBSTRING\_INDEX, яка містить три аргументи – власне <рядок>, у якому ведеться пошук, тип розділювача між елементами списку <separator> та порядковий номер розділювача <count>.

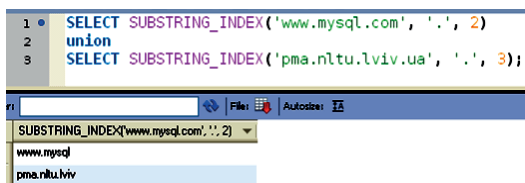
Якщо аргумент <count> рівний нулю, тоді функція SUBSTRING\_INDEX повертає порожній рядок (не NULL!).

Якщо аргумент <count> менший за нуль, тоді функція SUBSTRING\_INDEX повертає останніх <count> елементів рядка. (рис. 7.35, а).

Якщо ж аргумент <count> більший за нуль, тоді функція SUBSTRING\_INDEX повертає перших <count> елементів рядка. (рис. 7.35, б).



а)



б)

Рис. 7.35. Використання функції `SUBSTRING_INDEX` з: а) від'ємним аргументом `<count>`; б) додатнім аргументом `<count>`

#### 7.4.4. Функції порівняння рядків

Логічний оператор порівняння рядків повертає або 1 (істину), якщо рядки ідентичні (регістр та пробіли в кінці рядка не враховуються), або 0 (хибу), якщо рядки відрізняються значеннями (рис. 7.36).

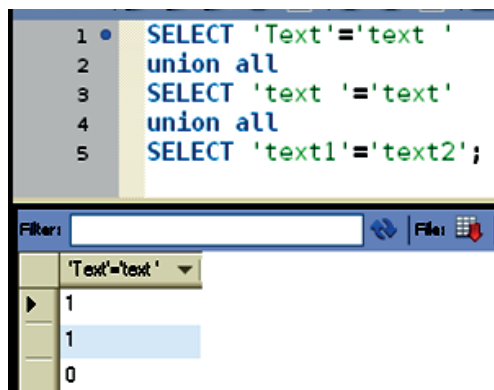


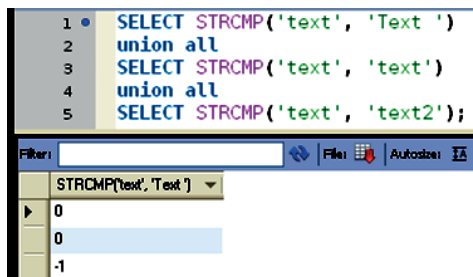
Рис. 7.36. Використання оператора порівняння рядків

Для порівняння рядків можна також використовувати функцію `STRCMP`, яка порівнює рядки, задані аргументами `<рядок1>` та `<рядок2>`:

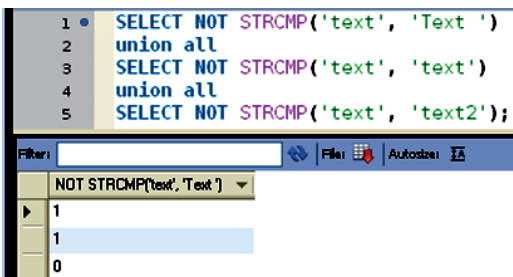
- **`STRCMP(<рядок1>, <рядок2>)`.**

Особливістю цієї функції є те, що у випадку рівності аргументів `<рядок1>` та `<рядок2>` результатом буде 0 (регістр та пробіли в кінці рядка не враховуються), якщо ж `<рядок1>` менший за `<рядок2>` – тоді -1 (мінус один), і 1 в решти випадках (рис. 37, а).

Для перетворення результату функції `STRCMP` в аналогічний оператору порівняння рядків, необхідно використовувати заперечення `NOT` (рис. 37, б).



а)



б)

Рис. 7.37. Використання функції STRCMP

СКБД MySQL дозволяє використовувати ще один спосіб порівняння рядків – за шаблоном. Такий спосіб вже розглянуто раніше, при розгляді задавання умови WHERE за допомогою оператора LIKE. Розглянемо ще ряд функцій, які також реалізують порівняння рядків за допомогою шаблону:

- <рядок> REGEXP <шаблон пошуку>

або

- <рядок> RLIKE <шаблон пошуку>.

Функція RLIKE є синонімом для REGEXP, вона передбачена для сумісності з MySQL. Обидві функції повертають 1, якщо <шаблону пошуку> є частиною аргументу <рядок>, в інших випадках – 0 (рис. 7.38), при цьому не враховується регістр символів.

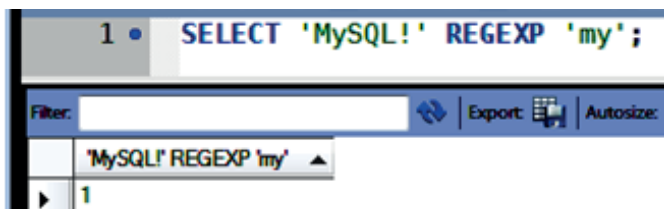


Рис. 7.38. Використання функцій:

Аргумент <шаблон пошуку> може бути розширеним регулярним виразом і включати один або декілька символів підстановки (рис. 7.39).

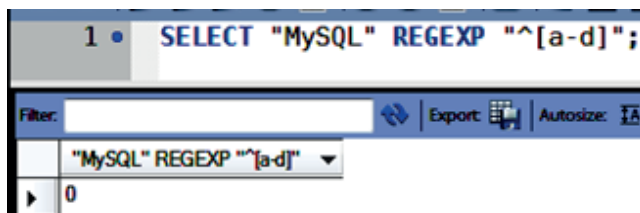


Рис. 7.39. Використання шаблонів у функціях:

#### 7.4.5. Чутливість до регістру

Зазвичай при порівнянні рядкових змінних регістр не враховується. Якщо ж необхідно розрізнити у рядкових змінних регістр, тоді необхідно використовувати функцію BINARY (рис. 7.40).

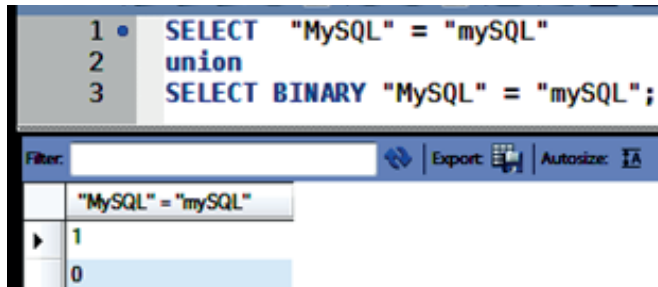


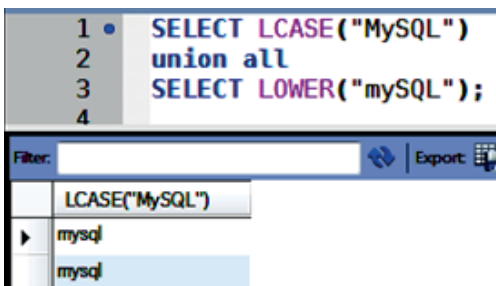
Рис. 7.40. Порівняння рядкових змінних, записаних у різних регістрах

Функція BINARY перетворює рядок, записаний за нею, у двійкові дані.

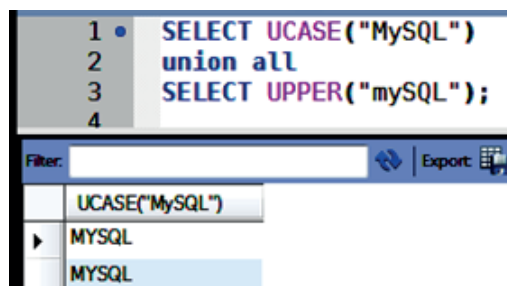
Для перетворення рядків у верхній або нижній регістри використовуються функції (рис. 7.41)

- **LCASE(<рядок>)** або **LOWER(<рядок>)** – повертають аргумент <рядок>, замінивши в ній всі великі букви на малі;
- **UCASE(<рядок>)** або **UPPER(<рядок>)** – повертають аргумент <рядок>, замінивши в ній всі малі букви на великі.

Але зазвичай у БД використовується підхід, коли всі дані вносяться в одному з регістрів, зазвичай, верхньому. Прикладами можуть бути БД «Пенсійний фонд», «Зарплата-Тріола» тощо.



а)



б)

Рис. 7.41. Використання функцій: а) LCASE та LOWER; б) UCASE та UPPER

## 7.5. Системні функції

Поряд з розглянутими функціями, в MySQL є ще багато інших.

Окремий розділ представляють функції для адміністрування сервера та БД, які розглянемо в подальшому викладенні матеріалу.

Крім того, корисними в багатьох випадках можуть бути системні функції:

- **USE <DATABASE>** – задає вказану БД як за замовчуванням. Це дозволяє при звертанні до таблиць уникати складних імен, таких як <ім'я БД>.<ім'я таблиці>;
- **DATABASE()** – повертає назву активної БД. Якщо в поточний момент немає активної БД, результатом буде порожній рядок;
- **USER(), SYSTEM\_USER(), SESSION\_USER()** – функції тотожні та повертають ім'я поточного активного користувача БД;
- **VERSION()** – повертає рядок з номером версії сервера MySQL. Може бути корисна для уточнення синтаксису SQL команд;
- **CONNECTION\_ID()** – повертає ідентифікатор для даного з'єднання. Кожне з'єднання має власний унікальний ідентифікатор.
- **INET\_ATON("<IP-адрес>")** – повертає ціле число, аналог числового значення даного адреса (SELECT INET\_ATON("172.16.2.250") → 2886730490). У такому форматі IP-адрес зручніше зберігати в БД та обробляти;
- **INET\_NTOA(<числове значення IP адреса> )** – функція зворотна INET\_ATON та повертає IP-адрес за його числовим представленням (SELECT INET\_NTOA(2886730490) → 172.16.2.250);
- **DEFAULT(<ім'я поля>)** – повертає значення за замовчуванням для поля таблиці (SELECT DEFAULT(Ido) FROM Osoba). Якщо ж під час створення таблиці не вказано значення за замовчуванням для поля, тоді виникає помилка.

## Контрольні запитання

1. Як оголосити локальну змінну в MySQL?
2. Які бувають типи змінних в MySQL?
3. Якою командою переглянути повний список системних змінних?
4. Чи можна перевіряти значення на рівність NULL за допомогою звичайних арифметичних операторів порівняння?
5. Під час порівняння чи NULL дорівнює NULL?
6. Які є функції в MySQL щоб визначити, чи величина дорівнює NULL?
7. Які функції використовуються для перевірки результату на значення NULL?
8. Які оператори керування можна використовувати у запитах типу SELECT в якості обчислювальних полів?
9. Які форми запису має оператор CASE? Наведіть приклади.
10. Які функції в MySQL дозволяють отримувати різні види підсумкової інформації? Охарактеризуйте їх.
11. Як за допомогою функції RAND() вивести випадкові числа в діапазоні від 2 до 10?
12. Які функції використовуються для отримання поточного значення дати і часу?
13. Для отримання року з календарних значень дати які використовуються функції перетворення?
14. Як представити дату у форматі DD-MM-YYYY?
15. Яке значення століття зі скороченим відображенням року приймає значення 75 року? А 65 року?
16. Які функції здійснюють арифметичні дії над датою і часом?
17. Чи може приймати від'ємне значення аргумент INTERVAL у функції DATE\_ADD? Якщо так, тоді який буде результат?
18. Яка функція визначає довжину рядка у символах?
19. Як відсікти зайві пробіли на початку і в кінці рядка?
20. Проаналізуйте, як відрізняються функції INSERT та REPLACE.
21. Як в MySQL об'єднати декілька рядків в один?

22. Які в MySQL є функції пошуку позиції входження в рядок?
23. Як в MySQL видобути частину рядка?
24. Які функції реалізують порівняння рядків за допомогою шаблону?
25. Які функції призначені для перетворення рядків у верхній або нижній регістри?

## Тема 8. Використання підзапитів в MySQL

### 8.1. Визначення підзапитів

*Підзапит* – це запит типу SELECT, вкладений в іншу команду SQL. При цьому підзапит (внутрішній запит) генерує значення, яке перевіряється в предикаті зовнішнього запиту. Виконання підзапитів завжди починається із *внутрішнього запиту*, а результати передаються у *зовнішній запит*, якщо лише не є корельованими.

Існує два типи обробки підзапитів – некорельований і корельований.

Корельований (залежний, зв'язаний) підзапит – обчислення якого залежить від процесу обчислення в зовнішньому запиті. Такі запити обробляються системою в зворотному порядку. Спочатку вибирається поточний рядок з таблиці зовнішнього запиту і на підставі значень її полів виконується обчислення підзапиту (тобто в умові обчислення підзапиту присутні значення полів з зовнішнього запиту).

Як корельовані, так і некорельовані підзапити бувають трьох типів, залежно від елементів у пропозиції WHERE зовнішнього запиту:

- 1) підзапити, що не повертають жодного чи повертають кілька елементів (починаються з IN або з оператора порівняння, містять ключові слова ANY чи ALL);
- 2) підзапити, що повертають єдине значення (починаються з простого оператора порівняння);
- 3) підзапити, що є тестом на існування (починаються з EXISTS).

Підзапити можуть використовуватися в командах SELECT, UPDATE, INSERT, DELETE, CREATE TABLE. Наприклад, кожна команда SELECT може включати декілька інших команд SELECT.

У команді SELECT підзапит може розміщуватись в переліку полів запиту як обчислювальне поле, в пропозиціях FROM, WHERE та HAVING. Підзапит може також містити групи і групові функції.



Вкладені запити можуть знаходитися практично в будь-якій частині SQL команди – у списку цілей, в секції WHERE і т. д.

### **Правила складання підзапитів:**

1. Підзапит завжди повинен міститись у круглих дужках;
2. Підзапит може звертатися до таблиць відмінних від тих, до яких звертається основний запит.
3. Підзапит може задаватися в складних критеріях пошуку зовнішніх запитів з використанням логічних зв'язків AND і OR та операторів ANY (SOME), ALL.
4. Не можна використовувати операцію BETWEEN стосовно підзапиту.
5. Пропозиція ORDER BY ставиться останньою в основному запиті і не може міститися в підзапиті.
6. Імена стовпців в запиті SELECT внутрішнього запиту повинні стояти в тій же послідовності, що і імена стовпців в лівій частині оператора порівняння зовнішнього запиту. Типи стовпців повинні попарно відповідати.

## **8.2. Використання підзапитів в якості розрахункових полів**

Розрахункові поля зазвичай описуються в переліку полів оператора SELECT перед пропозицією FROM. Такі поля можуть бути також результатом виконання іншого запиту (підзапиту).

**Приклад 8.1.** Відобразимо, наприклад, кількість прочитаних книг кожним абонентом:

```
SELECT PIPO,  
      (SELECT COUNT(IdKn) FROM formyljar f  
       WHERE o.IdO = f.IdOs) AS KILK  
FROM Osoba o;
```

Перший оператор SELECT відображає два стовпці – PIPO та KILK. Поле KILK є розрахунковим, воно формується в результаті виконання підзапиту, який взятий в круглі дужки. Цей підзапит виконується по одному разу для кожного запису в полі PIP та в загальному буде виконаний стільки раз, скільки є читачів в таблиці Osoba.

Речення WHERE в підзапиті виконує функцію поєднання, оскільки за допомогою WHERE ми з'єднали дві таблиці за полями IdO та IdOs, використавши псевдоніми таблиць (*Псевдонім\_таблиці.Поле*).

### 8.3. Підзапити в реченні FROM

У ряді випадків нам треба порівняти результати агрегації рядків, тобто підзапит формує таблицю, у якій основний запит виконує деяку агрегативну функцію. Це означає, що після речення FROM основного запиту поміщається віртуальна таблиця, яка містить результати вибірки підзапиту типу SELECT.

Таблиця, яка формується підзапитом, повинна обов'язково мати аліас (псевдонім)!

**Приклад 8.2.** Підрахувати кількість боржників у бібліотеці.

```
SELECT COUNT(IdOs) as KILK_BORZ FROM
    (SELECT DISTINCT IdOs
     FROM formyljar
     WHERE (dat_pov > NOW())
     OR (dat_pov IS NULL) ) Z ;
```

### 8.4. Вкладений запит в умові WHERE

SQL допускає вкладання допоміжного запиту в основний. При цьому допоміжний запит береться в дужки і виконується в першу чергу. Зовнішній запит оперує результуючою таблицею допоміжного запиту.

Використовується, коли внутрішній запит повертає 1 запис.

**Приклад 8.3.** Вивести дані про особу, яка позичила в бібліотеці книгу автора Аркаса «Історія України-Русі», 1912 р.

```
SELECT * FROM formyljar f WHERE f.IdKn =
    (SELECT k.IdK
     FROM Katalog k
     WHERE (k.AvtorK LIKE '%Аркас%')
     AND (k.NameK LIKE '%Історія України%' )
     AND (k.RikVydK=1912)
```

**Приклад 8.4.** Вивести дані про найдавнішу книгу.

```
SELECT * FROM Katalog k
WHERE k.RikVydK =
      (SELECT MIN(k1.RikVydK) FROM Katalog k1);
```

## 8.5. Вкладений запит в умові HAVING

Підзапити можна використовувати для відбору груп за умовою HAVING. Умова HAVING в запиті SELECT розміщується після речення [GROUP BY список полів групи] та перед реченням [ORDER BY список полів сортування], звичайно, якщо вони присутні в ньому запиті. **Варто згадати, що послідовність запису речень в команді SELECT є строго визначеною і змінювати її не можна.** Речення HAVING аналог WHERE і обробляється останнім, безпосередньо перед відсиланням даних клієнту, і без будь-якої оптимізації. Не використовуйте цей вираз для визначення того, що повинно бути визначено в умові WHERE.

**Приклад 8.5.** Вивести прізвища читачів, які протягом попереднього року взяли в бібліотеці більше книг, ніж в середньому по університету.

Перш за все, попередній рік дати видачі книги визначається як:

```
YEAR(NOW())-1
```

По-друге, взято книг в середньому по університету у попередньому році обчислюється як загальна кількість книг, що взято у цьому році, поділена на загальну кількість читачів у цьому є році:

```
(SELECT COUNT(idK) FROM Formujiar WHERE
YEAR(dat_vud)=YEAR(NOW())-1) /
(SELECT COUNT(idO) FROM (SELECT DISTINCT idO
FROM Formujiar WHERE YEAR(dat_vud)=YEAR(NOW())-1) z);
```

Таким чином, запит буде наступним:

```
SELECT Os.PIPO, COUNT(f.idK)
FROM Osoba Os INNER JOIN Formujiar f
ON f.idO=Os.idO
WHERE YEAR(dat_vud)=YEAR(NOW())-1
```

```

GROUP BY PIPO
HAVING COUNT(f.idK) >
(SELECT COUNT(idK)
FROM Formujjar WHERE YEAR(dat_vud)=YEAR(NOW())-1) /
(SELECT COUNT(idO) FROM (SELECT DISTINCT idO
FROM Formujjar WHERE YEAR(dat_vud)=YEAR(NOW())-1) z);

```

## 8.6. Оператор EXIST у вкладених запитах

Оператор EXIST повертає істинне значення, якщо множина значень, яку генерує підзапит, непуста. Виконання оператора EXIST зупиняється при появі першого запису у підзапиті, тим самим економлячи обчислювальні витрати. При цьому в підзапиті використовувати групування не можна.

**Приклад 8.6.** Вивести дані про читачів, що заборгувала книги (умовою боргу вважати порожнє поле дати повернення).

```

SELECT * FROM Osoba o
WHERE EXISTS
(SELECT f.idO FROM Formujjar f
WHERE (o.IdO = f.idO)
AND (f.dat_pov IS NULL));

```

## 8.7. Підзапити з операторами IN, ANY, ALL

Якщо внутрішній запит повертає більш ніж один запис, тоді для формування умови у зовнішньому запиті використовуємо оператори вибору із списку значень.

1. Оператори IN, ANY істинні, коли є співпадання хоча б із одним значенням внутрішнього запиту, якому вони передують.
2. =NOT IN істинне, коли нема співпадання із жодним значенням, отриманим у запиті.
3. >ANY, >=ANY істинні коли значення більше або більше-рівне довільного значення внутрішнього запиту.
4. <ANY, <=ANY істинні коли значення менше або менше -рівне довільного значення внутрішнього запиту

5. =ALL істинний, коли значення рівне всім значенням внутрішнього запиту.  
Така ситуація може спостерігатися, коли внутрішній запит повертає лише одне значення.
6. ALL істинний, коли значення не рівне жодному значенню внутрішнього запиту.
7. >ALL, >=ALL істинні, коли значення більші або більші-рівні всіх значень внутрішнього запиту
8. <ALL, <=ALL істинні, коли значення менші або менші-рівні всіх значень внутрішнього запиту
9. Слід враховувати, що коли результат внутрішнього запиту пустий, то оператори (=, >, <) ALL приймають істинне значення, а аналогічні оператори із ANY – хибне.

**Приклад 8.7.** Вивести на екран назви про книги, що позичались протягом 2015 та 2016 років.

```
SELECT * FROM Katalog k
      WHERE k.IdK IN
      (SELECT f.IdK
      FROM Formujar f
      WHERE YEAR(f.dat_vud)=2015
      OR YEAR(f.dat_vud)=2016);
```

## **8.8. Підзапити в операторах маніпулювання даними**

### **8.8.1. Підзапити в операторі DELETE**

Оператор DELETE вилучає записи з таблиці або всі, або лише ті, що відповідають умові WHERE. Якщо ж в умові WHERE необхідно отримати попередньо дані з деякої таблиці, тоді використовується підзапит SELECT в операторі DELETE.

Базовий синтаксис підзапиту в операторі DELETE наступний:

```
DELETE FROM <ім'я_таблиці> WHERE <ім'я_стовпця> ОПЕРАЦІЯ  
(SELECT <ім'я_стовпця> FROM <ім'я_таблиці> [WHERE])
```

**Приклад 8.8.** Вилучити з таблиці Formular осіб, які були зареєстровані 3 роки тому і раніше. Дані про реєстрацію читачів знаходяться в таблиці Osoba. У підзапиті виділимо цих осіб за допомогою функції поточної дати NOW(), від якої віднімемо 3 роки та передамо їх у списку для операції IN оператора DELETE:

```
DELETE FROM Formular WHERE Id_ch IN  
      (SELECT o.Ido FROM Osoba o  
      WHERE o.DRo < (DATE(NOW()) - INTERVAL 3 YEAR))) ;
```

### 8.8.2. Підзапити в операторі UPDATE

Оператор UPDATE використовується для редагування записів таблиці БД значень вказаних полів для всіх записів або декількох, що задовольняють умові WHERE. Якщо ж в умові WHERE необхідно отримати попередньо дані з деякої таблиці, тоді використовується підзапит SELECT в операторі UPDATE.

Формат підзапиту в операторі UPDATE наступний:

```
UPDATE <ім'я_таблиці>  
      SET  
<ідентифікатор_поля>=<значення>,[ідентифікатор_поля=значення]  
      WHERE <ім'я_стовпця> ОПЕРАЦІЯ  
      (SELECT <ім'я_стовпця>  
      FROM <ім'я_таблиці> [WHERE])
```

**Приклад 8.9.** Збільшити удвічі кількість примірників книжок, які позичались протягом минулого року. Кількість книг вказана в таблиці Katalog, а інформація про те, чи позичались книги – в таблиці Formular. У підзапиті виділимо ці книги та передамо їх у списку для операції IN оператора UPDATE:

```
UPDATE Katalog k  
      SET k.KilkPrym = k.KilkPrym*2  
      WHERE k.Idk IN  
      (SELECT f.idK FROM Formujiar f  
      WHERE YEAR(f.dat_pov)=YEAR(now()) - INTERVAL 1 YEAR));
```

### 8.8.3. Підзапити в операторі INSERT

Оператор INSERT дозволяє вставляти у таблицю записи з інших таблиць. Для цього список значень замінюється підзапитом (командою SELECT) або збереженою процедурою. Як виняток за допомогою підзапиту у команді INSERT можна додавати записи в межах однієї таблиці.

Оператор INSERT використовує дані, що повертаються підзапитом, для додавання їх в таблицю. Вибрані в підзапиті дані можна модифікувати за допомогою символьних або числових функцій, а також функцій дати і часу.

Формат запису оператора INSERT з підзапитом:

```
INSERT INTO <ім'я_таблиці> [ (стовпець1 [,стовпець2 ...]) ]  
SELECT { *| стовпець1 [,стовпець2 ...]}  
FROM таблиця1 [, таблиця2 ] [WHERE <умова> ]
```

***На відміну від всіх решти підзапитів, у цьому випадку підзапит SELECT у круглі дужки не береться!***

Такий синтаксис дозволяє внести в таблицю велику кількість записів за один раз з різних таблиць.

Якщо поля таблиць в операторі INSERT та підзапиті SELECT співпадають, тоді записи просто додаються в кінець таблиці, в інших випадках вказані поля перекопійовуються, решта отримують значення за замовчуванням або NULL.

***Приклад 8.10.*** Необхідно доповнити у таблиці Osoba1 поля PIPo та DRo записами з таблиці Osoba:

```
INSERT INTO osoba1 (PIPo, DRo)  
SELECT PIPo, DRo FROM Osoba;
```

Після виконання цього запиту записи із таблиці Osoba перекопійовуються в поля PIPo та DRo таблиці Osoba1, а решта полів отримують значення NULL.

***Приклад 8.11.*** Якщо ж необхідно доповнити таблицю Osoba1 для всіх полів записами із таблиці Osoba з умовою, що читачі проживають у Львові, тоді в команді INSERT поля не вказуються:

```
INSERT INTO BIBLIO.Osoba1  
SELECT * FROM BIBLIO.Osoba  
WHERE ADRo LIKE "%Львів%";
```

Після виконання цього запиту тільки записи для читачів, що проживають у Львові, із таблиці Osoba перекопійовуються в таблицю Osoba1.

**Приклад 8.12.** Якщо ж необхідно доповнити таблицю Osoba1 записами із таблиці Osoba з умовою, що адреса проживання не заповнена, тоді:

```
INSERT INTO BIBLIO.Osoba1
SELECT * FROM BIBLIO.Osoba
WHERE ADRo IS NULL;
```

Необхідно відмітити, що підзапити в операторі INSERT можуть виконуватися *в межах однієї таблиці*.

**Приклад 8.13.** Необхідно доповнити таблицю osoba1 записами з таблиці Osoba, у яких Ido > 7:

```
INSERT INTO
osoba1 (PIPo, RNo, ADRo, DRo)
SELECT oc.PIPo, oc.RNo, oc.ADRo, oc.DRo
FROM Osoba oc
WHERE oc.Ido > 7;
```

В таких випадках поле первинного ключа, якщо воно AUTO\_INCREMENT, буде автоматично заповнюватися. Якщо ж необхідно заповнювати в тому числі і поле первинного ключа, тоді необхідно його запрограмувати деяким чином.

**Приклад 8.14.** Необхідно доповнити таблицю osoba1 записами, у яких RNo більший за 1990 та врахувати, що поле первинного ключа не є AUTO\_INCREMENT:

```
SET @a=(SELECT COUNT(Ido) FROM osoba1)+1;
INSERT INTO
osoba1 (Ido, PIPo, RNo, ADRo, DRo)
SELECT (oc.Ido+@a), oc.PIPo, oc.RNo, oc.ADRo, oc.DRo
FROM osoba1 oc
WHERE oc.RNo >1990;
```

Для заповнення поля первинного ключа у підзапитах INSERT...SELECT не тільки в межах однієї таблиці можна використовувати і інші підходи.



## 8.9. Корельовані підзапити

*Корельований підзапит* – обчислення якого залежить від процесу обчислення в *зовнішньому запиті*. Такі запити обробляються СКБД в зворотному порядку. Спочатку вибирається поточний рядок з таблиці зовнішнього запиту і на підставі його значень виконується обчислення підзапиту (тобто в умові обчислення підзапиту присутні значення полів з зовнішнього запиту).

Послідовність виконання корельованого підзапиту:

- зовнішній запит вибирає рядок;
- виконується внутрішній запит, використовуючи значення рядка зовнішнього запиту;
- результат виконання внутрішнього запиту повертається в зовнішній запит, де перевіряється його відповідність вибраному рядку;
- вибирається наступний рядок зовнішнього запиту.

При використанні вкладених запитів допускаються вживання операторів ANY, EXISTS, ALL і логічних операторів.

**Приклад 8.15.** Необхідно вивести прізвища бібліотекарів, які видали у користування книжки автора Cameron Hughes:

```
SELECT PIPBf
FROM Formular
WHERE 'Cameron Hughes' IN
      (SELECT AVk FROM Katalog
       WHERE ldk = Formular.Id_kn);
```

Так як внутрішній підзапит залежить від даних із зовнішнього запиту – Formular.Id\_kn, він не може бути оброблений раніше за зовнішній.

Виконання таких запитів здійснюється наступним чином.

1) СКБД перевіряє перший запис таблиці Formular, у якого, наприклад, Formular.Id\_kn має значення 21. Тоді система опрацьовує внутрішній запит (підзапит), де Formular.Id\_kn = 21:

```
SELECT AVk
FROM Katalog
WHERE ldk = 21;
```

отримуючи в результаті з таблиці Katalog прізвище автора для книги з Idk = 21. Тепер СКБД може завершити обробку першого запису таблиці Formular.

2) Надалі СКБД буде здійснювати опрацювання почергово наступних записів таблиці Formular для отримання значення Formular.Id\_kn і порівняння зі значеннями аналогічних Idk в таблиці Katalog, і так до останнього запису таблиці Formular.

**Приклад 8/16.** Необхідно вивести номери абонентів, у яких рік народження такий же, як і в абонента з номером 7:

```
SELECT * FROM Osoba
WHERE RNo IN
      (SELECT RNo FROM Osoba
       WHERE   IdO = 7);
```

Іноді користувач може знати, що підзапит повинен повернути тільки одне значення. У такому випадку замість оператора IN можна використовувати інші оператори порівняння (наприклад, =, > і т. д.). І навіть якщо підзапит не поверне взагалі жодного значення, помилка не виникне. Але, якщо підзапит повертає більше одного значення і не використовується оператор IN, буде виникати помилка. Більш того, підзапит не може включати фразу GROUP BY, якщо він використовується з простим оператором порівняння, наприклад з =, > і т. д.

### Контрольні запитання

1. Що таке підзапит?
2. У яких командах SQL можуть бути використані підзапити?
3. Який порядок виконання підзапиту?
4. Коли неможливо використовувати підзапит?
5. Який порядок виконання корельованих підзапитів?
6. У яких випадках підзапити можуть передати декілька рядків в головну пропозицію?
7. Скільки разів виконується основний запит при реалізації корельованого підзапиту?

8. Де розташований підзапит в логічному вираженні пропозиції основного запиту: до оператора порівняння, після оператора порівняння?
9. Які пропозиції допустимі для основного запиту не можуть використовувати в підзапиті?
10. Які існують типи підзапитів?
11. У яких командах SQL можуть бути використані підзапити?
12. У яких пропозиціях SELECT можуть бути використані підзапити?
13. Які пропозиції SELECT не можна використовувати в підзапитах?
14. Де повинен знаходитися підзапит?
15. Які оператори порівняння працюють з декількома рядками?
16. Які оператори порівняння працюють лише з одним рядком?
17. У яких підзапитах введена команда SELECT виконується першою?
18. Які підзапити називаються однорядковими?
19. Який принцип роботи корельованого підзапиту?

# Розділ 3. OLTP ТА OLAP СИСТЕМИ

## Тема 9. Збережені підпрограми. OLTP технологія

### 9.1.Збережені процедури і функції на стороні сервера

#### 9.2.3. Поняття збережених процедур і функцій

В MySQL на стороні сервера можна створювати та зберігати підпрограми – процедури та функції. Під терміном «збережена процедура» розуміється послідовність операцій, що зберігається як єдине ціле в каталозі бази даних на SQL-сервері.

*Збережена процедура* – це спосіб інкапсуляції повторюваних дій. У збережених процедурах можна оголошувати змінні, управляти потоками даних, а також застосовувати інші техніки програмування.

Збережена процедура може бути корисна, коли одну і ту ж підпрограму необхідно викликати з різних додатків або з різних платформ або як засіб інкапсуляції функціональних можливостей. Збережені процедури в базах даних можна вважати аналогом об'єктно-орієнтованого підходу у програмуванні. Вони дозволяють керувати способом доступу до даних.

*Збережена підпрограма* є процедурою або функцією. Процедура викликається за допомогою інструкції CALL і може передавати аргументи як в тіло підпрограми, так і повертати зворотні значення, використовуючи змінні виведення. *Збережена функція* ж може повертати лише скалярне значення. Збережені підпрограми, зрозуміло, можуть викликати інші збережені підпрограми.

Переваги використання збережених підпрограм:

1. Поділ логіки з іншими додатками. Збережені процедури інкапсулюють функціональність, це забезпечує зв'язність доступу до даних і управління ними між різними додатками. Збережені процедури запускаються у всіх середовищах, і немає необхідності перестроювання логіки, адже логіка

процедури міститься на сервері БД і не залежить від того, який додаток і яка мова програмування використовується для виклику процедури.

2. Ізоляція користувачів від таблиць бази даних. Це дозволяє давати доступ до збережених процедур, але не до самих даних таблиць. Таким чином, інші користувачі не в змозі стерти дані з БД через команду DELETE.
3. За рахунок виконання збережених процедур на стороні сервера досягається скорочення мережевого трафіку. Тобто у збережених процедурах може бути об'єднана доволі значна кількість запитів без необхідності відсилання проміжних результатів у застосунок користувача.

Поряд з перевагами, збережені процедури мають ряд недоліків:

1. У зв'язку з тим, що більша частина роботи виконується на серверній частині, а менша – на клієнтській, підвищується навантаження на сервер бази даних.
2. Відбувається дублювання логіки застосунку в двох місцях: серверний код і код для збережених процедур, тим самим ускладнюється процес маніпулювання даними.
3. При використанні декількох типів СКБД (DB2, SQL Server тощо) можуть виникати проблеми.

#### **9.2.4. Створення процедур і функцій та звертання до них**

Для виділення збереженої процедури або функції завжди використовують обмежувач, який задається командою DELIMITER.

**Обмежувач** – це символ або рядок символів, який використовується для вказівки MySQL, що підпрограма завершена.

Не рекомендується у якості обмежувача застосовувати «;» – символ завершення SQL команди, адже у підпрограмі може бути декілька команд, кожна з яких завершується тим же символом і це може призвести до неоднозначності закінчення підпрограми.

##### **9.2.4.1. Збережені процедури**

Для розроблення *збереженої процедури* використовується оператор CREATE PROCEDURE, який має таку структуру:

## CREATE PROCEDURE

```
< ім'я процедури> (опис аргументів)
    [LANGUAGE SQL]
    [DETERMINISTIC | NOT DETERMINISTIC]
    [{CONTAINS SQL | NO SQL | READS SQL DATA |
      MODIFIES SQL DATA}]
    [SQL SECURITY {DEFINER | INVOKER}]
    [COMMENT <коментар>]
    BEGIN
        <тіло процедури>;
    END;
```

Назви збережених процедур чутливі до регістру. Також не можна створювати декілька процедур з однаковою назвою. Усередині збереженої процедури не може бути виразів, що змінюють саму базу даних.

Опис вхідних чи вихідних аргументів задається так само, як і опис полів таблиці:

**<ім'я\_аргументу> <тип>.**

Крім того ім'я аргументу не повинне співпадати з іменами полів.

У збереженій процедурі аргументи, які передаються у підпрограму та аргументи, які повертаються нею, вказуються у круглих дужках у списку через кому за допомогою операторів IN (вхідні), OUT (вихідні) та INOUT (одночасно вхідні та вихідні):

- CREATE PROCEDURE proc() – порожній список аргументів;
- CREATE PROCEDURE proc(IN varname DATA-TYPE) – один вхідний аргумент. Слово IN необов'язкове, тому що аргументи за замовчуванням – IN (вхідні);
- CREATE PROCEDURE proc(OUT varname DATA-TYPE) – один вихідний аргумент;
- CREATE PROCEDURE proc(INOUT varname DATA-TYPE) – один аргумент, одночасно вхідний і вихідний.

Після оголошення вхідних і вихідних аргументів можуть задаватися наступні параметри:

- **LANGUAGE:** з метою забезпечення переносимості, за замовчуванням вказано SQL;
- **DETERMINISTIC:** якщо процедура весь час повертає один і той же результат, і приймає одні й ті ж вхідні аргументи. Це для реплікації і процесу реєстрації. Значення за замовчуванням – NOT DETERMINISTIC;
- Декілька характеристик забезпечують інформацію щодо характеру використання даних підпрограмою:
  - **CONTAINS SQL** вказує, що підпрограма не містить інструкції, які читають або записують дані;
  - **NO SQL** вказує, що підпрограма не містить ніяких інструкцій SQL;
  - **READS SQL DATA** вказує, що підпрограма містить інструкції, які читають дані, але не записують;
  - **MODIFIES SQL DATA** вказує, що підпрограма містить інструкції, які можуть записувати дані;
  - **CONTAINS SQL** значення за замовчуванням, якщо жодна з цих характеристик не дана явно.

Ці характеристики тільки консультативні. Сервер не використовує їх, щоб обмежити види інструкцій, які підпрограмі дозволено виконати.

- **SQL SECURITY:** під час виклику йде перевірка прав користувача: **INVOKER** – це користувач, що викликає збережену процедуру, **DEFINER** – це "творець" процедури. Значення за замовчуванням – **DEFINER**;
- **COMMENT:** коментар, значення за замовчуванням – "".

Тіло процедури записується на мові процедур та тригерів, а також може містити опрацювання винятків **EXCEPTION**. Якщо в тілі збереженої процедури використано оператор **SELECT**, тоді повертається лише один запис, множину записів повертають збережені процедури вибору.

Локальні змінні створюються і зберігаються у процедурах. Їх необхідно оголошувати на початку блоку **BEGIN/END** разом з їх типами даних.

Формат оголошення локальної змінної:

**DECLARE <ім'я\_змінної> <тип\_даних>**

**[DEFAULT <значення\_за\_замовчуванням>];**

Приклади оголошення локальних змінних:

DECLARE a, b INT DEFAULT 5;

DECLARE str VARCHAR(25);

DECLARE today TIMESTAMP DEFAULT CURRENT\_DATE;

Оголошеній змінній можна задати початкові значення за допомогою команди SET:

SET str = 'Hello, world!';

SET b = 45;

**Приклад 9.1.** Розробити процедуру, яка об'єднує два рядки через два тире за допомогою функції CONCAT (рис .9.1).

```
DELIMITER |
CREATE PROCEDURE procedure_CONCAT
    (a VARCHAR(5), INOUT b VARCHAR(15))
DETERMINISTIC
COMMENT 'Procedure CONCAT'
BEGIN
    SET b = CONCAT(a, "--", b);
    SELECT b AS RESULT;
END |
```

*Рис. 9.1. Створення процедури з функцією CONCAT*

Деякі пояснення до прикладу 9.1:

У першому рядку вказується обмежувач.

У другому рядку після оператора CREATE вказується ім'я БД за замовчуванням, ключове слово PROCEDURE, ім'я цієї процедури та її аргументи (імена і типи).

У 3-4 рядках вказуються додаткові параметри процедури.

У 5-8 рядках наведено, власне, визначення самої процедури між BEGIN...END.

**Щоб викликати збережену процедуру,** необхідно використати ключове слово CALL, а потім назву процедури, а в дужках вказати аргументи (змінні або значення). Дужки обов'язкові (рис. 9.2):



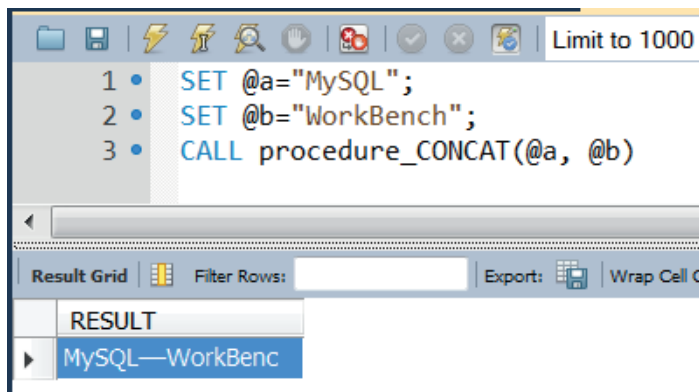


Рис. 9.2. Виклик збереженої процедури

**Приклад 9.2.** Розробити процедуру, яка додаватиме в таблицю Katalog нові записи. Така процедура матиме тільки вхідні аргументи:

```

DELIMITER |
CREATE PROCEDURE proc2
    (PIdk int, PAVk varchar(25),
    PNAMk varchar(25), PRVk int, PVYDk varchar(15),
    PNTOMk int, PKPRIMk int)
LANGUAGE SQL
DETERMINISTIC
SQL SECURITY DEFINER
COMMENT 'My procedure 2'
BEGIN
    INSERT INTO Katalog (ldk, AVk, NAMk, RVk, VYDk, NTOMk, KPRIMk)
    \VALUES
        (PIdk, PAVk, PNAMk, PRVk, PVYDk, PNTOMk, PKPRIMk);
END; |

```

Виклик процедури:

```
CALL proc2(111,'Pirko','Computer circuitry',2017,'NLTU',1,100)
```

або, попередньо задавши аргументи @a1-@a7:

```
CALL proc2(@a1,@a2,@a3,@a4,@a5,@a6,@a7)
```

**Приклад 9.3.** Розробити процедуру, яка здійснюватиме пошук авторів книг за заданою назвою книги або її фрагментом. Дана процедура матиме тільки один вхідний аргумент:

```
DELIMITER |  
CREATE PROCEDURE proc3(PNAMk VARCHAR(25))  
COMMENT 'My procedure 3'  
BEGIN  
    SELECT AVk, NAMk FROM Katalog  
        WHERE NAMk LIKE PNAMk;  
END; |
```

В межах збереженої процедури для управління потоками MySQL підтримує конструкції IF, CASE, ITERATE, LEAVE LOOP, WHILE і REPEAT. Серед них найбільш часто використовуються IF, CASE та WHILE.

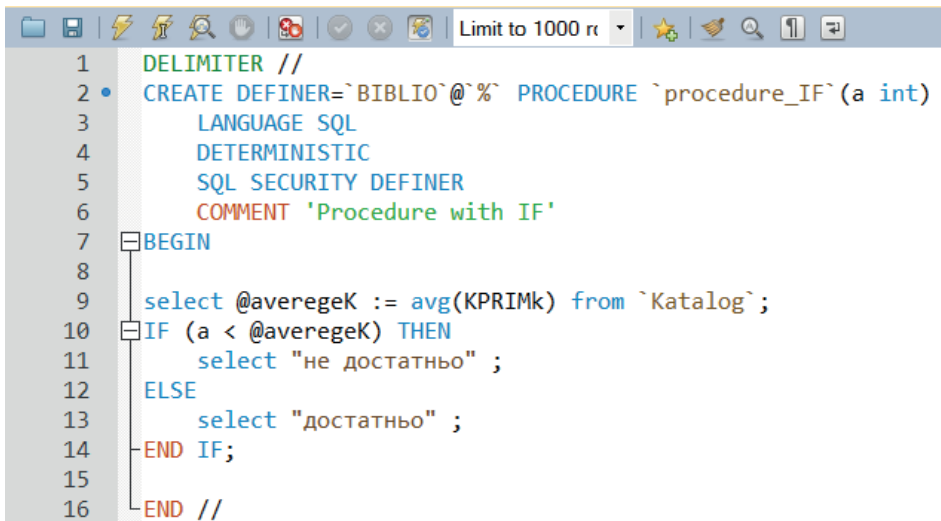
### Конструкція IF

Формат запису конструкції IF аналогічний форматам запису у більшості мов програмування:

```
IF <логічний вираз> THEN <команди>  
    [ELSEIF <логічний вираз> THEN <команди>]  
    ...  
    [ELSE <команди>]  
END IF
```

Необхідно зазначити, що команда IF відрізняється від функції IF(), яка була розглянута раніше. Використання команди IF можливе тільки в збережених процедурах і збережених функціях.

**Приклад 9.4.** Розробити процедуру, яка визначає за заданим числом N книг, достатньо чи не достатньо цієї кількості. Якщо число N менше за середню кількість книг у бібліотеці, тоді процедура виводить «не достатньо», інакше – «достатньо» (рис. 9.3).



```
1 DELIMITER //
2 • CREATE DEFINER=`BIBLIO`@`%` PROCEDURE `procedure_IF`(a int)
3     LANGUAGE SQL
4     DETERMINISTIC
5     SQL SECURITY DEFINER
6     COMMENT 'Procedure with IF'
7 BEGIN
8
9     select @averageK := avg(KPRIMk) from `Katalog`;
10 IF (a < @averageK) THEN
11     select "не достатньо" ;
12 ELSE
13     select "достатньо" ;
14 END IF;
15
16 END //
```

Рис. 9.3. Скрипт створення процедури з функцією IF

## Конструкція CASE

Конструкцію CASE можна описати двома способами, надаючи гнучкість в управлінні багатьох умовних виразів. Формат запису конструкції CASE:

**CASE <змінна вибору>**

**WHEN <значення змінної вибору> THEN <команди>**

**[WHEN <значення змінної вибору> THEN <команди>] ...**

**[ELSE <команди>]**

**END CASE**

або:

**CASE**

**WHEN <логічний вираз> THEN <команди>**

**[WHEN <логічний вираз> THEN <команди>] ...**

**[ELSE <команди>]**

**END CASE**

Варто зазначити, що синтаксис інструкції CASE для використання всередині збережених процедур і функцій дещо відрізняється від синтаксису в SQL. Інструкція CASE не може мати пропозицію ELSE NULL, і вона завершується END CASE замість END.

**Приклад 9.5.** Розробити процедуру, яка виводить різноманітні записи про читачів бібліотеки за вказаним курсом:

```
DELIMITER //
CREATE DEFINER=`BIBLIO`@`%`
    PROCEDURE `procedure_CASE1`(kurs INT)
    LANGUAGE SQL
    DETERMINISTIC
    SQL SECURITY DEFINER
    COMMENT 'Procedure with CASE змінна вибору'
BEGIN
CASE kurs
    WHEN 1
        THEN SELECT idO,PIPO,SpecialtyO
            FROM Osoba WHERE KursStaz=kurs;
    WHEN 2
        THEN SELECT idO,PIPO,DNO from Osoba where KursStaz=kurs;
    WHEN 3
        THEN SELECT idO, PIPO,AdresO from Osoba where
KursStaz=kurs;
    WHEN 4
        THEN SELECT idO,PIPO,DRO from Osoba where KursStaz=kurs;
    WHEN 5
        THEN SELECT idO,PIPO,PlaceWorkStudyO
            from Osoba where KursStaz=kurs;
    WHEN 6
        THEN SELECT idO,PIPO,Stat from Osoba where KursStaz=kurs;
    ELSE SELECT "Читач закінчив ВУЗ";
END CASE;
END //
```

**Приклад 9.6.** Аналогічна процедура прикладу 9.3 зі зміною формату запису конструкції CASE:

```
DELIMITER //
CREATE          DEFINER='BIBLIO'@'%`          PROCEDURE
`procedure_CASE2`(kurs INT)
    LANGUAGE SQL
    DETERMINISTIC
    SQL SECURITY DEFINER
    COMMENT 'Procedure with CASE логічний вираз'
BEGIN
CASE
    WHEN kurs=1
        THEN SELECT idO,PIPO,SpecialtyO
                FROM Osoba WHERE KursStaz=kurs;
    WHEN kurs=2
        THEN SELECT idO,PIPO,DNO from Osoba where KursStaz=kurs;
    WHEN kurs=3
        THEN  SELECT   idO,   PIPO,AdresO   from   Osoba   where
KursStaz=kurs;
    WHEN kurs=4
        THEN SELECT idO,PIPO,DRO from Osoba where KursStaz=kurs;
    WHEN kurs=5
        THEN SELECT idO,PIPO,PlaceWorkStudyO
                from Osoba where KursStaz=kurs;
    WHEN kurs=6
        THEN SELECT idO,PIPO,Stat from Osoba where KursStaz=kurs;
    ELSE SELECT "Читач закінчив ВУЗ";
END CASE;
END //
```

Технічно в MySQL існує три види циклів для використання їх у збережених процедурах і функціях:

- цикл WHILE;
- цикл LOOP;
- цикл REPEAT.

Всі ці конструкції циклів повторюють набір операторів до виконання певної умови на початку (WHILE), в кінці (REPEAT) або з явним виходом з нього (LOOP).

### **Конструкція WHILE**

Конструкція WHILE має наступний формат:

**WHILE <логічний вираз> DO**

**<набір операторів>**

**END WHILE**

Набір операторів всередині інструкції WHILE буде повторюватися, доки логічний вираз буде рівний true.

**Приклад 9.7.** Обчислити суму парних чисел в діапазоні від 1 до заданого числа N. Парні числа – це числа, які можуть бути розділені на 2 без залишку. У наступній процедурі користувач задає число через параметр IN та отримує суму парних чисел від 1 до встановленого числа:

```
DELIMITER $$
CREATE PROCEDURE proc_WHILE(IN n INT)
BEGIN
    DECLARE sum INT DEFAULT 0;
    DECLARE x INT DEFAULT 1;
    WHILE x < n DO
        IF mod(x, 2) = 0 THEN
            SET sum = sum + x;
        END IF;
        SET x = x + 1;
    END WHILE;
END$$
```

## Конструкція REPEAT

Конструкція REPEAT має наступний формат:

**REPEAT**

**<набір операторів>**

**UNTIL <логічний вираз>**

**END REPEAT**

Набір операторів всередині інструкції REPEAT буде повторюватися, доки логічний вираз буде рівний false.

Умова виконання циклу перевіряється не на початку, як в циклі WHILE, а в кінці, тобто цикл гарантовано виконується хоча би один раз.

**Приклад 9.8.** Обчислити суму непарних чисел в діапазоні від 1 до заданого числа N. Непарні числа – це числа, які при цілочисельному діленні дають залишок 1. У наступній процедурі користувач задає число через параметр IN та отримує суму непарних чисел від 1 до встановленого числа:

```
DELIMITER $$
CREATE PROCEDURE proc_REPEAT (IN n INT)
BEGIN
    SET @sum = 0;
    SET @x = 1;
    REPEAT
        IF mod(@x, 2) = 1 THEN
            SET @sum = @sum + @x;
        END IF;
        SET @x = @x + 1;
    UNTIL @x > n
    END REPEAT;
END $$
```

## Конструкція LOOP

Конструкція LOOP має наступний формат:

**LOOP**

**<набір операторів>**

**END LOOP**

Цей цикл взагалі не має умов виходу з циклу, тому обов'язково в тілі циклу повинен містити оператор явного виходу з циклу – LEAVE або RETURN.

Зазвичай для виходу з циклу LOOP використовується оператор LEAVE, який має задану мітку. Якщо мітка задана для останнього блоку MySQL, що зберігається, LEAVE виходить з програми.

LEAVE може використовуватися також в BEGIN...END або в інших конструкціях циклу (REPEAT, WHILE). Синтаксис наступний:

**LEAVE мітка;**

Оператор RETURN використовується не тільки для виходу зі всіх видів циклів, але і для завершення процедур та функцій з поверненням результату. Синтаксис наступний:

**RETURN result;**

***Приклад 9.9.*** Обчислити суму непарних чисел в діапазоні від 1 до заданого числа N (аналогічно до прикладу 9.8), використавши оператор LOOP.

```
DELIMITER $$
```

```
CREATE PROCEDURE proc_LOOP (IN n INT)
```

```
BEGIN
```

```
    SET @sum = 0;
```

```
    SET @x = 1;
```

```
    cycle: LOOP
```

```
        IF mod(@x, 2) = 1 THEN
```

```
            SET @sum = @sum + @x;
```

```
        END IF;
```

```
        SET @x = @x + 1;
```

```
        if @x < n THEN
```



```
        LEAVE cycle ;
    END IF;
END LOOP;
END $$
```

#### 9.2.4.2. Збережені функції

Залежно від типу підпрограми – процедура або функція, список параметрів у них буде різним. *Збережена функція* повертає скалярну величину, маючи довільну кількість вхідних параметрів.

Для розроблення збереженої функції використовується оператор, аналогічний оператору CREATE PROCEDURE, який має наступну структуру:

```
CREATE FUNCTION
    <ім'я функції> (опис вхідних параметрів)
    RETURNS <тип результату>
    [LANGUAGE SQL]
    [DETERMINISTIC | NOT DETERMINISTIC]
    [{CONTAINS SQL | NO SQL | READS SQL DATA |
        MODIFIES SQL DATA}]
    [SQL SECURITY {DEFINER | INVOKER}]
    [COMMENT <коментар>]
BEGIN
    Тіло функції;
END;
```

Визначення збереженої функції, так само, як і збереженої процедури, обов'язково маркується обмежувачами за допомогою команди DELIMITER.

**Приклад 9.10.** Створити функцію, яка виводить прізвище бібліотекаря, який видав книгу з назвою або фрагментом назви, що вводиться:

```
SET @a = "%PHP%";
USE BIBLIO;
DROP FUNCTION IF EXISTS func4;
DELIMITER |
CREATE FUNCTION func4(n VARCHAR(20))
```

```

        RETURNS VARCHAR(20)
BEGIN
    DECLARE x int;
    DECLARE y VARCHAR(5);
    SELECT Idk INTO x FROM Katalog WHERE NAMk LIKE n;
    SELECT PIPBf INTO y FROM Formular WHERE Id_kn=x;
    RETURN y;
END |
SELECT func4(@a);

```

### 9.2.5. Редагування і видалення збережених процедур і функцій

Починаючи з версії MySQL 5.0.3, у збережених процедурах або функціях можна редагувати характеристики.

Формат запису команди редагування характеристик збережених підпрограм:

```

ALTER {PROCEDURE | FUNCTION} <ім'я підпрограми>
[характеристики ...]

```

де характеристики:

```

[LANGUAGE SQL]
[DETERMINISTIC | NOT DETERMINISTIC]
[{CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }]
[SQL SECURITY { DEFINER | INVOKER }]
[COMMENT <коментар>]

```

Але, як згадувалося раніше, ці характеристики тільки консультативні, а от редагувати тіло підпрограми цією командою не можна.

Тому зазвичай простіше видалити існуючу підпрограму, а потім створити нову, за потребою з внесеними змінами.

Вилучити підпрограму можна оператором

```

DROP <{PROCEDURE | FUNCTION}>
IF EXISTS <ім'я процедури>;

```

IF EXISTS відловлює помилку у випадку, якщо такої підпрограми не існує.

## 9.2. Технологія OLTP. Основні поняття та вимоги до транзакцій

**Транзакція** – це набір команд, які виконуються як єдине ціле. У транзакції або всі команди будуть виконані, або жодна з них не виконається. Якщо хоча б одна із команд транзакції не може бути виконана, тоді базу даних буде повернено в стан, який вона мала до початку транзакції.

Деколи під транзакцією також розуміють не групу SQL-операторів, а інтервал часу, необхідний для виконання цієї групи SQL-операторів як єдиної команди.

**OLTP (Online Transaction Processing** – онлайнова обробка транзакцій) є способом організації баз даних (БД), при якому система працює з короткими, не пов'язаними одна з одною транзакціями, організованими у великий потік, з метою отримання максимально швидкої відповіді на запити користувачів і програмних додатків.

Системи OLTP характеризуються:

- підтримкою великого числа користувачів;
- малим часом відгуку на запит;
- відносно короткими запитами;
- участь в запитах невеликого числа таблиць.

OLTP-системи використовують нормалізовані моделі даних і надійні та безпечні методи маніпулювання даними і оперативної звітності. Практично всі запити до оперативних реляційних БД в OLTP-додатках регламентовані, тому ефективність OLTP-системи залежить від рівня нормалізації БД і оптимізується за швидкістю і надійністю виконання коротких операцій оновлення даних.

**Нормалізація** БД спричиняє нездатність обробки даних і неочікуваних (ad hoc) запитів у режимі реального часу. Неочікувані і складні аналітичні запити до оперативної інформації змінюють поточні дані, вимагають виконання ресурсомістких операцій об'єднання таблиць і гальмують поточну роботу, надовго блокуючи таблиці і захоплюючи ресурси сервера. Таким чином, OLTP-системи добре справляються зі створенням звітів і аналізом діяльності організації, але спроби отримати комплексні звіти приводять до втрати продуктивності. Вони є негнучкі і обмежені у розв'язанні аналітичних проблем.

Облікові OLTP-системи широко використовуються у банківській, страховій, біржовій, обліковій, комерційній, промисловій діяльності для виконання дискретних операцій читання-запису: касових операцій, переведення коштів, формування рахунків, введення замовлень, управління запасами тощо.

Практично всі запити до бази даних в OLTP-системах складаються з команд вставки, оновлення, видалення, а запити на вибірку в основному призначені для надання користувачам можливості вибору з різних довідників. Велика частина запитів, таким чином, відома заздалегідь ще на етапі проектування системи і критичним для OLTP-технології є швидкість і надійність виконання коротких операцій над даними.

Ситуація, коли декілька прикладних програм користувача одночасно виконують операції читання і запису одних і тих же даних, називається одночасним конкурентним (паралельним) доступом (concurrency).

При паралельному доступі до даних виникає ряд проблем:

- **втрачені оновлення** (коли деяка група команд вклинюється посеред виконання поточної групи команд, а вони пишуть і читають одні і ті ж дані – виникають спотворення в записі / читанні);

- **“грязне читання”** – перша група команд ще не зафіксувала зміни у фізичній БД, а друга вже почала читання (зазвичай доволі розповсюджений випадок втрати оновлень);

- **неузгоджена обробка** – одна група команд прочитала значення і почала його опрацьовувати, а інша обновила до завершення групи команд, в результаті спотворення інформації.

- **читання рядків фантомів...** – вклинювання в довготривалу групу команд і спотворення.

Таким чином, кожна система управління базами даних повинна володіти деяким механізмом управління одночасного конкурентного доступу.

Розглянемо приклад. Користувач А прочитав із таблиці певний запис і коригує його, але в цей самий момент користувач В читає цей самий запис та коригує або вилучає його. Як зміняться дані, коли користувач А намагатиметься зберегти результати своєї роботи і які дані зчитує користувач В? Крім того як

впливають збої на достовірність даних? Традиційний приклад – про переведення грошей з рахунку на рахунок. З рахунку А знято певну суму і переведено на рахунок В, проте під час цієї операції відбувся збій, і гроші на рахунок В не поступили.

Ці і багато інших проблем вирішують за допомогою механізму транзакцій.

Транзакції мають такі властивості, які всі разом позначаються скороченням ACID (Atomicity, Consistency, Isolation, Durability):

- атомарність (Atomicity);
- узгодженість (Consistency);
- ізолюваність (Isolation);
- довговічність (Durability).

Атомарність забезпечує неподільність набору інструкцій, який модифікує дані в базі даних і є частиною транзакції. Це означає, що або виконуються всі зміни даних в транзакції, або в разі будь-якої помилки здійснюється відкат всіх виконаних змін.

Дії, спрямовані на підтримку атомарності транзакції під час її аварійного завершення спричиненого помилками вводу/виводу, перевантаженням системи чи блокуваннями називається відновленням транзакцій. Дії, спрямовані на підтримку атомарності під час виходу з ладу системи, називаються відновленням в разі виникнення перебоїв.

Узгодженість – транзакція не повинна порушувати цілісності бази даних. Терміном цілісність даних позначають достовірність і точність інформації, що зберігається в базі даних. Цілісність досягається забезпеченням відповідності даних певним додатковим обмеженням, крім тих, які накладаються під час створення бази даних на структуру даних та їх типи.

Ізолюваність означає, що на роботу транзакції не можуть впливати інші транзакції. Транзакція бачить дані в тому стані, в якому вони перебували до початку роботи іншої транзакції або в тому стані, в якому вони перебувають після її завершення. Одна транзакція не може переглянути проміжні стани даних, що використовуються іншими транзакціями. Якщо транзакція зчитує кілька разів ті

самі дані, то вона повинна отримувати їх щоразу в тому ж стані, в якому вони були під час першого зчитування.

Іншими словами, активна транзакція не може бачити модифікації даних в паралельній або незавершеній транзакції. Це означає, що для забезпечення ізоляції для деяких транзакцій може знадобитися виконати відкат.

Довговічність. Властивість довговічності забезпечує одну з найбільш важливих вимог баз даних: збереження даних. Після того, як було підтверджено успішне завершення роботи транзакції, система має гарантувати, що ці результати не будуть втрачені, незважаючи на можливі перебої.

Іншими словами, ефект транзакції повинен залишатися дієвим навіть в разі системної помилки. З цієї причини, якщо в процесі виконання транзакції відбувається системна помилка, то здійснюється відкат для всіх виконаних інструкцій цієї транзакції.

Для вирішення проблеми одночасного доступу розроблено *спеціальний стандарт*, який визначає 4 рівні ізоляції транзакцій і, відповідно, надійності роботи з даними.

*Рівень 0*. Читання “забруднених” (непідтверджених) даних – Read Uncommitted, Dirty Read. На цьому рівні можливе читання незафіксованих змін як поточної транзакції, так і паралельних транзакцій. Немає гарантії, що дані, змінені іншими транзакціями, не будуть у будь-який момент змінені в результаті їх відкату, тому таке читання є потенційним джерелом помилок. Непоможливі втрачені зміни (lost changes), можливі повторного читання і фантоми.

*Рівень 1*. Читання підтверджених даних (Read Committed). На цьому рівні не можливе читання незафіксованих змін (заборона некоректного зчитування), тобто здійснюється читання всіх змін своєї транзакції і зафіксованих змін паралельних транзакцій. Якщо поточна транзакція розпочала змінювати дані, то жодна інша транзакція не може зчитати ці дані доти, поки перша не завершиться. Втрачені зміни і брудне читання не допускається, можливі повторне читання і фантоми.

Але таблиці цілком не блокуються. Через це залишається проблема «фантомного читання». «Фантомне читання» – це коли під час виконання однієї транзакції результат одних і тих же вибірок може змінюватися у зв’язку з тим, що

блокується не вся таблиця, а тільки ті рядки, які беруть участь у вибірці. Це означає, що паралельні транзакції можуть вставляти рядки в таблицю, в якій відбувається вибірка, тому два запити `SELECT * FROM table` можуть дати різний результат в різний час при вставці даних паралельними транзакціями.

*Рівень 2. Повторне зчитування (Repeatable Read, Snapshot).* На цьому рівні можливе читання всіх змін поточної транзакції, але будь-які зміни, внесені паралельними транзакціями після початку поточної, недоступні. Втрачені зміни, “брудне” і повторне читання неможливі, можливі фантоми.

*Рівень 3. Заборона фантомів (Serializable).* Якщо транзакція звертається до даних, то жодна інша транзакція не зможе додати чи видалити рядки, що можуть бути зчитані під час виконання транзакції. На цьому рівні взагалі відсутні будь-які проблеми паралельних транзакцій, але за це доведеться платити швидкодією системи, а швидкодія в більшості випадків вкрай важлива.

Отже, чим вищий рівень ізоляції, тим більше необхідно ресурсів, щоб його забезпечити. Відповідно, підвищення ізолюваності може призводити до зниження швидкості виконання паралельних транзакцій, що є «платою» за підвищення надійності.

В СКБД рівень ізоляції транзакцій можна вибрати як для всіх транзакцій відразу, так і для однієї конкретної транзакції. За замовчуванням в MySQL використовується рівень 2 (Repeatable Read), так як він найбільш вдалий для більшості випадків.

На перший погляд може здатися, що найкращий варіант № 3 – він найнадійніший, але на практиці можна отримати великі незручності через дуже повільну роботу СКБД.

Рівень 0 використовується в основному для відстеження змін тривалих транзакцій або для читання даних, які рідко змінюються.

Пам’ятайте, що багато що залежить не від того, наскільки хороший рівень ізоляції транзакцій в БД, а від того, як спроектовано вашу програму.

### **9.2.1. Реалізація транзакцій в SQL**

За замовчуванням СКБД працює в режимі автоматичного початку транзакцій, коли кожна команда розглядається як окрема транзакція. Якщо команда виконана

успішно, тоді її результати фіксуються в БД. Якщо ж при виконанні команди виникла помилка, тоді зроблені зміни відміняються і система повертається в початковий стан – так званий відкат.

Повноцінна реалізація відкату є досить складною задачею. Обробка даних, що надходять, призводить до великої кількості маленьких змін, включаючи оновлення як самих таблиць, так і індексів. Ці зміни потенційно можуть зазнати невдачі: закінчилося місце на диску, операція займає надто багато часу (timeout) і т. д. Система повинна в разі невдачі коректно повернути базу даних в стан до початку виконання транзакції.

Перші комерційні СКБД користувалися виключно блокуванням доступу до даних для забезпечення властивостей ACID (атомарність, узгодженість, ізолюваність, довговічність). Але велика кількість блокувань призводить до суттєвого зменшення продуктивності.

Для зниження кількості блокувань використовуються два підходи:

- журналізація змін (write ahead logging, WAL);
- механізм тіньових сторінок (shadow paging).

В обох випадках блокування повинні бути розставлені на всю інформацію, яка оновлюється. Залежно від рівня ізоляції та імплементації, блокування також розставляються на інформацію, яка була прочитана транзакцією.

При випереджаючій журналізації, яка використовується в Sybase і MS SQL Server до версії 2005, всі зміни записуються в журнал, і тільки після успішного завершення – в базу даних. Це дозволяє СКБД повернутися в робочий стан після несподіваного падіння системи. Тіньові сторінки містять копії тих сторінок бази даних на початок транзакції, в яких відбуваються зміни. Ці копії активізуються після успішного завершення. Хоча тіньові сторінки легше реалізуються, випереджаюча журналізація ефективніша.

Подальший розвиток технологій управління базами даних призвів до появи безблокувальних технологій. Ідея контролю над паралельним доступом за допомогою тимчасових міток (timestamp-based concurrency control) була розвинена і привела до появи багатоверсійності архітектури MVCC. Ці технології не потребують ні журналізації змін, ні тіньових сторінок. Архітектура,



реалізована в Oracle 7.x і вище, записує старі версії сторінок в спеціальний сегмент відкату, але вони все ще доступні для читання. Якщо транзакція при читанні потрапляє на сторінку, тимчасова мітка якої новіша початку читання, дані беруться з сегмента відкату (тобто використовується «стара» версія). Для підтримки такої роботи ведеться *журнал транзакцій*, але на відміну від випереджаючої журналізації, він не містить даних. Робота з ним складається з трьох логічних кроків:

- записати намір провести деякі операції;
- виконати завдання, копіюючи оригінали змінюваних сторінок в сегмент відкату;
- записати, що все зроблено без помилок.

*Журнал транзакцій* в поєднанні з сегментом відкату (область, в якій зберігається копія всіх змінюваних в ході транзакції даних) гарантує цілісність даних. У разі збою запускається процедура відновлення, яка переглядає окремі його записи таким чином:

- Якщо пошкоджений запис, то збій стався під час здійснення відмітки в журналі. Значить, нічого важливого не загубилося, ігноруємо цю помилку.
- Якщо всі записи позначені як успішно виконані, то збій стався між транзакціями, тут також немає втрат.
- Якщо в журналі є незавершена транзакція, то збій стався під час запису на диск. В цьому випадку ми відновлюємо стару версію даних з сегменту відкату.

Платформа MySQL зазвичай працює в режимі автоматичної фіксації, тобто за замовчуванням включений режим AUTOCOMMIT. Це означає, що за замовчуванням кожен запит еквівалентний одній транзакції, а зміни записуються на диск автоматично після завершення команд або транзакцій. Якщо з якоїсь причини зміни внести не вдається, відбувається автоматичний відкат.

Відключення режиму автоматичної фіксації працює тільки для «таблиць з безпечною транзакцією» (transaction-safe tables), таких, як таблиці InnoDB і BDB.

Відключення цього режиму в інших таблицях ніякого ефекту не надає. Режим залишається включеним.

Транзакція починається зі спеціального запиту «START TRANSACTION», або «BEGIN». Щоб закінчити транзакцію, потрібно або зафіксувати зміни (запит COMMIT), або відкотити їх (запит ROLLBACK).

Інструкція START TRANSACTION дозволяє створити нову транзакцію. При цьому режим автоматичного завершення транзакцій автоматично відключиться тільки для послідовності команд в поточній транзакції (режим AUTOCOMMIT = 0). Формат запису:

```
START TRANSACTION [READ ONLY | READ WRITE]  
[ISOLATION LEVEL  
  {READ COMMITTED |  
  READ UNCOMMITTED |  
  REPEATABLE READ |  
  SERIALIZABLE}
```

Відповідно до стандарту ANSI, єдина відмінність між інструкціями SET і START полягає в тому, що START вважається початком нової транзакції, а SET TRANSACT застосовується до наступної транзакції.

Оператор START TRANSACTION – це стандартний синтаксис SQL і рекомендований спосіб починати транзакції.

Інструкції-аналоги START TRANSACTION – BEGIN [TRANSACTION]] і її синонім BEGIN [WORK] доступні, починаючи з версії MySQL 3.23.17 і MySQL 3.23.19, відповідно.

Інструкція BEGIN TRANSACTION оголошує явну транзакцію, але не вказує рівні ізоляції.

Найголовніше правило використання інструкції START TRANSACTION полягає в тому, що необхідно використовувати цю інструкцію для управління режимом доступу, рівнем ізоляції або об'ємом діагностичної інформації для поточної транзакції. Як тільки запускається нова транзакція, необхідно або вказувати нові значення параметрів, або використовувати задані за замовчуванням.

Більшість платформ дозволяють неявним чином управляти транзакціями, для чого використовується режим автоматичної фіксації (autocommit mode). В цьому режимі база даних розглядає кожну інструкцію як окрему транзакцію, з неявними інструкціями BEGIN TRAN і COMMIT TRAN.

Альтернативою режиму автоматичної фіксації є явне ручне управління кожною транзакцією. При явному управлінні кожна нова транзакція оголошується за допомогою інструкції START TRANSACTION. Нова транзакція також може бути запущена неявно при кожному запуску інструкції, яка ініціює транзакцію. Серед інструкцій, які ініціюють транзакції, можна виділити INSERT, UPDATE, DELETE або SELECT.

***Транзакція не буде фіксувати зміни у БД або здійснювати відкат до того часу, поки не буде явно виконана інструкція COMMIT або ROLLBACK.***

Кожна явно оголошена транзакція після успішного завершення буде зафіксована (будуть внесені і збережені зміни в БД) тільки після виконання інструкції COMMIT. Аналогічно, кожна транзакція, що завершилася невдало, повинна скасовуватися явно інструкцією ROLLBACK (відкат).

Таким чином, інструкцію START необхідно використовувати тільки в парі з COMMIT або ROLLBACK. Інакше СКБД може не завершити транзакцію до того часу, поки не зустріне інструкцію COMMIT або ROLLBACK. Відсутність необхідних інструкцій COMMIT (і ROLLBACK) може привести до величезних, погано спланованих транзакцій.

Варто зазначити, що декілька команд SQL автоматично завершують транзакцію, ніби була здійснена операція COMMIT:

- ALTER TABLE;
- BEGIN;
- DROP DATABASE;
- DROP TABLE;
- CREATE INDEX;
- TRUNCATE;
- RENAME TABLE.

При використанні таблиць, що підтримують транзакції (таких як InnoDB, BDB), в MySQL можна відключити режим autocommit за допомогою наступної команди:

```
SET AUTOCOMMIT = 0
```

Після цього необхідно застосувати команду COMMIT для запису змін на диск або команду ROLLBACK, яка дозволяє ігнорувати зміни, вироблені з початку даної транзакції.

Доброю практикою є використання явних інструкцій COMMIT і ROLLBACK після однієї або декількох інструкцій, оскільки транзакції, які довго виконуються, можуть блокувати ресурси, не даючи доступу до них іншим користувачам. І не забувайте про фіксацію (або відкат)! Об'ємні транзакційні пакети, або транзакційні пакети, що довго виконуються, можуть переповнити сегменти відкату або журнали транзакцій бази даних.

Приклад з COMMIT:

```
SET AUTOCOMMIT = 0; // Відключаємо autocommit
START TRANSACTION; (Також, можна написати BEGIN;)
... якісь дії з БД (INSERT, UPDATE, DELETE ...)
COMMIT; // Фіксація дій, запис їх в фізичну БД.
```

Приклад з ROLLBACK:

```
SET AUTOCOMMIT = 0; // Відключаємо autocommit
START TRANSACTION;
... Якісь дії з БД (INSERT, UPDATE, DELETE ...)
ROLLBACK; // Скасування серії дій, запис в фізичну БД не здійснюється.
```

В MySQL не існує механізму вкладених транзакцій.

**Приклад 9.11.** Визначити книгу та абонента, який найдовше користується цією книгою та змінити термін повернення цієї книги на кінець поточного року.

```
START TRANSACTION;
SELECT @A:= os.IdO as Abonent, @B:= kt.IdK as Book, MIN(DataIN)
FROM
    BIBLIO. Formuljar fr INNER JOIN BIBLIO.Osoba os ON os.IdO=fr.IdO
```

```
INNER JOIN BIBLIO.Katalog kt ON fr.IdK=kt.IdK;  
UPDATE BIBLIO.Formuljar fr SET fr.DataOut="31.12.21"  
WHERE fr.IdO=@A AND fr.IdK=@B;  
COMMIT;
```

У прикладі 9.11 перша команда START TRANSACTION повідомляє серверу про початок транзакції. Це означає, що до отримання сервером команди завершення транзакції COMMIT всі зміни є тимчасовими. Отже, якщо на сервері відбудеться збій після першої або другої команди (відповідно SELECT або UPDATE), відбудеться відкат транзакції. Термін «відкат» означає, що всі наслідки транзакції будуть скасовані; дані на сервері будуть виглядати так, немов ніяких змін не було. Щоб забезпечити узгодженість транзакцій, модифікована інформація блокується до завершення транзакції. Термін «блокування» (lock) означає, що ніякий процес не зможе звернутися до даних до їх звільнення (за допомогою закріплення або відкату транзакції).

**Порада.** Постарайтеся скоротити обсяг транзакцій і прибрати з них зайві команди SELECT. Тим самим ви скорочуєте час, протягом якого зберігається блокування об'єктів. Додаток ніколи не повинен тримати блокування в очікуванні введення даних від користувача.

### 9.2.2. Точки відкату

Починаючи з версії MySQL 4.1.1, під час відкату транзакції можна анулювати роботу не всіх операторів поточної транзакції, що зазнала невдачі, а лише деяку частину, яка вказується у тілі транзакції так званою точкою збереження. Оператор у транзакції, до якого буде здійснено відкат у разі невдачі всієї транзакції, попереду виділяється оператором SAVEPOINT.

Точки збереження дуже корисні в ситуаціях, коли необхідно відновити частину незавершеної транзакції.

Оператор SAVEPOINT встановлює іменовану точку відкату транзакції з іменем *ідентифікатор*. Якщо поточна транзакція уже має точку збереження з таким же іменем, тоді ця попередня точка ігнорується.

Оператор ROLLBACK TO SAVEPOINT здійснює відкат транзакції до іменованої точки збереження. Модифікації рядків, які виконувалися поточною

транзакцією після цієї точки, відміняються відкатом, але InnoDB не знімає блокувань рядків, які були встановлені в пам'яті після точки збереження. Точки збереження, які встановлені пізніше іменованої точки, видаляються.

Синтаксис SAVEPOINT і ROLLBACK TO SAVEPOINT :

**SAVEPOINT <ідентифікатор>**

**ROLLBACK TO SAVEPOINT <ідентифікатор>**

Якщо ж у транзакції вказується оператор COMMIT або ROLLBACK без вказування імені точки збереження (без ідентифікатора), тоді всі точки збереження транзакцій, встановлені оператором SAVEPOINT, анулюються.

*Приклад 9.12* показує, що в програмному блоці зберігаються тільки ті зміни, які були внесені до опису точки збереження:

```
START TRANSACTION;
```

```
SELECT @A:= os.IdO as Abonent, @B:= kt.IdK as Book, MIN(DataIn)
```

```
FROM BIBLIO.Formuljar fr
```

```
INNER JOIN BIBLIO.Osoba os ON os.IdO=fr.IdO
```

```
INNER JOIN BIBLIO.Katalog kt ON fr.IdK=kt.IdK;
```

```
SAVEPOINT p1;
```

```
UPDATE BIBLIO.Formuljar fr SET fr.DataOut="01.12.16"
```

```
WHERE fr.IdO=@A AND fr.IdK=@B;
```

```
ROLLBACK TO SAVEPOINT p1;
```

```
COMMIT;
```

При задаванні в операторі ROLLBACK TO SAVEPOINT імені точки збереження, якої не існує, Workbench видасть повідомлення:

```
ERROR 1181: Got error 153 during ROLLBACK.
```

### **9.2.3. Блокування таблиць**

#### **9.2.3.1. Типи таблиць в MySQL**

MySQL підтримує два різних типи таблиць: транзакційні (InnoDB і BDB) і без підтримки транзакцій (HEAP, ISAM, MERGE і MyISAM).

Прийнятим за замовчуванням типом таблиць в MySQL є MyISAM. При створенні нової таблиці, відмінної від MyISAM, необхідно вказувати тип таблиці:

CREATE TABLE ...

ENGINE=InnoDB;

**Переваги транзакційних таблиць** (Transaction-safe tables, TST):

- Надійність. Навіть якщо станеться збій в роботі MySQL або виникнуть проблеми з обладнанням, тоді дані можна відновити: або методом автоматичного відновлення, або за допомогою резервної копії та журналу транзакцій.
- Можна поєднувати декілька операторів і приймати (підтверджувати виконання) всіх цих операторів однією командою COMMIT.
- Можна запустити команду ROLLBACK, щоб скасувати внесені зміни (якщо робота не проводиться в режимі автоматичної фіксації).
- Якщо станеться збій під час оновлення, всі зміни будуть відновлені (в таблицях без підтримки транзакцій всі внесені зміни не можуть бути скасовані).

**Переваги таблиць без підтримки транзакцій** (non-transaction-safe tables, NTST):

- Працювати з ними набагато швидше, тому що не виконуються додаткові транзакції.
- Для них потрібно менше дискового простору, так як не застосовуються додаткові транзакції.
- Для оновлення використовується менше пам'яті.

Перетворювати таблиці з одного типу в інший можна за допомогою оператора ALTER TABLE.

### 9.2.3.2. Типи блокувань

На даний час MySQL підтримує такі типи блокувань:

- табличне для таблиць типів ISAM / MyISAM і HEAP;
- сторінкове для таблиць BDB;
- рядкове для таблиць InnoDB.

Щоб отримати високу швидкість блокування, в MySQL на таблицях всіх типів, за винятком InnoDB і BDB, використовується табличне блокування (замість

блокування рядків або стовпців). Якщо таблиці великі, тоді для більшості застосунків табличне блокування набагато краще за рядкове, але далеко не у всіх випадках.

Табличне блокування забезпечує можливість одночасного виконання читання з таблиці декількома потоками, але якщо деякий з потоків спробує здійснити запис в таблицю, тоді він повинен володіти такими правами доступу. Під час оновлення таблиці всі інші потоки, які звертаються до цієї конкретної таблиці, будуть очікувати, доки дане оновлення не буде завершено.

Для таблиць типів InnoDB та BDB в MySQL табличне блокування використовується тільки у випадку, якщо таблиця явно блокується командою LOCK TABLES.

### 9.2.3.3. Оператори LOCK TABLES та UNLOCK TABLES

Блокування та розблокування таблиці здійснюється операторами LOCK TABLES / UNLOCK TABLES відповідно.

Основна причина застосування LOCK TABLES – емуляція транзакцій або підвищення швидкості оновлення таблиць.

Формат запису:

```
LOCK TABLES <ім'я_таблиці> [AS <псевдонім>]  
    {READ [LOCAL] | [LOW_PRIORITY] WRITE}  
    [, <ім'я_таблиці> [AS <псевдонім>]  
    {READ [LOCAL] | [LOW_PRIORITY] WRITE}] ...
```

#### **UNLOCK TABLES**

Команда LOCK TABLES блокує зазначені в ній таблиці для даного потоку.

Команда UNLOCK TABLES знімає будь-які блокування, що утримуються поточним потоком. Всі таблиці, які заблоковані в поточному потоці, неявно розблоковуються, коли потік виконає інший оператор LOCK TABLES або коли закривається з'єднання з сервером.

Якщо потік встановлює блокування з читання (READ) на таблиці, то цей потік (і всі інші) може тільки читати дані з таблиці. Якщо потік встановлює блокування запису (WRITE) таблиці, то лише цей потік може читати і писати в таблицю. Доступ інших ниток до таблиці блокується.



Різниця між READ LOCAL і READ полягає в тому, що READ LOCAL дозволяє виконуватися неконфліктним операторам INSERT (паралельним вставкам), поки блокування утримується. Але при зовнішньому звертанні до таблиць БД в той час, поки утримується блокування, неконфліктні оператори INSERT виконаними бути не можуть.

Блокування WRITE зазвичай має більш високий пріоритет, ніж блокування READ, щоб гарантувати, що зміни обробляються так швидко, як можливо. Це означає, що якщо один потік отримує блокування READ і потім інший потік запитує блокування WRITE, наступні запити на блокування READ чекатимуть, поки потік WRITE не отримає блокування і не зніме її. Можна використовувати блокування LOW\_PRIORITY WRITE, що дозволяють іншим потокам отримувати блокування READ в той час, як основний потік знаходиться в стані очікування блокування WRITE. Блокування LOW\_PRIORITY WRITE можуть бути використані тільки якщо є впевненість, що в кінцевому підсумку буде період часу, коли жоден з потоків не матиме блокування READ.

Використання для таблиці блокування LOW\_PRIORITY WRITE означає, що MySQL буде виконувати дане конкретне блокування, поки не з'явиться потік із блокуванням READ.

У деяких випадках, коли необхідно встановити пріоритети отримання даних, для команд INSERT, UPDATE або DELETE можна вказувати опцію LOW\_PRIORITY, а для команди SELECT – HIGH\_PRIORITY.

Починаючи з версії MySQL 4.0.2, для того, щоб виконувати LOCK TABLES, необхідно мати привілеї LOCK TABLES і привілеї SELECT для відповідних таблиць. В MySQL 3.23 необхідно мати привілеї SELECT, INSERT, DELETE і UPDATE для цих таблиць.

У разі застосування LOCK TABLES необхідно блокувати всі таблиці, які використовуються в запитах.

**Приклад 9.13.** Додати всі записи з таблиці osoba1 до таблиці Osoba, а після додавання – очистити таблицю osoba1. При цьому використати блокування, щоб інші користувачі не змогли внести зміни:

```
USE BIBLIO;
```

```

LOCK TABLES Osoba WRITE, osoba1 WRITE;
INSERT INTO Osoba
SELECT * FROM osoba1;
TRUNCATE TABLE osoba1;
UNLOCK TABLES;

```

Якщо одна і та ж таблиця використовується декілька разів у запиті (через псевдоніми), тоді необхідно отримати блокування на кожен псевдонім.

**Приклад 9.14.** Нехай у таблиці Osoba зустрічаються абоненти, які зареєстровані в один і той же день. Необхідно вивести попарно прізвища цих абонентів, щоб записи не повторювалися. Для такого запиту необхідно використати самооб'єднання таблиць із блокуванням кожного псевдоніму:

```

USE BIBLIO;
LOCK TABLES Osoba t1 WRITE, Osoba t2 WRITE;
SELECT DISTINCT t1.PIPO, t2.PIPO, t1.DRO
FROM Osoba t1, Osoba t2
WHERE t1.DRO = t2.DRO AND t1.IDO < t2.IDO;
UNLOCK TABLES;

```

Поки блокування активне, отримати доступ до інших таблиць, що не були заблоковані цим оператором LOCK TABLES, неможливо.

### Контрольні запитання

1. Яка різниця між збереженою процедурою та збереженою функцією?
2. Які види аргументів має збережена процедура, а які – збережена функція?
3. Що таке DELIMITER? Його призначення?
4. Проаналізуйте особливості декларування локальних змінних в збережених процедурах.
5. Охарактеризуйте відмінності команди IF від функції IF().
6. Які форми запису має оператор вибору CASE?
7. Які типи операторів циклів можна використовувати у збережених процедурах?

8. Як викликаються збережені процедури та збережені функції? Навести приклади виклику.
9. Як видалити збережені процедури або функції? Навести приклади.
10. Проаналізуйте характеристики OLTP систем.
11. Які переваги та недоліки використання OLTP систем?
12. Які виникають проблеми при паралельному доступі до даних?
13. Що таке транзакція?
14. Наведіть приклад опрацювання даних бази, коли необхідне використання транзакцій?
15. Які вимоги повинні задовольняти транзакції?
16. Для чого призначені інструкції COMMIT та ROLLBACK?
17. Поясніть принципи встановлення точок відкату.
18. Які типи блокувань є в MySQL?
19. Проаналізуйте встановлення пріоритетів отримання даних.
20. Скільки є рівнів блокувань і яке їх функціональне призначення?

# Тема 10. Big Data рішення. OLAP технологія

## 10.1. Великі дані (Big Data)

«Великі дані (Big Data) – позначення структурованих і неструктурованих даних величезних обсягів і значного розмаїття, що піддаються ефективній обробці програмних інструментів, які горизонтально масштабуються та з'явилися у кінці 2000-х років, і альтернативних традиційних систем управління базами даних і рішенням класу рішень Business Intelligence».

Як бачимо, у цьому визначенні присутні такі неоднозначні терміни, як «величезних», «значного», «ефективній» та «альтернативних». Навіть сама назва доволі суб'єктивна. Для прикладу, 4 Терабайти (ємність сучасного зовнішнього жорсткого диску для ноутбуку) – це вже великі дані чи ні? Вікіпедія трактує наступне: «в широкому сенсі про «великі дані» говорять, як про соціально-економічний феномен, пов'язаний з появою технологічних можливостей аналізувати величезні масиви інформації, у деяких проблемних галузях – весь світовий об'єм даних, і трансформаційні наслідки, які з цього випливають»

Можна користуватись і більш простим визначенням, яке цілком відповідає усталеним і більш простим визначенням, що цілком відповідають думці журналістів і маркетологів. «Великі дані – це сукупність технологій, покликаних здійснювати три операції:

1. Обробляти більші, у порівнянні зі «стандартними» сценаріями, об'єми даних.
2. Уміти працювати з даними, що швидко надходять у дуже великих об'ємах. Тобто даних не просто багато, а їх постійно стає все більше й більше.
3. Вміти працювати зі структурованими і малоструктурованими даними паралельно і у різних аспектах.

Вважається, що ці «вміння» дозволяють виявляти приховані закономірності, що вислизують від обмеженого людського сприйняття. Це дає безпрецедентні можливості оптимізації багатьох сфер нашого життя: державного управління, медицини, телекомунікацій, фінансів, транспорту, виробництва і так далі. Не

дивно, що журналісти і маркетологи так часто використовували словосполучення Big Data, що багато експертів вважають цей термін дискредитованим і пропонують від нього відмовитись.

Більш того, у жовтні 2015 року компанія Gartner виключила Big Data з числа популярних трендів. Своє рішення аналітики компанії пояснили тим, що до складу поняття «великі дані» входить значна кількість технологій, які вже активно застосовуються на підприємствах, вони частково стосуються інших популярних сфер і тенденцій і стали повсякденним робочим інструментом.

Та як би не було, термін Big Data як і раніше широко використовується, а підтвердженням тому є нові технології та тенденції роботи з Big Data.

В останні роки у світі сформувався ряд нових технологій збереження і аналізу великих обсягів даних:

- 1) сховища даних, або склади даних (Data Warehouse);
- 2) оперативна аналітична обробка (On-Line Analytical Processing, OLAP);
- 3) інтелектуальний аналіз даних – ІАД (Data Mining).

#### **10.1.1. Принципи роботи з великими даними (V V V)**

Визначальними характеристиками для великих даних є, окрім їх фізичного об'єму, й інші, які підкреслюють складність задачі обробки і аналізу цих даних. Компанією Meta Group у 2001 році було розроблено три принципи роботи з великими даними, так звані V V V:

**volume** – фізичний об'єм;

**velocity** – швидкість приросту даних;

**variety** – необхідність їх швидкої обробки, здатність обробляти дані різних типів.

У подальшому з'явилась інтерпретація з :

- чотирма V, додалась veracity (достовірність),
- п'ятьма V (viability – життєздатність або value – цінність),
- сімома V (variability – змінність та visualization – візуалізація).

Але компанія IDC, наприклад, інтерпретує саме четверте V як value (цінність), підкреслюючи економічну доцільність обробки великих об'ємів даних у відповідних умовах.

Виходячи з вищеназваних визначень, основні принципи роботи з великими даними такі:

**Горизонтальна масштабованість.** Це – базовий принцип обробки великих даних. Як вже було зазначено, великих даних з кожним днем стає все більше. Відповідно, необхідно збільшувати кількість обчислювальних вузлів, за якими розподіляються ці дані, при тому обробка має відбуватись без погіршення продуктивності.

**Відмовостійкість.** Цей принцип витікає з попереднього. Оскільки обчислювальних вузлів у кластері може бути багато (іноді десятки тисяч) та їх кількість, не виключено, буде збільшуватись, зростає ймовірність виходу машин з ладу. Методи роботи з великими даними мають враховувати ймовірність таких ситуацій і передбачати превентивні заходи

**Локальність даних.** Оскільки дані розподілені по великій кількості обчислювальних вузлів, то, якщо вони фізично знаходяться на одному сервері, а обробляються на іншому, витрати на передачу даних можуть бути не виправдано великими. Тому обробку даних бажано проводити на тій же машині, на якій вони зберігаються

Ці принципи відрізняються від тих, які характерні для традиційних, централізованих, вертикальних моделей зберігання добре структурованих даних. Власне, для роботи з великими даними розробляються підходи і технології.

### 10.1.2. Технології і тенденції роботи з Big Data

Початково у сукупність підходів і технологій включались засоби масово-паралельної обробки невизначено-структурованих даних, такі як СКБД NoSQL, алгоритми MapReduce і засоби проекту Hadoop. У подальшому до технологій великих даних почали відносити й інші рішення, що забезпечують схожі за характеристиками можливості обробки надвеликих масивів даних, а також деякі апаратні засоби.

**MapReduce** – модель розподілених обчислювань у комп'ютерних кластерах, представлена компанією Google. Згідно з цією моделлю, додаток розділяється на значну кількість однакових елементарних

завдань, що виконуються на вузлах кластера і потім, природнім шляхом зводяться у кінцевий результат.

**NoSQL** (від англ. Not Only SQL, не лише SQL) – загальний термін для різних нереляційних баз даних і сховищ, що не означає якусь конкретну технологію чи продукт. Звичайні реляційні бази даних добре підходять для досить швидких і однотипних запитів, а на складних і гнучко побудованих запитах, характерних для великих даних, навантаження перевищує розумні межі і використання СКБД стає неефективним.

**Hadoop** – набір утилітів, бібліотек і фреймворків, що вільно розповсюджується, для розробки і виконання розподілених програм, які працюють на кластерах із сотень і тисяч вузлів. Вважається однією з основоположних технологій більшості даних.

**R** – мова програмування для статистичної обробки даних і роботи з графікою. Широко використовується для аналізу даних і фактично стала стандартом для статистичних програм.

**Апаратні рішення.** Корпорації Teradata, EMC та ін. др. пропонують апаратно-програмні комплекси, призначені для обробки великих даних. Ці комплекси поставляються як готові до установки телекомунікаційні шафи, що містять кластер серверів і керівне програмне забезпечення для масово-паралельної обробки. Сюди іноді відносять апаратні рішення для аналітичної обробки в оперативній пам'яті, зокрема, апаратно-програмні комплекси Hana компанії SAP і комплекс Exalytics компанії Oracle, незважаючи на те, що така обробка початково не є масово-паралельною, а об'єми оперативної пам'яті одного вузла обмежуються кількома терабайтами.

Консалтингова компанія McKinsey, окрім технологій NoSQL, MapReduce, Hadoop, R, які розглядає більшість аналітиків, включає у контекст придатності для обробки великих даних також технології Business Intelligence і реляційні системи управління базами даних з підтримкою мови SQL.

### 10.1.3. Методи і техніка аналізу великих даних

Міжнародна консалтингова компанія McKinsey, що спеціалізується на розв'язанні задач, пов'язаних зі стратегічним управлінням, виділяє 11 методів і технік аналізу, що застосовуються до великих даних.

- 1) **Методи класу Data Mining** (видобуток даних, інтелектуальний аналіз даних, глибинний аналіз даних) – сукупність методів виявлення у даних раніше невідомих, нетривіальних, практично корисних знань, необхідних для прийняття рішень. До таких методів, зокрема, належать: навчання асоціативним правилам (association rule learning), класифікація (розгалуження на категорії), кластерний аналіз, регресійний аналіз, виявлення і аналіз відхилень тощо.
- 2) **Краудсорсинг** – класифікація і збагачення даних силами широкого, неозначеного кола особистостей, що виконують цю роботу без вступу у трудові стосунки.
- 3) **Змішання та інтеграція даних** (data fusion and integration) – набір технік, що дозволяють інтегрувати різномірні дані з розмаїття джерел з метою проведення глибинного аналізу (наприклад, цифрова обробка сигналів, обробка природньої мови, включно з тональним аналізом).
- 4) **Машинне навчання**, включаючи навчання з учителем і без учителя – використання моделей, побудованих на базі статистичного аналізу чи машинного навчання для отримання комплексних прогнозів на основі базових моделей.
- 5) **Штучні нейронні мережі**, мережевий аналіз, оптимізація, у тому числі генетичні алгоритми (genetic algorithm – евристичні алгоритми пошуку, що використовуються для розв'язання задач оптимізації і моделювання шляхом випадкового підбору, комбінування і варіації потрібних параметрів з використанням механізмів, аналогічних натуральному відбору у природі)
- 6) **Розпізнавання образів**
- 7) **Прогнозна аналітика**
- 8) **Імітаційне моделювання** (simulation) – метод, що дозволяє будувати моделі, що описують процеси так, як вони би проходили у дійсності.



Імітаційне моделювання модна розглядати як різновид експериментальних випробувань.

**9) Просторовий аналіз** (spatial analysis) – клас методів, що використовують топологічну, геометричну і географічну інформацію, що вилучається із даних.

**10) Статистичний аналіз** – аналіз часових рядів, А/В-тестування (А/В testing, split testing – метод маркетингового дослідження; при його використанні контрольна група елементів порівнюється із набором тестових груп, у яких один чи кілька показників були змінені, щоб з'ясувати, які зі змін покращують цільовий показник.

**11) Візуалізація аналітичних даних** – подання інформації у вигляді малюнків, діаграм, з використанням інтерактивних можливостей і анімації, як для отримання результатів, так і для використання у якості вихідних даних для подальшого аналізу. Дуже важливий етап аналізу великих даних, що дозволяє показати найважливіші результати аналізу у найбільш зручному для сприйняття вигляді.

## **10.2. Технологія OLAP**

### **10.2.1. Основні поняття OLAP**

Технологія OLAP – це не окремо взятий програмний продукт, і не мова програмування. Якщо постаратися охопити OLAP в усіх його проявах, то це сукупність концепцій, принципів і вимог, що лежать в основі програмних продуктів, що полегшують аналітикам доступ до великих об'ємів даних.

Технологія комплексного багатовимірного аналізу даних отримала назву OLAP (On-Line Analytical Processing). OLAP – це ключовий компонент організації *сховищ даних* (СД).

Основне призначення OLAP систем – це підтримка аналітичної діяльності, довільних запитів користувачів-аналітиків. Метою OLAP аналізу є перевірка гіпотез, які виникають.

В основі OLAP технології лежить багатовимірне концептуальне представлення даних (multidimensional conceptual view).

Концепція OLAP була описана в 1993 році Едгаром Коддом, відомим дослідником баз даних і автором реляційної моделі даних. У 1995 році на основі вимог, викладених Коддом, було сформульовано так званий *мечт FASMI* (Fast Analysis of Shared Multidimensional Information) – швидкий аналіз багатовимірної інформації шляхом розділення з такими вимогами для багатовимірного аналізу:

**Fast** (Швидкий) – надання користувачеві результатів аналізу за прийнятний час (зазвичай не більше 5 с), нехай навіть ціною менш детального аналізу;

**Analysis** (Аналіз) – можливість здійснення будь-якого логічного і статистичного аналізу, характерного для цього додатка, і його збереження в доступному для кінцевого користувача вигляді;

**Shared** (Спільний) – система повинна надавати широкі можливості розмежування доступу до даних і одночасної роботи багатьох користувачів;

**Multidimensional** (Багатомірний) – багатовимірне концептуальне представлення даних, включаючи повну підтримку для ієрархій і множинних ієрархій (це ключова вимога OLAP);

**Information** (Інформація) – система повинна мати можливість звертатися до будь-якої потрібної інформації, незалежно від її обсягу і місця зберігання.

Слід зазначити, що OLAP-функціональність може бути реалізована різними способами, починаючи з найпростіших засобів аналізу даних в офісних додатках і закінчуючи розподіленими аналітичними системами, заснованими на серверних продуктах.

### 10.2.2. Структура OLAP-куба

Можливість аналізу залежностей між різними параметрами припускає можливість представлення даних у вигляді багатовимірної моделі – гіперкуба (рис. 10.1), або OLAP-куба.

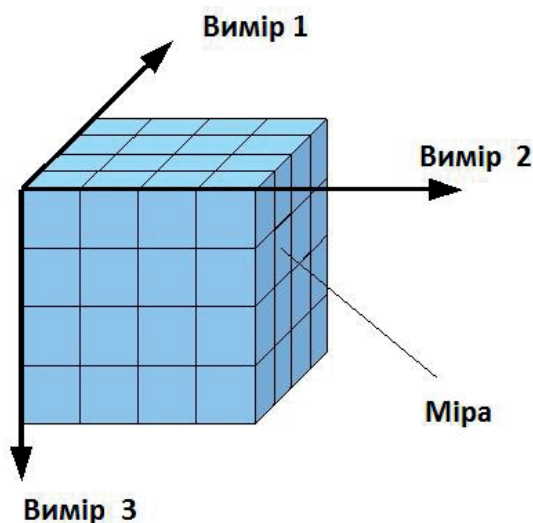


Рис. 10.1. Гіперкуб

Він складається з чисельних мір, розподілених за вимірами. OLAP-куб підтримує всі багатовимірні операції: довільне розміщення вимірів і мір, фільтрація, сортування, угруповання, різні способи агрегації і деталізації.

У процесі аналізу даних часто виникає необхідність побудови залежностей між різними параметрами, число яких може бути значним.

Під **виміром** будемо розуміти послідовність значень одного з аналізованих параметрів. Наприклад, для параметра “час” – це послідовність днів, місяців, кварталів, років.

Осі куба є вимірами, за якими відкладають параметри, що відносяться до аналізованої предметної області, наприклад у випадку задачі продажів товарів – це назви товарів та назви місяців року.

На перетині ліній, паралельних осям вимірів, розташовуються дані, що кількісно характеризують аналізовані факти – міри, наприклад, обсяги продажів, виражені в одиницях продукції.

У найпростішому випадку двовимірного куба виходить таблиця, що показує значення обсягів продажів по товарах і місяцях.

Подальше ускладнення моделі даних можливо по декількох напрямках:

1. Збільшення кількості вимірів даних про продажі не тільки за місяцями і товарами, але і за регіонами. У цьому випадку куб стає тривимірним.

2. Ускладнення вмісту комірки, наприклад, нас може цікавити не тільки рівень продажів, але і чистий прибуток або залишок на складі. У цьому випадку в комірці буде кілька значень.

3. Введення ієрархії в межах одного виміру. Загальне поняття “час” пов’язане з ієрархією значень: рік складається з кварталів, квартал з місяців і т.д.

### 10.2.3. Ієрархія вимірів OLAP-кубів

Кожен з вимірів OLAP-куба може бути представлений у вигляді ієрархічної структури. Наприклад, вимір "Регіон" може мати наступні рівні ієрархії: “країна – область – місто – район”.

Деякі виміри можуть мати декілька рівнів ієрархічного представлення, наприклад вимір “час” може мати представлення “рік – квартал – місяць – день” або “рік – тиждень – день”.

Так само в рамках виміру “географія” можна ввести рівні “країна”, “регіон”, “область” і “місто”.

### 10.2.4. Операції над гіперкубом

Над гіперкубом можуть виконуватися такі операції:

1. **Зріз** (рис. 10.2) – формується підмножина багатовимірному масиву даних, яка відповідає єдиному значенню одного або декількох елементів вимірів (параметрів), що не входять в цю підмножину.

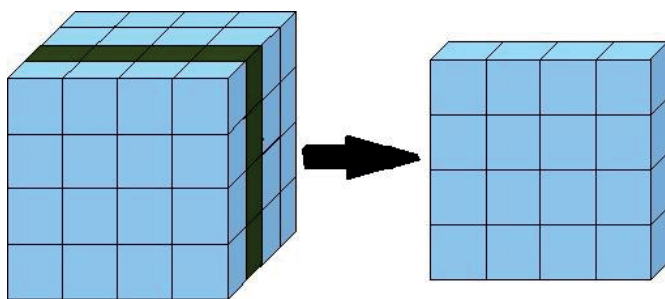


Рис. 10.2. Зріз куба

2. **Обертання** (рис. 10.3) – зміна розташування вимірів, представлених у звіті або на відображуваній сторінці. Наприклад, операція обертання може полягати в перестановці місцями рядків і стовпців таблиці. Крім того, обертанням куба даних

є переміщення позатабличних вимірів на місце вимірів, представлених на відображуваній сторінці, і навпаки.

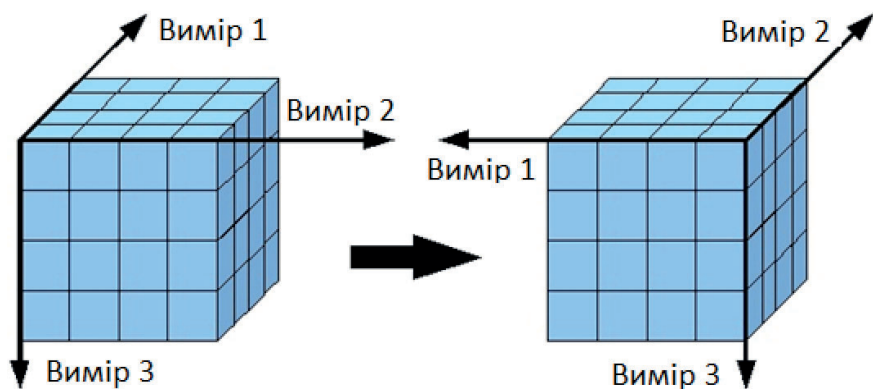


Рис. 10.3. Обертання

3. **Агрегація** (консолідація) (рис. 10.4) і деталізація (рис. 10.5) – операції, які визначають перехід вгору у напрямку від детального представлення даних до агрегованого і навпаки, відповідно. Напрямок агрегації (деталізації) може бути заданий як за ієрархією окремих вимірів, так і згідно інших відношень, встановлених в межах вимірів або між вимірами.

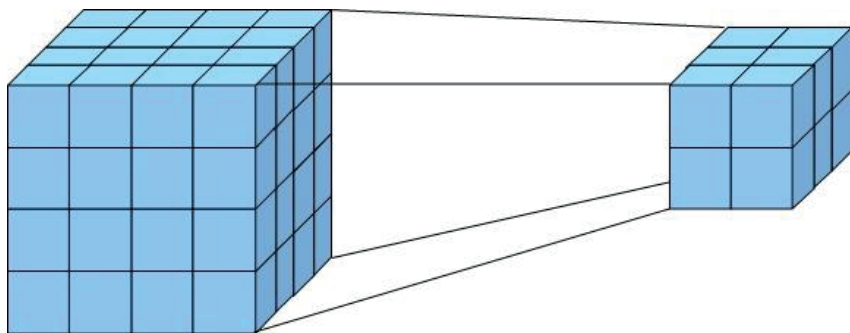


Рис. 10.4. Агрегація

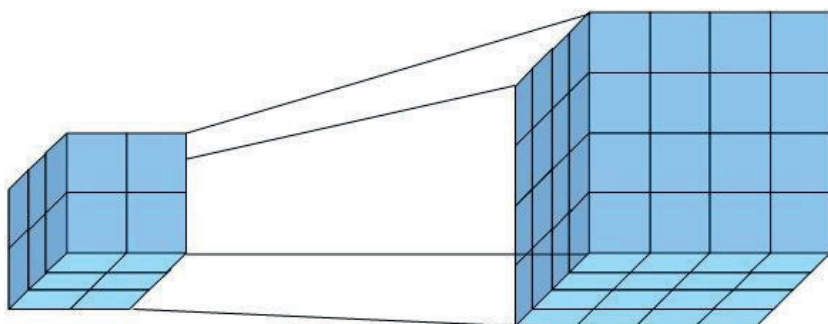


Рис. 10.5. Деталізація

Наприклад, якщо при аналізі даних про продажі в Східній Європі виконати операцію деталізації для виміру “регіон”, то будуть відображені такі елементи, як “Польща”, “Україна”, “Чехія” тощо. В результаті подальшої деталізації елемента “Україна” будуть відображені елементи “Київ”, “Львів”, “Харків” і т.д.

Головна причина використання OLAP для обробки запитів – це швидкість. Реляційні БД зберігають сутності (об’єкти) в окремих таблицях, які зазвичай добре нормалізовані. Ця структура зручна для онлайнових (OLTP) баз даних, але складні багатотабличні запити в ній виконуються відносно повільно. Зручнішою моделлю для виконання запитів (але не для внесення змін) є просторова БД. OLAP робить миттєвий знімок реляційної БД і структурує її в просторову модель для запитів. Заявлений час обробки запитів в OLAP становить близько 0,1% від аналогічних запитів до реляційної БД.

Звернемося до аналізу продажів (рис. 10.6). Припустимо, що керівнику необхідно знати обсяги продажів за деякий період, (наприклад, за місяць). При цьому, компанія продає не один, а множину товарів і має велику кількість торговельних точок (ТТ) – магазинів.

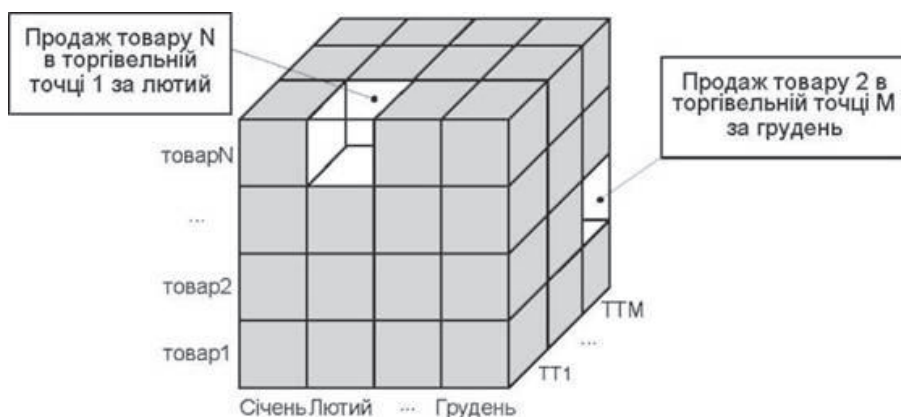


Рис. 10.6. Схема організації розміщення даних для технології OLAP

Допустимо питання, на яке потрібно дати відповідь, – це обсяги продажів всіх товарів в кожній торговельній точці та за кожний місяць.

Відповідь на це питання оформляється у вигляді двовимірної таблиці. Рядками і стовпцями цієї таблиці відповідно будуть назви ТТ і місяці, а значеннями – обсяги продажів всіх товарів.

Хоча OLAP і не представляє собою необхідний атрибут сховища даних, він все ж частіше застосовується для аналізу накопичених в цьому сховищі відомостей.

### 10.3. Сховища даних (Data Warehouse)

#### 10.3.1. Поняття сховища даних

Можна з впевненістю сказати, що практично в будь-якій організації склалася парадоксальна ситуація: інформація начебто, десть і є, її навіть забагато, але вона неструктурована, неузгоджена, розрізнена, не завжди достовірна, її практично неможливо знайти і отримати. В результаті можна говорити про відсутність потрібної інформації при її наявності і навіть надлишку.

Для того, щоб витягувати корисну інформацію з даних, вони повинні бути організовані у спосіб, відмінний від прийнятого в OLTP-системах.

В результаті дані, що застосовуються для аналізу, стали виділяти в окремі спеціальні бази даних, що згодом отримали назву сховищ даних (Data Warehouse).

Наведемо визначення, сформульоване "батьком-засновником" сховищ даних Біллом Інмона: "**Сховище даних** – предметно орієнтований, інтегрований, незмінний набір даних, що підтримує хронологію і здатний бути комплексним джерелом достовірної інформації для оперативного аналізу та прийняття рішень".

В основі концепції сховища даних (СД) лежить розподіл інформації, що використовують в системах оперативної обробки даних (OLTP) і в системах підтримки прийняття рішень (СППР). Такий розподіл дозволяє оптимізувати як структури даних оперативного зберігання для виконання операцій введення, модифікації, знищення та пошуку, так і структури даних, що використовуються для аналізу.

Дані в сховище потрапляють з оперативних систем (OLTP-систем) на основі клієнт-серверної архітектури, СКБД і утиліт підтримки прийняття рішень і призначене для зберігання і аналізу великих об'ємів даних, які призначені для автоматизації бізнес-процесів. Крім того, сховище може поповнюватися за рахунок зовнішніх джерел, наприклад статистичних звітів.

Дані, що надходять у сховище, стають доступні тільки для читання. Потрапивши один раз в сховище, дані там зберігаються і не змінюються. Дані в сховищі можуть лише додаватися.

Навіщо будувати сховища даних – адже вони містять завідомо надлишкову інформацію, яка і так "живе" в базах або файлах оперативних систем? Відповісти можна коротко: аналізувати дані оперативних систем безпосередньо неможливо або дуже важко. Це пояснюється різними причинами, в тому числі розрізненістю даних, зберіганням їх в форматах різних СКБД і в різних "куточках" корпоративної мережі. Але навіть якщо на підприємстві всі дані зберігаються на центральному сервері БД (що буває вкрай рідко), аналітик майже напевно не розбереться в їх складних, часом заплутаних структурах.

Таким чином, завдання сховища – надати "сировину" для аналізу в одному місці і в простій, зрозумілій структурі.

Є і ще одна причина, що виправдує появу окремого сховища – складні аналітичні запити до оперативної інформації гальмують поточну роботу компанії, надовго блокуючи таблиці і захоплюючи ресурси сервера.

Під сховищем можна розуміти не обов'язково гігантське скупчення даних – головне, щоб воно було зручне для аналізу. Взагалі кажучи, для маленьких сховищ призначається окремий термін – Data Marts (кіоски даних).

### **10.3.2. Категорії даних, що зберігаються в сховищах даних**

Всі дані у сховищах даних поділяються на три категорії:

1. **Детальні дані** – дані, які переносяться безпосередньо з підсистем онлайнної обробки транзакцій (OLTP). Транзакція (транзакція) – це здійснення закінчених дій стосовно визначеного об'єкта, що переводить цей об'єкт з одного постійного стану в інший, вона гарантує збереження цілісності бази даних. Детальні дані відповідають елементарним подіям, фіксуються в OLTP-системах та поділяються на:

- виміри – набори даних, необхідні для опису подій (товар, продавець, покупець, магазин, ...);
- факти (міри) – дані, що відображають сутність події (кількість проданого товару, сума продажів, ...);



2. **Агреговані (узагальнені) дані** – дані, одержувані на підставі детальних даних шляхом підсумовування по певним вимірам;

3. **Метадані** – дані про дані, що містяться в сховищах даних. Вони можуть описувати:

- об’єкти предметної області, інформація про які міститься в сховищах даних;
- категорії користувачів, що використовують дані в сховищах даних;
- місця і способи зберігання даних;
- дії, що виконуються над даними;
- час виконання різних дій над даними;
- причини виконання різних дій над даними.

### 10.3.3. Властивості сховища даних. Вітрина даних

Сховища даних мають наступні властивості:

**предметна орієнтація** – інформацію організовано відповідно до основних аспектів діяльності;

**інтегрованість даних** – дані в сховище надходять з різних джерел і відповідно агрегуються, тобто об’єднуються;

**стабільність, інваріантність у часі** – записи в сховищі ніколи не змінюються, являючи собою відбитки даних, зроблені у певний час;

**мінімізація надмірності інформації** – перед завантаженням у сховище дані сортуються, фільтруються, зберігаються у певній послідовності, а також формується деяка підсумкова інформація.

В сховищах даних надмірність даних є мінімальною (приблизно 1%), оскільки:

- при завантаженні у сховище, дані сортуються і фільтруються;
- інформація у сховищах зберігається в хронологічному порядку, що майже повністю виключає перекриття даних;
- при завантаженні у сховище, дані зводяться до єдиного формату, включаючи обчислення сумарних (агрегованих) показників.

Сервери багатовимірних баз даних можуть зберігати дані по-різному, крім агрегованих показників формується ще й додаткова інформація: поля часу, дати,

адресні посилання, таблиці метаданих тощо. Це приводить до значного збільшення інформації. Вхідний масив розміром 200 Mb може розростись до об'єму 5 Gb. Сховище даних повинне бути оптимально організованою базою даних, яка забезпечує максимально швидкий і оперативний пошук інформації.

**Вітрина даних** – це спрощений варіант сховища даних, що містить лише тематично об'єднані, агреговані дані.

Вітрина даних містить дані, орієнтовані на конкретного користувача, і є істотно меншою за обсягом від сховища даних. Для реалізації вітрини даних потрібно менше витрат, адже вони можуть будуватися як самостійно, так і разом зі сховищами даних. Вітрини даних впроваджуються набагато швидше і швидше видно ефект від їх використання. Недоліками вітрин даних є багаторазове зберігання одних і тих же даних в різних вітринах даних і відсутність консолідованості на рівні предметної області.

Зазвичай інформація потрапляє до вітрин даних зі сховищ, в цьому випадку вітрини даних називаються залежними. Можлива також ситуація, коли джерелом інформації для поповнення вітрин даних служать безпосередньо OLTP-системи. Такі вітрини даних, що отримали назву незалежних, як правило, розглядаються як тимчасове рішення, що дозволяє достатньо швидко і з невеликими витратами вирішити найбільш важливі завдання, оцінити переваги нового підходу, сформулювати деякі рекомендації для більш масштабного проекту розробки загального сховища даних.

Можливо також поєднання сховищ даних та вітрин даних в рамках однієї системи підтримки прийняття рішень. Сховище даних в цьому випадку представляє собою єдине джерело даних для всієї предметної області, а вітрини даних є підмножинами даних зі сховища, організованими для подання інформації за тематичними розділами даної області. У тому випадку, якщо користувачеві даних у вітрині даних є недостатньо, то він може звернутися до сховища даних.

#### **10.3.4. Глобальне сховище даних**

Глобальне сховище даних складається з трьох рівнів:

- 1) сховище агрегованих даних;
- 2) вітрини даних, які базуються на інформації зі сховища даних;

3) клієнтські робочі місця, на яких встановлено засоби оперативного аналізу даних.

У розпорядженні розробників прикладних програмних засобів є три різні технології роботи з базами даних:

- DAO (Data Access Objects) – доступ до локальних баз даних;
- RDO (Remote Data Objects) – доступ до віддалених баз даних;
- ADO (ActiveX Data Objects) – доступ до Windows-застосунків через Інтернет, що в основному використовується з міркувань безпеки.

Одним з перспективних напрямів удосконалення доступу до даних є гнучке конфігурування системи, коли розподіл між клієнтською і серверною частинами можливий за допомогою використання механізму віддалених процедур.

Поряд з потоками даних існують і потоки метаданих, які розміщуються в депозитарії. Він дає змогу визначити семантичну структуру застосунку у вигляді опису термінів предметної галузі, їхні взаємозв'язки й атрибути.

### 10.3.5. Архітектура та наповнення сховищ даних

Типова архітектура сховища даних наведена на рис. 10.7.

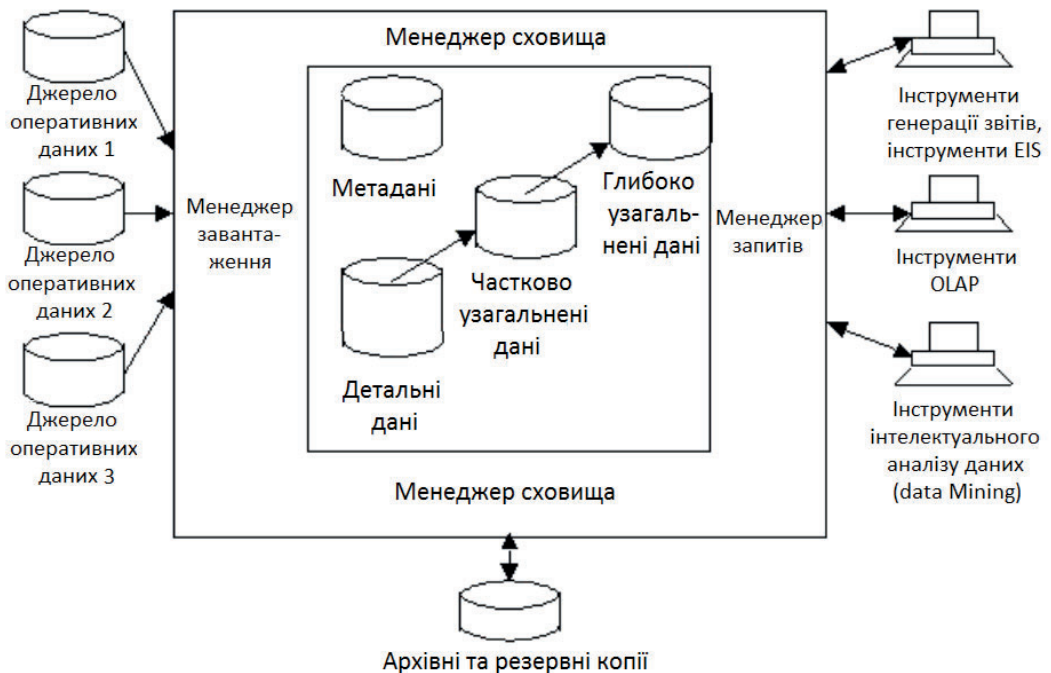


Рис. 10.7. Типова архітектура сховища даних

*Менеджер завантаження (load manager)* виконує операції, пов'язані з отриманням і завантаженням даних в сховищі.

*Менеджер сховища (warehouse manager)* виконує операції, пов'язані з управлінням інформацією, розміщеної в сховище даних:

- аналіз несуперечності даних;
- створення індексів і уявлень для базових таблиць;
- денормалізація даних (за необхідності);
- узагальнення даних (за необхідності);
- резервне зберігання та архівування.

*Менеджер запитів (query manager)* виконує операції, пов'язані з управлінням призначеними для користувача запитами.

Наповнення інформаційних сховищ відбувається в декілька етапів:

- **екстракція** (витяг) – імпорт даних у сховище з інформаційних підсистем, виробничих відділів та інших джерел;
- **трансформація** – консолідація, агрегація даних, розбиття їх на фракції, коригування та трансформування у відповідні формати;
- **завантаження** даних у сховище, синхронізація з датою або зовнішніми подіями.

**Обслуговуюча функція** інформаційних сховищ полягає в:

- копіюванні баз даних;
- налаштуванні;
- тиражуванні;
- надсиланні застарілих баз даних до архіву;
- управлінні правами користувачів;
- створенні та редагуванні графічних діаграм баз даних, тощо.

Як правило, дані копіюються в сховище з оперативних баз даних і інших джерел відповідно до певних правил.

Типи архівації у сховищах даних поділяють на:

- звичайна;
- копіювальна;

- додаткова;
- диференціальна;
- щоденна.

Звичайна архівація каталогів на всіх серверах здійснюється раз на тиждень у вихідні дні, а диференціальна – щодня в робочі дні. У річному архіві, як правило, зберігаються дані останнього тижня місяця. Усі зміни в каталозі сервера, а також в особистих і загальних сховищах записуються у файли, які називаються журналами транзакцій (*transaction log files*).

Під час виконання додаткової архівації каталогу або інформаційного сховища архівуванню підлягають лише журнали транзакцій.

Для ефективної роботи зі сховищем даних, необхідно зібрати максимум інформації про процес. Наприклад, для прогнозування обсягів продажів можуть бути використані бази даних облікових систем компанії, маркетингові дані, відгуки клієнтів, дослідження конкурентів і т.д.

#### **10.3.6. Моделі сховищ даних**

Забезпечуючи багатовимірне концептуальне уявлення з боку призначеного для користувача інтерфейсу до вихідної бази даних, всі продукти OLAP діляться на декілька класів за типом вихідної БД. Багатовимірний гіперкуб, який використовується в OLAP-технології, може бути реалізований в рамках реляційної моделі або існувати як окрема база даних спеціальної багатовимірної структури. Залежно від цього прийнято розрізняти багатовимірний (MOLAP) і реляційний (ROLAP) підходи до побудови сховища даних.

##### **10.3.6.1. MOLAP (Multidimensional OLAP)**

У MOLAP-моделі багатовимірне представлення даних реалізується фізично. У спеціалізованих СКБД, основаних на багатовимірному поданні даних, дані організовані не в формі реляційних таблиць, а у вигляді впорядкованих багатовимірних масивів:

- 1) *гіперкубів* (всі збережені в базі даних клітинки повинні мати однакову розмірність, тобто перебувати в максимально повному базисі вимірювань);

2) *полікубів* (кожна змінна зберігається з власним набором вимірювань, і всі пов'язані з цим складнощі обробки перекладаються на внутрішні механізми системи).

Використання багатовимірних баз даних в системах оперативної аналітичної обробки має такі переваги:

- Висока продуктивність. У разі використання багатовимірних СКБД пошук і вибірка даних здійснюється значно швидше, ніж при багатовимірному концептуальному погляді на реляційну базу даних, так як багатовимірна база даних денормалізована, містить заздалегідь агреговані показники і забезпечує оптимізований доступ до підзапитних клітинок.
- Багатовимірні СКБД легко справляються з завданнями включення в інформаційну модель різноманітних вбудованих функцій, тоді як об'єктивно існуючі обмеження мови SQL роблять виконання цих завдань на основі реляційних СКБД досить складним, а іноді і неможливим.

Недоліки MOLAP-моделі:

- Багатовимірні СКБД не дозволяють працювати з великими базами даних.
- Багатовимірні СКБД в порівнянні з реляційними дуже неефективно використовують зовнішню пам'ять. У переважній більшості випадків інформаційний гіперкуб є сильно розрідженим, а оскільки дані зберігаються в упорядкованому вигляді, невизначені значення вдається видалити тільки за рахунок вибору оптимального порядку сортування, що дозволяє організувати дані в максимально великі безперервні групи. Але навіть в цьому випадку проблема вирішується тільки частково. Крім того, оптимальний з точки зору зберігання розріджених даних порядок сортування швидше за все не буде збігатися з порядком, який найчастіше використовується в запитах.

Отже, використання багатовимірних СКБД виправдане тільки за таких умов:

1. Обсяг вихідних даних для аналізу не дуже великий (не більше декількох гігабайт), тобто рівень агрегації даних досить високий.
2. Набір інформаційних вимірювань стабільний (оскільки будь-яка зміна в їх структурі майже завжди вимагає повної перебудови гіперкуба).

3. Час відповіді системи на нерегламентовані запити є найбільш критичним параметром.

Приклади OLAP-серверів, що використовують MOLAP-архітектуру: Oracle Express Server фірми Oracle, IBM Informix MetaCube, IBM DB2 OLAP, Arbor Essbase.

### 10.3.6.2. ROLAP (Relational OLAP)

Системи оперативної аналітичної обробки реляційних даних (ROLAP) дозволяють представляти дані, що зберігаються в реляційній базі, в багатовимірній формі, забезпечуючи перетворення інформації в багатовимірну модель через проміжний шар метаданих. В цьому випадку гіперкуб емулюється СКБД на логічному рівні.

Для більшості сховищ даних найбільш ефективним способом моделювання N-мірного куба фактів є *схема "зірка" (star schema)*.

Основними складовими структури сховищ даних є таблиця фактів (fact table) і таблиці вимірювань (dimension tables).

**Таблиця фактів** є основною таблицею сховища даних. Як правило, вона містить відомості про об'єкти або події, сукупність яких буде надалі аналізуватися. Якщо проводити аналогію з багатовимірною моделлю, то рядок таблиці фактів відповідає осередку гіперкуба. Зазвичай говорять про чотири типи фактів, що найбільш часто зустрічаються. До них відносяться:

- факти, пов'язані з транзакціями (Transaction facts). Вони засновані на окремих подіях (типовими прикладами яких є телефонний дзвінок або зняття грошей з рахунку за допомогою банкомату);
- факти, пов'язані з "миттєвими знімками" (Snapshot facts). Засновані на стані об'єкта (наприклад, банківського рахунку) в певні моменти часу, наприклад на кінець дня чи місяця. Типовими прикладами таких фактів є обсяг продажів за день або денна виручка;
- факти, пов'язані з елементами документа (Line-item facts). Засновані на тому чи іншому документі (наприклад, рахунку за товар або послуги) і містять детальну інформацію про елементи цього документа (наприклад, кількості, ціні, відсотку знижки);

- факти, пов'язані з подіями або станом об'єкта (Event or state facts). Представляють виникнення події без подробиць про нього (наприклад, просто факт продажу або факт відсутності такої без інших подробиць).

Таблиця фактів індексується за складним ключем, складеним з ключів окремих змін. При цьому як ключові, так і деякі неключові поля таблиці фактів повинні відповідати майбутнім вимірам OLAP-куба. Крім цього, таблиця фактів містить одне або кілька числових полів, на підставі яких в подальшому будуть отримані агрегатні дані.

**Таблиця вимірів** містить незмінні або дані, що рідко змінюються. У кожній таблиці вимірювань перераховані можливі значення одного з вимірів гіперкуба. У переважній більшості випадків ці дані представляють собою по одному запису для кожного члена нижнього рівня ієрархії у вимірі. Таблиці вимірювань також містять як мінімум одне описове поле (зазвичай з ім'ям члена вимірювання) і, як правило, цілочисельне ключове поле (зазвичай це сурогатний ключ) для однозначної ідентифікації члена вимірювання. Кожна таблиця вимірювань повинна знаходитися у відношенні "один до багатьох" з таблицею фактів.

Причина, за якою дана схема названа "зіркою", досить очевидна: в центрі розміщуються факти і агреговані дані, а "променями" є виміри. Кожна "зірка" описує певну дію, наприклад, продаж товару, його відвантаження, надходження коштів й інше. Одна таблиця фактів (fact table) і декілька таблиць вимірів (dimensional table) пов'язані ідентифікуючими зв'язками. При цьому агреговані дані зберігаються спільно з початковими (рис. 10.8).

У складних задачах з багаторівневими вимірами використовуються різні розширення схеми "зірка" – **схема "сніжинка"** (snowflake schema).

Це розширення може проявлятися в двох різновидах.

1. У разі великого числа складних атрибутів в таблиці вимірювань, деякі атрибути можуть бути деталізовані в окремих таблицях вимірів. Іншими словами, окремі вимірювання містяться не в одній, а в декількох пов'язаних між собою таблицях. Додаткові таблиці вимірювань в такій схемі, зазвичай відповідні верхніх рівнів ієрархії вимірювання і знаходяться в співвідношенні "один до



багатьох" в головній таблиці вимірювань, що відповідає нижньому рівню ієрархії, іноді називають консольними таблицями (outrigger table).

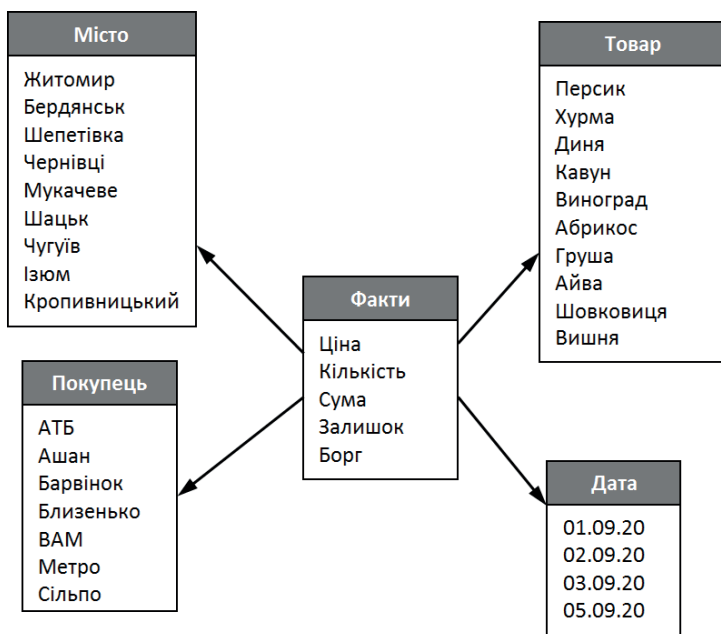


Рис. 10.8. Схема організації сховища даних за типом "зірка"

Наприклад, з таблиці "Місто" можна вилучити опису країни та типу населеного пункту (залишивши лише їх ключі) і зберігати їх в окремих додаткових таблицях. Це зменшить ступінь дублювання інформації, але знижує швидкість виконання запитів, оскільки збільшує ступінь нормалізації. Тому навіть при наявності ієрархічних вимірів з метою підвищення швидкості виконання запитів до сховища даних нерідко перевага віддається схемою "зірка".

2. Інше розширення пов'язано зі створенням окремих таблиць фактів для всіх можливих поєднань рівнів узагальнення різних вимірів.

Збільшення числа таблиць фактів в базі даних може виникати не тільки з множинності рівнів різних вимірів, але і з тієї обставини, що в загальному випадку факти мають різні множини вимірів. При абстрагуванні від окремих вимірювань, користувач повинен отримувати проекцію максимально повного гіперкуба, причому далеко не завжди значення показників в ній повинні бути результатом елементарного сумування. Таким чином, при великому числі

незалежних вимірювань необхідно підтримувати багато таблиць фактів, відповідних кожному можливому поєднанню обраних в запиті вимірювань.

Це дозволяє домогтися кращої продуктивності, але часто призводить до надмірності даних і до значних ускладнень в структурі бази даних, в якій виявляється величезна кількість таблиць фактів.

Отже, схема типу “сніжинка” є різновидом схеми типу “зірка”. Таблиця фактів і декілька таблиць вимірів теж пов’язані ідентифікуючими зв’язками. Агреговані дані можуть зберігатися окремо від початкових (рис. 10.9).

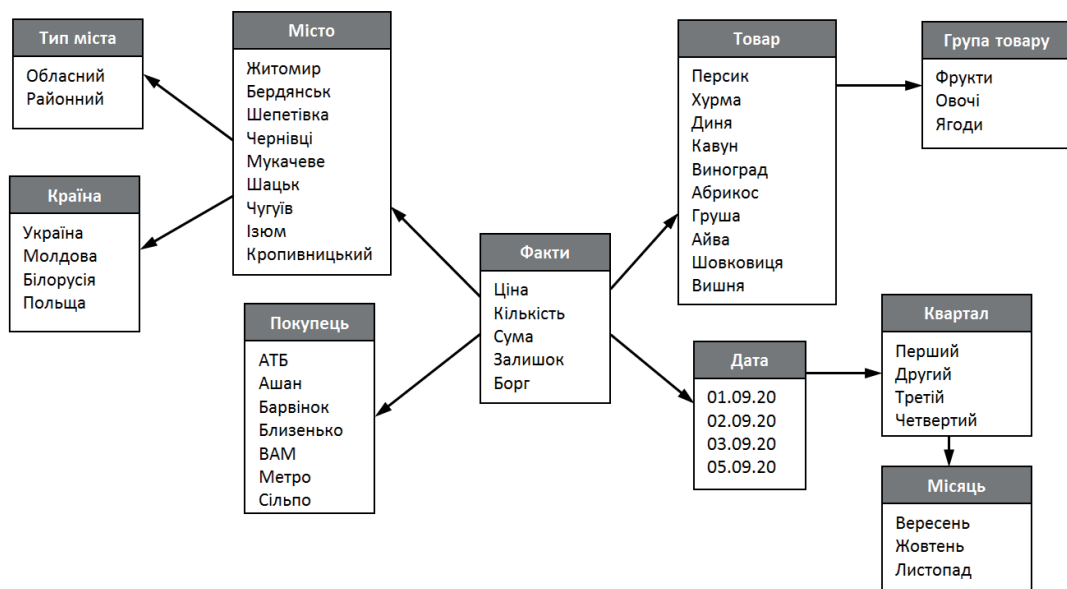


Рис. 10.9. Схема організації сховища даних за типом “сніжинка”

Переваги використання реляційних баз даних в системах аналітичної оперативної обробки:

1. При використанні ROLAP розмір сховища не є таким критичним параметром, як у випадку MOLAP.
2. Внесення змін до структури вимірів не вимагає фізичної реорганізації бази даних, як у випадку MOLAP.
3. Реляційні СКБД забезпечують значно вищий рівень захисту даних і хороші можливості розмежування прав доступу.

Головний недолік ROLAP в порівнянні з багатовимірними СКБД – менша продуктивність.

Приклади OLAP-серверів, що використовують ROLAP-архітектуру: IBM Informix Red Brick, HighGate Project фірми Sybase, Microsoft SQL Server 2000 Analysis Services фірми Microsoft.

### **10.3.6.3. Інші моделі побудови сховищ даних**

Гібридні системи (Hybrid OLAP, HOLAP) розроблені з метою поєднання переваг та мінімізації недоліків, властивих попереднім класам. До цього класу належить Media / MR компанії Speedware. За твердженням розробників, він об'єднує аналітичну гнучкість і швидкість відповіді MOLAP з постійним доступом до реальних даних, властивим ROLAP.

Приклади OLAP-серверів, що використовують HOLAP-архітектуру: Microsoft SQL Server 2000 Analysis Services фірми Microsoft, SAS Institute.

Крім перерахованих коштів існує ще один клас – інструменти керованого середовища запитів (MQE), доповнені функціями OLAP або інтегровані з зовнішніми засобами, які виконують такі функції. Ці добре розвинені системи здійснюють вибірку даних з вихідних джерел (реляційні бази даних, електронні таблиці), перетворюють їх і поміщають в динамічну багатовимірну базу даних, що функціонує на клієнтській станції кінцевого користувача. Побудований куб даних аналізується засобами багатовимірного OLAP, зберігається і супроводжується локально.

Переваги:

- відносна простота інсталяції, адміністрування та супроводу;
- здатність кожного користувача створювати свої власні куби даних.

Основними представниками цього класу є BusinessObjects однойменної компанії, PowerPlay компанії Cognos.

## **10.4. Інтелектуальний аналіз даних (Data Mining)**

Інтелектуальний аналіз даних (ІАД - Data Mining – це процес підтримки прийняття рішень, заснований на пошуку в даних схованих закономірностей. При цьому накопичені відомості автоматично узагальнюються до інформації, яка може бути охарактеризована як знання.

*Концепція інтелектуального аналізу даних* визначає завдання пошуку функціональних і логічних закономірностей в накопиченій інформації, побудову

моделей і правил, які пояснюють знайдені аномалії і / або прогнозують розвиток деяких процесів.

Орієнтовна структура інформаційно-аналітичної системи, побудованої на основі сховища даних, показана на рис. 10.10. У конкретних реалізаціях окремі компоненти цієї схеми часто відсутні.

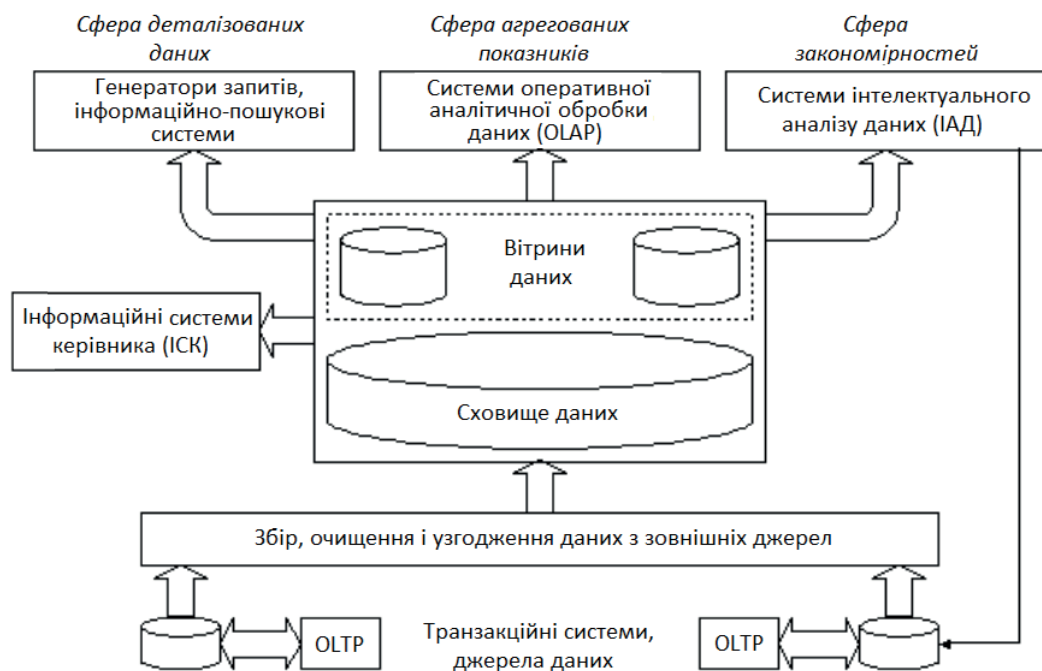


Рис. 10.10. Структура інформаційно-аналітичної системи на основі сховища даних

У загальному випадку процес ІАД складається з трьох стадій (рис. 10.11):

- виявлення закономірностей (вільний пошук);
- використання виявлених закономірностей для передбачення невідомих значень (прогностичне моделювання);
- аналіз винятків, призначений для виявлення і тлумачення аномалій в знайдених закономірностях.

Іноді в явному вигляді виділяють проміжну стадію перевірки достовірності знайдених закономірностей між їх перебуванням і використанням (стадія валідації).

## Контрольні запитання

1. Що таке великі дані?
2. Які технології призначені для роботи з великими даними?
3. Назвіть принципи роботи з великими даними.
4. Що таке горизонтальна масштабованість?
5. Яке основне призначення технології OLAP?
6. Скільки вимірів може мати OLAP гіперкуб?
7. Які є операції над гіперкубом?
8. Що таке сховище даних та кіоск даних?
9. Яка відмінність сховища даних від БД?
10. Що таке вітрина даних?
11. З яких рівнів складається глобальне сховище даних?
12. Проаналізуйте типову архітектуру сховищ даних.
13. Для чого потрібні журналами транзакцій в сховищах даних?
14. Які є моделі сховищ даних?
15. Які відмінності між гіперкубом та полікубом?
16. За яких умов використання багатовимірних СКБД виправдане?
17. Яка схема є найбільш ефективним способом моделювання N-мірного куба фактів для більшості сховищ даних?
18. В чому схожі та відмінні схеми зірка та сніжинка?
19. В чому полягає концепція інтелектуального аналізу даних?
20. На які групи діляться методи інтелектуального аналізу даних?

# Розділ 4. АДМІНІСТРУВАННЯ ТА ЗАХИСТ БАЗ ДАНИХ

## Тема 11. Перегляди, курсори та тригери в MySQL

### 11.1. Поняття переглядів VIEW

#### 11.1.1. Призначення переглядів

Типи таблиць, з якими ми працювали до цього часу, називаються базовими таблицями. Це таблиці, які містять дані. Однак є інший вид таблиць – перегляди.

Перегляд (VIEW) – об'єкт даних, який не містить жодних даних його власника. Це – тип таблиці, вміст якої вибирається з інших таблиць за допомогою виконання запиту типу SELECT.

Фактично перегляди схожі з вікнами в базових таблицях. Вони показують частину таблиці, але не обов'язково всю таблицю. Вони можуть обмежувати показ певних полів або рядків, а також показувати поля під псевдонімами, здійснювати ряд дій над полями – від сортування до агрегативних функцій. Таким чином, призначення переглядів – це своєрідний захист даних БД.

Перегляд VIEW з точки зору реалізації відрізняється від таблиці – це тимчасовий набір, який не зберігається в базі даних, там зберігається його визначення View. Перегляд View створюється і працює на стороні сервера, може містити складні зв'язки між таблицями і бути джерелом даних для компонентів для опрацювання баз даних. Оскільки у View можна задавати поля з окремих зв'язаних таблиць, тому його часто використовують адміністратори баз даних для задавання рівня доступу.

Доцільно використовувати перегляди VIEW також при звертанні застосунку користувача до БД із складним запитом, запитом до декількох таблиць тощо.

Перегляди VIEW можуть ґрунтуватися як на таблицях, так і на інших переглядах VIEW, тобто перегляди можуть бути вкладеними (до 32 рівнів вкладення).

### 11.1.2. Переваги використання переглядів

Використання переглядів VIEW надає:

1. можливість гнучкого налаштування прав доступу до даних за рахунок того, що права надаються не на таблицю, а на перегляд VIEW. Це дуже зручно в тому разі, якщо користувачеві потрібно надати права на окремі рядки таблиці або можливість отримання не самих даних, а результату деяких дій над ними;
2. можливість розділити логіку зберігання даних і програмного забезпечення. Можна міняти структуру даних, не зачіпаючи програмний код, потрібно лише створити перегляд VIEW, аналогічний таблиці, до якої раніше звертався застосунок. Це дуже зручно, коли немає можливості змінити програмний код або до однієї бази даних звертаються декілька застосунків з різними вимогами до структури даних;
3. зручність у використанні за рахунок автоматичного виконання таких дій, як доступ до певної частини рядків і / або стовпців, отримання даних з декількох таблиць і їх перетворення за допомогою різних функцій.

### 11.1.3. Обмеження для переглядів в MySQL

До версії MySQL 5.1 при створенні переглядів VIEW необхідно враховувати наступні обмеження:

- таблиці та перегляди VIEW, присутні у визначенні перегляду VIEW, повинні існувати;
- не можна створювати тимчасовий перегляд VIEW та перегляд VIEW на основі тимчасових таблиць;
- у визначенні перегляду VIEW не можна використовувати підзапит в частині FROM;
- у визначенні перегляду VIEW не можна використовувати системні і призначені для користувача змінні; всередині збережених процедур у визначенні перегляду VIEW можна використовувати локальні змінні або параметри процедури;
- у визначенні перегляду VIEW не можна використовувати параметри підготовлених виразів (PREPARE);

- не можна “повісити” тригер на перегляд VIEW;
- тільки перегляди VIEW, що задовольняють ряду вимог, допускають запити типу UPDATE, DELETE і INSERT.

#### 11.1.4. Алгоритми переглядів

При створенні перегляду VIEW є можливість явно вказати, який алгоритм використовується за допомогою необов’язковою конструкції

[ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}].

Інструкція UNDEFINED означає, що MySQL сам вибирає, який алгоритм використовувати при зверненні до перегляду VIEW. Це значення встановлене за замовчуванням, якщо дана конструкція відсутня.

У разі алгоритму MERGE, MySQL при зверненні до перегляду VIEW додає в оператор SELECT виклику перегляду VIEW відповідну частину з визначення перегляду VIEW і виконує оператор, що утворився таким об’єднанням.

У разі алгоритму TEMPTABLE, MySQL заносить вміст перегляду VIEW в тимчасову таблицю, над якою потім виконується оператор SELECT, що звертається до перегляду VIEW.

Варто зазначити, що немає серйозних причин явно вказувати алгоритм при створенні перегляду VIEW, оскільки:

1. У разі використання UNDEFINED MySQL намагається використовувати MERGE всюди, де це можливо, так як цей алгоритм більш ефективний, ніж TEMPTABLE і, на відміну від нього, не робить перегляду VIEW, які не оновлюються.

2. Якщо явно вказати алгоритм MERGE, а визначення перегляду VIEW містить конструкції, що забороняють його використання, тоді MySQL видасть попередження і встановить значення UNDEFIND.

#### 11.1.5. Створення переглядів

Для створення перегляду VIEW використовується оператор CREATE VIEW, який має наступний синтаксис:

**CREATE [OR REPLACE]**

**[ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]**



**VIEW <ім'я\_перегляду>**

**[(список\_полів\_перегляду)]**

**AS <оператор\_вибору>**

**[WITH [CASCADED | LOCAL] CHECK OPTION]**

1. Використання необов'язкової конструкції OR REPLACE дозволяє при створенні перегляду VIEW під іменем, яке вже існує в БД, уникнути помилки. При цьому існуючий перегляд VIEW буде затерто новоствореним.

2. Використання необов'язкової конструкції ALGORITHM визначає, який алгоритм буде реалізовано при зверненні до перегляду VIEW. Детальніше алгоритми розглянемо далі.

3. Використання конструкції VIEW <ім'я\_перегляду> є обов'язковим. При задаванні імені перегляду VIEW необхідно дотримуватись правил написання імен змінних, при цьому бажано контролювати унікальність (неповторюваність) імен переглядів VIEW.

4. Конструкція [(список\_полів\_перегляду)] є необов'язковою, оскільки за замовчуванням колонки перегляду VIEW мають ті ж імена, що і поля, які повертаються оператором SELECT у визначенні перегляду VIEW. При явному задаванні імен полів перегляду VIEW список\_полів\_перегляду повинен включати по одному імені для кожного поля, розділених комою у круглих дужках. Зазвичай використовують назви полів з таблиць. Але у списку полів можна також задавати нові імена полям, а також створювати обчислювальні поля. Якщо ж задаються нові імена полів, тоді необхідно робити це для кожного поля в перегляді VIEW.

*Порада.* Бажано використовувати явне задавання імен полів перегляду VIEW.

5. Використання конструкції AS <оператор\_вибору> є обов'язковим, де оператор\_вибору визначається оператором SELECT, тобто звичайним SQL-запитом на вибірку. В такому SQL-запиті використовують умову WHERE, але не використовують сортування ORDER BY і не можна використовувати параметри.

6. Конструкція [WITH [CASCADED | LOCAL] CHECK OPTION] є необов'язковою, при її використанні всі рядки, що добавляються або змінюються, будуть перевірятися на відповідність визначенню перегляду VIEW (не дозволяє користувачеві вставити запис, що виходить за межі допустимих значень полів). У

випадку невідповідності дана зміна виконана не буде. Варто зазначити, що при вказівці даної конструкції для перегляду VIEW, який не оновлюється, виникає помилка і перегляд VIEW створено не буде.

**Приклад 11.1.** Створити перегляд VIEW з іменем ViewNameK, який відображатиме з таблиці Katalog тільки ті записи, у яких назва книжки NameK починається на букву Р (рис.11.1).

При попередженні, що база даних не вибрана, необхідно використовувати конструкцію <база\_даних>.<ім'я\_перегляду> та < база\_даних >.< ім'я\_таблиці >.

Після виконання запиту, наведеного на рис. 11.1, у вікні провідника об'єктів Object Browser інструменту MySQL Workbench для розроблення і адміністрування СКБД, відобразиться створений перегляд VIEW.

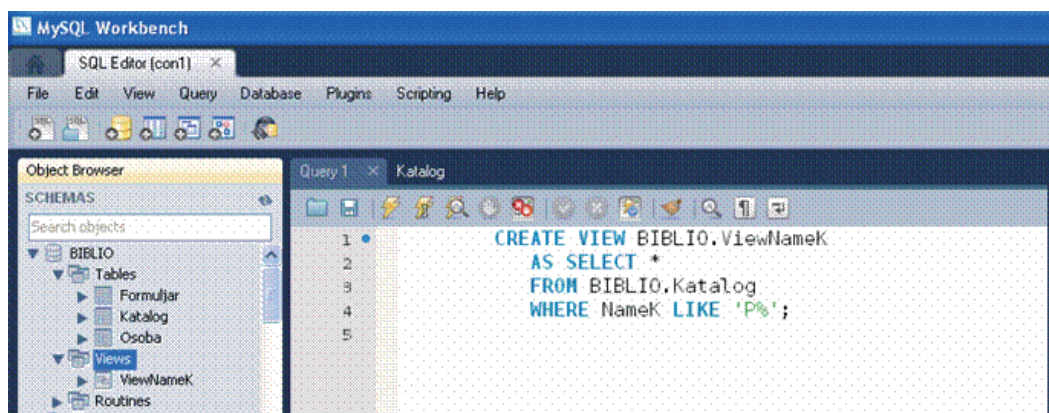


Рис. 11.1. Створення перегляду VIEW з іменем ViewNameK

Для відображення роботи створеного перегляду VIEW використовується команда SELECT, яка аналогічна у випадку вибірки з базової таблиці (рис. 11.2):

**SELECT <список\_полів\_перегляду>**

**FROM <ім'я\_перегляду>...**

Але з іншої сторони, оператор SELECT є в самому визначенні перегляду VIEW, тобто виходить вкладена конструкція – запит в запиті. При цьому, деякі конструкції оператора SELECT можуть бути присутніми в обох операторах. Можливі три варіанти розвитку подій:

- вони обидві будуть виконані;
- одна з них буде проігнорована;
- результат невизначений.

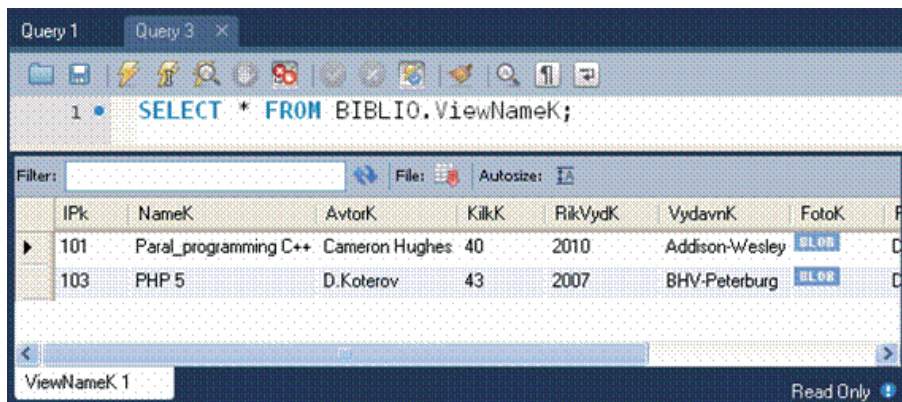


Рис.11.2. Відображення перегляду VIEW з іменем ViewNameK

Розглянемо докладніше ці випадки:

1. Якщо в обох операторах зустрічається умова WHERE, то обидві ці умови будуть виконані, так як би вони були об'єднані оператором AND.

2. Якщо у визначенні перегляду VIEW є конструкція ORDER BY, тоді вона буде працювати тільки за умови, що така конструкція відсутня в зовнішньому операторі SELECT, який звертається до перегляду VIEW. При наявності конструкції сортування ORDER BY у зовнішньому операторі SELECT, наявна конструкція сортування у перегляді VIEW буде проігнорована.

3. При наявності в обох операторах SELECT модифікаторів, що впливають на механізм блокування, таких як HIGH\_PRIORITY, результат їх спільної дії невизначений. Для уникнення невизначеності не рекомендується використовувати подібні модифікатори у визначенні перегляду VIEW.

## 11.2. Модифікації переглядів

### 11.2.1. Групові перегляди

Групові перегляди VIEW – які містять інструкцію GROUP BY або базуються на інших групових переглядах VIEW.

Але особливістю групових переглядів є те, що сортування ORDER BY у визначенні переглядів не використовується.

**Приклад 11.2.** Створити перегляд VIEW з іменем ViewSumBook, який виводитиме сумарну кількість книг кожного з видавництв, наявних в бібліотеці:

```
CREATE VIEW BIBLIO.ViewSumBook
```

```
AS SELECT k.VYDk, SUM(k.KPRIMk) AS SUM
FROM BIBLIO.Katalog k
GROUP BY k.VYDk;
```

Виклик створеного перегляду здійснюється командою SELECT:

```
SELECT *
FROM BIBLIO.ViewSumBook;
```

### 11.2.2. Перегляди і об'єднання та підзапити

У визначенні перегляду може бути використаний будь-який запит, що відповідає синтаксису SQL та може виводити інформацію з будь-якого числа базових таблиць, або з інших переглядів.

Для об'єднання таблиць у перегляді можна застосовувати всі відомі способи об'єднання таблиць в операторі SELECT *за винятком* об'єднань UNION та UNION ALL, які заборонено використовувати у переглядах.

**Приклад 11.3.** Створити перегляд VIEW з іменем ViewDebtor, який виводитиме інформацію про абонентів, які мають на руках книжки з бібліотеки:

```
CREATE VIEW BIBLIO.ViewDebtor
AS SELECT os.Ido, os.PIPO, os.ADRo, os.RNo
FROM BIBLIO.Osoba os
INNER JOIN BIBLIO.Formular frm
ON os.idOs = frm.IdO
WHERE frm.DataOut IS NULL OR frm.DataOut > NOW();
```

Тепер можна вибирати будь-які дані абонентів-боржників.

**Приклад 11.4.** Використовуючи перегляд ViewDebtor, вивести всіх Іванів, які заборгували книги:

```
SELECT *
FROM BIBLIO.ViewDebtor
WHERE PIPO LIKE '%Іван%';
```

Створені перегляди можна об'єднувати з іншими таблицями, переглядами, або з базовими таблицями та переглядами.

**Приклад 11.5.** Використовуючи перегляд ViewDebtor, вивести наступну інформацію про всіх абонентів-боржників: Прізвище, рік народження та дату

реєстрації. Так як у створеному перегляді відсутнє поле «Дата реєстрації», тоді необхідно об'єднати цей перегляд з таблицею, яка містить таке поле і зв'язати їх згідно синтаксису SQL:

```
SELECT v.PIPO, v.RNo, o.DRO
FROM BIBLIO.ViewDebtor v
JOIN BIBLIO.Osoba o
ON v.Ido=o.Ido;
```

### 11.2.3. Перегляди та підзапити

Аналогічно можна об'єднувати перегляди з іншими таблицями та переглядами, перегляди VIEW можуть також містити і підзапити, включаючи вкладені підзапити, при цьому вважаючи, що перегляд – це аналог запиту SELECT.

Під час створення переглядів можна використовувати підзапити в якості розрахункових полів, підзапити в реченні FROM, вкладені підзапити в умові WHERE та HAVING, підзапити з операторами EXIST, IN, ANY, ALL тощо.

**Приклад 11.6.** Створити перегляд, який відображатиме кількість прочитаних книг кожним абонентом:

```
CREATE VIEW BIBLIO.ViewCountBook
AS SELECT PIP,
(SELECT COUNT(IdKn) FROM BIBLIO. Formular
WHERE Osoba.IdO = Formular.IdOs) AS KILK
FROM BIBLIO.Osoba;
```

### 11.3. Перегляди в операторах маніпулювання даними

Але використання переглядів в операторах маніпулювання даними (INSERT, DELETE, UPDATE) або підзапити з ними – один з найбільш важких і неоднозначних аспектів переглядів, так як команди модифікації DML фактично впливають на значення в базовій таблиці переглядів.

Якщо запит змінює вміст базової таблиці, тоді і результат перегляду також повинен змінитися.

**Приклад 11.7.** Створимо перегляд, який виводитиме прізвища авторів та назви книг, виданих у 2015 році.

```
CREATE VIEW BIBLIO.ViewBook2015  
AS SELECT AVk, NAMk FROM BIBLIO.Katalog  
WHERE RVk=2015;
```

Такий перегляд обмежує доступ до певних рядків і стовпців таблиці Katalog. Якщо ж доповнити таблицю за допомогою створеного перегляду:

```
INSERT INTO BIBLIO.ViewBook2015  
VALUES ('О.Н.Евсеева', 'РАБОТА С БАЗАМИ ДАННЫХ НА ЯЗЫКЕ  
C#');
```

а така команда успішно виконається, базова таблиця доповниться ще одним записом, у якому два поля вказано явно, а решту – приймуть значення за замовчуванням або NULL.

Але зі зміною базової таблиці результат перегляду не зміниться. Більше того, доданий запис за допомогою перегляду ViewBook2015 користувач так і не побачить, а, отже, цей запис як змінити, так і видалити користувачу проблематично.

Щоб запобігти модифікаціям базових таблиць такого типу необхідно у визначенні перегляду ViewBook2015 включити опцію перевірки WITH CHECK OPTION (все або нічого):

```
CREATE VIEW BIBLIO.ViewBook2015  
AS SELECT AVk, NAMk FROM BIBLIO.Katalog  
WHERE RVk=2015  
WITH CHECK OPTION;
```

Така зміна визначення перегляду VIEW спонукає до перевірки команд DML, які використовують перегляди, і тому наведена вище команда додавання запису INSERT буде відхилена.

Зазвичай опція перевірки WITH CHECK OPTION може бути корисна, коли необхідно, щоб без певної на то причини змінювати вміст базової таблиці за допомогою перегляду, якщо цей перегляд не може відобразити ці зміни.

## 11.4. Модифікація і видалення переглядів

Для модифікації створених переглядів можна використовувати опцію OR REPLACE команди створення перегляду CREATE ... VIEW або команду ALTER ... VIEW.

Синтаксис команди ALTER ... VIEW такий же, як і CREATE ... VIEW:

**ALTER [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]**

**VIEW <ім'я\_перегляду>**

**[(список\_полів\_перегляду)]**

**AS <оператор\_вибору>**

**[WITH [CASCADED | LOCAL] CHECK OPTION]**

Ця команда змінює визначення існуючого перегляду VIEW і була додана в MySQL 5.0.1.

***Приклад 11.8.*** Змінити наявний перегляд ViewBook2015, який виводитиме прізвища авторів та назви книг, виданих у 2012 році за допомогою команди ALTER ... VIEW:

```
ALTER VIEW BIBLIO.ViewBook2015
AS SELECT AVk, NAMk FROM BIBLIO.Katalog
WHERE RVk=2012;
```

***Приклад 11.9.*** Змінити наявний перегляд ViewBook2015, який виводитиме прізвища авторів та назви книг, виданих у 2010 році за допомогою команди CREATE ... VIEW:

```
CREATE OR REPLACE VIEW BIBLIO.ViewBook2015
AS SELECT AVk, NAMk FROM BIBLIO.Katalog
WHERE RVk=2010;
```

Та, зазвичай доцільно видалити перегляд і створити новий, ніж його модифікувати. Для видалення перегляду використовують оператор

**DROP VIEW <ім'я\_перегляду>.**

***Приклад 11.10.*** Видалити перегляд ViewBook2015:

```
DROP VIEW BIBLIO.ViewBook2015;
```

Перегляди в MySQL надають повний комплекс надзвичайно гнучких і потужних інструментів для визначення використання даних з БД. Вони також



спрощують роботу, переформатовуючи дані у зручний спосіб і виключивши подвійну роботу.

## 11.5. Курсори

*Курсор* – це віртуальна таблиця, яка є альтернативним сховищем даних. Але курсор дозволяє звертатися до власних даних, як до даних звичайного масиву. Використовуються курсори у збережених процедурах і функціях.

Курсор не може використовуватися в MySQL сам по собі. Він є важливим компонентом збережених процедур.

За допомогою курсору є можливість перебрати набір даних та обробити кожен запис у відповідності з певними завданнями.

Оскільки курсор реалізується у збереженій процедурі, то він має усі переваги та недоліки, властиві збереженим процедурам (контроль доступу, прекомпіляція, труднощі налагодження та інше).

Властивості курсорів:

- Не чутливий: відкритий курсор не буде відображати зміни в таблиці, здійсненні після його відкриття.
- Доступний тільки для читання: курсори неможливо змінювати.
- Без перемотки: курсор здатний проходити записи тільки в одному напрямку – вперед, без будь-яких пропусків рядків.

Курсори повинні бути оголошені до їх використання. Змінні з умовами оголошуються поперед курсорів. Обробники виключень і помилок оголошуються обов'язково після оголошення курсорів.

Алгоритм створення та використання курсору:

1. *Оголошення курсору і його заповнення*. Курсор оголошується всередині збереженої процедури або функції в області оголошення локальних змінних і має такий формат:

```
DECLARE <ім'я курсору> CURSOR FOR  
      SELECT <ім'я поля> FROM <ім'я таблиці> [WHERE  
      <умова>...];
```



**2. Обробка помилок в збережених процедурах.** При виникненні помилки усередині збереженої процедури, важливо виправити її відповідним чином, наприклад, продовжуючи або зупиняючи виконання операції і видаючи повідомлення про помилку.

MySQL надає простий спосіб визначити обробники, які обробляють помилки, виходячи із загальних умов, таких як попередження або виключення з умов, наприклад, конкретні коди помилок.

Для оголошення обробника виключень або помилок використовується оператор DECLARE HANDLER:

**DECLARE <дія> HANDLER FOR**  
**{<значення умови>} [оператори]**

Якщо значення помилки співпаде зі <значенням умови>, MySQL виконає <оператори> і продовжить або завершить поточний блок коду, виходячи зі значення <дія>.

*Дія* може набувати таких значень:

- CONTINUE: виконання блокованого коду (BEGIN ... END) триває;
- EXIT: виконання блокованого коду, в якому було оголошено обробник, завершується.

*Значення умови* задає конкретну умова або клас умови, які активують обробник і може приймати одне з наступних значень:

- код помилки MySQL;
- стандартне значення SQLSTATE: SQLWARNING, NOT FOUND або SQLEXCEPTION;
- назва умови, пов'язаної або з кодом помилка MySQL, або зі значенням SQLSTATE.

Якщо у курсорі при переході на наступний рядок більше рядків не доступно, тоді відбувається зміна значення змінної SQLSTATE на 02000 (SQLSTATE=00000 свідчить про успішне виконання оператора). Для відслідковування цієї події необхідно встановити обробник HANDLER FOR SQLSTATE на значення 02000.

У якості **[операторів]** може використовуватися простий, наприклад SET, або складовий оператор, вшитий за допомогою ключових слів BEGIN і END. При

досягненні курсором останнього запису набору даних можна встановити деякій змінній значення 1:

```
SQLEXCEPTION SET has_error = 1.
```

### 3. Відкриття курсору:

```
OPEN <ім'я курсору>;
```

4. *Ітераційне задавання значення змінній*, рівній поточному значенню поля. Встановлення покажчика на наступний рядок (якщо рядок існує). При цьому, значення полів поточного запису присвоюються змінним, які вказані оператором FETCH:

```
FETCH <ім'я курсору> INTO <ім'я змінної> [, <ім'я змінної>];
```

### 5. Закриття курсору:

```
CLOSE <ім'я курсору>;
```

*Приклад 11.11.* Розглянемо простий приклад. Створити збережену процедуру з курсором, яка буде виводити за допомогою запиту типу SELECT деяке значення поточного запису, наприклад, прізвища абонентів:

```
USE BIBLIO;  
DELIMITER \\  
CREATE PROCEDURE cursor_demo()  
BEGIN  
    DECLARE k INT DEFAULT 0;  
    DECLARE a CHAR(25);  
    DECLARE cur1 CURSOR FOR SELECT PIPo FROM osoba1;  
    DECLARE CONTINUE HANDLER FOR SQLSTATE '02000' SET k = 1;  
    OPEN cur1;  
    REPEAT  
        FETCH cur1 INTO a;  
        SELECT a;  
    UNTIL k END REPEAT;  
    CLOSE cur1;  
END \\  

```

Результатом виконання збереженої процедури з курсором буде стільки запитів до БД, скільки є записів в таблиці `osoba1` (рис. 11.3).

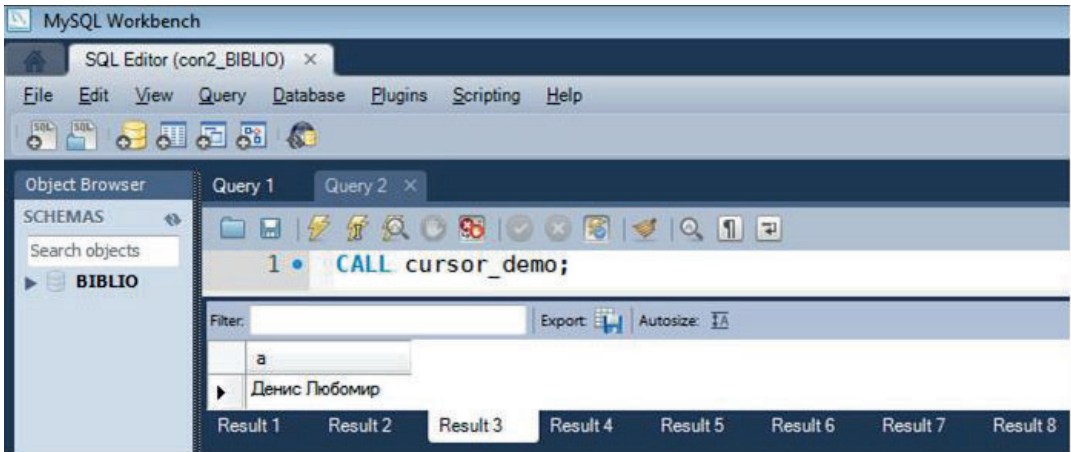


Рис. 11.3. Результат виконання збереженої процедури з курсором

## 11.6. Тригери

*Тригер* – це процедура, яка знаходиться на сервері і викликається автоматично з модифікацією таблиці бази даних (під час зміни значень полів, їх вилученні або доповненні), тобто тригер не можна явно викликати з клієнтської процедури, оскільки він активується тільки подіями, з якими пов'язаний.

Використовуються тригери зазвичай для підтримання цілісності БД, коректного введення даних та відслідковування подій, зв'язаних зі зміною контенту таблиць БД.

Тригер в MySQL створюється оператором `CREATE TRIGGER`, який має такий формат запису:

**CREATE**

**[DEFINER = { <ім'я користувача> | CURRENT\_USER }]**

**TRIGGER <ім'я тригера>**

**<час тригера>**

**<подія для спрацювання тригера>**

**ON <ім'я таблиці>**

**FOR EACH ROW**

**<вираз, що виконується при спрацюванні тригера>;**

Як і будь-яку збережену підпрограму, визначення тригера доцільно виділяти обмежувачами за допомогою оператора DELIMITER.

Момент спрацювання тригера визначає аргумент <час тригера>, який може приймати значення BEFORE або AFTER. BEFORE означає, що тригер виконається до завершення події, яка його викликала, а AFTER означає, що після.

Події, які можуть викликати спрацювання тригера: INSERT, UPDATE та DELETE.

Для доступу до полів таблиці використовуються конструкції OLD.<ім'я поля>, NEW.<ім'я поля>.

Для опису визначення тригерів використовують такі ж оператори, як і для програмування процедур, але вираз, що виконується при спрацюванні тригера, не може містити оператори типу SELECT (ті, що виводять результат).

Для вилучення тригера використовують оператор

**DROP TRIGGER <ім'я тригера>.**

***Приклад 11.12.*** Допустимо, під час створення таблиць у дочірній таблиці Formular зв'язок з батьківською таблицею Osoba у розділі REFERENCES задано як ON DELETE NO ACTION. Для того, щоб при видаленні з батьківської таблиці деякого запису, який зв'язаний з дочірньою таблицею, у дочірній таблиці також видалялися відповідні записи, необхідно було б на етапі проектування задати REFERENCES як ON DELETE CASCADE, а це, в свою чергу, забезпечить збереження цілісності даних бази. Якщо ж цього не зроблено, подібні ситуації можна контролювати за допомогою тригерів.

```
USE BIBLIO;
```

```
DELIMITER \\\
```

```
CREATE TRIGGER Del_Osoba
```

```
    BEFORE
```

```
    DELETE
```

```
    ON BIBLIO Osoba
```

```
    FOR EACH ROW
```

```
        DELETE FROM Formular
```

```
        WHERE Id_ch = OLD.Ido
```

\\

Тепер при видаленні з батьківської таблиці Osoba будь-якого абонента тригер буде вилучати відповідні записи з таблиці Formular, якщо такі є.

**Приклад 11.13.** Необхідно вести облік зміни вмісту таблиці Osoba, тобто всі зміни, які може здійснити користувач, записувати в окрему таблицю. Для цього створимо спочатку таку таблицю, наприклад, History\_Osoba, яка міститиме чотири поля:

```
CREATE TABLE IF NOT EXISTS `BIBLIO`.`History_Osoba` (  
    `id` INT NOT NULL AUTO_INCREMENT ,  
    `idOh` INT NULL ,  
    `PIPoh` VARCHAR(25) NULL ,  
    `date_changesh` DATETIME NULL ,  
    PRIMARY KEY (`id`))  
DEFAULT CHARACTER SET utf8 COLLATE utf8_bin ENGINE=InnoDB;
```

у яку тригер буде вносити відповідні зміни.

Для відслідковування всіх змін у таблиці Osoba необхідно три тригери, які контролюватимуть події додавання нових записів (INSERT), їх редагування (UPDATE) та вилучення (DELETE):

```
USE BIBLIO;  
DELIMITER |  
CREATE TRIGGER ins_Osoba  
    BEFORE INSERT ON Osoba  
    FOR EACH ROW  
    BEGIN  
        INSERT INTO History_Osoba  
            SET idOh = NEW.Ido, PIPoh = NEW.PIPo, date_changesh=  
NOW();  
END;  
|  
CREATE TRIGGER upd_Osoba  
    BEFORE UPDATE ON Osoba
```

```

FOR EACH ROW
BEGIN
    INSERT INTO History_Osoba
        SET idOh = NEW.Ido, PIPoh = NEW.PIPo, date_changesh=
NOW();
END;
|
CREATE TRIGGER del_Osoba
    BEFORE DELETE ON Osoba
    FOR EACH ROW
    BEGIN
        INSERT INTO History_Osoba
            SET idOh = OLD.Ido, PIPoh = OLD.PIPo, date_changesh=
NOW();
END;
|

```

Таким чином, тепер будь-які зміни вмісту таблиці Osoba будуть вноситися в таблицю History\_Osoba для подальшого аналізу.

### Контрольні запитання

1. Що таке перегляд View?
2. Чи містить дані перегляд?
3. Які при обмеження необхідно враховувати при створенні переглядів VIEW?
4. Яка інструкція задає автоматичний вибір алгоритму при зверненні до перегляду VIEW?
5. Запишіть та проаналізуйте синтаксис для створення перегляду VIEW.
6. Де зберігається перегляд?
7. Як викликається перегляд?
8. Які є модифікації переглядів?
9. Як змінити або видалити перегляд?

10. Для чого потрібні курсори?
11. Який алгоритм створення та використання курсору?
12. Які властивості курсорів?
13. Чи може бути визначений курсор за межами збереженої процедури?
14. Для чого призначені тригери?
15. Який синтаксис створення тригера?
16. Для яких подій призначені тригери?
17. Як викликаються тригери?
18. Які значення може приймати аргумент час тригера?
19. Чи можна модифікувати тригер?
20. Як видалити тригер?

## Тема 12. Управління обліковими записами користувачів та виняткові ситуації в MySQL

### 12.1. Управління обліковими записами користувачів

#### 12.2.1. Права (привілеї) в MySQL

Привілеї – це права або можливості користувачів стосовно доступу до баз даних, таблиць, структур, даних тощо.

Зазвичай назви привілеїв співзвучні з відповідними командами SQL і поділяються на три групи (рис. 12.1):

- привілеї адміністрування БД (управління правами користувачів);
- привілеї до зміни структури;
- привілеї доступу до даних (управління та зміна даних в БД).

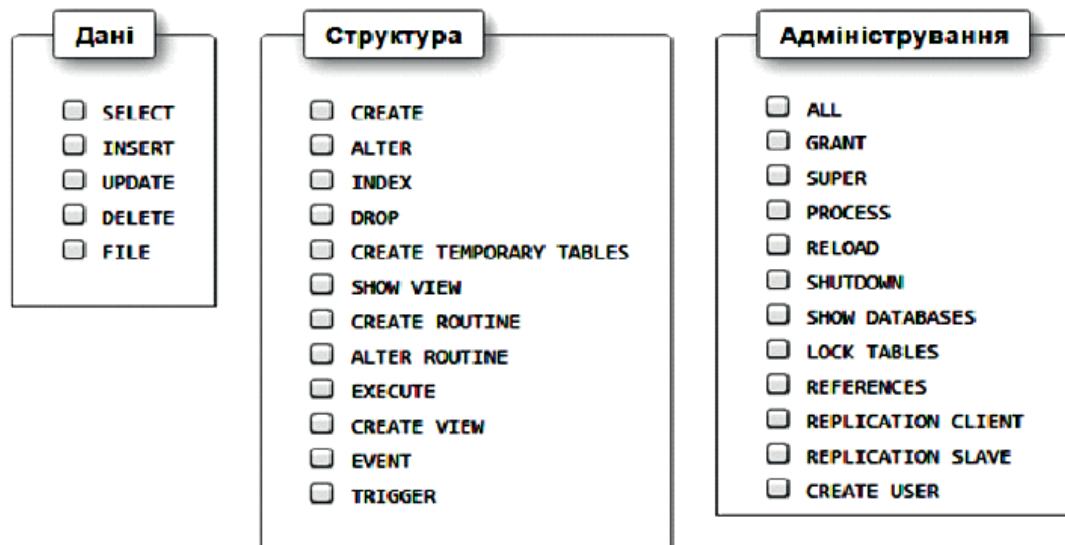


Рис. 12.1. Групи привілеїв в MySQL

Привілеї, що стосуються даних та структури, пояснень не потребують, тому детальніше зупинимося на найважливіших привілеях в ієрархії прав – групи «Адміністрування».



**ALL** – можливість створювати користувача і надавати йому права суперкористувача, який має право робити все, у тому числі і надавати привілеї іншим користувачам.

**GRANT** – можливість створювати нового користувача БД. Права нового користувача не можуть бути ширше прав того, хто створює. Тому, якщо новому користувачеві потрібен розширений варіант прав, їх повинен надавати адміністратор.

**SUPER** – це право дозволяє використовувати команду KILL (відключення іншого користувача від з'єднання з БД).

**PROCESS** – право, яке дає можливість побачити PROCESSLIST (всі з'єднання інших користувачів з БД).

**RELOAD** – можливість працювати з файлами журналів (відкривати, закривати), а також зчитувати права користувачів.

**SHUTDOWN** – можливість запуску команди QUIT або SHUTDOWN для виключення (припинення роботи) сервера.

**SHOW DATABASES** – перегляд всіх БД на даному SQL сервері.

**LOCK TABLES** – можливість заблокувати таблицю при з'єднанні певного користувача.

**EXECUTE** – можливість запуску збережених процедур.

**REPLICATION CLIENT** – дозволяє відстежити місцезнаходження головного (MASTER) і ведених (SLAVES) серверів.

**REPLICATION SLAVE** – право, що дозволяє веденим читати журнал ведучого сервера.

**USAGE** – синонім для «без привілеїв».

Значення USAGE можна задавати, якщо необхідно створити користувача без привілеїв.

Варто зазначити, що для забезпечення надійної роботи БД необхідно надавати привілеї з групи «Адміністрування» тільки тим користувачам, яким вони дійсно необхідні. Крім того, до надання деяких з них слід ставитися особливо уважно:

- Привілей GRANT дозволяє користувачам передавати свої привілеї іншим користувачам. Два користувачі з неоднаковими привілеями, маючи привілей GRANT, здатні об'єднати свої привілеї.
- Привілей ALTER може бути використано для перейменування таблиць і руйнування таким чином всієї системи привілеїв.
- Привілей FILE може використовуватися зловмисно для зчитування будь-якого засекреченого файлу на сервері в таблицю бази даних, до вмісту якої можна потім отримати доступ за допомогою команди SELECT. Це відноситься і до вмісту всіх баз даних, які знаходяться під управлінням сервера.
- Привілей SHUTDOWN може використовуватися для повного припинення роботи сервера і, таким чином, повної заборони обслуговування інших користувачів.
- Привілей PROCESS може бути використано для перегляду відкритого тексту запитів, які виконуються в даний момент, включаючи запити на установку або зміну паролів.
- Привілеї доступу до бази даних MySQL можуть бути використані для зміни паролів і іншої інформації, що відноситься до привілеїв доступу. Отримавши доступ до таблиці користувачів `mysql.user`, будь-який користувач може увійти на сервер MySQL під ім'ям іншого користувача та, маючи достатні привілеї, замінити пароль.

Система привілеїв MySQL забезпечує користувачам можливість виконувати тільки ті дії, які їм дозволені відповідно до їх статусу. Під час з'єднання користувача з сервером MySQL ідентифікуються ім'я хоста, ім'я користувача та привілеї, які знаходяться на сервері у БД MySQL і таблиці `user`.

У випадку, якщо запит стосується таблиць бази даних, СКБД може додатково звернутися до таблиць `tables_priv` і `columns_priv`.

Якщо коротко, то сервер використовує таблиці привілеїв наступним чином:

- Поля таблиці `user` визначають, чи дозволити вхідне з'єднання або відмовити в ньому. Для дозволених з'єднань будь-які привілеї, надані в таблиці `user`,

означають глобальні привілеї користувача (повноваження супер). Ці привілеї поширюються на всі бази даних, розміщені на сервері.

- Таблиці `db` і `host` використовуються спільно:
  - поля таблиці `db` визначають, яким користувачам, при з'єднанні з яких хостів і до яких баз даних дозволений доступ. Поля привілеїв визначають дозволені операції;
  - таблиця `host` використовується в якості розширення таблиці `db` в разі, якщо необхідно застосувати деякий запис з таблиці `db` до різних хостів.
- Таблиці `tables_priv` і `columns_priv` подібні до таблиці `db`, але областю їх дії є вже рівень таблиць і стовпців, а не баз даних.

Записи з таблиць привілеїв при операціях перейменування об'єктів (баз даних, таблиць, стовпців) не успадковуються. Будь-який привілей, прив'язаний до таблиці, після її перейменування більше не буде відноситися до даної таблиці. І цей принцип справедливий як для рівня баз даних, так і для таблиць, і стовпців.

Привілеї адміністрування задаються тільки в таблиці `user`. Це пов'язано з тим, що операції адміністрування є операціями над самим сервером, а не над базами даних, тому немає сенсу перераховувати такі привілеї в інших таблицях привілеїв.

У таблицях `user`, `db` і `host` всі поля привілеїв мають оголошений тип – список значень `ENUM` ('N', 'Y'), тобто можливе одне з двох значень – 'N' і 'Y', а за замовчуванням – 'N'.

У таблицях `tables_priv` and `columns_priv` поля привілеїв оголошуються як список значень `SET`.

Починаючи з версії `MySQL 4.0.2`, можна також обмежувати деякі ресурси, що виділяються користувачам:

- `MAX QUERIES PER HOUR` <значення> – максимальна кількість команд за годину, яку може виконати певний користувач;
- `MAX UPDATES PER HOUR` <значення> – максимальна кількість команд для створення, редагування та видалення даних в таблицях або БД певним користувачем або групою протягом години;

- **MAX CONNECTIONS PER HOUR** <значення> – максимальна кількість підключень користувача до БД за годину.
- **MAX USER CONNECTIONS** <значення> – максимальна кількість користувачів, які можуть одночасно працювати на сервері.

За замовчуванням всі користувачі не обмежені у використанні зазначених вище ресурсів тільки в разі, тільки якщо ці обмеження не накладені на них.

Якщо користувач протягом години досягає межі будь-якого з вищевказаних значень, його з'єднання буде перервано з видачею відповідного повідомлення про помилку.

### 12.2.2. Створення користувачів БД

Для створення нового користувача БД в СКБД MySQL можна використовувати різні способи (у режимі конструктора або в режимі командного рядка) та різні середовища адміністрування БД (наприклад, MySQL WorkBench, phpMyAdmin, dbForge Studio for MySQL тощо).

Головною умовою створення нового користувача є наявність привілеїв (прав або можливостей) в того, хто його створює. Привілеї нового користувача не можуть бути ширше привілеїв того, хто його створює. Якщо новому користувачеві потрібен розширений варіант можливостей, тоді їх повинен задавати адміністратор.

#### 12.2.2.1. Режим командного рядка

Частиною мови SQL є сімейство DCL, яке використовуються для контролю доступу до даних в базах даних. DCL командами є:

- **GRANT** (Надати) – дозволити визначеним користувачам виконувати визначені маніпуляції;
- **REVOKE** (Скасувати) – скасувати раніше надані права;
- **FLUSH PRIVILEGES** – оновлення прав, виконується завжди після зміни будь-яких прав.

Для створення нового користувача на SQL сервері використовується команда **GRANT**. Формат запису команди **GRANT**:

**GRANT** <привілеї> [(<стовпці>)]

**ON <рівень привілеїв>**

**TO <обліковий запис>**

**IDENTIFIED BY <пароль>**

**[WITH GRANT OPTION]**

**[REQUIRE ENCRYPTION REQUIREMENTS]**

**[WITH GRANT OR RESOURCE MANAGEMENT OPTIONS];**

1. Після обов'язкового слова GRANT вказуються ті <привілеї>, які присвоюються певному <обліковому запису>. Аргумент <стовпці> необов'язковий і використовується тільки при установці привілеїв для стовпців. Імена стовпців у списку відділяються один від одного комами.

2. Рівень застосування привілеїв вказується після ON і може бути:

**Глобальний рівень** – глобальні привілеї застосовуються до всіх баз даних на сервері. У команді GRANT цей рівень задається шаблоном: \*.\* – всі бази даних на сервері і всі таблиці;

**Рівень бази даних** – привілеї бази даних застосовуються до всіх таблиць вказаної бази даних і задаються шаблоном: <ім'я бази даних>.\* – всі таблиці вказаної БД, наприклад, BIBLIO.\*;

**Рівень таблиці** – привілеї таблиці застосовуються до всіх стовпців вказаної таблиці і задаються шаблоном: <ім'я бази даних>.<ім'я таблиці> – всі стовпці вказаної таблиці БД, наприклад, BIBLIO.Osoba;

**Рівень стовпця** – привілеї стовпця застосовуються до окремих стовпців вказаної таблиці. Вказуються стовпці таблиці у круглих дужках через кому після переліку привілеїв та перед задаванням рівня привілеїв (перед ON), наприклад:

**GRANT UPDATE (Telo, Adreso) ON BIBLIO.Osoba TO ...;**

де Telo, Adreso – назва стовпців таблиці Osoba бази даних BIBLIO.

3. Обліковий запис вказується після інструкції TO та має таку структуру:

**<ім'я користувача>@<хост>,**

оскільки в СКБД MySQL задаються не тільки ті, хто підключається, але і місце, звідки вони можуть це зробити. Таким чином, можна задати окремі облікові записи для двох користувачів, що мають одне і те ж ім'я, але що підключаються з

різних місць. СКБД MySQL дозволяє робити відмінність між ними і присвоювати привілеї кожному з них незалежно один від одного.

Кожен елемент структури облікового запису необхідно брати в одинарні лапки, наприклад, 'dmv'@'localhost'.

Ім'я користувача в MySQL – це ім'я, яке використовується для ідентифікації користувача при під'єднанні до серверу. У цього імені немає ніякого зв'язку з реєстраційним ім'ям в операційних системах Windows або UNIX.

Аргумент <хост> задає, з певного або з різних комп'ютерів даний користувач може під'єднуватися до сервера. В першому випадку оператор GRANT виглядатиме:

```
GRANT ALL ON BIBLIO.* TO 'dmv'@'localhost' IDENTIFIED BY ...
```

Застосування в аргументі <хост> значення localhost дозволить під'єднатися користувачу до серверу з локального вузла, задавши значення host, таке як localhost або 127.0.0.1 (IP-адреса локального вузла).

Іншою крайністю є дозвіл користувачу, який багато подорожує, під'єднуватися з будь-якого комп'ютера, розташованого в будь-якій точці світу:

```
GRANT ALL ON BIBLIO.* TO 'dmv'@'%' IDENTIFIED BY ...
```

Символ "%" замінює всі можливі значення адрес і виконує ту ж функцію, що і при порівнянні з допомогою оператора LIKE. Це найпростіший, але й найнебезпечніший спосіб створити користувача.

При необхідності можна також дозволити користувачу підключатися з обмеженого числа комп'ютерів, задавши ім'я домена мережі або IP-адресу:

```
GRANT ALL ON BIBLIO.* TO 'dmv'@'%.zevs' IDENTIFIED BY ... ;
```

```
GRANT ALL ON BIBLIO.* TO 'dmv'@'172.16.1.250' IDENTIFIED BY ... ;
```

Ці адреси можна задавати як явно, так і за допомогою допоміжних символів, наприклад символу ('\_'), що відповідає за будь-який один символ:

```
GRANT ALL ON BIBLIO.* TO 'dmv'@'172.16._.250' IDENTIFIED BY ... ;
```

Крім того, у версії MySQL 3.23 з'явилася можливість визначати IP-адреси, задаючи маску мережі, встановлюючи число розрядів в мережевому номері:

```
GRANT ALL ON BIBLIO.* TO 'dmv'@'172.16.1.0/22' IDENTIFIED BY ... ;
```

де частина 193.93.216.0/22 визначає 22-розрядний мережевий номер і відповідає будь-якому комп'ютеру з адресою 193.93.216 в перших 22 розрядах IP-адреси.

Відсутність частини host в специфікації облікового запису аналогічна використанню '%' – всі можливі значення адрес. Тобто, команди:

```
GRANT ALL ON BIBLIO.* TO 'dmv' IDENTIFIED BY ... ;
```

```
GRANT ALL ON BIBLIO.* TO 'dmv'@'%' IDENTIFIED BY ... ;
```

у випадку прикладі будуть аналогічними.

4. Значення паролю вказується після інструкції IDENTIFIED BY, і, як зазначено в документації MySQL, цей аргумент не є обов'язковим. Тобто, якщо для нового користувача у команді GRANT не вказати вираз IDENTIFIED BY, тоді пароль йому присвоєний не буде, що є не зовсім розумно з точки зору безпеки.

Якщо ж цей оператор задається для вже існуючого користувача, введений пароль замінить старий пароль.

Якщо ж за допомогою команди GRANT для деякого користувача додаються тільки нові привілеї, введений пароль замінить старий пароль. У випадку, коли у такій команді пароль опущено, він залишиться без змін.

Рядок пароля, що задається за допомогою виразу IDENTIFIED BY, повинен бути літерним рядком, який при записі зашифрує оператор GRANT. Тому слід уникати використання функції PASSWORD(), що застосовується з оператором SET PASSWORD.

5. Аргумент WITH GRANT OPTION є необов'язковим. З його допомогою можна делегувати іншому користувачу будь-які, але тільки свої привілеї.

Починаючи з версії MySQL 4.0.2 пропозиція WITH також використовується для задавання параметрів управління ресурсами, що дозволяють обмежити кількість з'єднань або запитів, які може обробляти засновник даного облікового запису за годину. Ці параметри допоможуть запобігти "захоплення" серверу даним обліковим записом.

6. Аргумент REQUIRE є також необов'язковим. Опція REQUIRE з'явилася в MySQL 4.0.0 і використовується для установки облікових записів, які повинні проводити підключення за допомогою протоколу SSL.

7. Задавання параметрів управління ресурсами (спеціальних привілеїв) в опції [WITH GRANT OR RESOURCE MANAGEMENT OPTIONS] тільки обмежує користувача БД, адже за замовчуванням кожен параметр рівний 0, що значить "обмежень немає".

Спеціальні привілеї можуть бути накладені тільки за допомогою глобальної команди GRANT (\*.\*) з використанням наступного синтаксису:

**GRANT ...**

**WITH MAX\_QUERIES\_PER\_HOUR N1**

**MAX\_UPDATES\_PER\_HOUR N2**

**MAX\_CONNECTIONS\_PER\_HOUR N3**

**MAX\_USER\_CONNECTIONS N4**

де N1, N2 N3 і N4 є цілими числами і представляють, відповідно, значення кількостей запитів / оновлень / з'єднань на годину / користувачів, які можуть одночасно працювати на сервері:

**GRANT ALL ON BIBLIO.\* TO 'dmv'@% IDENTIFIED BY '\*\*\*\*\*'**

**WITH MAX\_CONNECTIONS\_PER\_HOUR 10**

**MAX\_QUERIES\_PER\_HOUR 200**

**MAX\_UPDATES\_PER\_HOUR 50;**

де користувач dmv має повний доступ до бази даних BIBLIO, але може під'єднуватися десять разів на годину і робити 200 запитів в годину, лише 50 з яких можуть бути оновленнями.

Якщо користувач протягом години досягає межі будь-якого з вищевказаних значень, його з'єднання буде перервано з видачею відповідного повідомлення про помилку.

Якщо привілеї надаються користувачеві командою GRANT, якого не існує, то цей користувач створюється.

І головне, щоб всі зміни, які здійснюються командою GRANT, вступили в дію, необхідно оновити всі привілеї:

**FLUSH PRIVILEGES;**

Цю команду необхідно виконувати завжди після будь-яких змін привілеїв.

#### **12.2.2.2. Режим конструктора**



Найкращим і найбільш розповсюдженим інструментом адміністрування БД MySQL залишається phpMyAdmin, так як цей додаток в більшості випадків дозволяє обійтися без введення команд SQL, а користуватися лише графічним інтерфейсом.

Практично phpMyAdmin є сторінкою на веб-браузері за адресою, іменем користувача та паролем, наданими адміністратором (рис. 12.2).

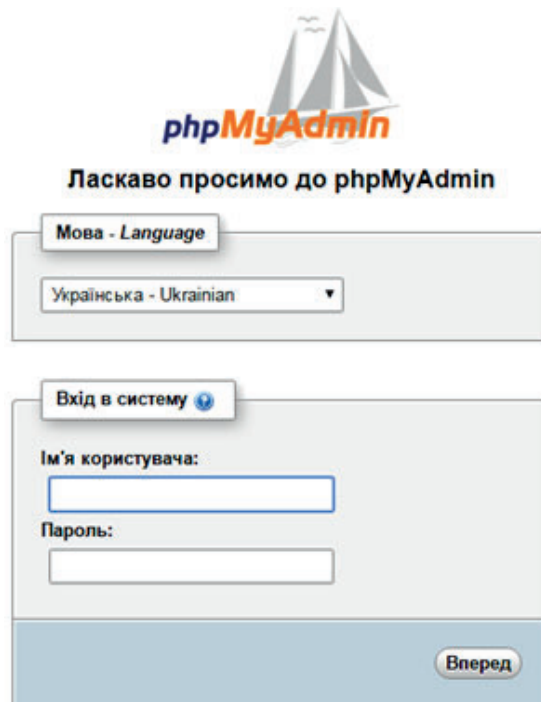


Рис. 12.2. Реєстрація у застосунку phpMyAdmin

Можливості phpMyAdmin:

- адміністрування БД і користувачів;
- управління процесами сервера і записами користувача;
- глобальний пошук по базах даних, користувачам і таблицями;
- імпорт та експорт даних в різні формати;
- створення, перегляд, видалення і редагування баз даних;
- створення, перейменування, копіювання і видалення таблиць;
- додавання нових і зміна існуючих полів в таблиці;
- створення, додавання, зміна індексів;
- обробка SQL-запитів;

- аналіз SELECT-запитів.

Для зручності роботи в phpMyAdmin передбачено підсвічування синтаксису і підказки в ході операцій.

Після ідентифікації прав користувача відкривається відповідне меню застосунку. Якщо користувач має права адміністратора (або привілей GRANT), тоді інтерфейс матиме пункти меню, доступні для створення нового користувача (рис. 12.3).

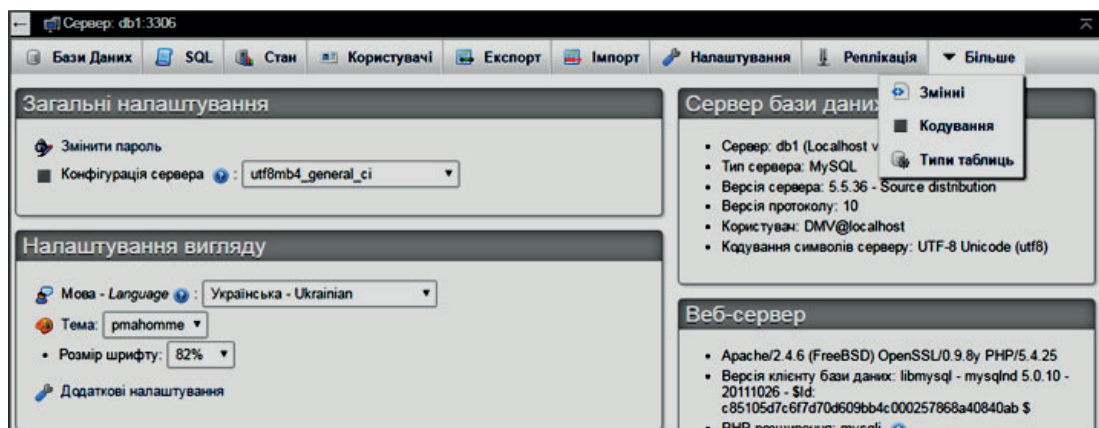


Рис. 12.3. Інтерфейс phpMyAdmin для користувача з привілеєм GRANT

Звичайно, такий користувач матиме і багато інших можливостей, на відміну від користувача типу гість без привілею GRANT (рис. 12.4).

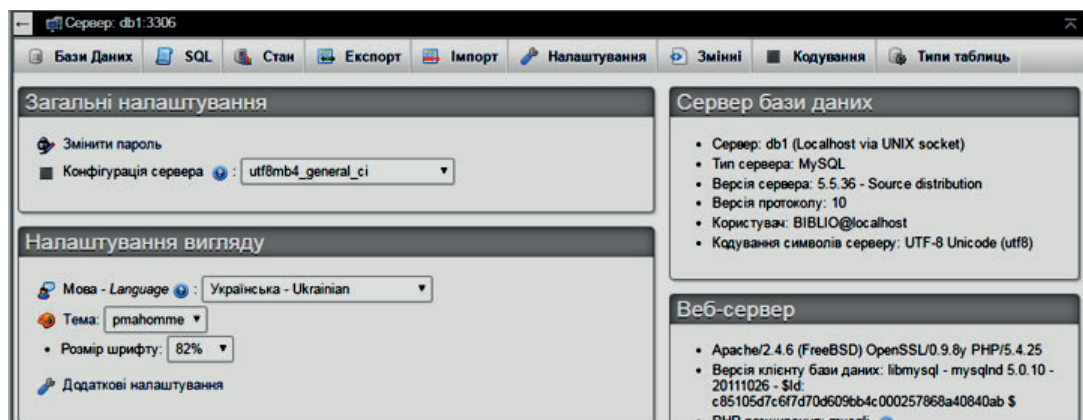


Рис. 12.4. Інтерфейс phpMyAdmin для користувача без привілею GRANT

Одна з таких можливостей і є створення нового користувача за допомогою меню «Користувачі» (рис. 12.5).

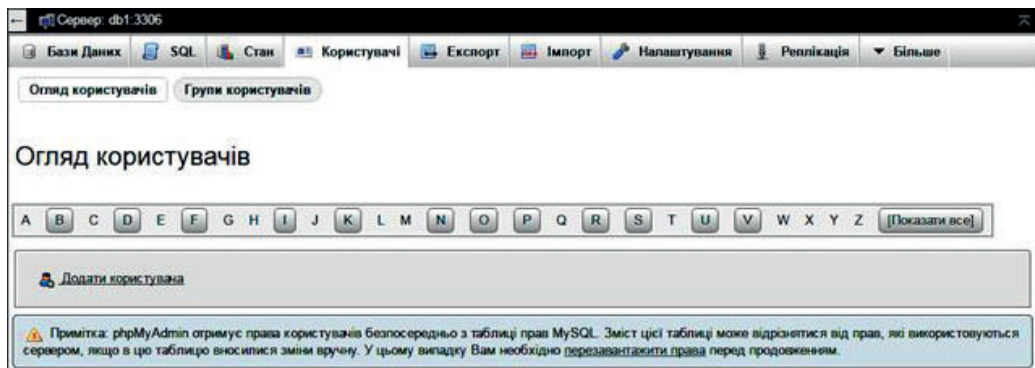


Рис. 12.5. Інтерфейс меню «Користувачі»

Вибравши команду «Додати користувача», необхідно заповнити поля «Ім'я користувача», задати хост, з якого користувач буде з'єднуватися з БД (за замовчуванням % – довільний хост), двічі (з підтвердженням) «Пароль» та зазначити, що під час створення облікового запису користувача також створити відповідну БД під іменем користувача і надати на цю БД повні привілеї (рис. 12.6).

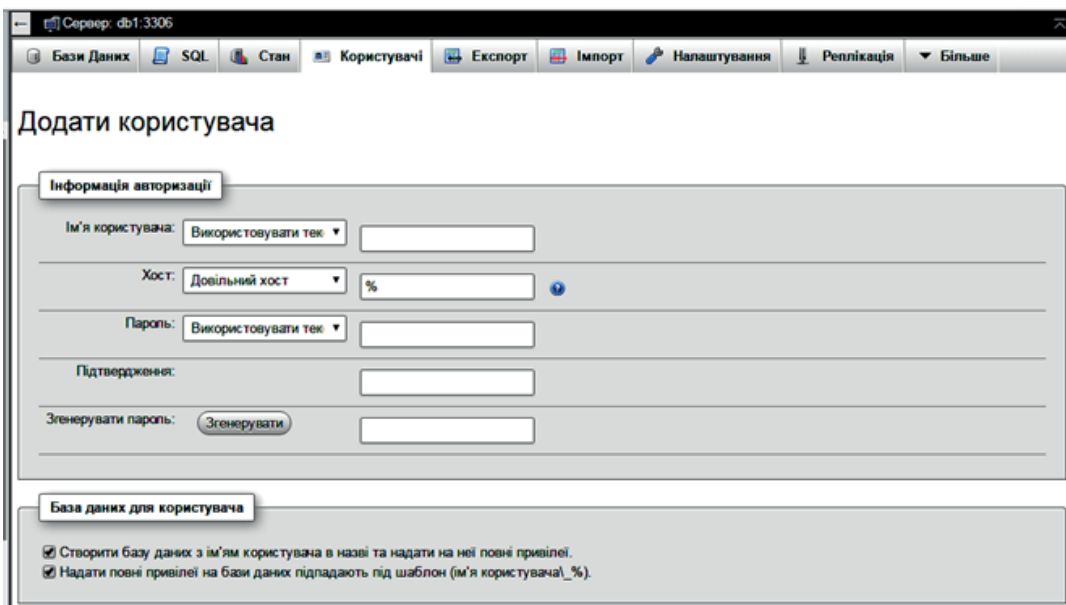


Рис. 12.6. Додавання облікового запису нового користувача SQL сервера

Якщо ж необхідно обмежити привілеї користувача, тоді у області задавання привілеїв необхідно задати тільки потрібні (рис. 12.7).

Також для нового користувача можна задавати обмеження на ресурси або залишити за замовчуванням 0 – без обмежень.

Створюється новий обліковий запис на сервері з вказаними вище параметрами натисканням кнопки «Вперед».

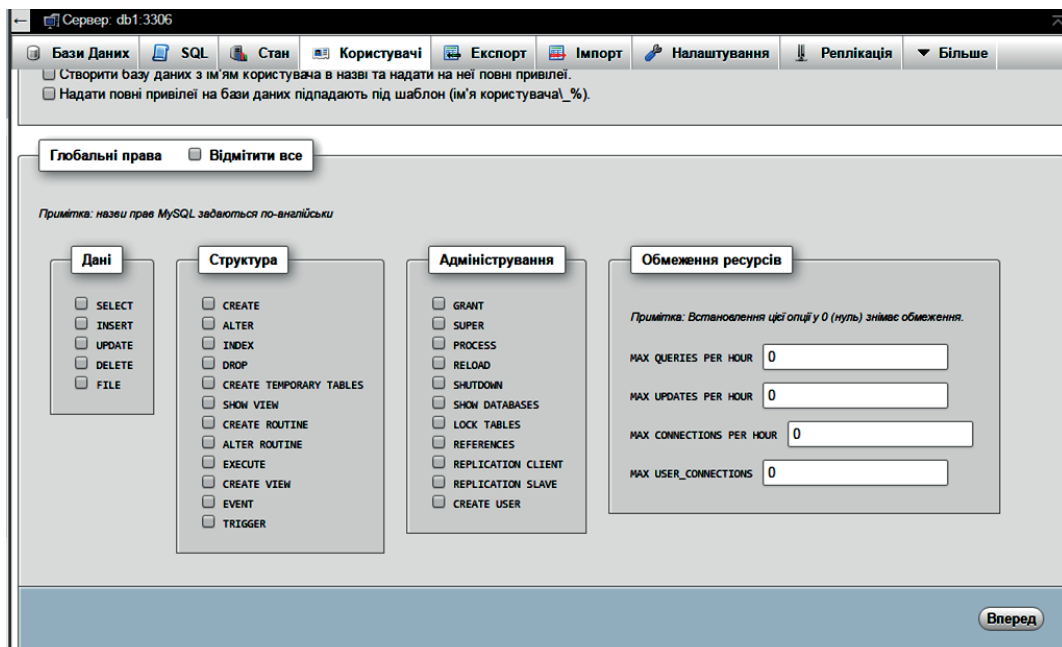


Рис. 12.7. Задавання привілеїв новому користувачу SQL сервера

### 12.2.3. Відміна привілеїв і видалення користувачів

Для відміни привілеїв користувача застосовується оператор REVOKE. За його допомогою можна відмінити або всі привілеї, або і деякі з них.

Синтаксис оператора REVOKE є скороченою формою запису оператора GRANT, лише з тією різницею, що опцію TO замінено на FROM, а опції IDENTIFIED BY, REQUIRE і WITH GRANT OPTION відсутні зовсім:

**REVOKE <привілеї> [(<стовпці>)]**

**ON <рівень привілеїв>**

**FROM <обліковий запис>**

Так як оператор REVOKE скасовує привілеї, надані командою GRANT, аргумент <обліковий запис> цього оператора повинен відповідати <обліковому запису> оператора GRANT для користувача, привілеї якого відмінюються.

Частина <привілеї> необов'язково повинна відповідати попереднім привілеям. Користуючись оператором REVOKE, можна відмінити тільки деякі з привілеїв, наданих оператором GRANT.

Виходячи з синтаксису команди REVOKE, змінити або відмінити такі аргументи, як пароль та обмеження на ресурси (які є в команді GRANT), нею неможливо.

Один із способів внесення змін або оновлення пароля користувача полягає у використанні оператора UPDATE, який задає значення для стовпців User і Host для запису таблиці user користувача:

```
UPDATE mysql.user SET Password = PASSWORD('новий пароль')
WHERE User = 'dmv' AND Host = '%';
FLUSH PRIVILEGES;
```

У даному випадку, як і для встановлення або зміни привілеїв, необхідно обов'язково оновити привілеї, адже команда FLUSH PRIVILEGES виконується завжди після зміни будь-яких привілеїв.

Ще одним зі способів зміни пароля є використання оператора SET PASSWORD, оскільки обліковий запис задається в тому ж форматі, що і GRANT, і немає необхідності оновлювати привілеї таким чином:

```
SET PASSWORD FOR 'dmv'@'%' = PASSWORD('новий пароль');
```

Інший спосіб зміни пароля полягає у використанні оператора GRANT USAGE з опцією IDENTIFIED, якою задається пароль:

```
GRANT USAGE ON *.* TO 'dmv'@'%' IDENTIFIED BY 'новий пароль';
```

Аналогічно, при установці обмеження ресурсів для користувача їх не можна видалити за допомогою оператора REVOKE. Натомість для установки значень обмеження в 0 ("немає обмежень") можна скористатися оператором GRANT з вказаною пропозицією USAGE:

```
GRANT USAGE ON *.* TO 'dmv'@'%'
WITH MAX_CONNECTIONS_PER_HOUR 0
MAX_QUERIES_PER_HOUR 0
MAX_UPDATES_PER_HOUR 0;
```

Привілей GRANT OPTION не включений в ALL, тому відмінити його можна тільки при явній згадці в частині <привілей> оператора REVOKE:

```
REVOKE GRANT OPTION ON *.* FROM 'dmv'@'%';
```

Оператор REVOKE застосовується для відміни привілеїв, але не для видалення користувачів. В таблиці user все одно залишається запис для користувача, навіть якщо всі привілеї для нього відмінено. Це означає, що користувач все ще має можливість під'єднуватися до сервера. Для повного видалення користувача необхідно видалити його запис з таблиці user. Для цієї мети застосовується оператор DELETE. Наприклад, для видалення облікового запису, створеного для dmv@%:

```
DELETE FROM mysql.user WHERE User = 'dmv' and Host = '%';
```

І так само, як і для команди GRANT, щоб зміни набули чинності, необхідно виконати команду:

```
FLUSH PRIVILEGES.
```

## 12.2. Резервне копіювання і відновлення БД MySQL

Однією з ключових виняткових ситуацій є відновлення бази даних. Зрозуміло, щоб було з чого відновлювати, перед тим необхідно створити деяку копію БД.

Традиційно рекомендовано зберігати 10 резервних копій (бекапів): по одному на кожен день тижня, а також бекапи двотижневої, місячної і квартальної давності – це дозволить достатньо глибоко здійснити відкат у випадку втрати будь-яких даних.

За способом зберігання копії БД або її частини можна виділити:

- копія БД або її частини зберігається на тому ж сервері, що і БД;
- копія БД або її частини зберігається на іншому ПК або зовнішніх носіях;
- копія БД або її частини зберігається в хмарі.

Кожен з цих способів має свої переваги та недоліки.

Якщо копія даних зберігається на тому ж сервері, що і основна БД, тоді перевагою буде простота і швидкість реалізації, але у випадку виходу з ладу сервера пропаде і резервна копія.

Найбільш поширеною утилітою для такого створення копій БД є `mysqldump` – ефективний інструмент створення резервної копії бази даних MySQL, що дозволяє створювати \*.sql файл бази даних (джерела) та, за необхідності, відновити БД з цього файлу.

Утиліта `mysqldump.exe` зазвичай знаходиться за шляхом `C:\Program Files\MySQL\MySQL Server X.X\bin`, тобто на цьому ж ПК, що і сервер.

Базовою командою для створення резервної копії бази даних MySQL за допомогою `mysqldump` є:

```
# mysqldump -u[користувач] -p[пароль_користувача] [ім'я_бази] >
[назва_файлу_резервної_копії_бази].sql
```

Наприклад:

```
# mysqldump -uroot -pqwerty my_db > my_db-dump1.sql
```

Для відновлення БД з резервної копії використовується утиліта `mysql.exe`, яка знаходиться за тим же шляхом, що і `mysqldump.exe`. Її синтаксис наступний:

```
# mysql -u[користувач] -p[пароль_користувача] [ім'я_бази] <
[назва_файлу_резервної_копії_бази].sql
```

Наприклад:

```
# mysql -uroot -pqwerty my_db < my_db-dump1.sql
```

Великою перевагою `mysqldump` є можливість автоматизації процесу резервного копіювання БД за допомогою завдання `cron`, яке буде запускати команду `mysqldump` в заданий час.

Зберігання даних на іншому ПК або зовнішніх носіях вимагає більше часу і для запису, так і для відновлення, але ймовірність одночасного виходу з ладу декількох пристроїв вже вдвічі менша.

Для створення резервних копій БД в іншому місці (не на SQL-сервері) і, відповідно, відновлення їх, достатньо функціоналу експорту/імпорту безкоштовних MySQL Workbench або PHPMyAdmin.

Крім того, є ряд комерційних ліцензованих утиліт резервного копіювання та відновлення для баз даних MySQL, серед яких можна виділити MySQL Enterprise



Backup, який інтегрується у MySQL Workbench та знаходиться на вкладці Адміністрування та має два розділи:

- Онлайн-резервне копіювання: встановлює профіль резервного копіювання, який визначає, what (що) потрібно зробити для резервного копіювання, where (де) зберігати резервну копію, а також when (коли) MySQL слід зробити резервну копію.
- Відновити : Відновлює сервер MySQL до певного моменту часу, як правило, шляхом відновлення резервної копії, створеної за допомогою функції Online Backup у робочому середовищі MySQL.

Надійним з огляду на попередні способи є зберігання резервних копій у хмарі. Не зважаючи на швидкодію, варто зазначити, що хмарні сервіси зазвичай платні.

Серед багатьох способів, що пропонують резервне копіювання та відновлення даних, можна виділити два підходи:

- під час зберігання БД копіюється повністю весь вміст таблиць;
- під час зберігання БД зберігаються лише зміни, зроблені за визначений період часу.

Створення копій таблиць цілком є простим, але часто вимагає значних затрат часу. Зберігання ж тільки змін вимагає значно менше часу, зате є складним з визначенням тих змін, що були зроблені у період деякого часу.

### **12.2.1. Копія БД або її частини на тому ж сервері**

Одним з різновидів резервної копії БД (бекапу), з якої можна відновити поточну базу, є її копія на цьому ж сервері.

Такий вид є недосконалим з огляду на те, що у разі виходу з ладу сервера відновити БД з бекапу буде вже неможливо.

Але на практиці використовується для часткового бекапу великих таблиць, які не можна скопіювати жодним іншим чином.

Для цього використовується розглянута раніше SQL-команда `SELECT * INTO OUTFILE 'file_name' FROM 'tbl_name'` та відповідна їй `LOAD DATA INFILE 'file_name' REPLACE ...`

Для використання цих команд потрібні привілеї FILE.



Цей різновид команди здійснює збереження вибраних у таблиці записів у вказаний файл 'file\_name. Файл створюється на сервері і до цього не повинен існувати.

Вивантаження проводиться порядково і можна відібрати для збереження лише потрібні рядки, як у звичайному SELECT. Структура таблиць ніде не вказується – про це має дбати програміст. Він також повинен подбати про включення команд SELECT INTO OUTFILE у транзакцію, якщо це необхідно для забезпечення цілісності даних.

Щоб уникнути повторення записів, у таблиці повинен бути первинний або унікальний ключ. При відновленні ключове слово REPLACE задає заміну старих записів новими у випадку, коли новий запис у значеннях унікального ключа повторює старий.

Особливістю команд SELECT INTO OUTFILE та LOAD DATA INFILE є те, що вказаний формат відображення у SELECT INTO OUTFILE повинен повністю відповідати формату читання даних у LOAD DATA INFILE.

Наприклад, щоб створити копію таблиці Katalog у файл mydata у форматі значень, розділених комами (CSV), необхідно виконати команду:

```
SELECT * INTO OUTFILE "mydata.csv"  
    FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY ""  
    LINES TERMINATED BY '\n'  
FROM Katalog;
```

Відповідно, для відновлення цієї таблиці необхідно скористатися командою:

```
LOAD DATA INFILE "mydata.csv"  
    REPLACE  
    INTO TABLE Katalog  
    FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY ""  
    LINES TERMINATED BY '\n'
```

## 12.2.2. Резервне копіювання і відновлення за допомогою MySQL

### Workbench

#### 12.2.2.1. Інструмент Reverse Engineer

Під час створення фізичної бази даних основою є розроблена схема даних з ER діаграмою, з яких і створюється фізична БД, і цей процес називається Forward Engineer.

Але бувають випадки, коли потрібно зробити навпаки – створити діаграму з існуючої схеми бази даних. Це завдання у MySQL Workbench називається Reverse Engineer, оскільки спочатку є робоча БД і з неї створюється модель.

Інструмент Reverse Engineer *не створює бекапу*, з якого можна повністю відновити структуру БД з даними, а створює лише модель БД, з якої можна відновити тільки структуру БД без даних одним з способів, розглянутих в темі «Етапи проектування баз даних».

До структури відносяться як таблиці, так і збережені процедури, функції, перегляди, тригери тощо.

Викликається у MySQL Workbench інструмент Reverse Engineer за допомогою одноіменної команди з пункту головного меню Database (рис. 12.8) і виконується в декілька кроків.

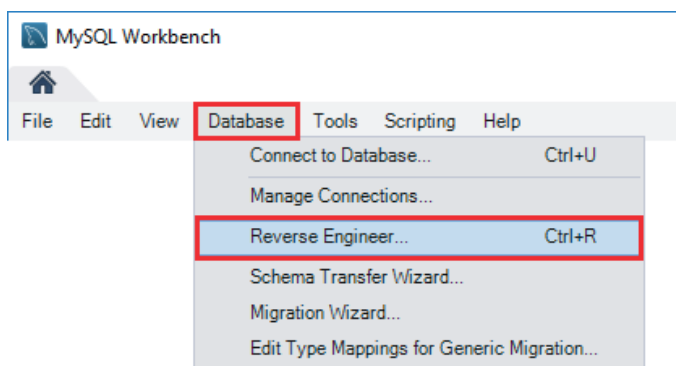


Рис. 12.8. Виклик інструменту Reverse Engineer

На першому кроці здійснюється підключення до MySQL сервера шляхом вибору наявних з'єднань (рис. 12.9) або створенням нового.

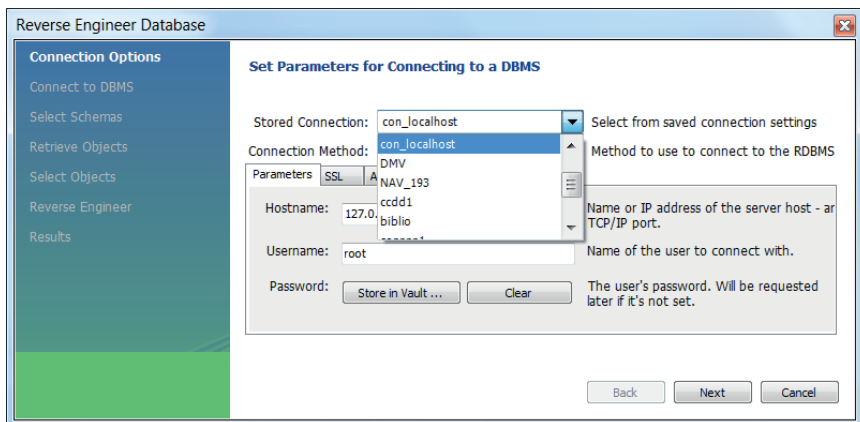


Рис. 12.9. Вибір наявних з'єднань з SQL сервером

Після успішного підключення майстер покаже список доступних схем на сервері. Для продовження необхідно обрати ті БД на сервері, для яких здійснюється реверс-інжиніринг (рис. 12.10).

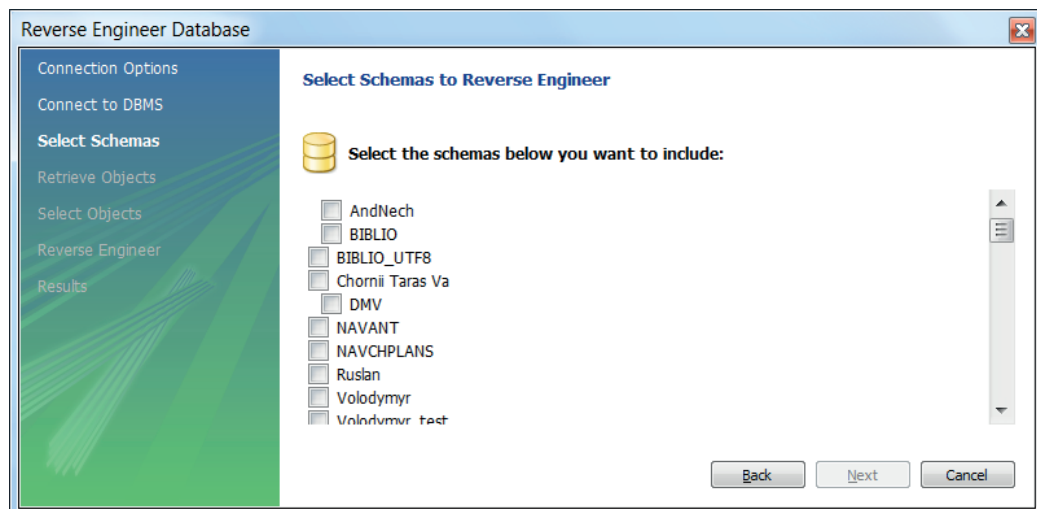


Рис. 12.10. Вибір БД на сервері, для яких здійснюється реверс-інжиніринг

Якщо обрано декілька БД для реверс-інжинірингу, тоді результатом буде одна ER діаграма і відповідна кількість моделей БД.

Наступним кроком є отримання об'єктів схеми відновлення на основі вказаних БД на попередньому кроці (рис. 12.11).

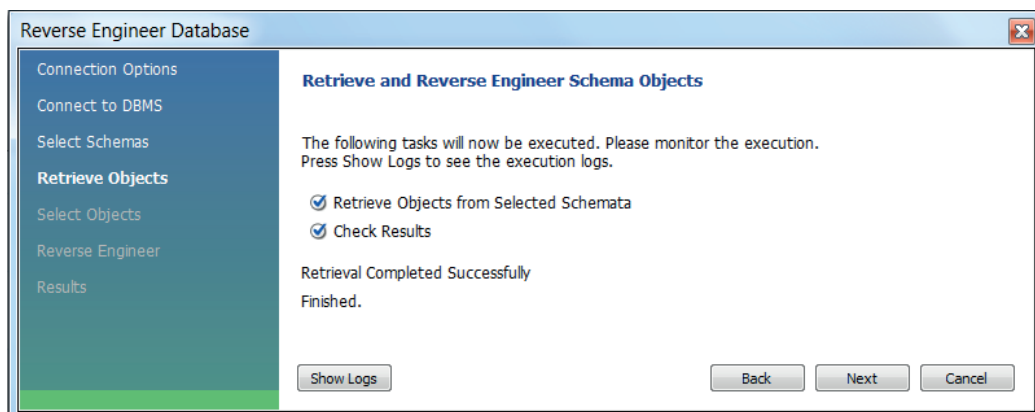


Рис. 12.11. Отримання об'єктів схеми відновлення реверс-інжинірингом

Отримані об'єкти можна на наступному кроці відфільтрувати, тобто вибрати не всі, а лише необхідні, відкривши відповідне вікно об'єктів за допомогою відповідної кнопки Show Filter (рис. 12.12).

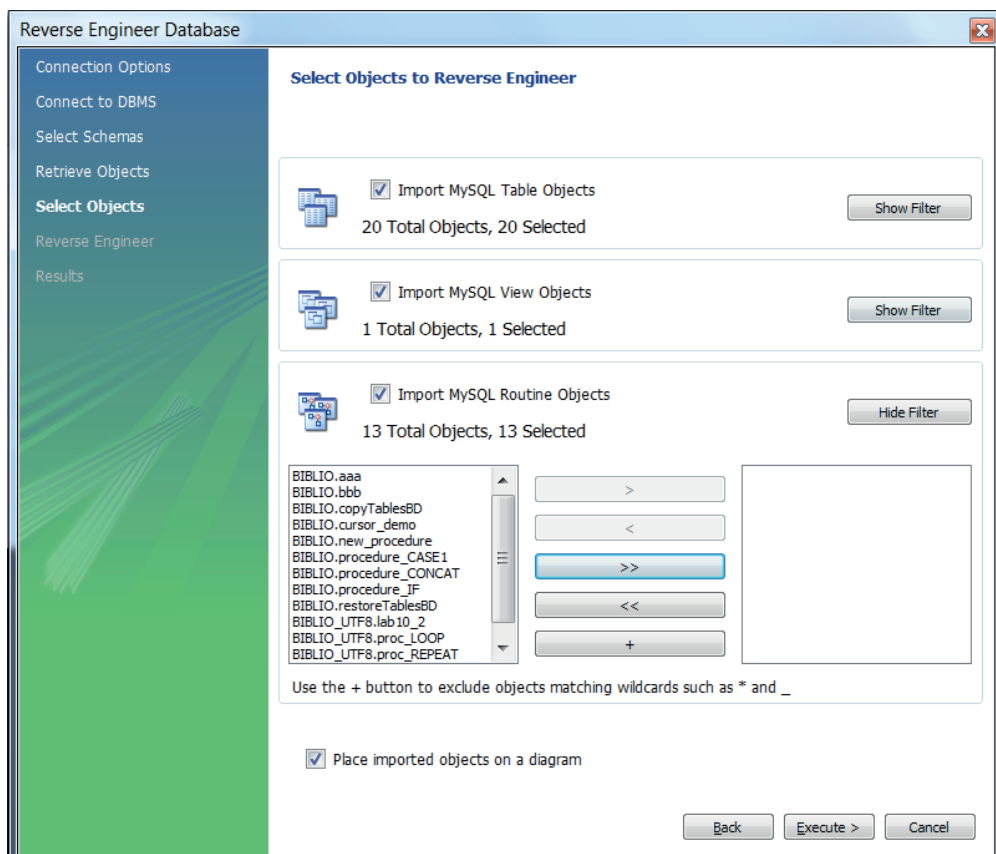


Рис. 12.12. Отримання об'єктів схеми відновлення реверс-інжинірингом

Для отримання схеми БД необхідно виконати команду Execute (реверс-інжиніринг), переглянути процес виконання реверс-інжинірингу (Next), Остаточний екран (Finish) покаже підсумок імпорту.

Після успішного завершення процесу реверс-інжинірингу буде отримано модель з:

- фізичні схеми з об'єктами бази даних, імпортованими з бази даних;
- діаграма ER за замовчуванням з усіма таблицями та представленнями з усіх імпортованих схем.

Для прикладу, було здійснено реверс-інжиніринг одночасно для двох схем даних (двох БД): BIBLIO та BIBLIO\_UTF8.

Результатом будуть дві відповідні схеми БД (рис. 12.13) та одна ER діаграма для двох імпортованих схем (рис. 12.14).

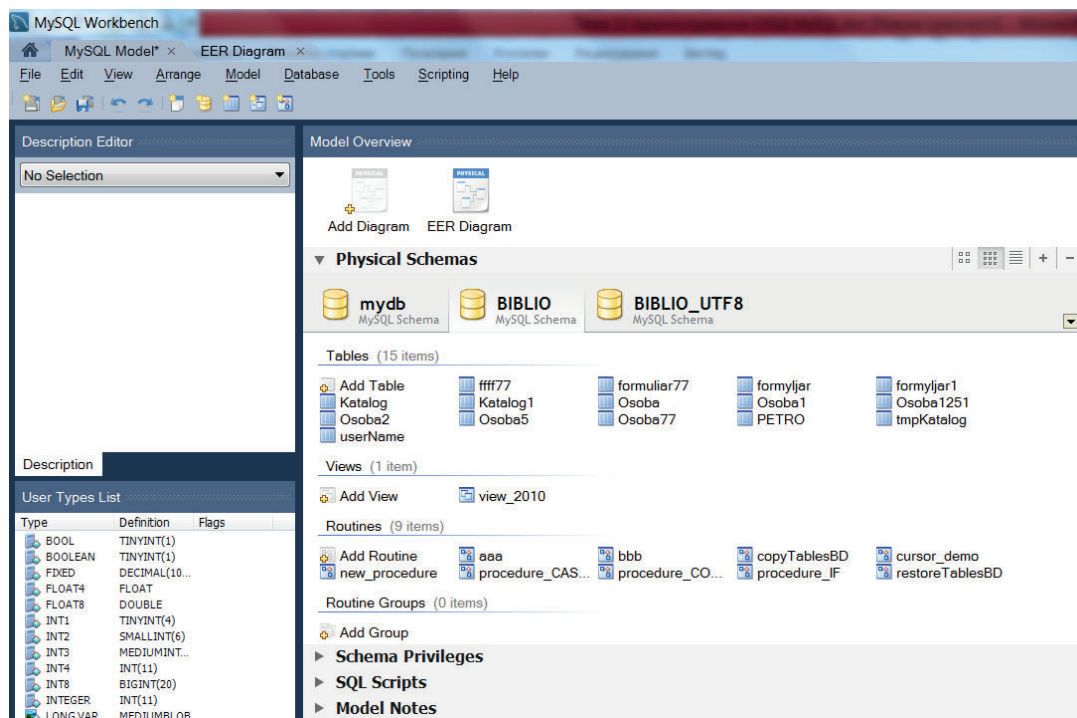


Рис. 12.13. Відновлені схеми БД реверс-інжинірингом

Даний функціонал доцільно використовувати на початкових етапах роботи з БД для оцінки створеної фізичної БД, а також для модифікації та оптимізації у разі потреби.

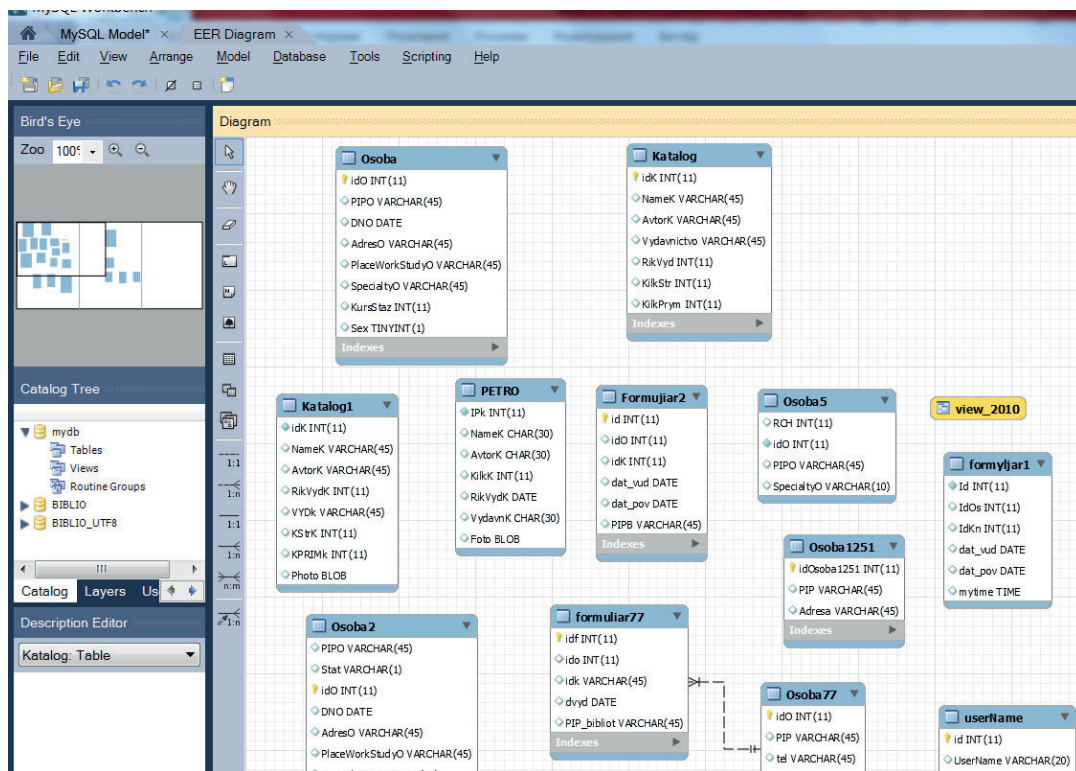


Рис. 12.14. Відновлена реверс-інжинірингом ER діаграма вибраних БД

#### 12.2.2.2. Майстер експорту та імпорту SQL-даних

Майстер експорту та імпорту SQL-даних надає змогу створювати резервні копії як всієї БД, так і її частин, а також даних результатів і, відповідно, відновлювати їх в разі потреби.

Для створення резервної копії БД або її частини майстер експорту SQL-даних викликається командою головного меню Server → Data Export (рис. 12.15).

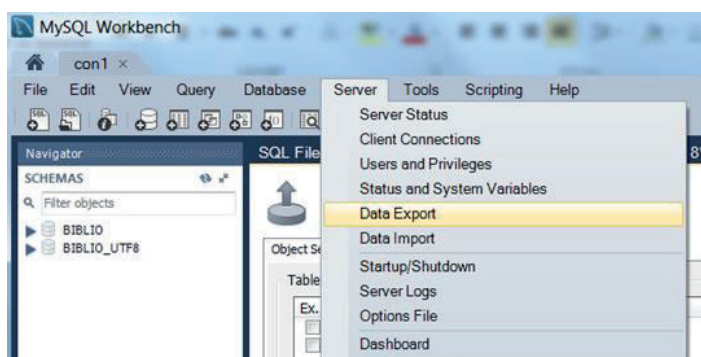


Рис. 12.15. Виклик Майстра експорту SQL-даних

Даний функціонал надає змогу створювати копію як всієї бази чи декілька баз, так і їх частин. Можна створювати копії як тільки структури, як тільки даних, так і структур з даними (рис. 12.16).

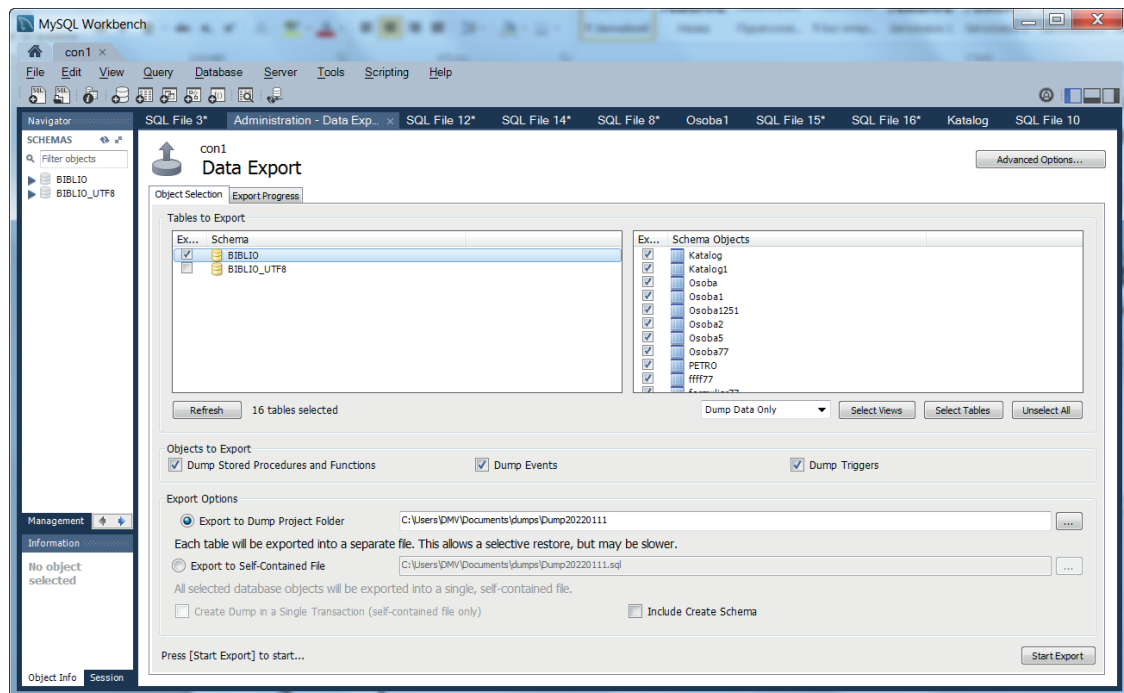


Рис. 12.16. Вікно Майстра експорту SQL-даних

Зберігати всі бекапи можна зручно та впорядковано. За замовчуванням MySQL Workbench пропонує збереження резервної копії БД у файлі з іменем Dump плюс поточна дата експорту.

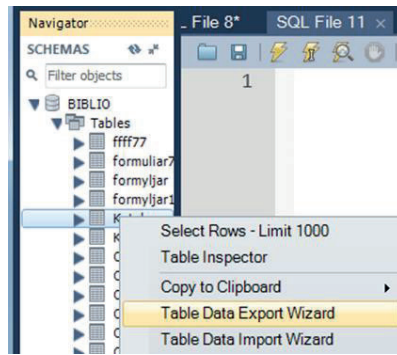
Відновити БД на сервері з експортованих даних також просто. Для цього необхідно виконати зворотну команду Server → Data Import.

Для коректного відновлення необхідно вибрати відповідну резервну копію у полі Import from Dump Project Folder та, власне, здійснити відновлення БД командою Start Import.

Відновлювати можна як окремі елементи БД, так і її цілком.

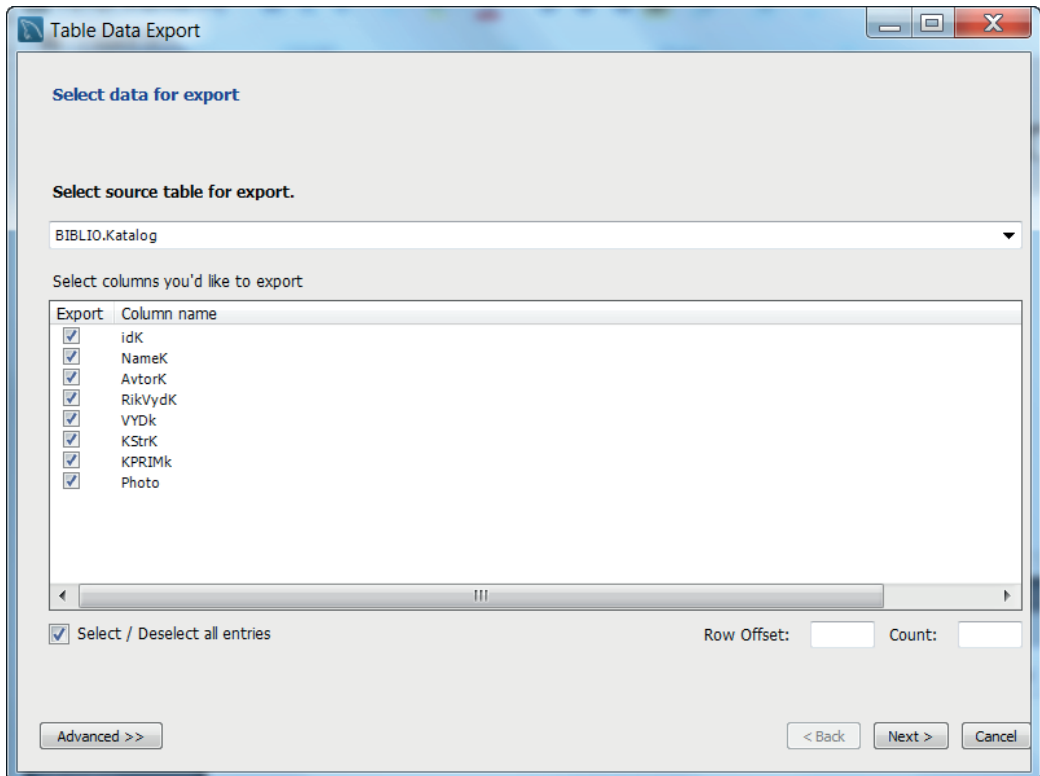
### 12.2.2.3. Майстер експорту та імпорту однієї таблиці

Зробити резервну копію таблиці та відновити її у MySQL Workbench можна також за допомогою контекстного меню і вікні Navigator (рис. 12.17).



*Рис. 12.17. Виклик Майстра експорту таблиці*

Даний функціонал надає змогу зробити бекап тільки вибраної однієї таблиці, при цьому можна також задавати поля (стовпці) таблиці для копіювання та кількість записів (рис. 12.18).



*Рис. 12.18. Вікно Майстра експорту таблиці*

Зберігати бекап таблиці можна на локальному комп'ютері у форматі CSV або JSON, вказавши відповідний шлях та назву бекапу (рис. 12.19).



**Формат CSV** (comma-separated values) – файловий формат, який зазвичай використовується для перенесення даних між базами даних і у якому поля відокремлюються символом коми та переходу на новий рядок. Поля, що містять коми, декілька рядків, або лапки, обмежуються з обох боків лапками.

Останнім часом роздільником у форматі CSV може бути не тільки кома, але й інші символи, такі як пробіл, крапка з комою, табуляція, тощо.

**Формат JSON** (JavaScript Object Notation) – текстовий формат обміну даними, заснований на JavaScript. Та незважаючи на походження від JavaScript, формат вважається незалежним від мови і може використовуватися практично з будь-якою мовою програмування.

Об'єкт JSON це формат даних типу ключ-значення, який зазвичай заключається у фігурні дужки.

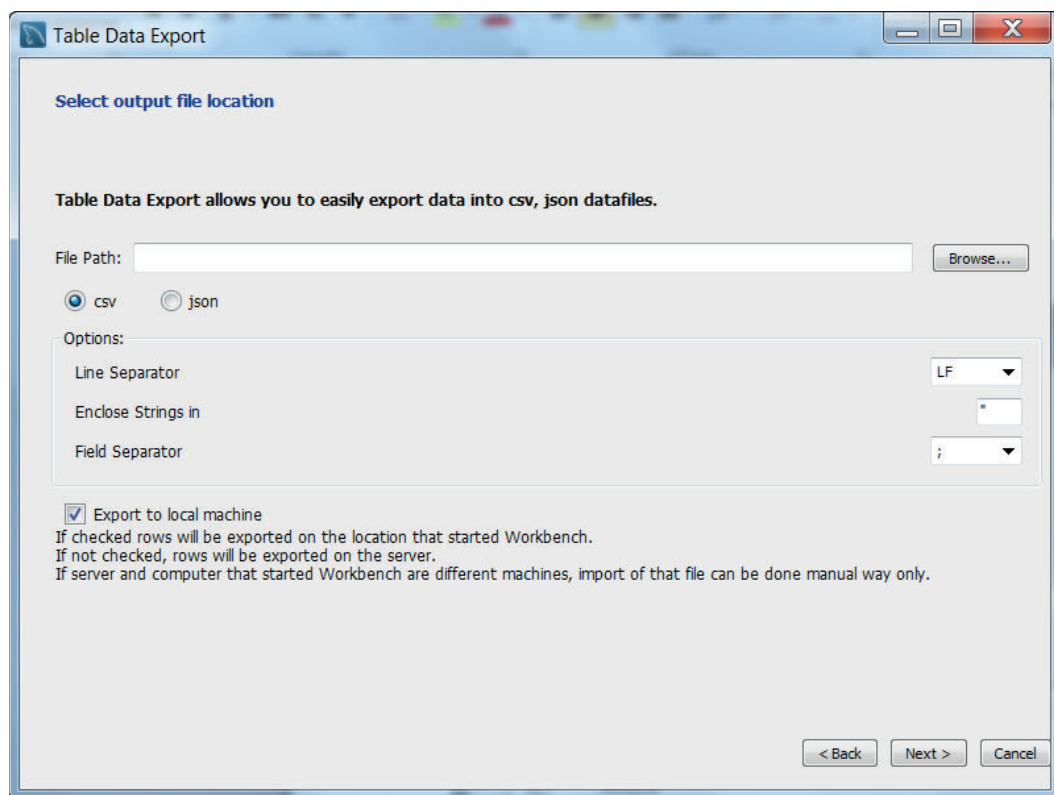
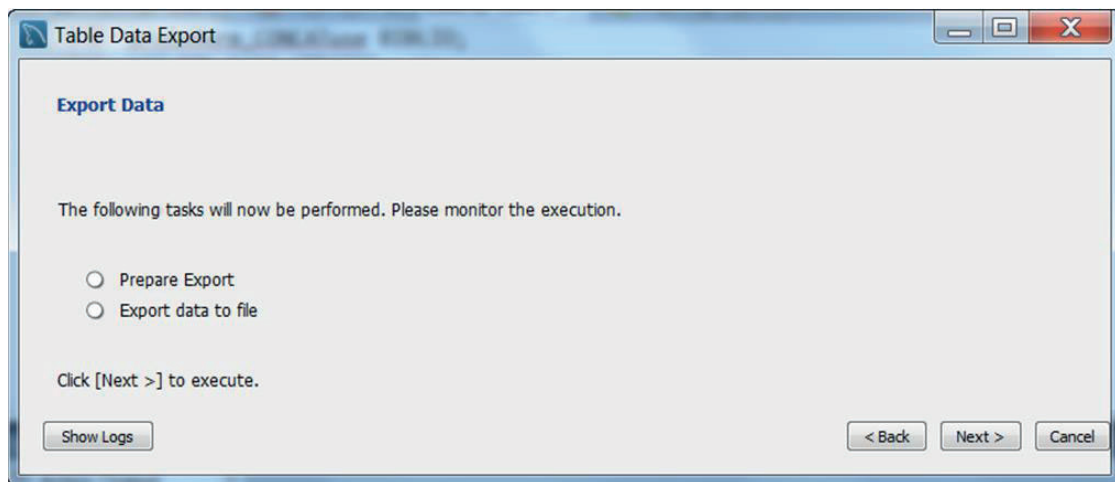


Рис. 12.19. Вікно експорту таблиці у файл – вибір шляху та формату зберігання даних

Для переходу до наступного кроку необхідно задати шлях та ім'я файлу, де буде зберігатися бекап таблиці.

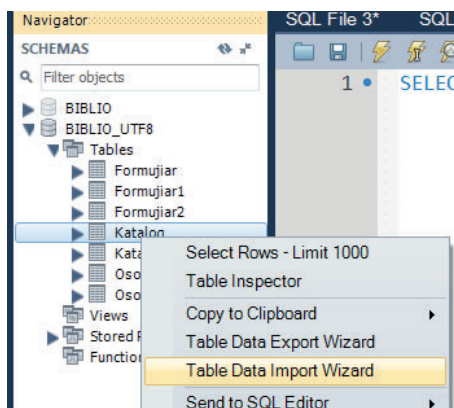
Крім того, щоб дійсно зробити резервну копію БД не на сервері, необхідно обрати опцію Export to local machine.

Наступні кроки підготують дані для експорту та створять заданий файл (рис. 12.20).



*Рис. 12.20. Вікно експорту таблиці у файл – підготовка та створення файлу резервної копії таблиці*

Відновлення таблиці у MySQL Workbench здійснюється за допомогою команди контекстного меню Table Data Import Wizard у вікні Navigator (рис. 12.21).



*Рис. 12.21. Виклик Майстра імпорту таблиці*

#### 12.2.2.4. Майстер експорту та імпорту даних результатів

Отримані результати виконання запитів у візуальному редакторі SQL можна також експортувати у файли форматів CSV, JSON, HTML та XML.

Майстер експорту / імпорту даних результатів викликається відповідними кнопками на панелі інструментів у вікні висвітлення виконання результату запиту (рис. 12.22).

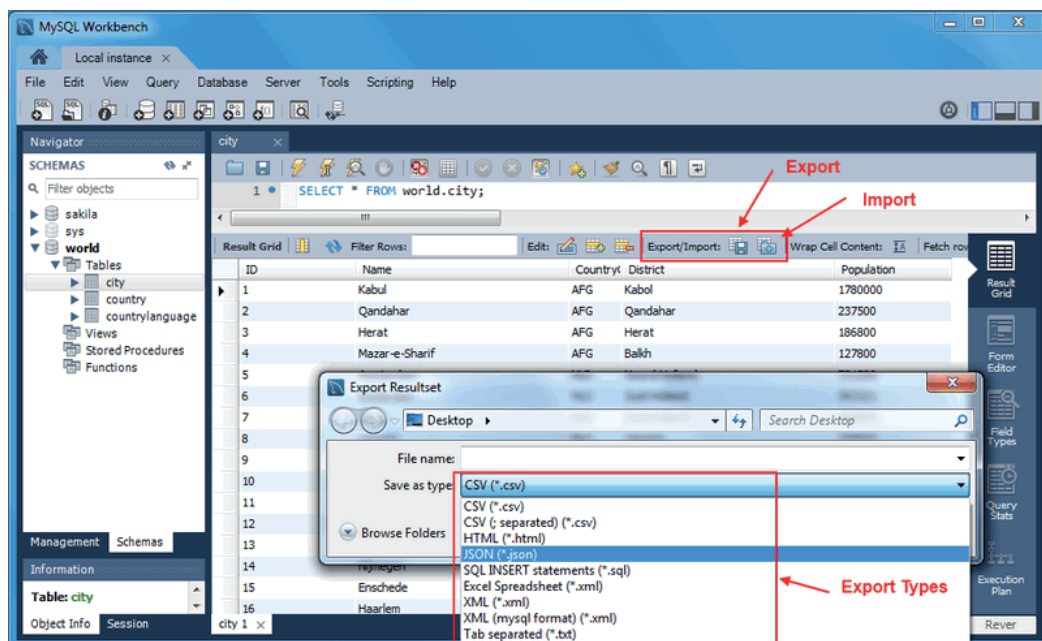


Рис 12.22. Експорт набору результатів

Набір результатів повинен мати унікальний ідентифікатор рядка (наприклад, унікальний індекс Primary Key або NOT NULL), оскільки інакше значення не можна імпортувати, оскільки набір результатів буде доступним лише для читання.

### 12.3. Характерні помилки в MySQL

#### 12.3.1. Діагностика причин виняткових ситуацій

При виникненні **помилки** найперше, що необхідно зробити, так це виявити, що викликає проблемні ситуації – обладнання чи програмне забезпечення.

Якщо після дослідження всіх можливих варіантів помилки з'ясовано, що проблема полягає в клієнті або сервері MySQL, необхідно зробити звіт про

помилку для списку розсилки або групи підтримки. У звіті помилки треба надати дуже детальний опис того, як система веде себе, і що з нею трапляється, а також обґрунтувати, чому це викликано саме MySQL.

Частіше помилок трапляються *виняткові ситуації*, які можна було б передбачити і які пов'язані з некоректними даними, неправильним синтаксисом тощо. Інакшими словами, виняткову ситуацію треба розглядати не як непоправиму помилку, із-за якої необхідно припинити роботу програми, а умову, яка дозволяє обійти цю проблему.

Надійне опрацювання виняткових ситуацій дозволяє значно покращити надійність програмного продукту. В той же час, виняткові ситуації, які невірно визначені або неправильно опрацьовуються, можуть завдати більше проблем, чим без їх опрацювання, оскільки вони утруднюють пошук реальних проблем. Виходячи з цього, для правильного визначення і реалізації методу опрацювання виняткових ситуацій необхідно приділяти достатньо уваги.

Всі проблемні ситуації, що виникають з використанням програмного забезпечення при роботі з MySQL, можна розділити на дві основні групи:

- проблеми, пов'язані встановленням та адмініструванням MySQL;
- проблеми експлуатації MySQL.

Якщо виникають проблеми першої групи, їх повинен вирішувати адміністратор. Зазвичай такі проблемні ситуації не дозволяють надалі працювати з MySQL до усунення їх причин і такі ситуації класифікуються як помилки – «Доступ заборонено» (Access denied).

Повідомлення про помилку «Access denied» інформує користувача про те, під яким ім'ям він намагається увійти в систему, про ім'я хоста, з якого він намагається встановити з'єднання, і про те, чи використовується при цьому пароль чи ні.

До причин другої групи відносяться проблеми, які створює сам користувач MySQL. Більшість таких ситуацій можна передбачити і вони класифікуються як виняткові ситуації.

### 12.3.2. Помилки Cannot Connect to Database Server

Для початку роботи з СКБД MySQL користувачу необхідно з'єднатися з сервером. Якщо таке з'єднання встановити не вдається, сервер видає помилку “Неможливо з'єднатися з сервером бази даних” (*Cannot Connect to Database Server*). І хоча більшість проблем, що виникають при під'єднанні користувача до сервера, відноситься до компетенцій адміністратора і їх рішення знаходиться поза привілеями звичайного користувача, для їх оперативної ідентифікації і виправлення необхідно знати основні з них.

1. Помилка 10060 – під час з'єднання з сервером невірно задана IP-адреса (рис. 12.23).

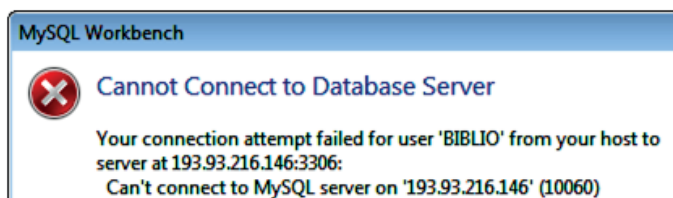


Рис. 12.23. Причина помилки *Cannot Connect to Database Server* – невірна IP-адреса

2. Помилка 10061 – під час з'єднання з сервером не задана IP-адреса (рис. 12.24).

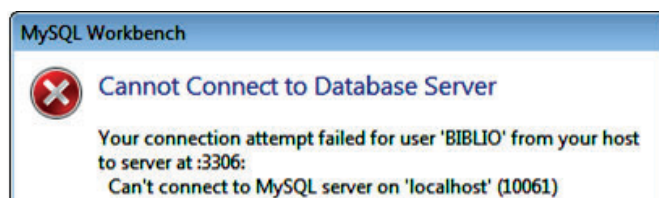


Рис. 12.24. Причина помилки *Cannot Connect to Database Server* – не задана IP-адреса

3. Помилка 10061 – під час з'єднання з сервером невірно заданий порт (рис. 12.25).

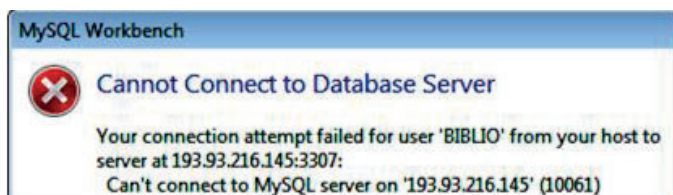


Рис. 12.25. Причина помилки *Cannot Connect to Database Server* – невірний порт

4. Помилка «Доступ заборонено» (Access denied) – під час з'єднання з сервером не задане ім'я бази даних (рис. 12.26).

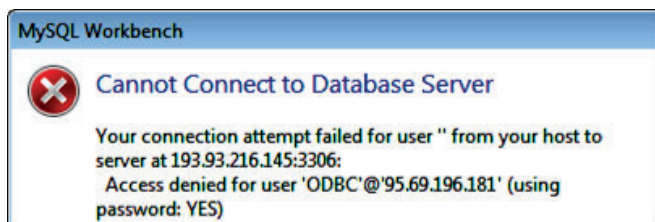


Рис. 12.26. Причина помилки *Cannot Connect to Database Server* – не задане ім'я БД

5. Помилка «Доступ заборонено» – під час з'єднання з сервером невірно задане ім'я бази даних (рис. 12.27).

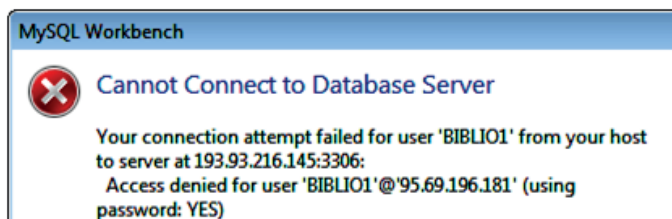


Рис. 12.27. Причина помилки *Cannot Connect to Database Server* – невірне ім'я БД

6. Помилка «Доступ заборонено» – під час з'єднання з сервером невірно заданий пароль (рис. 12.28).

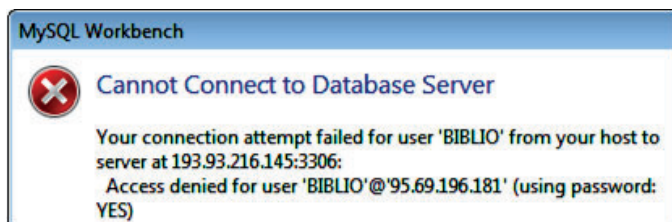


Рис. 12.28. Причина помилки *Cannot Connect to Database Server* – невірний пароль

7. Помилка «Доступ заборонено» – під час з'єднання з сервером не заданий пароль (рис. 12.29).



Рис. 12.29. Причина помилки *Cannot Connect to Database Server* – не заданий пароль

Якщо ж пароль не працює, необхідно врахувати, що при задаванні пароля в таблиці user за допомогою команд INSERT, UPDATE або SET PASSWORD треба використовувати функцію PASSWORD(), яка шифрує паролі. Якщо ж пароль задається за допомогою команди GRANT ... IDENTIFIED BY, тоді шифрування здійснюється цією функцією і, відповідно, функція PASSWORD() буде зайвою.

8. Помилка 10065 – відсутня мережа або сервер MySQL виключений (рис. 12.30).

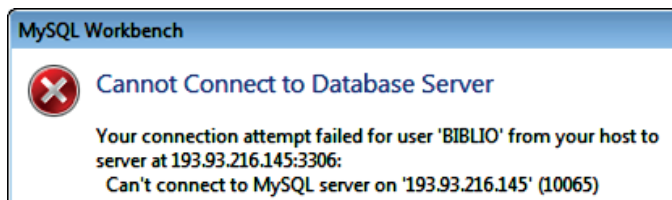


Рис. 12.30. Причина помилки *Cannot Connect to Database Server* – відсутня мережа або сервер MySQL виключений

При виникненні помилок з'єднання з SQL сервером необхідно:

- по-перше, перевірити правильність задавання IP-адреси, порту, імені БД та паролю;
- по-друге, звернутися до адміністратора.

### 12.3.3. Проблеми при роботі з сервером

#### 12.3.3.1. Причина «server has gone away»

Першою причиною повідомлення «server has gone away» може бути те, що у сервера закінчився час очікування і він закрив підключення. За замовчуванням сервер закриває підключення після 8 годин (28800 секунд) відсутності активності. Час очікування активності зберігається у системній змінній wait\_timeout (рис. 12.31).

Час очікування сервера можна змінити за допомогою команди SET без потреби зупинки і перезавантаження сервера, маючи привілеї SUPER.

Значення системної змінної wait\_timeout можна також використовувати у виразах.





Рис. 12.31. Час очікування сервером активності користувача

Другою причиною повідомлення «server has gone away» може бути те, що користувач закрив з'єднання і хоче виконати запит при закритому підключенні до сервера.

Зазвичай в цьому випадку кодами помилки можуть бути:

**CR\_SERVER\_GONE\_ERROR** Клієнт не може послати запит на сервер

Клієнт не отримав помилку при звертанні до

**CR\_SERVER\_LOST** сервера, але і не отримав повну відповідь (або будь-яку відповідь) на запит

Ці помилки можуть виникати також при надсиланні неправильного або занадто довгого запиту до сервера. Якщо `mysqld` отримує пакет, який є занадто великим або поза правилами, він вважає, що щось пішло неправильно з клієнтом і закриває підключення.

Якщо ж є необхідність великих запитів, зокрема для роботи з даними типу BLOB, можна збільшити обмеження запиту, запускаючи `mysqld` з опцією:

`-O max_allowed_packet=#` (за замовчуванням 1M).

Пам'ять додаткового простору розподілена на вимогу, так що `mysqld` використовує більшу кількість пам'яті тільки тоді, коли надсилається великий запит, або коли `mysqld` має повернути велику кількість результатів!

### 12.3.3.2. Помилка Host '...' is blocked

Якщо в ході роботи з сервером виникає забагато помилок з'єднання, MySQL сервер вважає, що відбувається щось не так, на зразок атаки хакерів на сервер, і блокує подальші з'єднання з цим комп'ютером на зразок:

Host 'hostname' is blocked because of many connection errors.

Unblock with 'mysqladmin flush-hosts'



Максимально допустима кількість некоректних спроб з'єднання з сервером зберігається у глобальній системній змінній `max_connect_errors`.

За замовчуванням MySQL блокує комп'ютер після 10 помилок підключення.

Для розблокування сервера необхідно виконати команду у Workbench з іншого комп'ютера:

```
FLUSH HOSTS;
```

Також можна збільшити значення системної змінної `max_connect_errors` під час запуску сервера, наприклад, до 1000:

```
safe_mysqld -O max_connect_errors=1000 &
```

Але в першу чергу необхідно перевірити правильність TCP/IP підключення для конкретного комп'ютера. Якщо TCP/IP підключення не працюють, збільшення змінної `max_connect_errors` не допоможе.

#### 12.3.3.3. Помилка Too many connections

Кількість користувачів, що можуть одночасно працювати з клієнт-серверною БД, в MySQL визначається системною змінною `max_connections`, яка за замовчуванням приймає значення 100.

У випадку перебільшення кількості одночасно під'єднаних користувачів до БД сервер видає помилку:

Too many connections

Максимальна кількість підключень залежить від того, наскільки добра бібліотека потоків на даній платформі. Linux або Solaris можуть підтримувати 500-1000 одночасних підключень.

Щоб збільшити кількість одночасно підключених користувачів БД, необхідно перезапустити MySQL з відповідним значенням для змінної `max_connections`.

Для виправлення ситуації MySQL дозволяє фактично (`max_connections+1`) підключень.

Фактичне значення змінної `max_connections` визначається командою (рис. 12.32):

```
SHOW VARIABLES LIKE "max_connection";
```

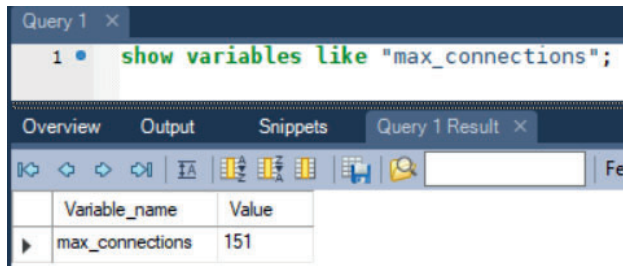


Рис. 12.32. Значення системної змінної *max\_connection*

Останнє підключення зарезервовано для користувача із привілеєм *process*. Не даючи цей привілей нормальним користувачам (вони не повинні потребувати її), адміністратор з цим привілеєм може увійти та використовувати *SHOW PROCESSLIST*, щоб з'ясувати те, що могло б бути неправильно.

Також можна збільшити значення змінної *max\_connections* за умови, що на сервері достатньо оперативної пам'яті для підтримки збільшення кількості з'єднань:

```
set global max_connections = 200;
```

Щоб застосувати зміни для *max\_connections* необхідно перевантажити сервер.

#### 12.3.3.4. Помилка *Out of memory*

Дана помилка стосується клієнта MySQL і її причиною можуть бути:

##### *1. Неоптимізовані таблиці та запити*

Більшість веб-сайтів створені на основі програмного забезпечення CMS і керуються базою даних. Налаштування, плагіни, програми сторонніх розробників – все це може включати складний код або запити низької якості, включаючи складні запити сортування та об'єднання, що спричиняє завантаження всієї пам'яті і призводить до збою MySQL. Першим кроком є пошук джерел витоку пам'яті.

З часом бази даних можуть збільшуватися в розмірах, а також у них можуть утворюватися фрагменти через видалення даних. Обидва ці фактори призводять до втрати пам'яті.

##### *2. Конфігурація сервера MySQL*

Пам'ять, виділена службі MySQL, залежить від різних параметрів, налаштованих у ній. Приблизна формула використання пам'яті MySQL:

Максимальне використання пам'яті MySQL = innodb\_buffer\_pool\_size + key\_buffer\_size + ((read\_buffer\_size + read\_rnd\_buffer\_size + sort\_buffer\_size + join\_buffer\_size) × max\_connections).

Розмір пулу буферів, розміри буферів для читання, сортування та приєднання, максимальні з'єднання, тимчасові таблиці, усі разом визначають задіяну пам'ять.

Оптимальне використання пам'яті не повинно перевищувати 60% загальної доступної оперативної пам'яті. Це робиться ітераційним способом, з налаштуваннями, які виконуються до досягнення ідеального значення

Визначення оптимальної пам'яті службі MySQL допомагає гарантувати, що MySQL не виходить з ладу через недостатню кількість пам'яті, і він отримує ідеальні ресурси, які їй потрібні.

### ***3. Ресурсоємні процеси***

Іноді можуть бути такі програми, як Java, Python, PHP, ColdFusion, програмне забезпечення для резервного копіювання, програми сторонніх розробників тощо, які споживають занадто багато пам'яті.

Можуть бути користувачі, які зловживають ресурсами сервера, якщо вони не обмежені належним чином. Такі процеси та користувачі можуть «з'їсти» всю пам'ять сервера та призвести до збою служби, як-от MySQL.

Щоб утримати один процес від зловживання ресурсами сервера, необхідно налаштувати всі серверні програми таким чином, щоб використовувати не більше фіксованої частки доступної пам'яті.

Встановлення лімітів облікових записів користувачів є ще одним запобіжним заходом.

### ***4. Недостатня оперативна пам'ять***

Коли ж всі перераховані вище причини виключаються, залишається одна — сервер не має достатньо пам'яті для виконання всіх процесів на ньому.

За допомогою інструментів моніторингу пам'яті можна відстежувати використання пам'яті різними процесами в часі. Постійні помилки «Недостатньо пам'яті», навіть при нормальному трафіку, вказують на недостатню пам'ять.

В цьому випадку залишається або обмежувати використання пам'яті, або ж збільшити фізично оперативну пам'ять сервера.

### **12.3.3.5. Помилка Table 'xxx' doesn't exist**

Існує кілька причин і каталізаторів цієї помилки 1146:

- ***Збій InnoDB і відсутні файли даних***

Оскільки InnoDB є безпечним для транзакцій і володіє такими механізмами, як відкат, фіксація та відновлення даних бази даних MySQL, він схильний до пошкоджень. І якщо з ним працювати не належним чином, існує ймовірність збою InnoDB.

В InnoDB файл `ibdata` та файл журналу діють як словник даних для нього. Якщо з будь-якої причини ці файли зникнуть, у InnoDB не буде шансів працювати, а MySQL покаже вам помилку 1146.

- ***Неправильні права доступу до каталогу***

MySQL має каталог зберігання даних, де зберігається вся база даних, зазвичай за шляхом – `'/var/lib/mysql'`. Якщо права доступу неправильні, відповідно MySQL не може отримати доступ за цим шляхом до каталогу, тому і буде помилка 1146.

- ***Пошкодження в таблицях даних***

Таблиці бази даних MySQL можуть бути пошкоджені через кілька причин, як-от неправильне вимкнення сервера, неповні запити, зловживання користувачами, неправильні формати, помилка обмеження зовнішнього ключа тощо. Отже, якщо таблиця пошкоджена, база даних може показати помилку 1146.

- ***Відсутній файл .frm***

Файл `.frm` містить формат і структуру бази даних. Якщо в будь-якому випадку цей файл не буде скопійований до каталогу бази даних або його буде видалено, з'явиться повідомлення про помилку з кодом помилки 1146.

***Методи виправлення помилки MySQL 1146 «Таблиця не існує»:***

### 1) Відновлення БД з резервної копії.

Це найкраща альтернатива, яку потрібно використовувати. Так, безсумнівно, буде відновлено усі втрачені таблиці з бази даних. Але для відновлення має бути файл резервної копії. Тому завжди зберігайте звичку час від часу створювати резервні копії, це хороша практика.

Крім того, відновлення БД з резервної копії усуває помилку MySQL ERROR 1146, яка не існує після резервного відновлення.

### 2) Перезавантажити Сервер.

Одним з методів щодо виправлення збою роботи сервера може бути перезавантаження сервера, Таким чином є шанс повернути сервер в робочий стан і видалити код помилки. Такі дії матимуть успіх, якщо збій відбувся після неправильного вимкнення сервера.

### 3) Ремонт таблиць бази даних

Зручним інструментом для ремонту таблиць є phpMyAdmin, у якому необхідно вибрати відповідну БД, у ній – відмітити потрібну таблицю (поставити галочку) та виконати команду «Ремонтувати таблицю» (рис. 12.33).

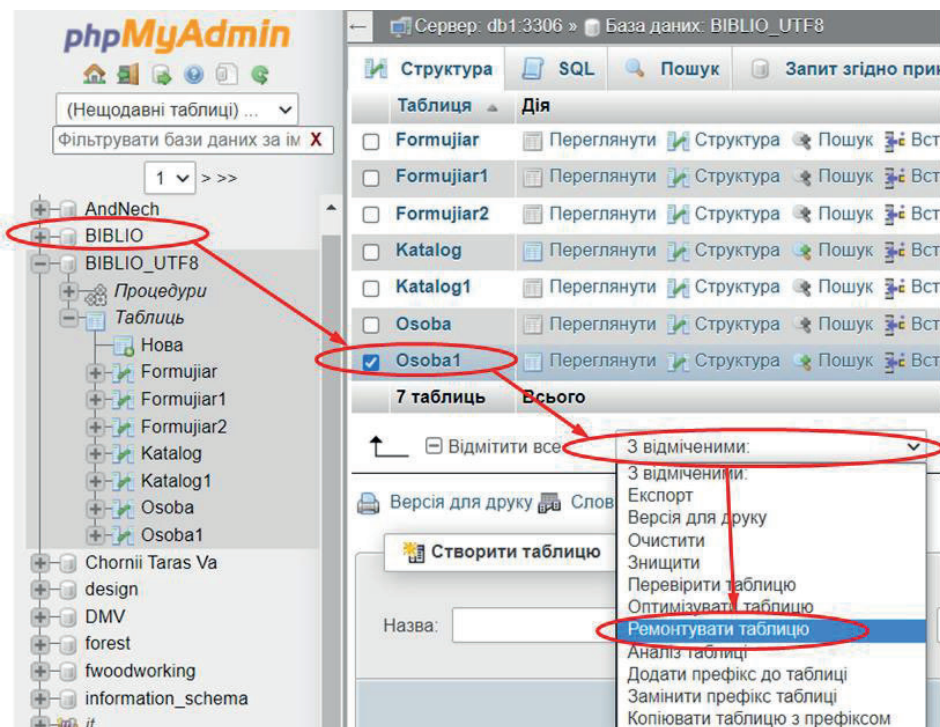


Рис. 12.33. Ремонт таблиці в phpMyAdmin

Можна також відновити таблиці бази даних за допомогою MySQL Command-Line Client (MySQL CLI). Для цього необхідно виконати декілька кроків:

- за допомогою протоколу SSH підключитися до сервера. Для цього у Workbench створити нове з'єднання, вибравши метод Standard TPC/IP over SSH;
- з командного рядка виконати команду:

```
mysql -u <ім'я користувача> -p
```

- ввести свій пароль для імені користувача;
- обрати необхідну БД за замовчуванням, ввівши команду:

```
use <назва бази даних>;
```

- за необхідності вивести список таблиць поточної БД:

```
show tables;
```

- відремонтувати таблицю за допомогою таких команд:

(a) перевірити наявність таблиць з помилкою:

```
check table <ім'я таблиці>;
```

(b) здійснити ремонт таблиці:

```
repair table <ім'я таблиці>;
```

- вийти з режиму SSH за допомогою команди quit .

#### 4) відновлення файлів .frm та ibdata

Можна зберігати копії файлів .frm та ibdata, і щоразу, коли виникає помилка 1146, відновлювати ці файли до папки схеми з його копії. Це може виправити помилку 1146.

І як висновок, для відновлення працездатності БД, в тому числі для відновлення таблиць, необхідно завжди зберігати резервну копію БД.

### 12.3.3.6. Помилка Can't initialize character set xxx

Проблеми з набором символів можуть призвести до такої помилки:

MySQL Connection Failed: Can't initialize character set xxx

Ця помилка може бути викликана наступними причинами:

- Кодування є багатобайтовим і не підтримується клієнтом. У цьому випадку необхідно перекомпілювати клієнта, запустивши `configure` з параметром:

`--with-charset=xxx`

або з

`--with-extra-charsets=xxx.`

Усі стандартні двійкові файли MySQL компілюються з опцією:

`--with-extra-character-sets=complex,`

що забезпечує підтримку всіх багатобайтових кодувань.

- Кодування є простим кодуванням, не скомпільованим у `mysql`, і файли визначення кодування знаходяться не там, де розраховує їх знайти клієнт. В цьому випадку необхідно:
  - перекомпілювати клієнта з підтримкою цього кодування;
  - вказати клієнту, де розташовані файли визначення кодування. Для багатьох клієнтів це можна зробити за допомогою опції:  
`--character-sets-dir=path-to-charset-dir;`
  - скопіювати файли визначення кодування туди, де, на думку клієнта, вони мають бути.

#### 12.3.4. Інші виняткові ситуації

Багато виняткових ситуацій виникає також при формуванні та виконанні запитів безпосередньо до даних БД. Серед них можна виокремити:

- проблеми з використанням значень `DATE`;
- проблеми з використанням значень `NULL`;
- проблеми з видаленням значень зв'язаних таблиць тощо.

Також можна звернути увагу на виняткові ситуації під час модифікації структури БД та таблиць, пов'язаних з використанням запиту `ALTER TABLE`.

Аналогічно, виняткові ситуації можуть мати місце при створенні збережених функцій і процедур, при розробленні транзакцій, тригерів, тощо.

І основою для запобігання таких ситуацій є прискіпливе моделювання БД. Тобто, як спроектована та нормалізована БД, так і вона буде працювати.

Звичайно, розглянути всі виняткові ситуації в одній темі не можливо. Але розглянуті вище, найбільш характерні, мають допомогти у роботі з MySQL БД.

## Контрольні запитання

1. Що таке привілеї користувачів?
2. Які є групи привілеїв?
3. Які привілеї має суперкористувач?
4. Якою командою встановлюються привілеї користувачу?
5. Як скасувати раніше надані права?
6. Як задати спеціальні привілеї користувачу?
7. Якою командою оновлюються надані привілеї командою GRANT?
8. Коли доцільно використовувати phpMyAdmin?
9. Чи можна змінити привілеї користувача командою UPDATE?
10. Як змінити пароль користувачу?
11. Які способи створення резервних копій БД є за способом зберігання?
12. Які утиліти використовуються для створення резервних копій БД та їх відновлення?
13. Чи може утиліта mysqldump.exe зберегти резервну копію БД на іншому комп'ютері, чи тільки на сервері?
14. В яких випадках доцільно використовувати команду SELECT \* INTO OUTFILE для створення резервної копії?
15. Чи може створити резервну копію БД з даними інструмент Reverse Engineer?
16. Який функціонал надає MySQL Workbench для створення повноцінних копій БД?
17. В яких форматах можна реалізувати бекап за допомогою команди Data Export в MySQL Workbench?
18. Як в MySQL Workbench викликати Майстер експорту таблиці?
19. Чи можна в MySQL Workbench зробити бекап результатів вибірок?
20. Що вважається винятковою ситуацією?
21. Як можна «відремонтувати» таблицю?



# Розділ 5. БАЗИ ДАНИХ NoSQL

## Тема 13. Основи баз даних NoSQL

### 13.1. Основні поняття NoSQL

Що не так з реляційними базами даних?

Справа в тому, що вони дійсно чудово працювали протягом багатьох років, але тепер з'явилася проблема, з якою вони більше не можуть впоратися. Сьогодні людство генерує 50 000 гігабайт даних в секунду. Це колосальний обсяг даних! Його зберігання і обробка є серйозним інженерним викликом. І цей обсяг постійно зростає...

Прикладами є Facebook, Google тощо.

Як виявилось, реляційні бази даних погано пристосовані до роботи з дійсно великими обсягами даних.

Вони спроектовані для роботи на одній машині, а для розподіленої роботи на кількох машинах нам потрібна інша технологія баз даних.

Перш за все, NoSQL (Not only SQL) – це не тільки реляційні бази даних.

NoSQL є загальним терміном для різних баз даних і сховищ, але він не виділяє деяку одну конкретну технологію або продукт.

Термін NoSQL застосовується до сховищ даних, які не використовують мову запитів SQL, а оперують з даними за допомогою інших мов і конструкцій. На практиці NoSQL означає "нереляційна база даних", навіть незважаючи на те, що багато які з цих баз даних не підтримують сумісні з SQL запити.

Нереляційна база даних NoSQL – це база даних, в якій, на відміну від більшості традиційних систем баз даних, не використовується таблична схема рядків і стовпців. У цих базах даних застосовується модель зберігання, оптимізована під конкретні вимоги типу збережених даних. Наприклад, дані можуть зберігатися як прості пари "ключ - значення", документи JSON або граф, що складається з ребер і вершин.

## 13.2. Категорії NoSQL

Кількість конкуруючих NoSQL баз даних дуже велика. А причина проста: різні бази даних NoSQL призначені для вирішення різних завдань.

NoSQL бази даних здебільшого поділяють на чотири основні категорії:

1. **Документно-орієнтовані бази даних** – надають можливість зберігання складних вкладених документів, в той час як більшість реляційних баз даних підтримує лише одновірні рядки
2. **Бази даних типу "ключ/значення"** – зазвичай реалізують найпростішу NoSQL-модель.
3. **Графові бази даних** – предметні області представлені у вигляді графів
4. **Стовпцеві бази даних** – створюють по файлу для кожного стовпця ваших таблиць.

### 13.2.1. Документно-орієнтовані бази даних

Сховище даних документів управляє набором значень іменованих рядкових полів і даних об'єкта по суті, яка називається документом. Зазвичай дані в цих сховищах містяться у вигляді документів JSON (рис. 13.1).

| Key  | Document                                                                                                                                                                                                                          |
|------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1001 | <pre>{   "CustomerID": 99,   "OrderItems": [     { "ProductID": 2010,       "Quantity": 2,       "Cost": 520     },     { "ProductID": 4365,       "Quantity": 1,       "Cost": 18     }   ],   "OrderDate": "04/01/2017" }</pre> |
| 1002 | <pre>{   "CustomerID": 220,   "OrderItems": [     { "ProductID": 1285,       "Quantity": 1,       "Cost": 120     }   ],   "OrderDate": "05/08/2017" }</pre>                                                                      |

Рис. 13.1. Документно-орієнтовані бази даних

Кожне значення поля може бути як скалярним елементом, наприклад число, так і складним об'єктом, наприклад список або колекція типу "батько - нащадок". Дані в полях документа можна закодувати різними способами, наприклад у форматі XML, YAML, JSON, BSON, або зберігати у вигляді звичайного тексту. Поля документів доступні системі управління сховищем, що дозволяє додатку виконувати запити і застосовувати фільтри, засновані на значеннях цих полів.

Документ, як правило, містить всі дані однієї сутності. Елементи, що становлять сутність, залежать від конкретного додатка. Наприклад, сутність може містити відомості про клієнта, замовлення або й ті, і інші. Один документ може містити відомості, які в реляційній СКБД зазвичай розподіляються по декількох реляційних таблицях. Сховище документів не зобов'язує використовувати однакову структуру для всіх документів. Підтримка вільної форми забезпечує більшу гнучкість. Наприклад, додатки можуть зберігати в документах різні дані відповідно до поточних вимог компанії.

Додаток може отримувати документи за ключем документа. Це унікальний ідентифікатор документа. Часто до нього застосовується хешування для рівномірного розподілу даних. Деякі бази даних документів створюють ключ документа автоматично. Інші дозволяють вказати, який атрибут документа слід використовувати в якості ключа. Також додаток може запитувати документи за значеннями одного або декількох полів. Деякі бази даних документів підтримують індексування, що прискорює пошук документів за одним або декількома індексованими полями.

Багато баз даних документів підтримують поновлення "на місці", тобто дозволяють додаткам змінювати значення окремих полів без перезапису всього документа. Операції читання і запису для кількох полів в одному документі зазвичай є атомарними.

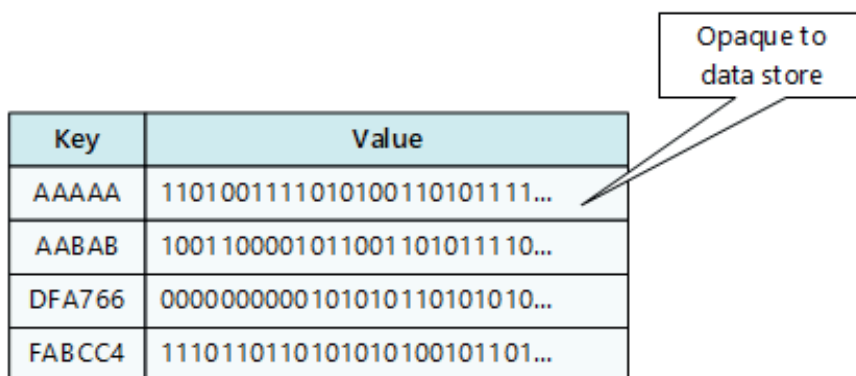
Документоорієнтовані СКБД служать для зберігання ієрархічних структур даних. Знаходять своє застосування в системах управління вмістом, видавничій справі, документальному пошуку.

Приклади СКБД даного типу – CouchDB, Couchbase, MongoDB, eXist, Berkeley DB XML.

### 13.2.2. Бази даних типу "ключ/значення"

Сховище пар "ключ - значення" по суті є великою хеш-таблицею (рис. 13.2). Кожне значення зіставляється з унікальним ключем, і сховище ключів використовує цей ключ для зберігання даних, застосовуючи до нього деяку функцію хешування. Вибір функції хешування повинен забезпечити рівномірний розподіл хешованих ключів по сховищу даних.

Більшість сховищ пар "ключ - значення" підтримують тільки найпростіші операції запитів, вставки і видалення. Щоб частково або повністю змінити значення, додаток завжди перезаписує існуюче значення цілком. У більшості реалізацій атомарної операцією вважається читання або запис одного значення. Запис великих значень займає відносно довгий час.



| Key    | Value                        |
|--------|------------------------------|
| AAAAA  | 110100111101010011010111...  |
| AABAB  | 1001100001011001101011110... |
| DFA766 | 0000000000101010110101010... |
| FABCC4 | 1110110110101010100101101... |

Рис. 13.2. Бази даних типу "ключ/значення"

Додаток може зберігати в наборі значень довільні дані, але деякі сховища пар "ключ - значення" накладають обмеження на максимальний розмір значень. Програмне забезпечення сховища нічого не знає про значення, які в ньому зберігаються. Всі відомості про схему підтримуються і застосовуються на рівні додатку. Ці значення по суті є великими двійковими об'єктами, які сховище витягує і зберігає за відповідним ключем.

Сховища пар "ключ - значення" розраховані на додатки, які виконують прості операції пошуку на основі значення ключа або діапазону ключів, але не дуже підходять для систем, яким потрібно запитувати дані з декількох таблиць сховищ пар "ключ - значення", наприклад приєднані дані в декількох таблицях.

Крім того, сховища пар "ключ - значення" незручні в сценаріях, де можуть виконуватися запити або фільтрація за значенням, а не тільки по ключам. Наприклад, за допомогою реляційної бази даних можна знайти запис, використовуючи пропозицію WHERE для фільтрації неключових стовпців, але в сховищах "ключ-значення" зазвичай відсутня можливість пошуку в значеннях, або, якщо вони є, потрібен повільний перегляд всіх значень.

Одне сховище пар "ключ - значення" дуже легко масштабується, оскільки дозволяє зручно розподілити дані серед декількох вузлів на різних комп'ютерах.

Модель «ключ - значення» є найпростішим варіантом, що використовують ключ для доступу до значення. Такі системи використовуються для зберігання зображень, створення спеціалізованих файлових систем, як кешів для об'єктів, а також в системах, спроектованих з прицілом на масштабованість.

Приклади таких сховищ – Berkeley DB, MemcacheDB, Redis, Riak, Amazon DynamoDB.

### **13.2.3. Графові бази даних**

Сховища даних графів керують відомостями двох типів: вузлами і ребрами. Вузли в цьому випадку представляють суті, а ребра визначають зв'язки між ними. Вузли та ребра мають властивості, які надають відомості про конкретний вузол або зв'язок, приблизно як стовпці в реляційної таблиці. Грані можуть мати напрямок, які вказують на характер зв'язку.

Сховища даних графів дозволяють додаткам ефективно виконувати запити, які проходять через мережу вузлів і ребер, а також аналізувати зв'язки між сутностями. На рис. 13.3 представлені дані персоналу організації, структуровані у вигляді графа.

Сутностями тут є співробітники і відділи, а ребра визначають відношення підпорядкування і відділ, в якому працює кожен співробітник. На цьому графі межі мають строгий напрям, який на схемі позначено стрілками.

Така структура дозволяє легко виконувати такі запити, як "знайти всіх співробітників, які прямо або побічно підкоряються Світлані" або "знайти всіх, хто працює в одному відділі з Дмитром". Для великих графів з великою кількістю сутностей і зв'язків можна швидко виконувати складний аналіз. Багато баз даних

графів надають мову запитів, який можна використовувати для ефективного обходу мережі зв'язків.

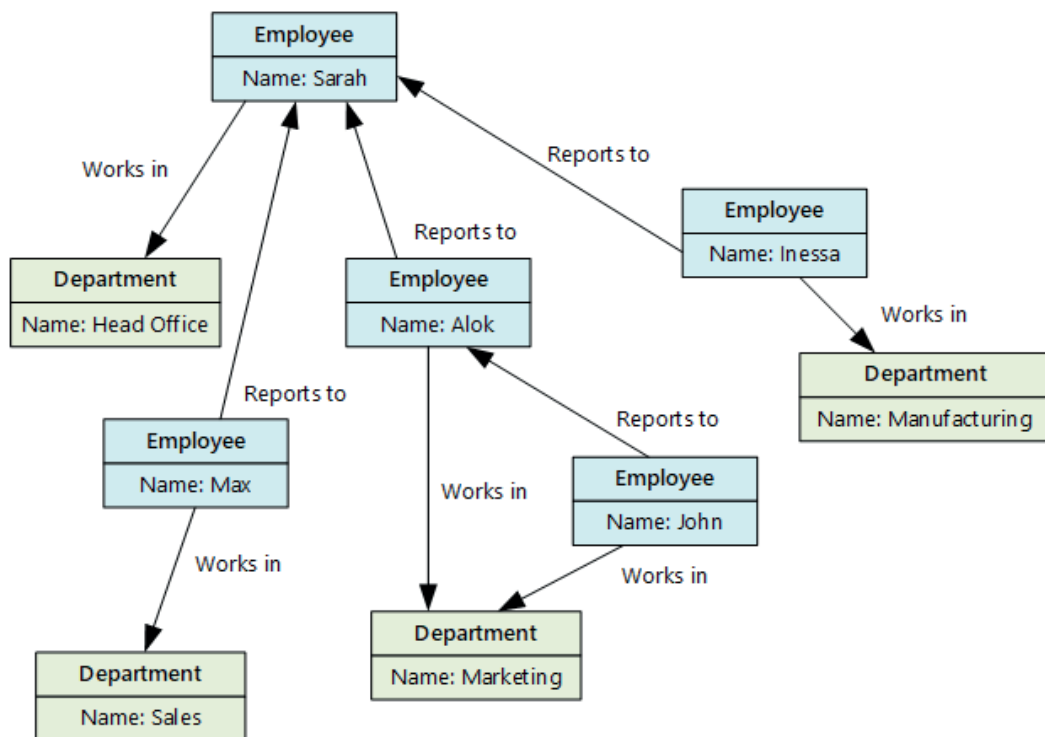


Рис. 13.3. Графові бази даних

Графові СКБД застосовуються для задач, в яких дані мають велику кількість зв'язків, наприклад, соціальні мережі, виявлення шахрайства.

Приклади: Neo4j, OrientDB, AllegroGraph, Blazegraph, InfiniteGraph, FlockDB, Titan.

Так як ребра графа матеріалізовані, тобто є збереженими, обхід графа не вимагає додаткових обчислень (як зв'язки в SQL), але для знаходження початкової вершини обходу потрібна наявність індексів. Графові СКБД як правило підтримують ACID, а також підтримують спеціалізовані мови запитів, такі як Gremlin, Cypher, SPARQL, GraphQL.

### 13.2.4. Стовпцеві бази даних

Стовпчасті сховища даних або сховища сімейств стовпців впорядковує дані по стовпцях і рядках. Стовпчасті сховища даних в простій формі майже не відрізняються від реляційної бази даних, по крайній мері організаційно. Справжня перевага стовпцевого сховища даних полягає в здатності денормалізовано структурувати розріджені дані, що пов'язано зі стовбцево-орієнтованим методом зберігання даних.

Стовпчасті сховища даних можна представити як набір табличних даних з рядками і стовпцями, в яких стовпці поділяються на певні групи або сімейства стовпців. Кожне сімейство стовпців включає набір логічно пов'язаних стовпців, які зазвичай беруться або управляються як єдине ціле. Інші дані, які використовуються в інших процесах, зберігаються окремо в інших родинках стовпців. У сімейство стовпців можна динамічно додати нові стовпці, а рядки можуть бути розрідженими (тобто рядки не зобов'язані мати значення для кожного стовпця).

На рис. 13.4 представлений приклад таблиці з двома родинками стовпців: Identity і ContactInfo. Дані однієї сутності мають однакові ключі рядків у всіх родинках стовпців. Така структура, в якій рядки будь-якого об'єкта в сімействі стовпців можуть динамічно змінюватися, визначає важливу перевагу цієї категорії сховищ. Сімейства стовпців дуже добре підходять для зберігання даних з різними схемами.

| CustomerID | Column Family: Identity                                      | CustomerID | Column Family: Contact Info                          |
|------------|--------------------------------------------------------------|------------|------------------------------------------------------|
| 001        | First name: Mu Bae<br>Last name: Min                         | 001        | Phone number: 555-0100<br>Email: someone@example.com |
| 002        | First name: Francisco<br>Last name: Vila Nova<br>Suffix: Jr. | 002        | Email: vilanova@contoso.com                          |
| 003        | First name: Lena<br>Last name: Adamczyk<br>Title: Dr.        | 003        | Phone number: 555-0120                               |

Рис. 13.4. Стовпцеві бази даних

На відміну від сховища пар "ключ - значення" і баз даних документів, більшість стовпчастих баз даних впорядковують збережені дані за допомогою самих значень ключів, а не хеш-кодів від них. Ключ рядка розглядається як первинний індекс і забезпечує доступ на основі певного ключа або їх діапазону. Деякі реалізації дозволяють створювати вторинні індекси за певними стовпцями в сімействі стовпців. Вторинні індекси дозволяють отримувати дані за значеннями стовпців, а не ключам рядки.

Всі стовпчики одного сімейства зберігаються на диску в одному файлі. Кожен файл містить певну кількість рядків. При використанні великих наборів даних цей підхід дозволяє підвищити продуктивність за рахунок зниження обсягу даних, які необхідно зчитувати з диска, коли відправляється запит на отримання кількох стовпців за раз.

Операції читання і запису для рядка зазвичай є атомарними в межах одного сімейства стовпців, хоча деякі реалізації забезпечують атомарність по всьому рядку, що охоплює кілька сімейств стовпців.

Прабатько цього типу – система Google BigTable. У таких системах дані зберігаються у вигляді розріджених матриць, рядки і стовпці якої використовуються як ключі. Типовим застосуванням цього типу СКБД є веб-індексування, а також завдання, пов'язані з великими даними, зі зниженими вимогами до узгодженості.

Прикладами СКБД даного типу є: Apache HBase, ApacheCassandra, ScyllaDB, ApacheAccumulo, Hypertable.

### **13.3. Основи MongoDB**

MongoDB реалізує новий підхід до побудови баз даних, де немає таблиць, схем, запитів SQL, зовнішніх ключів тощо, які наявні в об'єктно-реляційних базах даних.

На відміну від реляційних баз даних, MongoDB пропонує документо-орієнтовану модель даних, завдяки чому MongoDB працює швидше, володіє кращою масштабованістю, її краще використовувати.



Але, навіть враховуючи всі недоліки традиційних баз даних і переваги MongoDB, важливо розуміти, що задачі бувають різні і методи їх вирішення бувають також різні. В деяких ситуаціях MongoDB дійсно покращить продуктивність вашого застосунку, наприклад, якщо необхідно зберігати складні за структурою дані. В інших ситуаціях краще буде використання традиційних реляційних баз даних. Крім того, можна застосовувати змішаний підхід: зберігати один тип даних в MongoDB, а інший тип даних – в традиційних БД.

Вся система MongoDB може представляти не тільки одну базу даних, яка знаходиться на одному фізичному сервері. Функціональність MongoDB дозволяє розмістити декілька баз даних на декількох фізичних серверах, і ці бази даних зможуть легко обмінюватися даними і зберігати цілісність.

Одним з популярних стандартів обміну даними і їх зберігання є JSON (JavaScript Object Notation). JSON ефективно описує складні за структурою дані. Спосіб зберігання даних в MongoDB в цьому плані схожий на JSON, хоча формально JSON не використовується. Для зберігання в MongoDB застосовується формат, який називається BSON (BiCon) або скорочено від binary JSON.

BSON дозволяє працювати з даними швидше: швидше здійснюється пошук і обробка. Хоча варто зазначити, що BSON на відміну від зберігання даних в форматі JSON має невеликий недолік: загалом дані в JSON-форматі займають менше місця, чим у форматі BSON. З іншої сторони, даний недолік з надлишком перекривається швидкістю.

MongoDB написана на C++, тому її легко імпортувати на найрізноманітніші платформи. MongoDB може бути розгорнута на платформах Windows, Linux, MacOS, Solaris. Можна також завантажити початковий код і самому скопіювати MongoDB, але рекомендується використовувати бібліотеки з офіційного сайту.

### 13.3.1. Колекції

Якщо в традиційному світі SQL є таблиці, то в світі MongoDB є *колекції*. І якщо в реляційних БД в таблиці зберігаються однотипні жорстко структуровані об'єкти, то в колекції можуть міститися найрізноманітніші об'єкти, що мають різну структуру і різний набір властивостей.

Всю модель структури бази даних в MongoDB можна представити у вигляді *колекцій*, які, в свою чергу, складаються з документів (рис. 13.5).

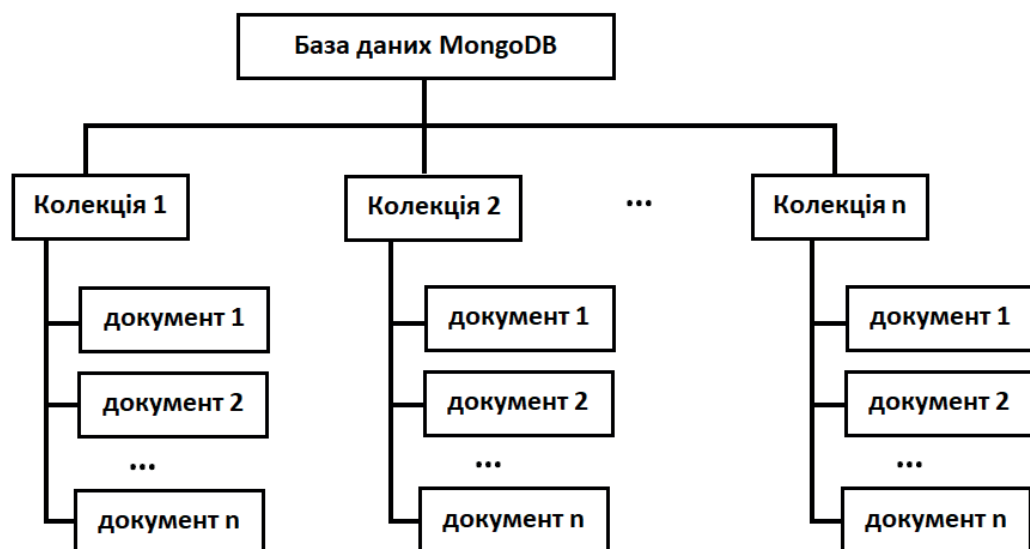


Рис. 13.5. Структура бази даних MongoDB

Кожна колекція має своє унікальне ім'я – довільний ідентифікатор, що складається з не більше ніж 128 різних алфавітно-цифрових символів і знаку підкреслення.

### 13.3.2. Документи замість рядків

Якщо реляційні бази даних зберігають рядки (записи), то MongoDB зберігає **документи**. На відміну від рядків, документи можуть зберігати складну за структурою інформацію. Документ можна представити як сховище ключів і значень.

**Ключ** є простою міткою, з яким асоційований певний діапазон даних.

**Приклад 13.1.** Типовий документ:

```
1  {
2    "name": "Bill",
3    "surname": "Gates",
4    "age": "48",
5    "company": {
6      "name" : "Microsoft",
```

```

7         "year" : "1974",
8         "price" : "300000"
9     }
10 }
```

Документ є набором пар ключ-значення. Наприклад, у виразі "name": "Bill" ключем виступає name, а значенням – Bill.

Однак при всіх відмінностях є одна особливість, яка зближує MongoDB і реляційні бази даних. У реляційних СКБД зустрічається таке поняття як **первинний ключ**. Це поняття описує якийсь стовпець, який має унікальні значення. У MongoDB для кожного документа є унікальний ідентифікатор, який називається `_id`. І якщо явно не вказати його значення, то MongoDB автоматично згенерує для нього значення.

Кожному ключу зіставляється певне значення. Але тут також треба враховувати одну особливість: якщо в реляційних базах є чітко окреслена структура, де є поля, і якщо якесь поле не має значення, йому (в залежності від налаштувань конкретної БД) можна привласнити значення NULL. У MongoDB все інакше. Якщо якомусь ключу не співставлено значення, то цей ключ просто опускається в документі і не вживається.

Ключі представляють рядки. Значення ж можуть відрізнятися за типом даних. В даному прикладі майже всі значення представляють рядковий тип, і лише один ключ (company) посилається на окремий об'єкт. В MongoDB є наступні типи значень:

- **String**: рядковий тип даних, використовується кодування UTF-8;
- **Array (масив)**: тип даних для зберігання масивів елементів;
- **Binary data (двійкові дані)**: тип для зберігання даних в бінарному форматі
- **Boolean**: булевий тип даних для зберігання логічних значень TRUE або FALSE, наприклад, {"married": FALSE};
- **Date**: зберігає дату в форматі часу Unix;
- **Double**: числовий тип даних для зберігання чисел з плаваючою точкою;
- **Integer**: використовується для зберігання цілочисельних значень, наприклад, {"age": 29}

- **JavaScript**: тип даних для зберігання коду javascript;
- **Min key/Max key**: використовується для порівняння значень з найменшим/найбільшим елементів BSON;
- **Null**: тип даних для зберігання значень Null;
- **Object**: рядковий тип даних, є складовим простих типів;
- **ObjectID**: тип даних для зберігання id документу;
- **Regular expression**: застосовується для зберігання регулярних виразів;
- **Symbol**: тип даних, ідентичний рядковому. Використовується переважно для тих мов, у яких є спеціальні символи;
- **Timestamp**: застосовується для зберігання часу.

**Приклад 13.2.** На відміну від рядків, документи можуть містити різномірну інформацію. Так, поруч з документом, наведеним вище, в одній колекції може знаходитися інший об'єкт.

```

1  {
2      "name": "Tom",
3      "birthday": "1985.06.28",
4      "place" : "Berlin",
5      "languages" :[
6          "english",
7          "german",
8          "spanish"
9      ]
10 }
```

Здавалося б різні об'єкти за винятком окремих властивостей, але всі вони можуть перебувати в одній колекції.

Ще пара важливих зауважень: в MongoDB запити володіють чутливістю до регістру і строгою типізацією. Тобто такі два документи не будуть ідентичні:

```

1  {"age" : "28"}
2  {"age" : 28}
```

Якщо в першому випадку для ключа age в якості значення визначений рядок, то в другому випадку – значенням є число.

Для кожного документа в MongoDB визначено унікальний ідентифікатор, який називається `_id`. При додаванні документа в колекцію даний ідентифікатор створюється автоматично. Однак розробник може сам явно задати ідентифікатор, вказавши відповідний ключ і його значення в документі.

Дане поле повинно мати унікальне значення в рамках колекції. І якщо ми спробуємо додати в колекцію два документа з однаковим ідентифікатором, то додасться лише один з них, а при додаванні другого ми отримаємо помилку.

Якщо ж ідентифікатор не заданий явно, то MongoDB створює спеціальне бінарне значення розміром 12 байт. Це значення складається з декількох сегментів: значення типу timestamp розміром 4 байта, ідентифікатор машини з 3 байт, ідентифікатор процесу з 2 байт і лічильник з 3 байт. Таким чином, перші 9 байт гарантують унікальність серед інших машин, на яких можуть бути репліки бази даних. А наступні 3 байта гарантують унікальність протягом однієї секунди для одного процесу. Така модель побудови ідентифікатора гарантує з високою часткою ймовірності, що він буде мати унікальне значення, адже вона дозволяє створювати до 16 777 216 унікальних об'єктів ObjectId в секунду для одного процесу.

### 13.3.3. Формат JSON

Формат JSON (JavaScript Object Notation) – формат передачі даних. Як зрозуміло з назви, JSON вийшов за межі JavaScript і доступний для використання в багатьох інших мовах програмування, включаючи Python, Ruby, PHP і Java.

Об'єкт JSON це формат даних типу ключ-значення, який зазвичай заключається у фігурні дужки.

**Приклад 13.3.** Ось так виглядає JSON об'єкт:

```
{
    "first_name" : "Sammy",
    "last_name"  : "Shark",
    "location"   : "Ocean",
    "online"     : true,
    "followers"  : 987
}
```

Цей простий приклад показує, що дані в форматі ключ-значення знаходяться між двома фігурними дужками, які виглядають так {}.

Пари ключ-значення розділені двокрапкою, як наприклад "key": "value". Ключі в JSON знаходяться з лівого боку від двокрапки і їх потрібно брати в лапки, як з "key" і це може бути будь-який рядок. У кожному об'єкті ключі повинні бути унікальними. Такі ключові рядки можуть містити пробіли, як в "first name", але такий підхід може ускладнити отримання доступу до них під час процесу розробки, так що кращим варіантом в таких випадках буде використання нижнього підкреслення: "first\_name".

Значення JSON знаходяться з правого боку від двокрапки і можуть бути одним з шести типів даних:

- рядок;
- число;
- об'єкт;
- масив;
- булеве значення;
- null.

На ширшому рівні, значення можуть також складатися їх складних типів даних, таких як JSON об'єкт або масив.

Хоч в .json файлах ми зазвичай бачимо формат декількох рядків, JSON також може бути написаний в одну суцільний рядок.

```
{"First_name": "Sammy", "last_name": "Shark", "online": true}
```

Такий підхід найбільш поширений в файлах інших форматів або при роботі з JSON рядком.

Робота з JSON в багаторядковому форматі часто робить його більш читабельним, особливо коли ви намагаєтеся впоратися з великим набором даних.

**Приклад 13.4.** Так як JSON ігнорує пробіли між елементами, щоб зробити дані більш читабельними, можна розділити їх тими ж самими пробілами:

```
{
```

```
    "first_name" : "Sammy",
    "last_name"  : "Shark",
    "online"     : true
}
```

Дуже важливо пам'ятати те, що хоч об'єкти JSON схожі візуально з об'єктами JavaScript, але об'єкти JSON мають інший формат і не можуть використовувати функції всередині об'єктів на відміну від об'єктів JavaScript.

Найважливішим властивістю формату JSON є те, що він може легко передаватися між мовами програмування в форматі, який розуміють практично всі з них. JavaScript об'єкти можуть працювати тільки безпосередньо через JavaScript.

JSON може містити вкладені об'єкти JSON. Такі об'єкти і масиви будуть також представлятися як пара "ключ-значення".

Розглянемо приклад вкладених об'єктів.

**Приклад 13.5.** Нехай у файлі `users.json` міститься інформація про чотирьох користувачів "Tom", "Bob", "Alice", "Bill". Для кожного користувача є вкладений об'єкт, який передає його значення, як, наприклад, "Tom" : "значення", але власне значення є також вкладеним об'єктом зі своїми парами "ключ-значення":

```
{
  "Tom" : {
    "username" : "SammyShark",
    "location" : "Indian Ocean",
    "online"   : true,
    "followers" : 987
  },
  "Bob" : {
    "username" : "JesseOctopus",
    "location" : "Pacific Ocean",
    "online"   : false,
    "followers" : 432
  },
}
```

```

"Alice" : {
    "username" : "DrewSquid",
    "location" : "Atlantic Ocean",
    "online"    : false,
    "followers" : 321
},
"Bill" : {
    "username" : "JamieMantisShrimp",
    "location" : "Pacific Ocean",
    "online"    : true,
    "followers" : 654
}
}

```

У наведеному вище прикладі фігурні дужки всюди використовуються для формування вкладеного JSON об'єкта з асоційованими іменами користувачів і даними локацій для кожного з них, а двокрапка використовується для поділу елементів у парі "ключ-значення".

Вкладеними даними також можуть бути масиви, які передаються як значення.

Для формування масивів JSON та JavaScript використовують квадратні дужки []. Масиви за своєю суттю – це впорядковані колекції і можуть включати в себе значення абсолютно різних типів даних.

**Приклад 13.6.** Використання простого масиву:

```

{
    "name" : "Alice",
    "age"  : 27,
    languages : ["german", "english"]
}

```

**Приклад 13.7.** Але дані в масиві можуть бути також складних структур:

```

{
    "first_name" : "Sammy",

```



```

"last_name" : "Shark",
"location" : "Ocean",
"websites" : [
    {
        "description" : "work",
        "URL" : "https://www.digitalocean.com/"
    },
    {
        "description" : "tutorials",
        "URL" :
            "https://www.digitalocean.com/community/tutorials"
    }
],
"social_media" : [
    {
        "description" : "twitter",
        "link" : "https://twitter.com/digitalocean"
    },
    {
        "description" : "facebook",
        "link" :
            "https://www.facebook.com/DigitalOceanCloudHosting"
    },
    {
        "description" : "github",
        "link" : https://github.com/digitalocean
    }
]
}

```

Ключі "websites" і "social\_media" використовують масив для вкладення інформації про сайти користувача і профайли в соціальних мережах. Ми знаємо, що це масиви - через квадратних дужок.

Використання вкладеності в форматі JSON дозволяє працювати з складними та ієрархічними даними.

У багатьох випадках JSON дуже схожий з XML, але простіший в написанні та інтерпретації. Крім того, на відміну від JSON, XML не може використовувати масиви.

JSON це нейтральний формат і він має безліч імплементацій для використання в багатьох мовах програмування.

JSON це легкий формат, який дозволяє вам легко ділитися, зберігати і працювати з даними. Як формат, JSON переживає зростаючу підтримку API, включаючи і Twitter API.

### **Контрольні запитання**

1. Що таке нереляційні бази даних?
2. Які є категорії NoSQL?
3. Які нереляційні БД надають можливість зберігання складних вкладених документів?
4. В якому форматі містяться дані в нереляційних документно-орієнтованих БД?
5. Які операції запитів підтримують сховища пар "ключ - значення"?
6. На які додатки розраховані сховища пар "ключ - значення"?
7. В яких сценаріях сховища пар "ключ - значення" є незручні?
8. Якими типами відомостей керують сховища даних графів?
9. Для яких типів задач застосовуються графові СКБД?
10. Які запити дозволяють ефективно виконувати додаткам сховища даних графів ?
11. Які типи нереляційних БД здатні денормалізовано структурувати розріджені дані?
12. Як представляються стовпчасті сховища даних?
13. За допомогою яких значень впорядковують збережені дані у стовпчастих БД?
14. Як зберігаються стовпчасті БД на диску?

15. Які задачі вирішуються за допомогою стовпчастих БД?
16. Який підхід реалізує MongoDB до побудови баз даних?
17. Чи дозволяє функціональність MongoDB розмістити декілька баз даних на декількох фізичних серверах?
18. Наведіть модель структури бази даних в MongoDB. Що таке колекції в MongoDB?
19. Що містять колекції в MongoDB?
20. Які типи значень є в MongoDB?
21. Що таке JSON?
22. Чи може JSON містити вкладені об'єкти JSON?
23. Як представляються масиви в JSON?

## Тема 14. Консоль MongoDB

### 14.1. Основи роботи з MongoDB

#### 14.1.1. Початок роботи з MongoDB

Встановлення MongoDB здійснюється з офіційного сайту: <https://www.mongodb.com/download-center/community>.

Офіційний сайт надає пакети дистрибутивів для різних платформ: Windows, Linux, MacOS, Solaris. І кожної платформи є кілька дистрибутивів.

Після інсталяції MongoDB перш за все необхідно створити на жорсткому диску каталог C:\data\db, так як MongoDB за замовчуванням зберігає бази даних за цим шляхом і без цього каталогу сервер MongoDB не стартує. В ОС Linux і MacOS каталогом за замовчуванням буде / data / db.

Якщо ж виникла необхідність використовувати якийсь інший шлях до файлів, то його можна передати при запуску MongoDB у прапорці – dbpath.

Отже, після створення каталогу для зберігання БД, можна запустити сервер MongoDB. Сервер є додатком mongod, який знаходиться за шляхом C:\Program Files\MongoDB\Server\XX.X\bin.

Для управління даними, тобто для роботи з БД, необхідно також запустити консольну оболонку для взаємодії з сервером. Ця оболонка є файлом mongo.exe, який також знаходиться в папці bin.

Запустивши оболонку, можна приступати до роботи з даними (рис. 14.1).

Розпочинаючи роботу з MongoDB, перш за все, необхідно встановити необхідну базу даних в якості поточної, щоб в подальшому її використовувати. Для цього використовується команда:

**use <ім'я БД> (аналогічна команді SQL).**

При цьому не важливо, чи існує вже така БД. Якщо такої БД немає, MongoDB створить її автоматично під час додавання в неї даних.

Для прикладу, введемо наступну команду:

```
> use info
switched to db info
>
```

Теперь поточною буде БД info.

```

C:\Program Files\MongoDB\Server\4.0\bin\mongod.exe
2020-11-19T10:31:59.818+0200 I CONTROL [main] Automatically disabling TLS 1.0,
to force-enable TLS 1.0 specify --sslDisabledProtocols 'none'
2020-11-19T10:31:59.818+0200 I CONTROL [initandlisten] ** server with
--bind_ip 127.0.0.1 to disable this warning.
2020-11-19T10:32:00.880+0200 I CONTROL [initandlisten]
2020-11-19T10:32:00.881+0200 I CONTROL [initandlisten] Hotfix KB2731284 or late
r update is not installed, will zero-out data files.
2020-11-19T10:32:00.882+0200 I CONTROL [initandlisten]
---
Enable MongoDB's free cloud-based monitoring service, which will then receive an
d display
metrics about your deployment (disk utilization, CPU, operation statistics, etc)
.
The monitoring data will be available on a MongoDB website with a unique URL acc
essible to you
and anyone you share the URL with. MongoDB may use this information to make prod
uct
improvements and to suggest MongoDB products and deployment options to you.
To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeM
onitoring()

```

Рис. 14.1. Консольна оболонка MongoDB

Щоб вивести назви всіх БД на консоль, використовується команда

```
> show dbs
admin    0.000GB
config   0.000GB
local    0.000GB
test     0.000GB
```

БД info в даному списку немає, оскільки в неї ще не додано дані.

Визначити ім'я поточної БД можна командою db:

```
> db
info
>
```

Крім баз даних можна переглянути список всіх колекцій в поточній БД за допомогою команди:

```
> show collections
```

Оскільки БД info, яка задана поточною, ще не існує, то і колекцій вона поки не містить.

Щоб видалити всю поточну базу даних, треба скористатися функцією `dropDatabase()`:

```
> db.dropDatabase()
```

#### 14.1.2. Типи даних

В MongoDB використовуються наступні типи значень:

- **String:** рядковий тип даних (використовується кодування UTF-8);
- **Array (масив):** тип даних для зберігання елементів масивів;
- **Binary data (двійкові дані):** тип для зберігання даних в бінарному форматі;
- **Boolean:** булевий тип даних, зберігає логічні значення TRUE або FALSE, наприклад, {"married": FALSE};
- **Date:** зберігає дату в форматі часу Unix;
- **Double:** числовий тип даних для зберігання чисел з плаваючою точкою;
- **Integer:** використовується для зберігання цілочисельних значень, наприклад, {"age": 29};
- **JavaScript:** тип даних для зберігання коду javascript;
- **Min key/Max key:** використовується для порівняння значень з найменшим/найбільшим елементів BSON;
- **Null:** тип даних для зберігання значення Null;
- **Object:** рядковий тип даних;
- **ObjectId:** тип даних для зберігання id документа;
- **Regular expression:** застосовується для зберігання регулярних виразів;
- **Symbol:** тип даних, ідентичний рядковому. Використовується переважно для тих мов, в яких є спеціальні символи;
- **Timestamp:** застосовується для зберігання часу.

### 14.1.3. Колекції і документи

Якщо в реляційних БД вмістом є таблиці, то в MongoDB база даних складається з *колекцій*.

Кожна колекція має своє унікальне ім'я – довільний ідентифікатор, що складається з не більше ніж 128 різних алфавітно-цифрових символів і знаку підкреслення.

На відміну від реляційних баз даних, MongoDB не використовує табличну структуру з чітко заданою кількістю стовпців і типів даних. MongoDB є документо-орієнтованою системою, в якій центральним поняттям є *документ*.

Якщо в команді додавання документа в БД вказана колекція відсутня, MongoDB створить її автоматично.

Документ можна уявити як об'єкт, який зберігає деяку інформацію. У певному сенсі він схожий на рядки в реляційних СКБД, де рядки зберігають інформацію про окремий елемент.

**Приклад 14.1** типового документа:

```
{  
  "name": "Bill",  
  "surname": "Gates",  
  "age": "48"  
}
```

Документ є набором пар ключ-значення. Наприклад, у виразі "name": "Bill" name є ключ, а Bill – значення; age – ключ, а 48 – значення рядкового типу, адже є в лапках.

Ключі є рядками (тип – рядок). Значення же можуть відрізнятися за типами даних.

На відміну від рядків, документи можуть містити різноманітну інформацію.

**Приклад 14.2.** Так, поряд з документом, наведеним у прикладі 14.1, в одній колекції може знаходитися інший об'єкт.

```
{  
  "name": "Tom",  
  "birthday": "1985.06.28",
```

```
"place" : "Vashington",  
"languages" : [  
  "english",  
  "german",  
  "spanish"  
]  
}
```

Здається, що різні об'єкти (за винятком окремих властивостей), але всі вони можуть знаходитися в одній колекції.

Запити в MongoDB володіють регістрозалежністю і строгою типізацією. Тобто наступні два документи не будуть ідентичними:

```
{"age" : "28"}  
{"age" : 28}
```

Якщо в першому випадку для ключа age визначено у якості значення рядок, то в другому випадку значенням є число.

Для кожного документу в MongoDB визначений унікальний ідентифікатор, який називається `_id`. При додаванні документу в колекцію даний ідентифікатор створюється автоматично. Однак розробник може сам явним чином задати ідентифікатор, а не покладатися на автоматично згенерований, указавши відповідний ключ і його значення в документі.

Дане поле повинно мати унікальне значення в рамках колекції. При спробі додати в колекцію два документа з однаковим ідентифікатором, додасться лише один з них, а при додаванні другого отримаємо помилку.

Якщо ідентифікатор не заданий явно, то MongoDB створює спеціальне бінарне значення розміром 12 байт. Це значення складається з декількох сегментів: значення типу timestamp розміром 4 байта, ідентифікатор машини з 3 байт, ідентифікатор процесу з 2 байт і лічильник з 3 байт. Таким чином, перші 9 байт гарантують унікальність серед інших машин, на яких можуть бути репліки бази даних. А наступні 3 байти гарантують унікальність протягом однієї секунди для одного процесу. Така модель побудови ідентифікатора гарантує з високою часткою ймовірності, що він буде мати унікальне значення, адже вона дозволяє



створювати до 16 777 216 унікальних об'єктів ObjectId в секунду для одного процесу.

## 14.2. Додавання даних

Встановивши поточну БД,

**use <ім'я БД>**,

можна додавати в неї дані. Всі дані в БД зберігаються у форматі BSON, і, відповідно, необхідно вводити дані в цьому форматі. Якщо в БД немає жодної колекції, при додаванні даних вона автоматично створиться.

### 14.2.1. Додавання даних командами

Для додавання даних в колекцію використовуються методи:

- **insertOne()**: додає один документ;
- **insertMany()**: додає декілька документів;
- **insert()**: може додавати як один, так і декілька документів;

#### *1) Додавання в БД одного документа.*

Формат запису:

**db.<ім'я\_колекції>.insertOne({ключ : значення, ключ : значення ...})**

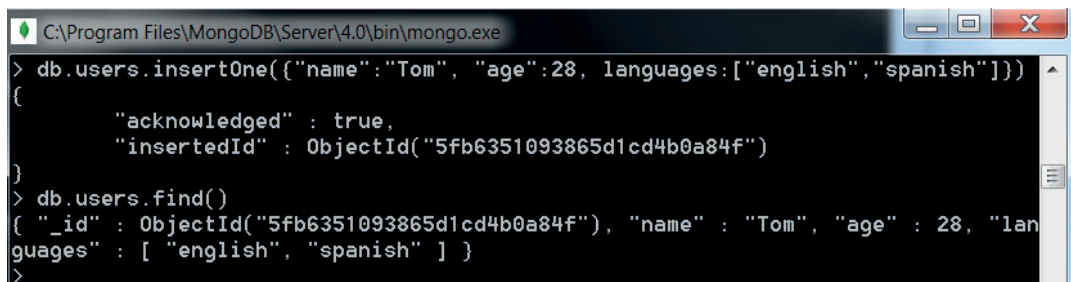
де db – поточна БД.

**Приклад 14.3.** Додавання в колекцію users одного документа:

```
> db.users.insertOne({ "name": "Tom", "age": 28,
  languages: ["english", "spanish"] })
```

У разі вдалого додавання документу у колекцію БД на консоль буде виведено згенерований автоматично ідентифікатор доданого документа.

І щоб переконатися, що документ додано в БД, виводимо його функцією find (рис. 14.2).



```
C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe
> db.users.insertOne({"name":"Tom", "age":28, "languages":["english","spanish"]})
{
  "acknowledged" : true,
  "insertedId" : ObjectId("5fb6351093865d1cd4b0a84f")
}
> db.users.find()
{ "_id" : ObjectId("5fb6351093865d1cd4b0a84f"), "name" : "Tom", "age" : 28, "languages" : [ "english", "spanish" ] }
>
```

Рис. 14.2. Додавання одного документа та відображення його в колекції

Документ є набором пар ключ-значення. В даному випадку документ, що додається, має три ключі: name, age, languages, і кожному з них зіставляє певне значення. Наприклад, ключу languages в якості значення зіставляється масив.

Деякі обмеження при використанні імен ключів:

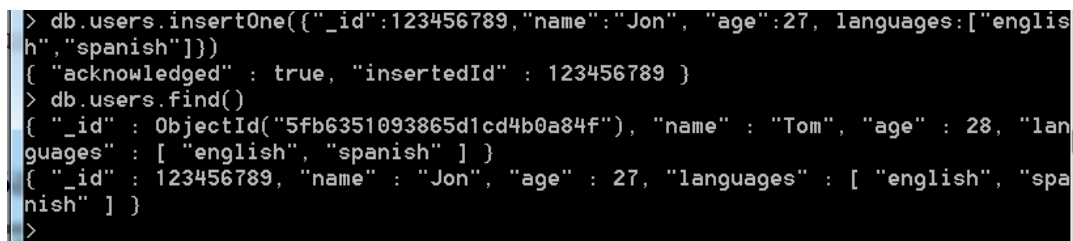
- Символ \$ не може бути першим символом в імені ключа
- Ім'я ключа не може містити символ точки (.).

При додаванні даних, якщо явно не задано значення для поля "\_id" (тобто унікального ідентифікатора документа), тоді воно генерується автоматично. Але в принципі при додаванні даних можна самостійно встановити цей ідентифікатор.

**Приклад 14.4.** Додавання одного документа з задаванням значення для унікального ідентифікатора документа "\_id":

```
> db.users.insertOne({"_id": 123456789, "name": "Jon",
"age": 27, "languages": ["english", "spanish"]})
```

Результат виконання цієї команди наведено на рис. 14.3.



```
> db.users.insertOne({"_id":123456789,"name":"Jon", "age":27, "languages":["english","spanish"]})
{ "acknowledged" : true, "insertedId" : 123456789 }
> db.users.find()
{ "_id" : ObjectId("5fb6351093865d1cd4b0a84f"), "name" : "Tom", "age" : 28, "languages" : [ "english", "spanish" ] }
{ "_id" : 123456789, "name" : "Jon", "age" : 27, "languages" : [ "english", "spanish" ] }
>
```

Рис. 14.3. Додавання одного документа з задаванням значення для унікального ідентифікатора документа "\_id" та відображення документів колекції

Варто зазначити, що назви ключів можуть використовуватися в лапках, а можуть і без лапок.

## 2) Додавання в БД одночасно декілька документів – метод insertMany().

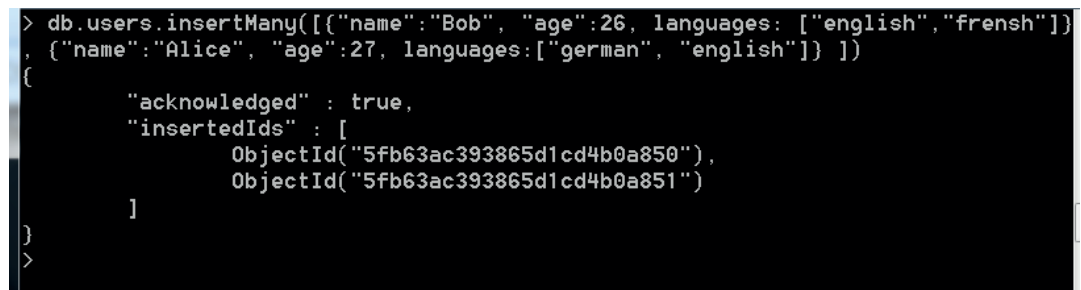
Формат запису:

```
db.<ім'я_колекції>.insertMany({ключ : значення, ключ : значення ...},  
                               {ключ : значення, ключ : значення ...},  
...)
```

*Приклад 14.5.* Додавання в БД одночасно декілька документів:

```
> db.users.insertMany([{"name": "Bob", "age": 26,  
  languages: ["english", "frensh"]}, {"name": "Alice",  
  "age": 27, languages:["german", "english"]}])
```

Після додавання консоль виводить ідентифікатори доданих документів (рис. 14.4):



```
> db.users.insertMany([{"name": "Bob", "age": 26, languages: ["english", "frensh"]},  
  {"name": "Alice", "age": 27, languages: ["german", "english"]}])  
{  
  "acknowledged" : true,  
  "insertedIds" : [  
    ObjectId("5fb63ac393865d1cd4b0a850"),  
    ObjectId("5fb63ac393865d1cd4b0a851")  
  ]  
}
```

Рис. 14.4. Додавання двох документів у колекцію users

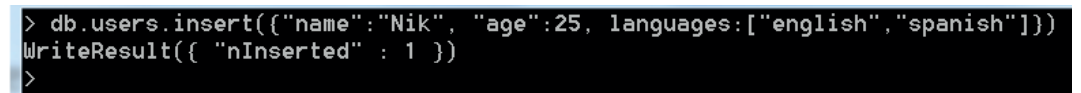
## 3) Універсальний спосіб додавання документів – метод insert().

Метод insert() – це об'єднання можливостей методів insertOne та insertMany.

*Приклад 14.6.* Додавання документу в колекцію users методом insert():

```
> db.users.insert({"name": "Nik", "age": 25, languages:  
  ["english", "spanish"]})
```

Після його виклику на консоль виводиться кількість доданих записів (рис. 14.5):



```
> db.users.insert({"name": "Nik", "age": 25, languages: ["english", "spanish"]})  
WriteResult({ "nInserted" : 1 })  
>
```

Рис. 14.5. Додавання документа методом insert()

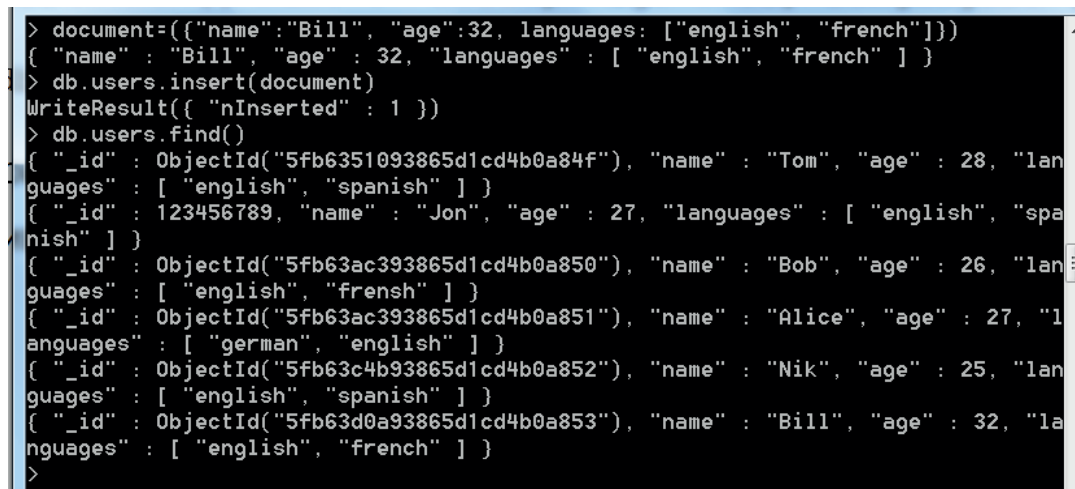
## 4) Додавання наперед визначеного документа.

Є ще один спосіб додавання в БД документа, який включає два етапи: визначення документа (`document = ({...})`) і власне його додавання.

**Приклад 14.7.** Додавання наперед визначеного документа:

```
> document = ({ "name": "Bill", "age": 32, languages:
  ["english", "french"] })
> db.users.insert(document)
```

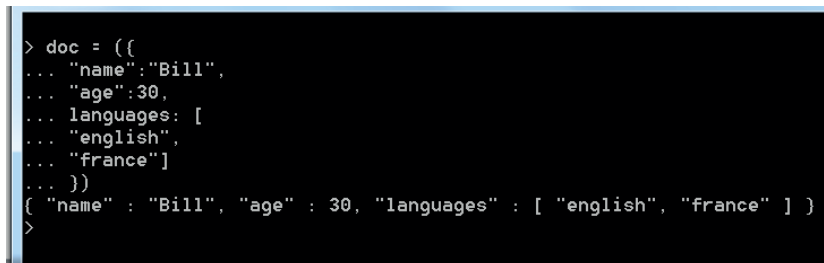
При бажанні знову ж можна за допомогою функції `db.users.find()` переконатися, що документ потрапив в БД (рис. 14.6).



```
> document=({ "name": "Bill", "age": 32, languages: ["english", "french"] })
{ "name": "Bill", "age": 32, "languages": [ "english", "french" ] }
> db.users.insert(document)
WriteResult({ "nInserted" : 1 })
> db.users.find()
{ "_id" : ObjectId("5fb6351093865d1cd4b0a84f"), "name" : "Tom", "age" : 28, "languages" : [ "english", "spanish" ] }
{ "_id" : 123456789, "name" : "Jon", "age" : 27, "languages" : [ "english", "spanish" ] }
{ "_id" : ObjectId("5fb63ac393865d1cd4b0a850"), "name" : "Bob", "age" : 26, "languages" : [ "english", "french" ] }
{ "_id" : ObjectId("5fb63ac393865d1cd4b0a851"), "name" : "Alice", "age" : 27, "languages" : [ "german", "english" ] }
{ "_id" : ObjectId("5fb63c4b93865d1cd4b0a852"), "name" : "Nik", "age" : 25, "languages" : [ "english", "spanish" ] }
{ "_id" : ObjectId("5fb63d0a93865d1cd4b0a853"), "name" : "Bill", "age" : 32, "languages" : [ "english", "french" ] }
>
```

*Рис. 14.6. Додавання наперед визначеного документа*

Інтерпретатор MongoDB дозволяє також вводити і багаторядкові команди. Якщо вираз не закінчено (з точки зору мови JavaScript), і натиснуто Enter, то введення наступної частини виразу автоматично переноситься на наступний рядок (рис. 14.7).

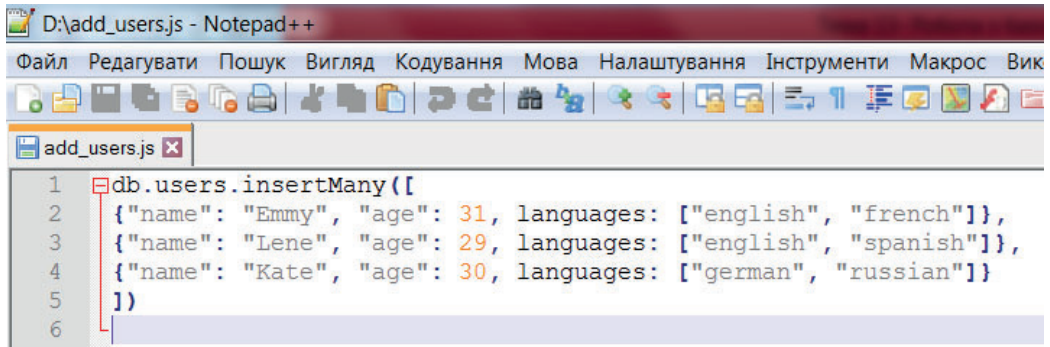


```
> doc = ({
...   "name": "Bill",
...   "age": 30,
...   languages: [
...     "english",
...     "france"
...   ]
... })
{ "name" : "Bill", "age" : 30, "languages" : [ "english", "france" ] }
>
```

*Рис. 14.7. Додавання наперед визначеного документа з форматуванням*

### 14.2.2. Завантаження даних з файлу

Дані для бази даних MongoDB можна визначати в звичайному текстовому файлі, що досить зручно, оскільки ми можемо переносити або пересилати цей файл незалежно від бази даних MongoDB. Наприклад, визначимо на диску D: файл `add_users.js` наступного змісту (рис. 14.8):



```
1 db.users.insertMany([
2   {"name": "Emmy", "age": 31, languages: ["english", "french"]},
3   {"name": "Lene", "age": 29, languages: ["english", "spanish"]},
4   {"name": "Kate", "age": 30, languages: ["german", "russian"]}
5 ])
6
```

Рис. 14.8. Файл типу JSON для додавання даних в поточну БД MongoDB в колекцію `users`

У файлі за допомогою методу `insertMany` записано команду додавання в колекцію `users` поточної БД трьох документів.

Для завантаження файлу в поточну БД застосовується функція `load()`,

в яку як параметр передається шлях до файлу.

**Приклад 14.8.** Додавання збереженого документа у файлі `add_users.js`:

```
> load("D:/add_users.js")
```

У разі успішного виконання команди `load()` MongoDB поверне повідомлення `true` (рис. 14.9).

```

> load("D:/add_users.js")
true
> db.users.find()
{ "_id" : ObjectId("5fc0c857fac1f17606f135ff"), "name" : "Tom", "age" : 28, "languages" : [ "english", "spanish" ] }
{ "_id" : 12345789, "name" : "Jon", "age" : 27, "languages" : [ "english", "spanish" ] }
{ "_id" : ObjectId("5fc0c857fac1f17606f13600"), "name" : "Bob", "age" : 26, "languages" : [ "english", "frensh" ] }
{ "_id" : ObjectId("5fc0c857fac1f17606f13601"), "name" : "Alice", "age" : 27, "languages" : [ "german", "english" ] }
{ "_id" : ObjectId("5fc0c857fac1f17606f13602"), "name" : "Nik", "age" : 25, "languages" : [ "english", "spanish" ] }
{ "_id" : ObjectId("5fc0c857fac1f17606f13603"), "name" : "Bill", "age" : 32, "languages" : [ "english", "french" ] }
{ "_id" : ObjectId("5fc0c857fac1f17606f13604"), "name" : "Emmy", "age" : 31, "languages" : [ "english", "french" ] }
{ "_id" : ObjectId("5fc0c857fac1f17606f13605"), "name" : "Lene", "age" : 29, "languages" : [ "english", "spanish" ] }
{ "_id" : ObjectId("5fc0c857fac1f17606f13606"), "name" : "Kate", "age" : 30, "languages" : [ "german", "russian" ] }
>

```

Рис. 14.9. Додавання збереженого документа у файлі *add\_users.js*

## 14.3. Вибірка з БД

### 14.3.1. Функція *find*

Найбільш простий спосіб отримання вмісту БД полягає у використанні функції *find*. Дія цієї функції багато в чому аналогічна звичайному запиту *SELECT \* FROM Table*, який витягує всі рядки.

Формат запису функції *find*:

***db.ім'я\_колекції.find()***,

де *db* – поточна БД.

Окремо для отримання першого документа колекції застосовується функція:

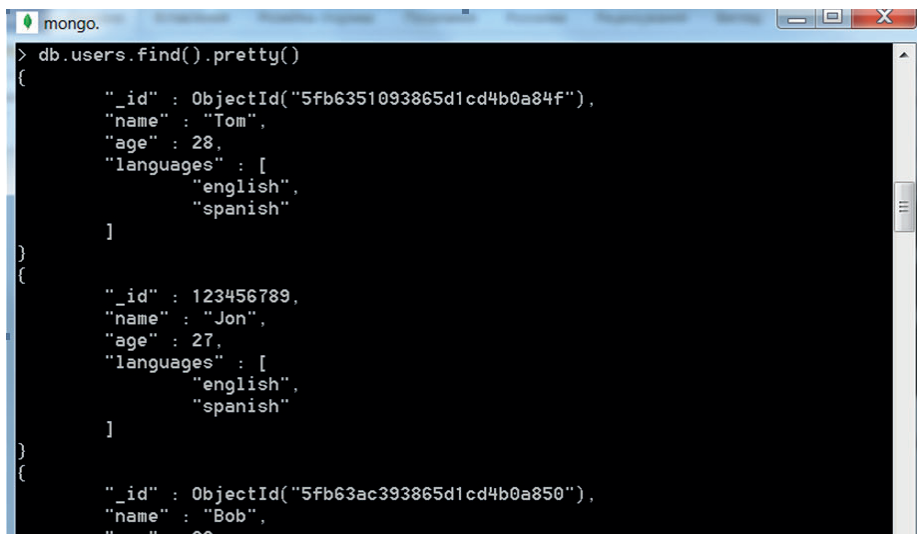
***db.ім'я\_колекції.findOne()***.

Наприклад, щоб витягти всі документи з колекції *users* бази даних *info*, використовується команда *db.users.find()*. І з таким форматом функції ми вже знайомі з попереднього матеріалу.

Для виведення документів в більш зручному наочному поданні ми можемо додати виклик методу *pretty()*.

**Приклад 14.9.** Виведення документів в більш зручному наочному поданні (рис. 14.10):

```
> db.users.find().pretty()
```



```
mongo.  
> db.users.find().pretty()  
{  
  "_id" : ObjectId("5fb6351093865d1cd4b0a84f"),  
  "name" : "Tom",  
  "age" : 28,  
  "languages" : [  
    "english",  
    "spanish"  
  ]  
}  
{  
  "_id" : 123456789,  
  "name" : "Jon",  
  "age" : 27,  
  "languages" : [  
    "english",  
    "spanish"  
  ]  
}  
{  
  "_id" : ObjectId("5fb63ac393865d1cd4b0a850"),  
  "name" : "Bob",  
  "age" : 26,  
  "languages" : [  
    "english",  
    "spanish"  
  ]  
}
```

Рис. 14.10. Виведення документів методом *pretty()*

Якщо ж треба отримати не всі документи, а тільки ті, які задовольняють певним обмеженням (аналог WHERE у SELECT), тоді ці обмеження записуються у дужках:

```
db.ім'я_колекції.find(обмеження),
```

а формат запису обмеження – JSON.

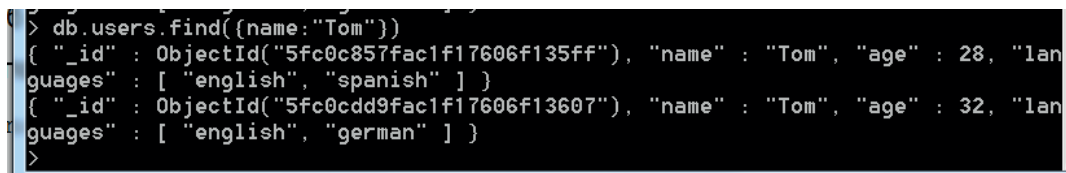
**Приклад 14.10.** Додамо у поточну базу у колекцію *users* ще одного Тома :

```
db.users.insertOne({"name": "Tom", "age": 32,  
languages: ["english", "german"]})
```

та виведемо всі документи, в яких *name* = Tom. Обмеження у форматі JSON виглядатиме: `{"name": "Tom"}`, хоча лапки для ключа можна опускати:

```
> db.users.find({name: "Tom"})
```

Такий запит виведе нам два документи, в яких *name* = Tom (рис. 14.11).



```
> db.users.find({name: "Tom"})  
{ "_id" : ObjectId("5fc0c857fac1f17606f135ff"), "name" : "Tom", "age" : 28, "languages" : [ "english", "spanish" ] }  
{ "_id" : ObjectId("5fc0cdd9fac1f17606f13607"), "name" : "Tom", "age" : 32, "languages" : [ "english", "german" ] }  
>
```

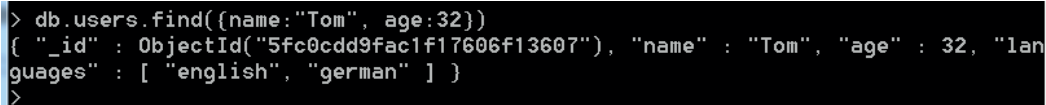
Рис. 14.11. Виведення документів, в яких *name* = Tom

**Приклад 14.11.** Тепер розглянемо складніший запит: необхідно вивести ті об'єкти, у яких `name = Tom` і `age = 32`. Тобто на мові SQL це могло б виглядати так:

```
SELECT * FROM Table WHERE name = 'Tom' AND age = 32,
```

а в MongoDB (рис. 14.12):

```
> db.users.find({name: "Tom", age: 32})
```

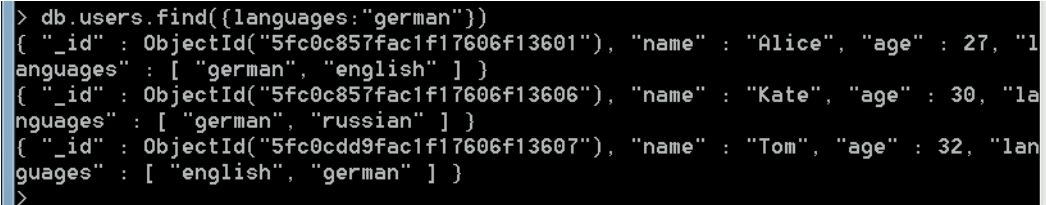


```
> db.users.find({name:"Tom", age:32})
{ "_id" : ObjectId("5fc0cdd9fac1f17606f13607"), "name" : "Tom", "age" : 32, "languages" : [ "english", "german" ] }
>
```

Рис. 14.12. Виведення документів, в яких `name = Tom` та `age = 32`

**Приклад 14.12.** Якщо ж необхідно здійснити пошук за елементом у масиві, тоді вказується назва масиву і можливі значення у масиві, наприклад `{languages: "german"}` (рис. 14.13):

```
> db.users.find({languages: "german"})
```



```
> db.users.find({languages:"german"})
{ "_id" : ObjectId("5fc0c857fac1f17606f13601"), "name" : "Alice", "age" : 27, "languages" : [ "german", "english" ] }
{ "_id" : ObjectId("5fc0c857fac1f17606f13606"), "name" : "Kate", "age" : 30, "languages" : [ "german", "russian" ] }
{ "_id" : ObjectId("5fc0cdd9fac1f17606f13607"), "name" : "Tom", "age" : 32, "languages" : [ "english", "german" ] }
>
```

Рис. 14.13. Виведення документів, в яких `languages: "german"`

Такий запит виведе всі документи, у яких в масиві `languages` є значення `"german"`.

**Приклад 14.13.** Запит типу:

```
> db.users.find({languages: ["german"]})
```

виведе тільки ті документи, у яких масив `languages` складається лише з одного елемента, в даному випадку `["german"]`. Якщо ж у списку більше 1 елемента, наприклад 2 мови, тоді цей запит з обмеженням тільки на одну мову такий список розглядати не буде. І для наведеної БД результат виконання такого запиту нічого не виведе, так як в кожному документі даний масив `languages` має по два значення (рис. 14.14).



```
> db.users.find({languages:["german"]})  
>
```

*Рис. 14.14. Виведення документів, в яких масив languages складається лише з одного елемента "german"*

**Приклад 14.14.** Для того, щоб отримати документи, у яких в масиві languages одночасно містить тільки дві мови, наприклад, "english" і "german", масив записується в квадратних дужках (рис. 14.15):

```
> db.users.find({languages: ["english", "german"]})
```

```
> db.users.find({languages:["english","german"]})  
{ "_id" : ObjectId("5fc0cdd9fac1f17606f13607"), "name" : "Tom", "age" : 32, "languages" : [ "english", "german" ] }  
>
```

*Рис. 14.15. Виведення документів, в яких масив languages одночасно містить тільки два елементи "english" і "german"*

**Приклад 14.15.** Результат залежить від вказаної послідовності елементів масиву і запит:

```
> db.users.find({languages: ["german", "english"]})
```

де елементи "english" і "german" змінено місцями, дасть інший результат (рис. 14.16):

```
> db.users.find({languages:["english","german"]})  
{ "_id" : ObjectId("5fc0cdd9fac1f17606f13607"), "name" : "Tom", "age" : 32, "languages" : [ "english", "german" ] }  
> db.users.find({languages:["german", "english"]})  
{ "_id" : ObjectId("5fc0c857fac1f17606f13601"), "name" : "Alice", "age" : 27, "languages" : [ "german", "english" ] }  
>
```

*Рис. 14.16. Виведення документів, в яких масив languages одночасно містить тільки два елементи "german" та "english" (поміняні місцями)*

**Приклад 14.16.** Тепер виведемо всі документи, в яких "german" в масиві languages знаходиться на першому місці (рис. 14.17):

```
> db.users.find({"languages.0": "german"})
```

```
> db.users.find({"languages.0":"german"})
{ "_id" : ObjectId("5fc0c857fac1f17606f13601"), "name" : "Alice", "age" : 27, "languages" : [ "german", "english" ] }
{ "_id" : ObjectId("5fc0c857fac1f17606f13606"), "name" : "Kate", "age" : 30, "languages" : [ "german", "russian" ] }
>
```

Рис. 14.17. Виведення документів, в яких "german" в масиві languages знаходиться на першому місці

**Приклад 14.17.** Відповідно, якщо необхідно вивести документи, де english на другому місці (наприклад, ["german", "english"]), тоді замість нуля ставимо одиницю: languages.1, оскільки нумерація елементів масивів починається з 0 (рис. 14.18).

```
> db.users.find({"languages.1":"german"})
{ "_id" : ObjectId("5fc0cdd9fac1f17606f13607"), "name" : "Tom", "age" : 32, "languages" : [ "english", "german" ] }
>
```

Рис. 14.18. Виведення документів, в яких у масиві languages значення "english" знаходиться на другому місці

### 14.3.2. Проекція

Документ може мати багато полів, але, зазвичай, не всі ці поля потрібні і важливі при запиті. Тоді, щоб включити у вибірку тільки потрібні поля, використовується *проекція*.

Формат запису:

**db.ім'я\_колекції.find({обмеження}, {ім'я поля: 1 або 0}).**

Аналог в SQL – SELECT <перелік полів> FROM...

Значення 1 вказує на те, що поле в результаті запиту буде відображено і 0 – навпаки, будуть відображені всі інші поля.

Поле \_id завжди буде виводитися. Щоб його виключити з запиту, необхідно задати його явно:

```
{ "_id": 0 }.
```

Якщо ж обмеження у запиті не потрібно, тобто необхідно вивести всі документи колекції, тоді фігурні дужки залишаються порожніми: {}.

Альтернативою замість 1 і 0 можна використовувати true або false.

**Приклад 14.18.** Вивести дані тільки для полів "age" колекції users, в яких name = Tom (рис. 14.19):

```
> db.users.find({name: "Tom"}, {age: 1})
```

```
> db.users.find({name:"Tom"}, {age:1})
{ "_id" : ObjectId("5fc0c857fac1f17606f135ff"), "age" : 28 }
{ "_id" : ObjectId("5fc0cdd9fac1f17606f13607"), "age" : 32 }
>
```

Рис. 14.19. Відображення для колекції users тільки полів, в яких name = Tom

**Приклад 14.19.** Задамо явно, щоб \_id не виводити (рис. 14.20):

```
> db.users.find({name:"Tom"}, {_id:0, age:1})
{ "age" : 28 }
{ "age" : 32 }
>
```

Рис. 14.20. Відображення тільки полів "age", в яких name = Tom без \_id

**Приклад 14.20.** І зворотна ситуація: необхідно вивести всі поля документів для name = Tom окрім властивості age. У цьому випадку в якості параметра вказується 0 (рис. 14.21):

```
> db.persons.find({name: "Tom"}, {age: 0})
```

```
> db.users.find({name:"Tom"}, {age:0})
{ "_id" : ObjectId("5fc0c857fac1f17606f135ff"), "name" : "Tom", "languages" : [
"english", "spanish" ] }
{ "_id" : ObjectId("5fc0cdd9fac1f17606f13607"), "name" : "Tom", "languages" : [
"english", "german" ] }
>
```

Рис. 14.21. Відображення всіх полів документів для name = Tom окрім поля age

**Приклад 14.21.** Тепер вивести всі поля документів для name = Tom, за винятком \_id та age (рис. 14.22):

```
> db.users.find({name:"Tom"}, {_id:0, age:0})
{ "name" : "Tom", "languages" : [ "english", "spanish" ] }
{ "name" : "Tom", "languages" : [ "english", "german" ] }
>
```

Рис. 14.22. Відображення всіх полів документів для name = Tom окрім полів \_id та age

**Приклад 14.22.** Альтернатива замість 1 і 0 – true або false (рис. 14.23):

```
> db.users.find({name:"Tom"}, {_id:false, age:true})
{ "age" : 28 }
{ "age" : 32 }
>
```

*Рис. 14.23. Відображення всіх полів документів для name = Tom окрім полів \_id та age, де замість 0/1 вказано true/false*

**Приклад 14.23.** Якщо ж не потрібно конкретизувати вибірку, а вивести всі документи колекції, то перші фігурні дужки проєкції залишаються порожніми (рис. 14.24):

```
> db.users.find({}, {_id: false, age: true })
```

```
> db.users.find({}, {_id:false, age:true})
{ "age" : 28 }
{ "age" : 27 }
{ "age" : 26 }
{ "age" : 27 }
{ "age" : 25 }
{ "age" : 32 }
{ "age" : 31 }
{ "age" : 29 }
{ "age" : 30 }
{ "age" : 32 }
>
```

*Рис. 14.24. Відображення поля age (без поля \_id ) у всіх документах колекції users*

### 14.3.3. Запит до вкладених об'єктів

Попередні запити застосовувалися до простих об'єктів. Але документи можуть бути дуже складними за структурою.

Нехай в колекції `persons` буде наступний документ:

```
> db.persons.insert({"name": "Alex", "age": 28,
company: {"name":"microsoft", "country":"USA"}})
```

Тепер, щоб звернутися до вкладеного (внутрішнього) ключа та його значення, необхідно спочатку вказати зовнішній ключ через крапку:

**<зовнішній\_ключ>.<внутрішній\_ключ>**

**Приклад 14.24.** Щоб знайти всі документи, у яких в ключі `company` вкладено властивість `name = microsoft`, необхідно використовувати оператор крапки (рис.14.25):

```
> db.users.find({"company.name": "microsoft"})
```

```
> db.persons.find({"company.name":"microsoft"})
{ "_id" : ObjectId("5fc21098b9aa1addedf722e6"), "name" : "Alex", "age" : 28, "company" : { "name" : "microsoft", "country" : "USA" } }
>
```

Рис. 14.25. Відображення значень вкладених документів

#### 14.3.4. Використання регулярних виразів

**Регулярний вираз** часто називають **шаблоном пошуку**. Синтаксис регулярних виразів залежить від інтерпретатора, що використовується для їхньої обробки. Однак, із незначними відхиленнями, майже всі поширені інтерпретатори регулярних виразів мають спільні правила.

Найпростішим регулярним виразом, з якого формуються складні, є звичайний символ. Більшість символів у регулярному виразі представляють самі себе, за винятком спеціальних символів (метасимволів) `[ ] \ ^ $ . | ? * + ( ) { }`, яким може передувати символ `\` (зворотна коса риска), котрий робить метасимволи «екранованими», «захищеними» для представлення, відображення їх самих як символів тексту.

Аналогічно можуть бути представлені інші спеціальні символи: знак табуляції (`\t`), повернення каретки (`\r`), новий рядок (`\n`), кінець сторінки (`\f`) тощо. Частина символів, які в тій або іншій реалізації не вимагають екранування (наприклад, кутові дужки `< >`), можуть бути екрановані з міркувань зручності читання.

Залежно від інтерпретатора регулярних виразів, метасимволи `«?»`, `«+»`, `«{»`, `«|»`, `«(»`, та `«)»` можуть втрачати своє спеціальне значення, замість цього слід вживати `«\?»`, `«\+»`, `«\{»`, `«\|»`, `«\(»`, та `«\)»`.

Набір символів у квадратних дужках `[ ]` називається символьним класом і дозволяє вказати інтерпретаторові регулярних виразів, що на даному місці в рядку може стояти один із перерахованих символів.

Зокрема, `[абв]` задає можливість появи в тексті одного із трьох зазначених символів, а `[1234567890]` задає відповідність одній із цифр. Можливе зазначення діапазонів символів: наприклад, `[0-9]`. Для включення усіх символів українського алфавіту можна використовувати `[Є-ЯҐ]`, `[а-їґ]`.

**Приклад 14.25.** Множина, що складається із слів «грати» та «прати», може бути описана регулярним виразом «[гп]рати».

Якщо потрібно вказати символи, які не входять у зазначений набір, то використовують символ ^ усередині квадратних дужок, наприклад, [^0-9] означає будь-який символ, крім цифр.

Для символічних класів, які часто використовуються, існують короткі позначення: будь-яка цифра (\d), будь який символ, за винятком цифр (\D), будь-який непробільний символ (\S), будь-який літерний символ, цифровий знак і знаку підкреслення (\w) тощо.

**Приклад 14.26.** Знайдемо всі документи, в яких значення ключа name містить велику букву Т (/T) серед будь-яких літер (\w), що зустрічаються один або більше раз (+/) (рис. 14.26):

```
> db.users.find({name: /T\w+/})
{ "_id" : ObjectId("5fc0c857fac1f17606f135ff"), "name" : "Tom", "age" : 28, "languages" : [ "english", "spanish" ] }
{ "_id" : ObjectId("5fc0cdd9fac1f17606f13607"), "name" : "Tom", "age" : 32, "languages" : [ "english", "german" ] }
>
```

*Рис. 14.26. Регулярний вираз, де значення ключа name містить велику букву Т (/T) серед будь-яких літер (\w), що зустрічаються один або більше раз (+/)*

Під цей шаблон попадають як Tom, так і Ted, Taras і навіть KaTe.

**Приклад 14.27.** Якщо ж задати замість великої малу букву t (/t), тоді побачимо, що регістр має значення (рис. 14.27).

```
> db.users.find({name: /t\w+/})
{ "_id" : ObjectId("5fc0c857fac1f17606f13606"), "name" : "Kate", "age" : 30, "languages" : [ "german", "russian" ] }
>
```

*Рис. 14.27. Регулярний вираз, де значення ключа name містить малу букву t (/t) серед будь-яких літер (\w), що зустрічаються один або більше раз (+/)*

**Приклад 14.28.** Вказавши, що регістр букв не має значення (/i), отримаємо всі документи, у яких ключ name містить як великі букви Т, так і малі t (рис. 14.28).

```
> db.users.find({name: /T\w+/i})
{ "_id" : ObjectId("5fc0c857fac1f17606f135ff"), "name" : "Tom", "age" : 28, "languages" : [ "english", "spanish" ] }
{ "_id" : ObjectId("5fc0c857fac1f17606f13606"), "name" : "Kate", "age" : 30, "languages" : [ "german", "russian" ] }
{ "_id" : ObjectId("5fc0cdd9fac1f17606f13607"), "name" : "Tom", "age" : 32, "languages" : [ "english", "german" ] }
>
```

Рис. 14.28. Регулярний вираз, де значення ключа *name* містить букву *T* (*/T*) серед будь-яких літер (*\w*), що зустрічаються один або більше раз (*+*) та регістр букв не має значення (*/i*)

### 14.3.5. Налаштування запитів та сортування

Для керування вибіркою в MongoDB використовуються функції:

- **limit();**
- **skip();**
- **sort().**

Функція **limit** задає максимально допустиму кількість документів у вибірці.

Кількість передається у вигляді числового параметра.

**Приклад 14.29.** Обмежити вибірку трьома документами (рис. 14.29):

```
> db.users.find().limit(3)
```

```
> db.users.find().limit(3)
{ "_id" : ObjectId("5fc0c857fac1f17606f135ff"), "name" : "Tom", "age" : 28, "languages" : [ "english", "spanish" ] }
{ "_id" : 12345789, "name" : "Jon", "age" : 27, "languages" : [ "english", "spanish" ] }
{ "_id" : ObjectId("5fc0c857fac1f17606f13600"), "name" : "Bob", "age" : 26, "languages" : [ "english", "frensh" ] }
>
```

Рис. 14.29. Обмежена вибірка з перших трьох документів колекції

В даному випадку ми отримаємо перші три документи (якщо в колекції 3 і більше документів). Якщо ж необхідно зробити вибірку не спочатку, а пропустивши деяку кількість документів, використовується функція **skip**.

**Приклад 14.30.** Обмежити вибірку трьома документами, пропустивши перші три записи (рис. 14.30):

```
> db.users.find().skip(3).limit(3)
```

```
> db.users.find().skip(3).limit(3)
{ "_id" : ObjectId("5fc0c857fac1f17606f13601"), "name" : "Alice", "age" : 27, "languages" : [ "german", "english" ] }
{ "_id" : ObjectId("5fc0c857fac1f17606f13602"), "name" : "Nik", "age" : 25, "languages" : [ "english", "spanish" ] }
{ "_id" : ObjectId("5fc0c857fac1f17606f13603"), "name" : "Bill", "age" : 32, "languages" : [ "english", "french" ] }
>
```

*Рис. 14.30. Обмежена вибірка з трьох документів, починаючи з четвертого елемента колекції (без перших трьох)*

MongoDB надає можливість отримати впорядкований набір даних з БД за допомогою функції `sort`. Передаючи в цю функцію значення 1 або -1, можна вказати, в якому порядку сортувати: за зростанням (1) чи за спаданням (-1). Багато в чому ця функція за дією аналогічна `ORDER BY` в SQL.

**Приклад 14.31.** Відсортувати за зростанням за полем `name` документи колекції `users` (рис. 14.31):

```
> db.users.find({}, {_id:0, name:1,
age:1}).sort({name:1})
```

```
> db.users.find({}, {_id: false, name:true, age:true}).sort({name:1})
{ "name" : "Alex", "age" : 28 }
{ "name" : "Alice", "age" : 27 }
{ "name" : "Bill", "age" : 32 }
{ "name" : "Bob", "age" : 26 }
{ "name" : "Emmy", "age" : 31 }
{ "name" : "Jon", "age" : 27 }
{ "name" : "Kate", "age" : 30 }
{ "name" : "Lene", "age" : 29 }
{ "name" : "Nik", "age" : 25 }
{ "name" : "Tom", "age" : 28 }
{ "name" : "Tom", "age" : 32 }
>
```

*Рис. 14.31. Сортування результатів вибірки за полем `name`*

Також варто зазначити, що поєднувати всі ці функції можна в одному ланцюжку.

**Приклад 14.32.** Вивести вибірку з трьох елементів колекції `users`, пропустивши перші три, та спроектувати (відобразити) тільки поля `name` та `age` без ключа `_id`, при цьому відсортувавши результат за зростанням за полем `name` (рис. 14.32):



```
> db.users.find({}, {_id: 0, name:1, age:1}).sort({name:1}).skip(3).limit(3)
{ "name" : "Bob", "age" : 26 }
{ "name" : "Emmy", "age" : 31 }
{ "name" : "Jon", "age" : 27 }
>
```

*Рис. 14.32. Поєднання багатьох функцій в одному запиті*

Результат не залежить від місця розташування в запиті функцій skip, limit та sort. (рис. 14.33):

```
> db.users.find({}, {_id: 0, name:1, age:1}).limit(3).sort({name:1}).skip(3)
{ "name" : "Bob", "age" : 26 }
{ "name" : "Emmy", "age" : 31 }
{ "name" : "Jon", "age" : 27 }
>
```

*Рис. 14.33. Запит до Прикладу 14.32 зі зміненою послідовністю функцій skip, limit та sort*

Але для сортування великого обсягу даних в MongoDB необхідно використовувати індекси.

### 14.3.6. Курсори

У документації MongoDB курсор визначається як покажчик на результуючий набір запиту. Іншими словами, результат вибірки, отриманої за допомогою функції find, називається **курсором**. Курсори інкапсулюють у собі набори об'єктів, одержаних з БД.

Реалізація курсорів дозволяє їх редагувати, наприклад пропускати непотрібні елементи, перебирати курсор для отримання результатів, обмежувати кількість елементів, сортувати за будь-яким ключем тощо.

Щоб отримати курсор і відразу не виводити всі дані, що містяться в ньому, можна використати один з підходів (або їх поєднати):

– до змінної додається ключове слово var;

```
> var cursor = db.users.find()
```

– після методу find() через крапку з комою додається вираз null;

```
> cursor = db.users.find(); null;
```

Для оброблення вмісту курсорів в MongoDB є більше 30 методів. Основні з них:

`hasNext()` – повертає `true`, якщо курсор має документи і може бути повторений;

`next()` – повертає наступний документ у курсорі;

`forEach()` – застосовує функцію циклу JavaScript до кожного документа в курсорі

`limit()` – обмежує розмір набору результатів курсору;

`sort()` – повертає результати, упорядковані відповідно до специфікації сортування;

`skip()` – повертає результати лише після пропуску ряду документів;

`count()` – повертає кількість документів у наборі результатів;

`pretty()` – налаштовує курсор для відображення результатів у зручному для читання форматі;

`toArray()` – повертає масив, що містить усі документи, повернуті курсором;

`close()` – закриває курсор і звільняє пов'язані ресурси сервера.

***За замовчуванням курсори відключаються через 10 хвилин бездіяльності.***

Розглянемо приклади створення і оброблення курсору згідно синтаксису мови javascript та застосування методів курсорів.

**Приклад 14.33.** Отримати курсор методом `var cursor = db.users.find()` та відобразити у циклі `while` методами `hasNext()` та `next()` значення ключів `name` (рис. 14.34):

```
> var cursor = db.users.find()
> while(cursor.hasNext()){
... obj = cursor.next();
... print(obj["name"]);
... }
Tom
Jon
Bob
Alice
Nik
Bill
Emmy
Lene
Kate
Tom
Alex
>
```

*Рис. 14.34. Отримання курсору методом `var cursor = db.users.find()` та оброблення результату у циклі `while` методами `hasNext()` та `next()`*

Курсор володіє методом `hasNext`, який показує при переборі, чи є ще в наборі документ. А метод `next` витягує поточний документ і переміщує курсор до наступного документу в наборі. У підсумку в змінній `obj` виявляється документ, до полів якого отримуємо доступ.

Альтернативою перебору документів у курсорі є метод `forEach`.

**Приклад 14.34.** Отримати курсор методом `var cursor = db.users.find()` та відобразити у циклі `forEach` значення ключів `name` (рис. 14.35).

```
> var cursor = db.users.find()
> cursor.forEach(function(obj){
...   print(obj.name);
... })
Tom
Jon
Bob
Alice
Nik
Bill
Emmy
Lene
Kate
Tom
Alex
>
```

*Рис. 14.35. Отримання курсору методом `var cursor = db.users.find()` та оброблення результату у циклі `forEach`*

Щоб обмежити розмір вибірки, використовується метод `limit`, що приймає кількість документів.

**Приклад 14.35.** Отримати курсор методом `var cursor = db.users.find()` та відобразити перші п'ять значень ключа `name` (рис. 14.36).

```
> var cursor = db.users.find()
> cursor.limit(5);null;
null
> cursor.forEach(function(obj){
...   print(obj.name);
... })
Tom
Jon
Bob
Alice
Nik
>
```

*Рис. 14.36. Отримання курсору методом `var cursor = db.users.find()`, обмеження його в діапазоні перших п'яти документів та оброблення результату у циклі `forEach`*

Використовуючи метод `sort()`, можна відсортувати документи в курсорі:

```
> cursor.sort({name:1});null;
```

Вираз `cursor.sort({name: 1})` сортує документи в курсорі за полем `name` за зростанням. Якщо необхідно впорядкувати за спаданням, то замість `1` використовуємо `-1`: `cursor.sort ({name: -1})`

Метод `skip()` дозволяє пропустити при вибірці певну кількість документів.

**Приклад 14.36.** Під час виведення курсору пропустити перші два документи:

```
> cursor.skip(2);null;
> cursor.forEach(function(obj) {
...  print(obj.name);
... })
```

І крім того, можна об'єднувати методи `skip`, `limit` та `sort` в ланцюжки. Як і в налаштуванні запитів, результат не залежить від місця розташування цих функцій в курсорі.

**Приклад 14.37.** Вивести курсор тільки для трьох документів, які відсортовані за полем `name` за зростанням, пропустивши перші 2 документи:

```
> var cursor = db.users.find();null;
null
> cursor.sort({name:1}).limit(3).skip(2);null;
null
> cursor.forEach(function(obj) {
...  print(obj.name);
... })
```

## 14.4. Команди групування

Окремо варто розглянути команди групування: `count`, `group`, `distinct`.

### 14.4.1. Кількість елементів в колекції

Функція `count()` повертає число елементів в колекції.

Формат запису:

```
db.ім'я_колекції.count()
```

**Приклад 14.38.** Знайти кількість документів в колекції users (рис. 14.37):

```
> db.users.count()
```



```
> db.users.count()  
11  
>
```

*Рис. 14.37. Отримання кількості документів колекції*

Тобто в колекції users поточної БД є 11 документів.

При поєднанні параметрів пошуку і функції count отримуємо фільтр на кількість документів.

**Приклад 14.39.** Визначити кількість документів у колекції users, у яких name = Tom (рис. 14.38):

```
> db.users.find({name: "Tom"}).count()
```



```
> db.users.find({name: "Tom"}).count()  
2  
>
```

*Рис. 14.38. Фільтр на кількість документів*

Більше того, можна створювати ланцюжки функцій skip та limit, щоб конкретизувати умови підрахунку.

Тут варто зазначити, що за замовчуванням функція count не використовується з функціями limit і skip. Для їх використання у функцію count треба передати логічне значення true.

**Приклад 14.40.** Визначити кількість документів у колекції users, пропустивши перші два документи, у яких name = Tom:

```
> db.users.find({name: "Tom"}).skip(2).count(true)
```

#### 14.4.2. Функція distinct

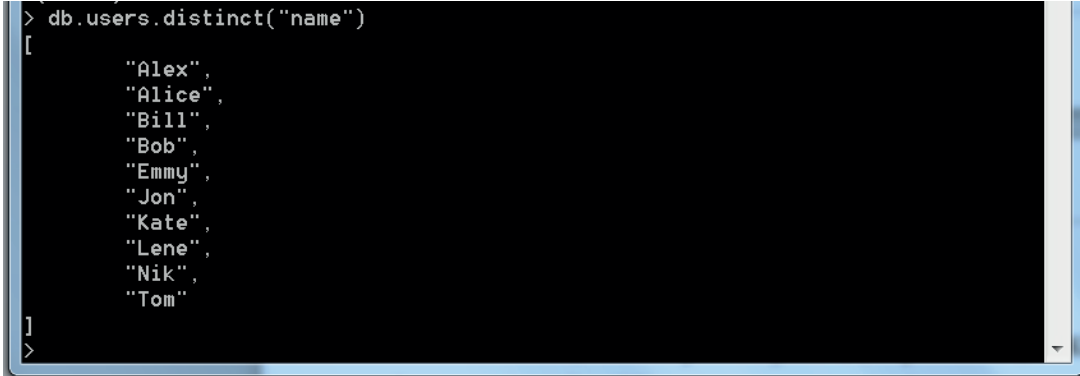
У колекції можуть бути документи, які містять однакові значення для одного або декількох полів. І якщо потрібно отримати унікальні значення поля, тобто перелік значень, що не повторюються, в MongoDB використовується функція distinct.

Формат запису функції:

**db.ім'я\_колекції.distinct(<ключ\_поля>)**

**Приклад 14.41.** В декількох документах визначено name: "Tom". А треба вивести тільки унікальні значення для поля name всіх документів колекції (рис. 14.39):

```
> db.users.distinct("name")
```



```
> db.users.distinct("name")
[
  "Alex",
  "Alice",
  "Bill",
  "Bob",
  "Emmy",
  "Jon",
  "Kate",
  "Lene",
  "Nik",
  "Tom"
]
```

*Рис. 14.39. Унікальні значення поля name документів колекції users*

В результаті буде виведено масив унікальних значень вказаного ключа.

### 14.4.3. Агрегація і метод group

Агрегація – це групування значень багатьох документів за допомогою операцій:

- \$sum;
- \$avg;
- \$min;
- \$max.

В MongoDB для агрегації застосовується метод **aggregate()**. Даний метод має наступний синтаксис:

```
db.<ім'я_колекції>.aggregate(
  [ { $group : { _id : <ім'я_ключа_результату>,
    <ім'я_результату> :
      {<операція_агрегації> : <поле, за яким здійснюється агрегація> } }
  ] )
```

Для зберігання результату обчислень необхідно явно задати ключ та його ім'я після методу `group`, а також вказати ім'я результату, якому буде присвоєно результат обчислень.

**Приклад 14.42.** Знайти середній вік персон з колекції `users` (рис. 14.40):

```
> db.users.aggregate([{$group:{_id:"111",Total:{$avg:"$age"}}}])
{ "_id" : "111", "Total" : 27.90909090909091 }
>
```

Рис. 14.40. Обчислення середнього операцією агрегації `avg` за полем `age`

Використання методу `group` подібне до використання виразу `GROUP BY` в SQL. Метод `group` приймає три обов'язкові параметри:

- **key:** вказує на ключ, за яким треба проводити групування;
- **initial:** ініціалізує поля документа, який буде представляти групу документів;
- **reduce:** представляє функцію, яка повертає кількість елементів. Ця функція приймає в якості аргументів два параметри: поточний елемент і агрегатний результуючий документ для поточної групи.

Формат запису команди групування:

```
> db.ім'я_колекції.group ({
    key: {ключ : true},
    initial: {total : числова ініціалізація результату},
    reduce : function (curr, res){функція обчислення результату}
})
```

**Приклад 14.43.** Реалізувати групування за полем `name` з виведенням підсумків за однотипними значеннями (групами) цього поля (рис. 14.41):

```
> db.users.group ({
    key: {name : true},
    initial: {total : 0},
    reduce : function (curr, res){res.total += 1}})
```

Розберемо вираз. Параметр `key` вказує, що групування буде проводитися за ключем `name` – `key: {name: true}`.

```

> db.users.group({
... key:{name : true},
... initial : {total : 0},
... reduce : function (curr, res){res.total += 1}
... })
[
  {
    "name" : "Tom",
    "total" : 2
  },
  {
    "name" : "Jon",
    "total" : 1
  },
  {
    "name" : "Bob",
    "total" : 1
  },
  {
    "name" : "Alice",
    "total" : 1
  }
]

```

*Рис. 14.41. Групування за полем name з виведенням підсумків за однотипними значеннями (групами) цього поля*

Значення параметра `initial` ініціалізує поля результуючого документа, який буде представляти групу. Так, в даному випадку встановлюється початкове значення поля `total`. Це поле буде представляти кількість елементів для групи. І так як елементів може і не бути, то ініціалізуємо його нулем.

Параметр `reduce` є функцією, де параметр `curr` посилається на поточний об'єкт в групі, а `res` представляє поточну групу. Якщо знайдений ще один об'єкт з певним значенням для поля `name`, то цей документ додається в групу, а в документі `res` на одиницю збільшується значення `total`.

Використовуючи групування, слід враховувати, що функція `group` не працює в сегментованих кластерах (`sharded cluster`). У цьому випадку замість функції `group` слід використовувати функціональність `map-reduce`.

## 14.5. Оператори керування

### 14.5.1. Умовні оператори

Умовні оператори задають умову, якій має відповідати значення поля документа:

- `$eq` (рівне);
- `$ne` (не рівне);



- **\$gt** (більше чим);
- **\$lt** (менше чим);
- **\$gte** (більше або рівне);
- **\$lte** (менше або рівне);
- **\$in** визначає масив значень, одне з яких має мати поле документа;
- **\$nin** визначає масив значень, одне з яких не повинно мати поле документа.

**Приклад 14.44.** Знайти всі документи, у яких значення ключа `age` менше 28 (рис. 14.42):

```
> db.users.find ({age: {$lt : 28}})
```



```

C:\Program Files\MongoDB\Server\4.0\bin>mongo.exe
> db.users.find({age:{$lt:28}})
{ "_id" : 123456789, "name" : "Jon", "age" : 27, "languages" : [ "english", "spanish" ] }
{ "_id" : ObjectId("5fb63ac393865d1cd4b0a850"), "name" : "Bob", "age" : 26, "languages" : [ "english", "frensh" ] }
{ "_id" : ObjectId("5fb63ac393865d1cd4b0a851"), "name" : "Alice", "age" : 27, "languages" : [ "german", "english" ] }
{ "_id" : ObjectId("5fb63c4b93865d1cd4b0a852"), "name" : "Nik", "age" : 25, "languages" : [ "english", "spanish" ] }
>
  
```

Рис. 14.42. Вивід документів з обмеженням на значення поля `age < 28`

Аналогічно буде використання інших операторів порівняння.

**Приклад 14.45.** Знайти всі документи, у яких значення ключа `age` більше 30 (рис. 14.43):

```
> db.users.find ({age: {$gt : 30}})
```



```

C:\Program Files\MongoDB\Server\4.0\bin>mongo.exe
> db.users.find({age:{$gt:30}})
{ "_id" : ObjectId("5fb63d0a93865d1cd4b0a853"), "name" : "Bill", "age" : 32, "languages" : [ "english", "french" ] }
{ "_id" : ObjectId("5fb6a70d829e3671d2cf81ae"), "name" : "Tom", "age" : 32, "languages" : [ "english", "german" ] }
>
  
```

Рис. 14.43. Вивід документів з обмеженням на значення поля `age > 30`

Варто відзначити, що порівняння проводиться над цілочисельними типами, а не рядками. Якщо ключ `age` представляє рядкові значення, то відповідно треба

проводити порівняння над рядками: `db.users.find ({age: {$ gt: "30"}})`, однак результат буде тим же.

**Приклад 14.46.** Знайти користувачів, вік яких дорівнює 27 (рис. 14.44):

```
> db.users.find ({age: {$eq : 27}})
```

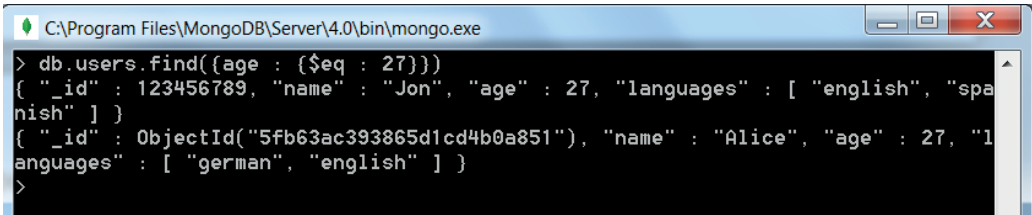


Рис. 14.44. Вивід документів з обмеженням на значення поля `age = 27`

По суті це аналогія наступного запиту:

```
> db.users.find ({age: 27})
```

**Приклад 14.47.** Зворотна операція – знайти користувачів, вік яких НЕ дорівнює 27 (рис. 14.45):

```
> db.users.find ({age: {$ne : 27}})
```

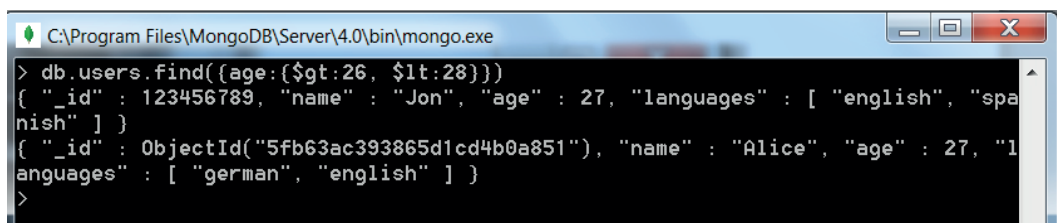


Рис. 14.45. Вивід документів з обмеженням на значення поля `age ≠ 27`

Для задавання діапазону значень логічні оператори поєднуються комою, що рівносильне функції AND.

**Приклад 14.48.** Знайти всі об’єкти зі значенням поля `age` більше 26, але менше 28 (рис. 14.46):

```
> db.users.find ({age: {$gt : 26, $lt: 28}})
```



```
C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe
> db.users.find({age:{$gt:26, $lt:28}})
{ "_id" : 123456789, "name" : "Jon", "age" : 27, "languages" : [ "english", "spanish" ] }
{ "_id" : ObjectId("5fb63ac393865d1cd4b0a851"), "name" : "Alice", "age" : 27, "languages" : [ "german", "english" ] }
>
```

Рис. 14.46. Вивід документів з обмеженням на значення поля  $26 < \text{age} < 28$

Оператор `$in` визначає масив можливих виразів і шукає ті ключі, значення яких є в масиві.

**Приклад 14.49.** Знайти всі об'єкти зі значенням поля `age` 27 та 32 (рис. 14.47):

```
> db.users.find ({age: {$in : [27, 32]}})
```



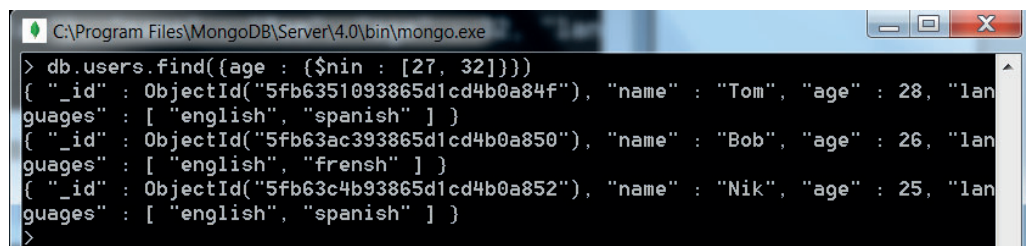
```
C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe
> db.users.find({age : {$in : [27, 32]}})
{ "_id" : 123456789, "name" : "Jon", "age" : 27, "languages" : [ "english", "spanish" ] }
{ "_id" : ObjectId("5fb63ac393865d1cd4b0a851"), "name" : "Alice", "age" : 27, "languages" : [ "german", "english" ] }
{ "_id" : ObjectId("5fb63d0a93865d1cd4b0a853"), "name" : "Bill", "age" : 32, "languages" : [ "english", "french" ] }
{ "_id" : ObjectId("5fb6a70d829e3671d2cf81ae"), "name" : "Tom", "age" : 32, "languages" : [ "english", "german" ] }
>
```

Рис. 14.47. Вивід документів зі значення поля `age` 27 або 32

Протилежним чином діє оператор `$nin` – він визначає масив можливих виразів і шукає ті ключі, значення яких відсутні в цьому масиві.

**Приклад 14.50.** Знайти всі об'єкти за винятком значень `age` 27 або 32 (рис. 14.48):

```
> db.users.find ({age: {$nin : [27, 32]}})
```



```
C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe
> db.users.find({age : {$nin : [27, 32]}})
{ "_id" : ObjectId("5fb6351093865d1cd4b0a84f"), "name" : "Tom", "age" : 28, "languages" : [ "english", "spanish" ] }
{ "_id" : ObjectId("5fb63ac393865d1cd4b0a850"), "name" : "Bob", "age" : 26, "languages" : [ "english", "frensh" ] }
{ "_id" : ObjectId("5fb63c4b93865d1cd4b0a852"), "name" : "Nik", "age" : 25, "languages" : [ "english", "spanish" ] }
>
```

Рис. 14.48. Вивід документів за винятком значень `age` 27 або 32

## 14.5.2. Логічні оператори

Логічні оператори виконуються над умовами вибірки:

- **\$or**: з'єднує дві умови, і документ повинен відповідати одній з цих умов;
- **\$and**: з'єднує дві умови, і документ повинен відповідати обом умовам;
- **\$not**: документ повинен НЕ відповідати умові;
- **\$nor**: з'єднує дві умови, і документ повинен НЕ відповідати обом умовам.

### Оператор \$or

Оператор \$or є логічною операцією АБО і визначає набір пар ключ-значення, які повинні бути в документі. І якщо документ має хоч одну таку пару ключ-значення, то він відповідає даному запиту і витягується з БД.

**Приклад 14.51.** Повернути всі документи, в яких name = Tom або age = 27 (рис. 14.49):

```
> db.users.find ({ $or : [{ name: "Tom" }, { age: 27 }] })
```

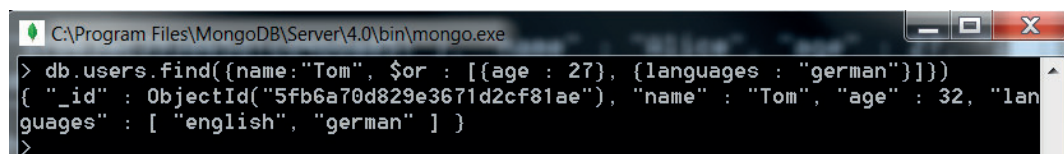


```
C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe
> db.users.find({ $or : [{ name : "Tom" }, { age : 27 }] })
{ "_id" : ObjectId("5fb6351093865d1cd4b0a84f"), "name" : "Tom", "age" : 28, "languages" : [ "english", "spanish" ] }
{ "_id" : 123456789, "name" : "Jon", "age" : 27, "languages" : [ "english", "spanish" ] }
{ "_id" : ObjectId("5fb63ac393865d1cd4b0a851"), "name" : "Alice", "age" : 27, "languages" : [ "german", "english" ] }
{ "_id" : ObjectId("5fb6a70d829e3671d2cf81ae"), "name" : "Tom", "age" : 32, "languages" : [ "english", "german" ] }
>
```

Рис. 14.49. Вивід документів, в яких або name = Tom, або age = 27

**Приклад 14.52.** Повернути всі документи, в яких name = Tom, а age рівне або 27, або серед значень languages є "german" (рис. 14.50):

```
> db.users.find({ name: "Tom", $or: [{ age: 27 }, { languages: "german" }] })
```



```
C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe
> db.users.find({ name: "Tom", $or : [{ age : 27 }, { languages : "german" }] })
{ "_id" : ObjectId("5fb6a70d829e3671d2cf81ae"), "name" : "Tom", "age" : 32, "languages" : [ "english", "german" ] }
>
```

Рис. 14.50. Вивід документів, в яких name = Tom, а age рівне або 27, або серед значень languages є "german"

У підвиразах ог можна застосовувати умовні оператори.

**Приклад 14.53.** Повернути всі документи, в яких АБО name = Tom, АБО age більше рівне 30 (рис. 14.51):

```
> db.users.find ({ $or : [{ name: "Tom" }, { age: { $gte: 30 } } ] })
```

В даному прикладі вибираються всі документи, де або name = "Tom", або поле age має значення від 30 і вище.



```
C:\Program Files\MongoDB\Server\4.0\bin> mongo.exe
> db.users.find ({ $or : [{ name: "Tom" }, { age: { $gte: 30 } } ] })
{ "_id" : ObjectId("5fb6351093865d1cd4b0a84f"), "name" : "Tom", "age" : 28, "languages" : [ "english", "spanish" ] }
{ "_id" : ObjectId("5fb63d0a93865d1cd4b0a853"), "name" : "Bill", "age" : 32, "languages" : [ "english", "french" ] }
{ "_id" : ObjectId("5fb6a70d829e3671d2cf81ae"), "name" : "Tom", "age" : 32, "languages" : [ "english", "german" ] }
```

Рис. 14.51. Вивід документів, в яких АБО name = Tom, АБО age більше рівне 30

### Оператор \$and

Оператор \$and є логічною операцією І (логічне множення) і визначає набір критеріїв, яким обов'язково повинен відповідати документ. На відміну від оператора \$or, документ повинен відповідати всім зазначеним критеріям.

**Приклад 14.54.** Повернути всі документи, в яких name = Tom І age = 32 (рис. 14.52):

```
> db.users.find ({ $and : [{ name: "Tom" }, { age: 32 } ] })
```



```
C:\Program Files\MongoDB\Server\4.0\bin> mongo.exe
> db.users.find ({ $and : [{ name: "Tom" }, { age: 32 } ] })
{ "_id" : ObjectId("5fb6a70d829e3671d2cf81ae"), "name" : "Tom", "age" : 32, "languages" : [ "english", "german" ] }
```

Рис. 14.52. Вивід документів, в яких І name = Tom, І age = 32

Тут обрані документи обов'язково повинні мати ім'я Tom і вік 32 – обидві ці ознаки.

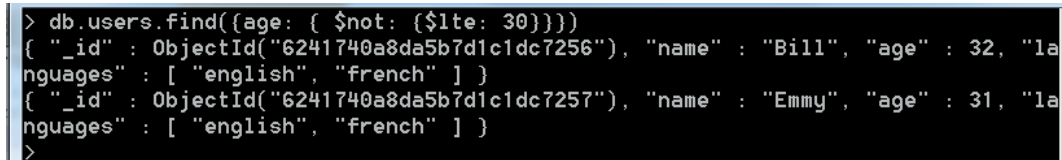
### Оператор \$not

Оператор \$not є логічним запереченням та повертає документи, які не відповідають умові після цього оператора.

Варто зазначити, що оператор `$not` впливає лише на інші оператори і не може самостійно перевіряти поля та документи.

**Приклад 14.55.** Повернути всі документи, в яких вік `age` більший за 30. Для цього використаємо заперечення `$not` для тих документів, в яких `age` менше рівне 30 (рис. 14.53):

```
> db.users.find ( age: { $not: { $lte: 30 } } )
```



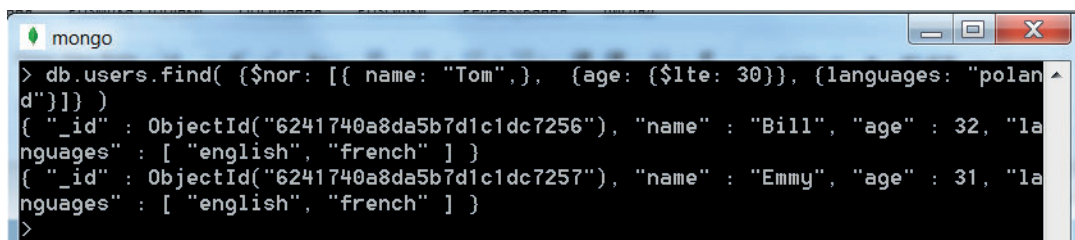
```
> db.users.find({age: { $not: { $lte: 30 }}})
{ "_id" : ObjectId("6241740a8da5b7d1c1dc7256"), "name" : "Bill", "age" : 32, "languages" : [ "english", "french" ] }
{ "_id" : ObjectId("6241740a8da5b7d1c1dc7257"), "name" : "Emmy", "age" : 31, "languages" : [ "english", "french" ] }
>
```

Рис. 14.53. Вивід документів, в яких `age` не менше 30

### Оператор `$nor`

Оператор `$nor` виконує логічну операцію заперечення NOR над двома або більше логічними виразами і повертає документи, для яких НЕ виконуються УСІ логічні вирази.

**Приклад 14.56.** Повернути всі документи, в яких `name` не "Tom", вік `age` не менший за 30 та мова `languages` не "poland" (рис. 14.54):



```
> db.users.find( { $nor: [{ name: "Tom", }, { age: { $lte: 30 } }, { languages: "poland" } ] } )
{ "_id" : ObjectId("6241740a8da5b7d1c1dc7256"), "name" : "Bill", "age" : 32, "languages" : [ "english", "french" ] }
{ "_id" : ObjectId("6241740a8da5b7d1c1dc7257"), "name" : "Emmy", "age" : 31, "languages" : [ "english", "french" ] }
>
```

Рис. 14.54. Вивід документів за умовами, поєднаними оператором `$nor`

### 14.5.3. Пошук у масивах

Для пошуку у масивах призначені такі оператори:

- **\$all:** визначає набір значень, які повинні бути в масиві;
- **\$size:** визначає кількість елементів, які повинні бути в масиві;
- **\$elemMatch:** визначає умову, якій повинні відповідати елементи в масиві.

## Оператор \$all

Оператор \$all застосовується для пошуку у масиві і визначає масив можливих виразів та вимагає, щоб документи мали цей набір виразів.

**Приклад 14.57.** В документах є масив languages, який зберігає іноземні мови, на яких говорить користувач. І щоб знайти всіх людей, які говорять одночасно і англійською, і німецькою, необхідно записати такий вираз (рис. 14.55):

```
> db.users.find({languages: {$all: ["english",  
"german"]}})
```

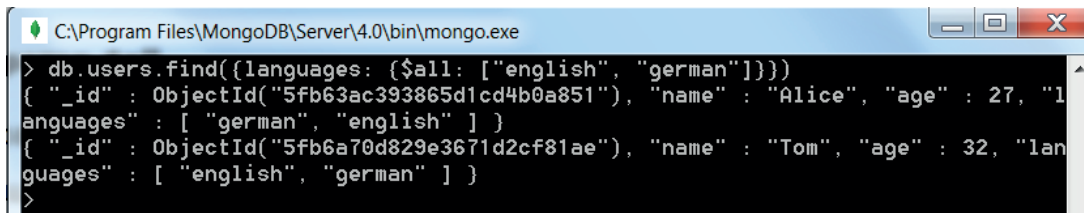


Рис. 14.55. Вивід документів, в яких в масиві languages одночасно є значення і "english", і "german"

## Оператор \$size

Оператор \$size використовується для знаходження документів, в яких масиви мають кількість елементів, рівну значенню \$size.

**Приклад 14.58.** Знайти всі документи, в яких в масиві languages є по три елементи:

```
> db.users.find ({languages: {$size:3}})
```

## Оператор \$elemMatch

Оператор \$elemMatch дозволяє вибрати документи, в яких масиви містять елементи, що потрапляють під певні умови.

**Приклад 14.59.** Нехай в Базі Даних буде колекція, яка містить оцінки користувачів за певними курсами. Додамо декілька документів:

```
> db.grades.insertMany([  
  {student: "Tom", courses:[{name: "Java", grade: 5},  
    {name: "MongoDB", grade: 3}]},  
  {student: "Alice", courses:[{name: "C++", grade: 3},  
    {name: "MongoDB", grade: 5}]}  
)
```



Кожен документ має масив `courses`, який складається з вкладених документів. Тепер знайдемо студентів, які для курсу MongoDB мають оцінку вище 4 (рис. 14.56):

```
> db.grades.find(  
  {courses: {$elemMatch:  
    {name: "MongoDB", grade: {$gt: 4}}}})
```

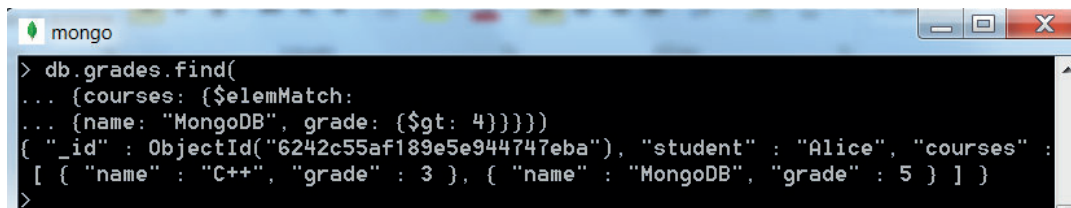


Рис.14.56. Вивід документів з колекції `grades`, в яких в масиві `courses` для `name = "MongoDB"` поле `grade` відповідає умові `grade > 4`

#### 14.5.4. Оператори `$exists`, `$type` та `$regex`

Оператор `$exists` дозволяє витягти тільки ті документи, в яких певний ключ присутній або відсутній.

Якщо ж задати в операторі `$exists` як параметр `false`, тоді запит поверне тільки ті документи, в яких ключ `company` не визначений.

Оператор `$type` витягує тільки ті документи, в яких заданий ключ має значення вказаного типу (рядок, число, дата тощо).

Оператор `$regex` задає регулярний вираз, яким має відповідати значення поля.

**Приклад 14.60.** Знайти всі документи, в яких є ключ `company` (оператор `$exists`). Для цього додамо в колекцію `users` ще один документ:

```
db.users.insert({"name": "Alex", "age": 28,  
  "company": {"name": "microsoft", "country": "USA"}  
})
```

і, відповідно, в колекції `users` знайдемо документи, в яких ключ є `"company"` (рис. 14.57):

```
> db.users.find ({company: {$exists:true}})
```



```
> db.users.find({company: {$exists: true}})
{ "_id" : ObjectId("62455c85f3662a787c8f411a"), "name" : "Alex", "age" : 28, "company" : { "name" : "microsoft", "country" : "USA" } }
>
```

Рис.14.57. Приклад застосування оператора *\$exists*

В колекції users є багато документів, і лише в одного ключем є "company".

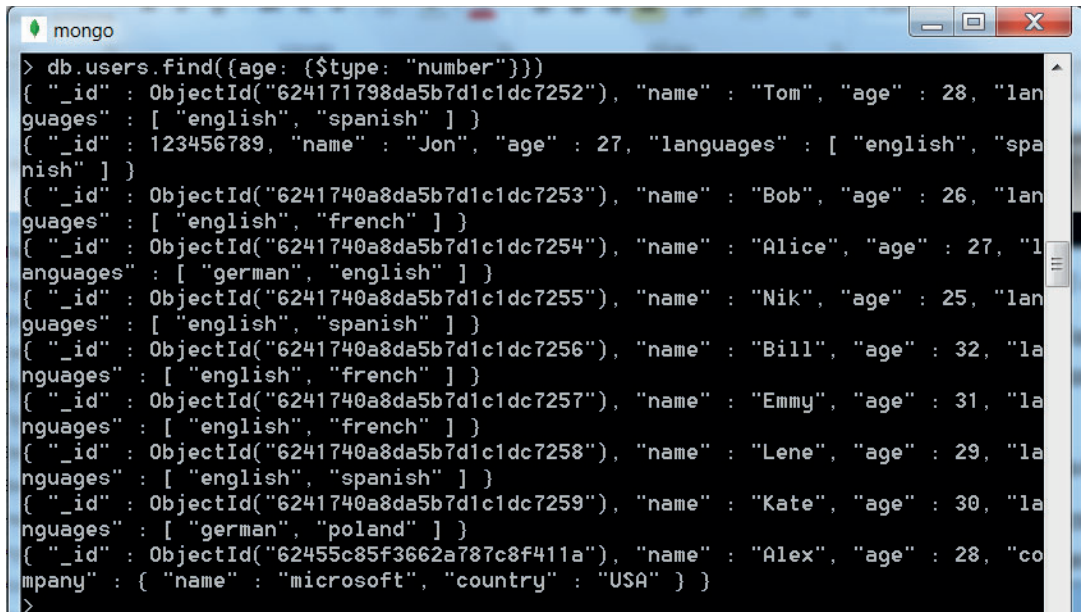
**Приклад 14.61.** Знайти всі документи, в яких поле age є рядкового типу (оператор \$type):

```
> db.users.find ({age: {$type:"string"}})
```

Але у всіх документах поле age є цілочисельного типу, тому в результаті такого запиту нічого не отримаємо. Відповідно, якщо цей запит змінити:

```
> db.users.find ({age: {$type:"number"}})
```

тоді отримаємо документи, в яких поле age є числового типу (рис. 14.58).



```
mongo
> db.users.find({age: {$type: "number"}})
{ "_id" : ObjectId("624171798da5b7d1c1dc7252"), "name" : "Tom", "age" : 28, "languages" : [ "english", "spanish" ] }
{ "_id" : 123456789, "name" : "Jon", "age" : 27, "languages" : [ "english", "spanish" ] }
{ "_id" : ObjectId("6241740a8da5b7d1c1dc7253"), "name" : "Bob", "age" : 26, "languages" : [ "english", "french" ] }
{ "_id" : ObjectId("6241740a8da5b7d1c1dc7254"), "name" : "Alice", "age" : 27, "languages" : [ "german", "english" ] }
{ "_id" : ObjectId("6241740a8da5b7d1c1dc7255"), "name" : "Nik", "age" : 25, "languages" : [ "english", "spanish" ] }
{ "_id" : ObjectId("6241740a8da5b7d1c1dc7256"), "name" : "Bill", "age" : 32, "languages" : [ "english", "french" ] }
{ "_id" : ObjectId("6241740a8da5b7d1c1dc7257"), "name" : "Emmy", "age" : 31, "languages" : [ "english", "french" ] }
{ "_id" : ObjectId("6241740a8da5b7d1c1dc7258"), "name" : "Lene", "age" : 29, "languages" : [ "english", "spanish" ] }
{ "_id" : ObjectId("6241740a8da5b7d1c1dc7259"), "name" : "Kate", "age" : 30, "languages" : [ "german", "poland" ] }
{ "_id" : ObjectId("62455c85f3662a787c8f411a"), "name" : "Alex", "age" : 28, "company" : { "name" : "microsoft", "country" : "USA" } }
>
```

Рис.14.58. Приклад застосування оператора *\$type*

**Приклад 14.62.** Знайти всі документи, в яких поле name обов'язково має букву "b" (рис. 14.59):

```
> db.users.find ({name: {$regex:"b"}})
```

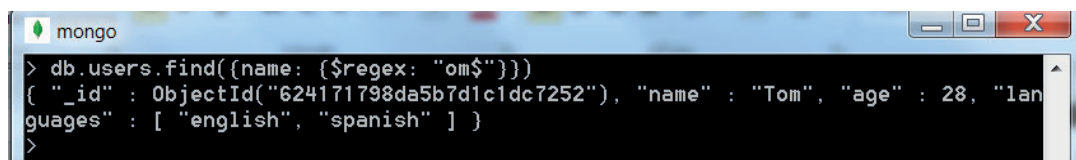
```
> db.users.find({name:{$regex:"b"}})
{ "_id" : ObjectId("5fc0c857fac1f17606f13600"), "name" : "Bob", "age" : 26, "languages" : [ "english", "frensh" ] }
>
```

Рис.14.59. Приклад застосування оператора \$regex

Важливо розуміти, що \$regex приймає не просто рядки, а саме регулярні вирази.

**Приклад 14.63.** Знайти всі документи, в яких значення name закінчується на "om":

```
> db.users.find ({name: {$regex: "om$"}})
```



```
mongo
> db.users.find({name: {$regex: "om$"}})
{ "_id" : ObjectId("624171798da5b7d1c1dc7252"), "name" : "Tom", "age" : 28, "languages" : [ "english", "spanish" ] }
>
```

Рис.14.60. Вивід документів, в яких значення name закінчується на "om"

## 14.6. Оновлення даних

### 14.6.1. Метод save

Як і інші системи управління базами даних, MongoDB надає змогу оновлення даних. Найбільш простим для використання є метод **save**. Як параметр цей метод приймає документ.

В цей документ в якості поля можна передати параметр `_id`. Якщо метод знаходить документ з таким значенням `_id`, то документ оновлюється. Якщо ж з подібним `_id` документів немає, тоді документ вставляється.

Якщо параметр `_id` не вказано, то документ вставляється, а параметр `_id` генерується автоматично (як при звичайному додаванні через команду `insert`).

**Приклад 14.64.** Замінити за відомим `_id` 123456789 вік `age` на 28, а `languages` "spanish" на "german", залишивши поле `name` без змін (рис. 14.61):

```
> db.users.save ({_id: 123456789, name: "Jon", age: 28,
  languages: ["english", "german"] }
  })
```

```

> db.users.find({_id:12345789})
{ "_id" : 12345789, "name" : "Jon", "age" : 27, "languages" : [ "english", "spanish" ] }
> db.users.save({_id:12345789, name:"Jon",age:28, languages:["englis","german"]})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.users.find({_id:12345789})
{ "_id" : 12345789, "name" : "Jon", "age" : 28, "languages" : [ "englis", "german" ] }
>

```

*Рис.14.61. Заміна значень полів методом save*

При цьому варто пам'ятати, що оператор save перезаписує документ з вказаним `_id`, тобто витирає попередній документ і створює новий тільки з тими полями, які вказані.

У прикладі зроблена синтаксична помилка для наочного відображення вище сказаного: у полі `languages` значення `englis` (а не `english`) записано з помилкою.

**Приклад 14.65.** Змінити документ колекції `users` за відомим `_id` 123456789, вказавши поля `name` та `age` і опустивши поле `languages` (рис. 14.62):

```

> db.users.save({_id:12345789, name:"Jon",age:28})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.users.find({_id:12345789})
{ "_id" : 12345789, "name" : "Jon", "age" : 28 }
>

```

*Рис.14.62. Заміна документу методом save*

## 14.6.2. Функція update

Більший функціонал при оновленні документа пропонує команда `update`. Вона приймає три параметри:

- **query**: приймає запит на вибірку документа, який треба оновити;
- **objNew**: представляє документ з новою інформацією, який замінить старий при оновленні;
- **options**: визначає додаткові параметри при оновленні документів. Може приймати два аргументи: `upsert` і `multi`.

Формат запису:

```

db.ім'я_колекції.update(
    {query запит на вибірку – ключ : значення},
    { objNew поля для зміни – ключ : значення },

```

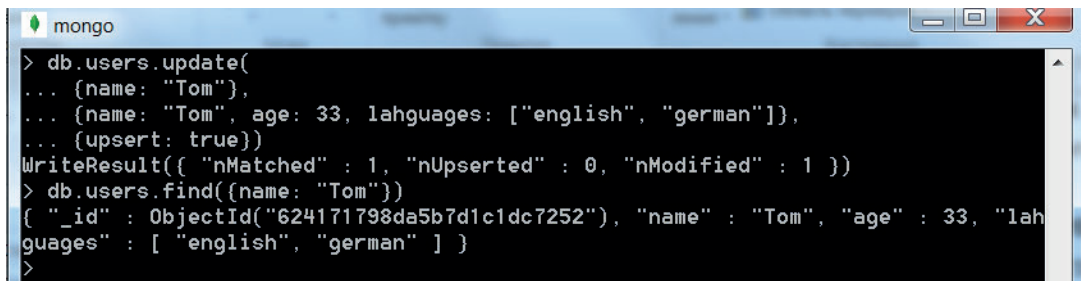
**{options – upsert/multi: true/false }}**

Якщо параметр `upsert` має значення `true`, то MongoDB буде оновлювати документ, якщо він знайдений, і створювати новий, якщо такого документа немає. Якщо ж він має значення `false`, то MongoDB не створюватиме новий документ, якщо запит на вибірку не знайде жодного документа.

Параметр `multi` вказує, чи повинен оновлюватися перший документ у вибірці (використовується за замовчуванням, якщо цей параметр не вказано) або ж повинні оновлюватися всі документи в вибірці.

**Приклад 14.65.** Оновити документ за умовою `{name : "Tom"}` значеннями `{name:"Tom", age: 33, languages: ["english", "german"]}` (рис. 14.63):

```
> db.users.update(  
  {name : "Tom"},  
  {name:"Tom", age: 33, languages: ["english",  
    "german"]}, {upsert: true})
```



*Рис.14.63. Оновлення документу методом update*

Тепер перший документ, знайдений запитом `{name: "Tom"}`, буде перезаписаний документом із заданими значеннями.

### 14.6.3. Оновлення окремого поля

Часто не потрібно оновлювати весь документ, а тільки значення одного з його ключів. Для цього застосовується оператор `$set`.

Формат запису:

```
db.ім'я_колекції.update(  
  {query запит на вибірку – ключ : значення }  
  {$set : { ключ : нове значення}}  
)
```

**Приклад 14.66.** У БД є документ {name : "Tom", age : 33}. Необхідно змінити у цьому документі тільки age на 34 (рис. 14.64):

```
{ "_id" : ObjectId("5fcbf8a1de9a0f06645d21d8"), "name" : "Tom", "age" : 33, "languages" : [ "english", "german" ] }
> db.users.update( {name:"Tom", age:33}, {$set:{age:34}} )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.users.find({name:"Tom", age:34})
{ "_id" : ObjectId("5fcbf8a1de9a0f06645d21d8"), "name" : "Tom", "age" : 34, "languages" : [ "english", "german" ] }
>
```

*Рис.14.64. Оновлення в першому знайденому документі тільки одного поля*

Таким чином, поєднання команди update та оператора \$set залишить всі решту поля документа без змін, модифікувавши при цьому тільки вказане поле.

Якщо документ не містить поле, яке необхідно оновити, то тоді таке поле створюється.

**Приклад 14.67.** Оновити перший знайдений у вибірці документ за умовою {name: "Tom", age: 34} шляхом зміни поля salary (рис. 14.65).

```
> db.users.update( {name:"Tom", age: 34} ,
                  {$set:{salary:3000}} )
```

```
> db.users.update( {name:"Tom", age:34}, {$set:{salary:3000}} )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.users.find({name:"Tom", age:34})
{ "_id" : ObjectId("5fcbf8a1de9a0f06645d21d8"), "name" : "Tom", "age" : 34, "languages" : [ "english", "german" ], "salary" : 3000 }
>
```

*Рис.14.65. Додавання в перший знайдений документ відсутнього поля*

В даному випадку оновився тільки один документ, перший у вибірці, а поле salary було додано до документа, оскільки його в документі не було.

Вказавши значення multi: true, можна оновити всі документи вибірки.

**Приклад 14.68.** Змінити у всіх Томів вік на 25, а якщо такого ключа немає в документі, тоді створити його.

Нехай в БД є 3 документи {name : "Tom"} (рис. 14.66):

```
> db.users.find({name:"Tom"})
{ "_id" : ObjectId("5fce1ce9e0430a34fddb6ace"), "name" : "Tom", "age" : 28, "languages" : [ "english", "spanish" ] }
{ "_id" : ObjectId("5fce1ce9e0430a34fddb6ad7"), "name" : "Tom", "languages" : [ "english", "french" ] }
{ "_id" : ObjectId("5fce1ce9e0430a34fddb6ad8"), "name" : "Tom", "age" : 34, "languages" : [ "english", "german" ], "salary" : 3000 }
```

*Рис.14.66. Вибірка з колекції users документів з умовою {name : "Tom"}*

Для зміни у всіх документах, що відповідають вказаній умові, необхідно у команді update задати значення multi : true (рис. 14.67):

```
> db.users.update(
... {name:"Tom"},
... {$set:{name:"Tom", age:25}},
... {multi:true}
... )
WriteResult({ "nMatched" : 3, "nUpserted" : 0, "nModified" : 3 })
```

Рис.14.67. Заміна значень в багатьох документах

В результаті буде змінено у всіх документах, де name : "Tom", ключ age на 25. Якщо такого ключа у документі не було, тоді він створюється автоматично, як для другого документа вибірки (рис. 14.68):

```
> db.users.find({name:"Tom"})
{ "_id" : ObjectId("5fce1ce9e0430a34fdbb6ace"), "name" : "Tom", "age" : 25, "languages" : [ "english", "spanish" ] }
{ "_id" : ObjectId("5fce1ce9e0430a34fdbb6ad7"), "name" : "Tom", "languages" : [ "english", "french" ], "age" : 25 }
{ "_id" : ObjectId("5fce1ce9e0430a34fdbb6ad8"), "name" : "Tom", "age" : 25, "languages" : [ "english", "german" ], "salary" : 3000 }
```

Рис.14.68. Результати виконання команди update з опцією multi: true

Для простого збільшення значення числового поля на певну кількість одиниць застосовується оператор \$inc. Якщо документ не містить поле, яке необхідно оновити, то тоді таке поле створюється. Даний оператор застосовується лише до числових значень.

**Приклад 14.69.** Замінити у першому документі, що відповідає умові {name : "Tom"} значення числового поля age на значення, збільшене на 2 (рис. 14.69):

```
> db.users.update({name : "Tom"}, {$inc: {age:2}})
```

```
> db.users.update({name:"Tom"}, {$inc:{age:2}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.users.find({name:"Tom"})
{ "_id" : ObjectId("5fce1ce9e0430a34fdbb6ace"), "name" : "Tom", "age" : 27, "languages" : [ "english", "spanish" ] }
{ "_id" : ObjectId("5fce1ce9e0430a34fdbb6ad7"), "name" : "Tom", "languages" : [ "english", "french" ], "age" : 25 }
{ "_id" : ObjectId("5fce1ce9e0430a34fdbb6ad8"), "name" : "Tom", "age" : 25, "languages" : [ "english", "german" ], "salary" : 3000 }
```

Рис.14.69. Приклад використання оператора \$inc

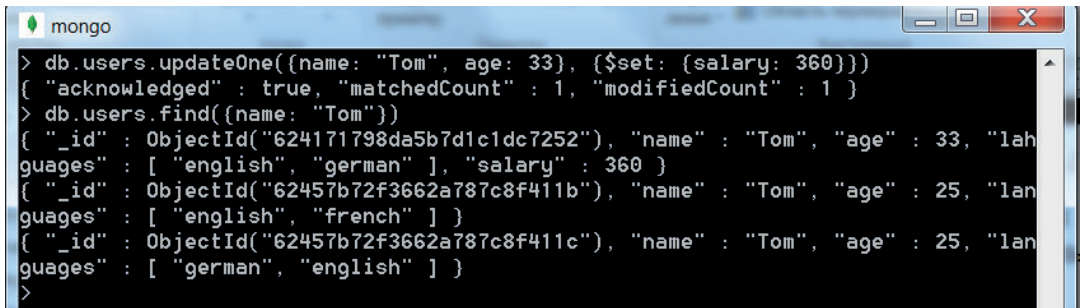
В результаті серед трьох документів з name : "Tom" зміниться тільки у першому документі поле age, збільшене на 2 – було 25, стало 27.

#### 14.6.4. Методи updateOne і updateMany

Метод updateOne схожий на метод update за тим винятком, що він оновлює тільки один документ.

**Приклад 14.70.** Замінити методом updateOne в першому знайденому документі за умовою {name: "Tom", age: 33} поле salary на 360, а якщо цього поля немає, тоді створити його (рис. 14.70):

```
> db.users.updateOne({name:"Tom", age:33},  
                    {$set:{salary: 360}})
```



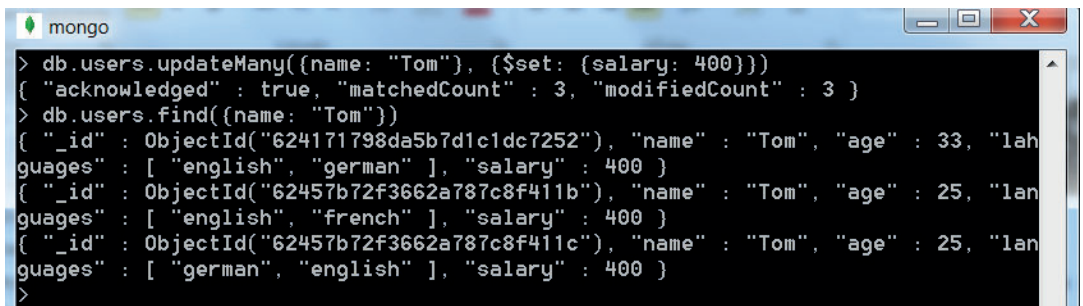
```
mongo  
> db.users.updateOne({name: "Tom", age: 33}, {$set: {salary: 360}})  
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }  
> db.users.find({name: "Tom"})  
{ "_id" : ObjectId("624171798da5b7d1c1dc7252"), "name" : "Tom", "age" : 33, "languages" : [ "english", "german" ], "salary" : 360 }  
{ "_id" : ObjectId("62457b72f3662a787c8f411b"), "name" : "Tom", "age" : 25, "languages" : [ "english", "french" ] }  
{ "_id" : ObjectId("62457b72f3662a787c8f411c"), "name" : "Tom", "age" : 25, "languages" : [ "german", "english" ] }
```

Рис.14.70. Приклад методу updateOne

Якщо ж необхідно оновити всі документи, що відповідають певному критерію, то застосовується метод updateMany().

**Приклад 14.71.** Оновити методом updateMany за умовою {name: "Tom"} поле salary на 400 у всіх документах колекції users (рис. 14.71):

```
> db.users.updateMany(  
    {name: "Tom"}, {$set: {salary: 400}})
```



```
mongo  
> db.users.updateMany({name: "Tom"}, {$set: {salary: 400}})  
{ "acknowledged" : true, "matchedCount" : 3, "modifiedCount" : 3 }  
> db.users.find({name: "Tom"})  
{ "_id" : ObjectId("624171798da5b7d1c1dc7252"), "name" : "Tom", "age" : 33, "languages" : [ "english", "german" ], "salary" : 400 }  
{ "_id" : ObjectId("62457b72f3662a787c8f411b"), "name" : "Tom", "age" : 25, "languages" : [ "english", "french" ], "salary" : 400 }  
{ "_id" : ObjectId("62457b72f3662a787c8f411c"), "name" : "Tom", "age" : 25, "languages" : [ "german", "english" ], "salary" : 400 }
```

Рис.14.71. Приклад методу updateMany



### 14.6.5. Оновлення масивів

#### Оператор \$push

Оператор \$push дозволяє додати ще одне значення в масив до вже існуючих.

Формат запису:

```
> db.ім'я_колекції.updateOne(  
    {query запит на вибірку – ключ : значення}  
    {$push: {ключ масиву : значення}})
```

*Приклад 14.72.* Додати до масиву languages ще одне значення "poland" у документ за умовою {name: "Tom", age: 33} (рис. 14.72):

```
> db.users.updateOne({name: "Tom", age: 33},  
    {$push: {languages: "poland"}})
```

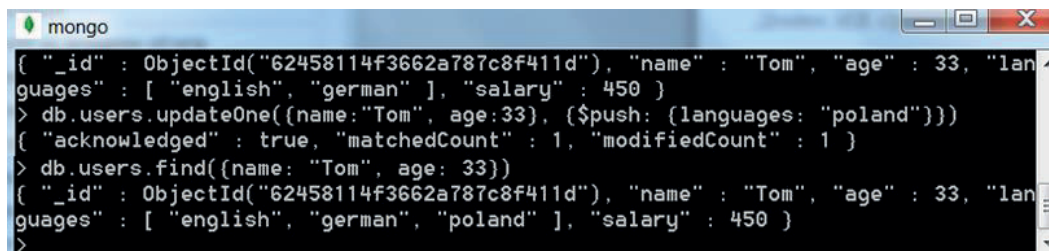


Рис.14.72. Оновлення масиву оператором updateOne

Якщо ключ, для якого додається значення, не є масивом, тоді виникає помилка Can not apply \$push / \$pushAll modifier to non-array.

Ще пара операторів дозволяє налаштувати вставку. Оператор \$position задає позицію в масиві для вставки елементів, а оператор \$slice вказує, скільки елементів залишити в масиві після вставки.

*Приклад 14.73.* Додати в масив languages з першої позиції три значення "german", "spanish" та "italian" та обмеживши кількість значень масиву до 5, для документів за умовою ({name: "Tom"}):

```
> db.users.update({name:"Tom"}, {$push: {languages: {$each:  
    ["german", "spanish", "italian"], $position: 1, $slice: 5}}})
```

В даному випадку вказані елементи вставляються в масив languages з 1-го індексу, і після вставки в масиві залишаються тільки 5 перших елементів.

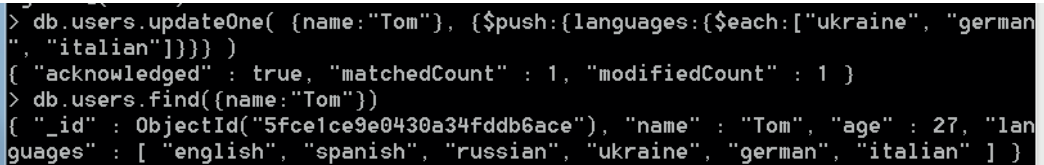


## Оператор \$each

Використовуючи оператор \$each, можна додати відразу кілька значень.

**Приклад 14.74:** Додати в кінець масиву languages три значення "ukraine", "german" та "italian" для документів за умовою ({name: "Tom"}) (рис. 17.73):

```
> db.users.update(  
  {name : "Tom"},  
  {$push: {languages:  
    {$each: ["ukraine", "german", "italian"]}  
  }})
```



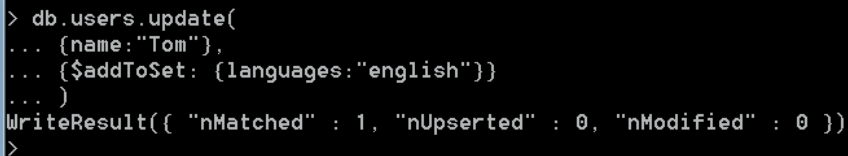
```
> db.users.updateOne( {name:"Tom"}, {$push:{languages:{$each:["ukraine", "german", "italian"]}} }  
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }  
> db.users.find({name:"Tom"})  
{ "_id" : ObjectId("5fce1ce9e0430a34fddb6ace"), "name" : "Tom", "age" : 27, "languages" : [ "english", "spanish", "russian", "ukraine", "german", "italian" ] }
```

Рис.14.73. Оновлення в масиві відразу декілька значень

## Оператор \$addToSet

Оператор \$addToSet подібно оператору \$push додає об'єкти в масив. Відмінність полягає в тому, що \$addToSet додає дані, якщо їх ще немає в масиві.

**Приклад 14.75.** Додамо до документу з name : "Tom" у масив languages ще одне значення, яке є в цьому масиві (рис. 17.74):



```
> db.users.update(  
... {name:"Tom"},  
... {$addToSet: {languages:"english"}}  
... )  
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 0 })  
>
```

Рис.14.74. Оператор \$addToSet для існуючих значень

Результатом буде повідомлення WriteResult, яке вкаже у параметрі модифікації nModified, що модифікація не відбулась (значення 0).

**Приклад 14.76.** А якщо ще такого значення ще немає, наприклад, languages : poland, тоді повідомлення WriteResult вкаже у параметрі модифікації nModified, що модифікація відбулась успішно (значення 1) (рис. 14.75):

```
> db.users.update( {name:"Tom"}, {$addToSet: {languages:"poland"}} )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
```

Рис.14.75. Оператор \$addToSet для відсутніх значень

## 14.7. Видалення даних

Для видалення документів в MongoDB передбачений метод `remove`:

```
> db.ім'я_колекції.remove(  
    {query запит на вибірку – ключ : значення},  
    options  
)
```

Параметр `options` не обов'язковий, за замовчуванням встановлений в значення `false`. Якщо ж цей параметр дорівнює `true`, то видаляється тільки перший елемент серед тих, що відповідають запиту на вибірку `query`, в іншому випадку – видаленню підлягають всі елементи, що відповідають умові.

**Приклад 14.77.** Видалити всі документи за умовою `{name : "Tom"}`:

```
> db.users.remove({name : "Tom"})
```

У підсумку всі знайдені документи з `name = Tom` будуть видалені. Причому, як і у випадку з `find`, можна задавати умови вибірки для видалення різними способами (у вигляді регулярних виразів, у вигляді умовних конструкцій і т.д.):

```
> db.users.remove({name : /T\w+/i})  
> db.users.remove({age: {$lt : 30}})
```

Щоб видалити разом всі документи з колекції, треба залишити порожнім параметр запиту:

```
> db.users.remove({})
```

### 14.7.1. Видалення поля

Для видалення окремого ключа (окремого поля) використовується конструкція `update` – оператор `$unset`:

```
> db.ім'я_колекції.update(  
    {query запит на вибірку – ключ : значення},
```

`{ $unset: { ім'я_ключа1 : 1, ім'я_ключа2 : 1, ... } }`

)

Якщо раптом подібного ключа в документі не існує, то оператор ігнорується.

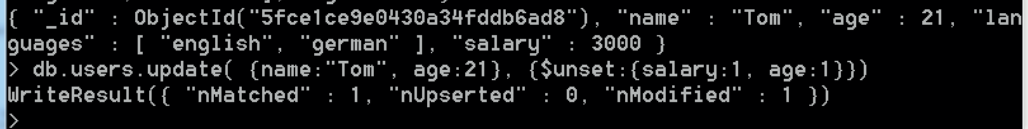
**Приклад 14.78.** Видалити з документів за умовою `{name : "Tom"}` поле salary:

```
> db.users.update(  
  {name : "Tom"},  
  { $unset: {salary: 1} })
```

Також можна видаляти відразу декілька полів:

**Приклад 14.79.** Видалити з документів за умовою `{name : "Tom"}` поля salary та age (рис. 14.76):

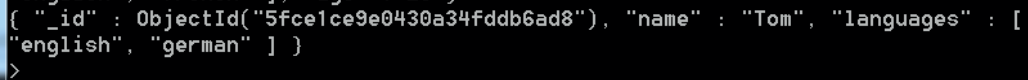
```
> db.users.update(  
  {name : "Tom", age : 21},  
  { $unset: {salary: 1, age: 1} })
```



```
{ "_id" : ObjectId("5fcea1ce9e0430a34fddb6ad8"), "name" : "Tom", "age" : 21, "languages" : [ "english", "german" ], "salary" : 3000 }  
> db.users.update( {name:"Tom", age:21}, { $unset:{salary:1, age:1} })  
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })  
>
```

*Рис.14.76. Видалення з документу відразу декілька полів*

В результаті в документі зазначені поля будуть видалені:



```
{ "_id" : ObjectId("5fcea1ce9e0430a34fddb6ad8"), "name" : "Tom", "languages" : [ "english", "german" ] }  
>
```

*Рис.14.77. Результат виконання прикладу 14.79*

### 14.7.2. Видалення елемента з масиву

Для видалення елементів з масиву використовуються конструкції:

- update – оператор `$pop`;
- update – оператор `$pull`;
- update – оператор `$pullAll`.

За допомогою конструкції update – оператор `$pop` видаляється один елемент з масиву за його номером, конструкція update – оператор `$pull` – видаляє кожне

входження елемента в масив за його значенням, а `$pullAll` – видаляє відразу декілька значень з масиву за його значенням.

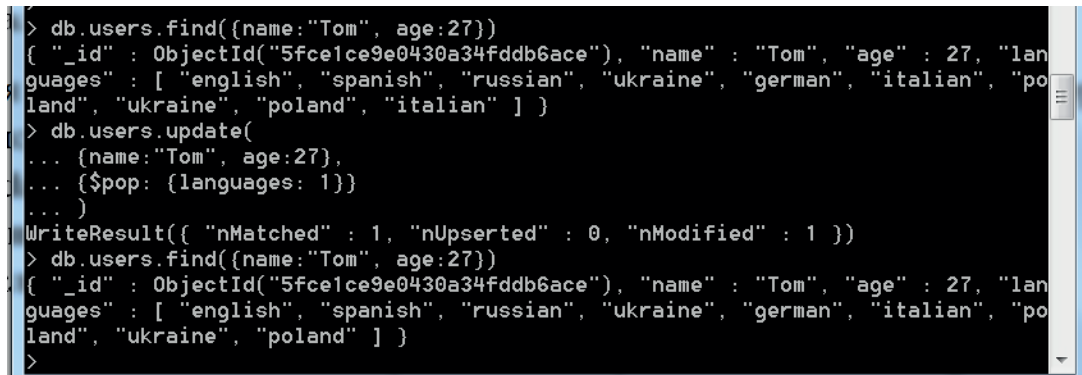
Формат запису update – оператор `$pop`:

```
> db.users.update(  
    {query запит на вибірку – ключ : значення },  
    {$pop: {ключ_масиву: номер_елемента масиву}}  
)
```

Якщо номер елемента масиву є додатнім числом, тоді видаляється вказаний елемент з кінця масиву.

**Приклад 14.80.** Видалити з масиву `languages` останнє значення (`italian`) для документу `{name : "Tom", age : 27}` (рис. 14.78). При цьому номер елемента масиву вказується додатнім:

```
> db.users.update(  
    {name : "Tom", age : 27},  
    {$pop: {languages: 1}}  
)
```



```
> db.users.find({name:"Tom", age:27})  
{ "_id" : ObjectId("5fce1ce9e0430a34fdbb6ace"), "name" : "Tom", "age" : 27, "languages" : [ "english", "spanish", "russian", "ukraine", "german", "italian", "poland", "ukraine", "poland", "italian" ] }  
> db.users.update(  
... {name:"Tom", age:27},  
... {$pop: {languages: 1}}  
... )  
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })  
> db.users.find({name:"Tom", age:27})  
{ "_id" : ObjectId("5fce1ce9e0430a34fdbb6ace"), "name" : "Tom", "age" : 27, "languages" : [ "english", "spanish", "russian", "ukraine", "german", "italian", "poland", "ukraine", "poland" ] }  
>
```

Рис.14.78. Видалення елемента з кінця масиву

Щоб вказати номер елемента з початку масиву, тоді число має бути від’ємним!

**Приклад 14.81.** Видалити з масиву `languages` перше значення (`english`) для документу `{name : "Tom", age : 27}` (рис. 14.79). При цьому номер елемента масиву вказується від’ємним:

```

> db.users.find({name:"Tom", age:27})
{ "_id" : ObjectId("5fce1ce9e0430a34fddb6ace"), "name" : "Tom", "age" : 27, "languages" : [ "english", "spanish", "russian", "ukraine", "german", "italian", "poland", "ukraine", "poland" ] }
> db.users.update( {name:"Tom", age:27}, {$pop: {languages: -1}} )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.users.find({name:"Tom", age:27})
{ "_id" : ObjectId("5fce1ce9e0430a34fddb6ace"), "name" : "Tom", "age" : 27, "languages" : [ "spanish", "russian", "ukraine", "german", "italian", "poland", "ukraine", "poland" ] }
>

```

Рис.14.79. Видалення елемента з початку масиву

Формат запису конструкції update – оператор \$pull аналогічний попередньому:

```

> db.users.update(
    {query запит на вибірку – ключ : значення },
    {$pull : {ключ_масиву: значення_елемента масиву}}
)

```

**Приклад 14.82.** Видалити всі входження в масив languages значення poland для документів з умовою {name : "Tom", age : 27} (рис. 14.80):

```

> db.users.update (
    {name : "Tom", age : 27},
    {$pull : {languages: "poland"}}
)

```

```

> db.users.find({name:"Tom", age:27})
{ "_id" : ObjectId("5fce1ce9e0430a34fddb6ace"), "name" : "Tom", "age" : 27, "languages" : [ "spanish", "russian", "ukraine", "german", "italian", "poland", "ukraine", "poland" ] }
> db.users.update(
... {name:"Tom", age:27},
... {$pull: {languages : "poland"}}
... )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.users.find({name:"Tom", age:27})
{ "_id" : ObjectId("5fce1ce9e0430a34fddb6ace"), "name" : "Tom", "age" : 27, "languages" : [ "spanish", "russian", "ukraine", "german", "italian", "ukraine" ] }
>

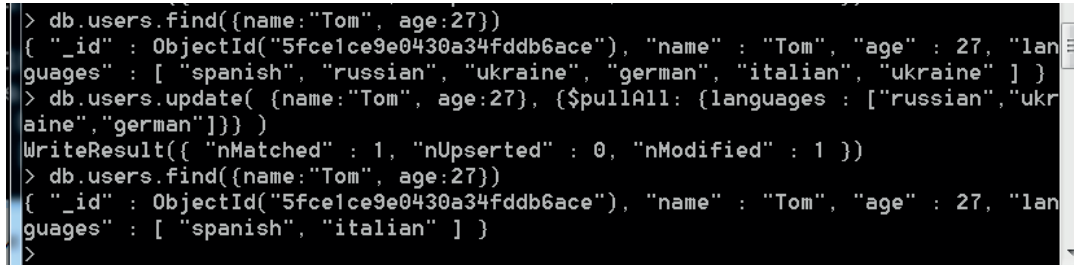
```

Рис.14.80. Видалення з масиву всіх входжень деякого значення

Для видалення відразу декілька значень застосовується оператор \$pullAll.

**Приклад 14.83.** Видалити з масиву `languages` відразу три значення `"russian", "ukraine"` та `"german"` для документів за умовою `{name : "Tom", age : 27}` (рис. 14.81):

```
db.users.update(  
  {name : "Tom", age : 27},  
  {$pullAll : {languages: "russian","ukraine","german"}}  
)
```



```
> db.users.find({name:"Tom", age:27})  
{ "_id" : ObjectId("5fce1ce9e0430a34fddb6ace"), "name" : "Tom", "age" : 27, "languages" : [ "spanish", "russian", "ukraine", "german", "italian", "ukraine" ] }  
> db.users.update( {name:"Tom", age:27}, {$pullAll: {languages : ["russian","ukraine","german"]}} )  
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })  
> db.users.find({name:"Tom", age:27})  
{ "_id" : ObjectId("5fce1ce9e0430a34fddb6ace"), "name" : "Tom", "age" : 27, "languages" : [ "spanish", "italian" ] }  
>
```

*Рис.14.81. Видалення з масиву відразу декілька значень*

І якщо в масиві вказаних значень декілька, то вони всі будуть видалені.

### 14.7.3. Видалення колекцій і баз даних

Ми можемо видаляти не тільки документи, а й колекції та бази даних. Для видалення колекцій використовується функція `drop`:

```
> db.<ім'я_колекції>.drop()
```

І якщо видалення колекції пройде успішно, то консоль виведе `true`.

Щоб видалити всю поточну базу даних, треба скористатися функцією `dropDatabase()`:

```
> db.dropDatabase()
```

## 14.8. Встановлення посилань в БД

У реляційних базах даних можна встановлювати зовнішні ключі, коли поля з однієї таблиці посилаються на поля в іншій таблиці. У MongoDB також можна встановлювати такі посилання.

### 14.8.1. Ручне встановлення посилань

Ручне встановлення посилань зводиться до присвоєння значення поля `_id` одного документа полю іншого документа.

**Приклад 14.84.** У колекції `companies` є документи з даними про компанії. Один з документів містить інформацію (рис. 14.82):

```
> db.companies.findOne()  
{ "_id" : 111, "name" : "microsoft", "year" : 1974 }  
>
```

Рис.14.82. Документ колекції `companies`

В іншій колекції, наприклад, `persons`, є дані про працівників (рис. 14.83).

```
> db.persons.find()  
{ "_id" : ObjectId("5fcf85300743c3fb845444ae"), "name" : "Tom", "age" : 27, "salary" : 3000, "company" : 111 }  
>
```

Рис.14.83. Документ колекції `persons`

Особливістю є те, що для працівників поле `companies` містить не дані про компанію, а ключ `_id` з документів колекції `companies`.

Для зв'язування таких документів з різних колекцій необхідна проміжна ланка – документ (рис. 14.84):

```
Tom: db.persons.findOne()  
{  
  "_id" : ObjectId("5fcf85300743c3fb845444ae"),  
  "name" : "Tom",  
  "age" : 27,  
  "salary" : 3000,  
  "company" : 111  
}
```

Рис.14.84. Документ `Tom`, що зв'язує документи колекцій `companies` та `persons`

І тепер знайдемо посилання на його компанію в колекції `companies` (рис. 14.85):

```
> db.companies.findOne({"_id": Tom.company})  
{ "_id" : 111, "name" : "microsoft", "year" : 1974 }  
>
```

Рис.14.85. Результат ручного зв'язування документів різних колекцій

## 14.8.2. Автоматичне зв'язування

Використовуючи функціональність DBRef, можна встановити автоматичне зв'язування між документами. Таке зв'язування здійснюється на етапі додавання документу в колекцію.

Формальний синтаксис DBRef наступний:

**ім'я\_ключа: new DBRef (ім'я\_колекції, ім'я\_документу.\_id)),**

де ім'я\_ключа – назва ключа в дочерньому документі, в якому буде прописаний зв'язок з іншим документом, ім'я\_колекції – назва колекції, де знаходиться батьківський документ, з яким встановлюється зв'язок, ім'я\_документу – ідентифікований батьківський документ з колекції, з яким встановлюється зв'язок.

*Приклад 14.85.* Зв'язати документ з даними про працівника steve з документом з даними про компанію apple за ключем компанії. Для цього додамо новий документ в колекцію companies:

```
> apple = ({ "_id" : 112, "name" : "apple", "year": 1976 })
> db.companies.save (apple)
```

І після збереження можемо вивести документ на консоль (рис. 14.86):

```
> apple = db.companies.find({name:"apple"})
{ "_id" : 112, "name" : "apple", "year" : 1976 }
```

Рис.14.86. Доданий документ в колекції companies

Тепер створимо новий документ для колекції person, у якого ключ company зв'яжемо з тільки що доданим документом apple:

```
> steve = ({ "name": "Steve", "age": 25,
             company: new DBRef( 'companies', apple._id) })
> db.persons.save (steve)
```

Після збереження виведемо документи колекції persons на консоль (рис. 14.87):

```
> db.persons.find()
{ "_id" : ObjectId("5fcf85300743c3fb845444ae"), "name" : "Tom", "age" : 27, "salary" : 3000, "company" : 111 }
{ "_id" : ObjectId("5fd1ec10de9ff403ced11106"), "name" : "Steve", "age" : 25, "company" : DBRef("companies", 112) }
>
```



*Рис.14.87. Зв'язаний документ steve колекції persons*

Для тестування необхідно ідентифікувати об'єкт steve як документ, а не додавати цей документ в БД за допомогою методу insert (рис. 14.88):

```
> steve
{
  "name" : "Steve",
  "age" : 25,
  "company" : DBRef("companies", 112),
  "_id" : ObjectId("5fd1eeadde9ff403ced11107")
}
```

*Рис.14.88. Ідентифікований документ steve*

При тестуванні в якості запиту на вибірку вказується вираз `_id: steve.company.$Id`, так як `person.company` є тепер об'єктом `new DBRef` (рис. 14.89):

```
> db.companies.findOne({_id: steve.company.$id})
```

```
> db.companies.findOne({"_id": steve.company.$id})
{ "_id" : 112, "name" : "apple", "year" : 1976 }
```

*Рис.14.89. Результат автоматичного зв'язування документів різних колекцій*

## 14.9. Робота з індексами

Пошук потрібних даних у великих об'ємах інформації може зайняти значний час, який може виявитися критичним для деякої задачі. Для підвищення швидкодії пошуку у базах даних використовують індекси, що дозволяють упорядкувати дані за певним полем, що згодом прискорить пошук.

**Індекс** (англ. index) – об'єкт бази даних, що створений зі значень одного чи декількох полів та вказівників на відповідні значення цих полів і має структуру, що оптимізована для пошуку, наприклад, збалансованого дерева.

MongoDB визначає індекси на рівні колекції та підтримує індекси за будь-яким полем або декількома полями.

На одну колекцію MongoDB дозволяє встановити до 64 індексів, а значення поля, за яким створюється індекс, не повинно перевищувати 1024 байт.

Для створення індексу використовується метод `createIndex()`, що має наступний формат запису:

```
db . <колекція> . createIndex (
    {<ключ_1>:<напрямок>,
    <ключ_N>:<напрямок>},
    {<параметри >})
```

Назва за замовчуванням для індексу – об'єднання назви поля та напрямку кожного ключа в індексі (тобто 1 за зростанням або -1 за спаданням ) з використанням підкреслення як роздільника, як `item_1`, `name_-1` і т.д.

Параметри необов'язковий аргумент, в якому може задаватися власна назва ключа, а не за замовчуванням, (наприклад, { `name: "myIndex"`}), унікальність значень ключа {`"unique" : true`} тощо.

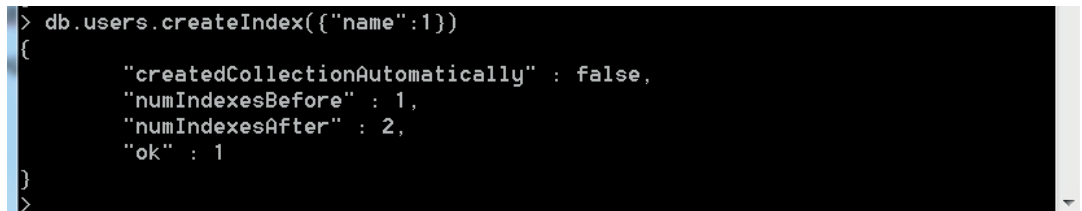
Задавання унікальності індексу за деяким полем забезпечує неможливість додавання у колекцію двох документів з однаковим значенням цього поля.

Крім того, під час створення колекції MongoDB автоматично створює унікальний індекс у полі `_id`, що надає захист від додавання у колекцію двох документів з однаковим `_id`. Цей індекс за полем `_id` заблокований і видалити його не можливо.

Всі решту додані індекси після створення модифікації не підлягають, тобто їх не можна змінити, в тому числі не можна і перейменувати! Натомість потрібно видалити необхідний індекс та заново створити його з новою назвою.

**Приклад 14.86.** Створити індекс для колекції `users` за полем `name` (рис. 14.90):

```
> db.users.createIndex({"name" : 1})
```



```
> db.users.createIndex({"name":1})
{
  "createdCollectionAutomatically" : false,
  "numIndexesBefore" : 1,
  "numIndexesAfter" : 2,
  "ok" : 1
}
```

Рис.14.90. Приклад створення індексу за одним полем

**Приклад 14.87.** Створити індекс для поточної БД і колекції users за полями name та age (рис. 14.91):

```
> db.users.createIndex({name : 1, age : -1})
```

```
> db.users.createIndex({name : 1, age : -1})
{
  "createdCollectionAutomatically" : false,
  "numIndexesBefore" : 1,
  "numIndexesAfter" : 2,
  "ok" : 1
}
```

Рис.14.91. Приклад створення індексу за двома полями: name – за зростанням, age – за спаданням

Для перегляду інформації про індекси колекції використовується метод getIndexes, що має наступний формат запису:

```
db.<колекція>.getIndexes()
```

Дана команда поверне результат на зразок (рис. 14.92):

```
> db.users.getIndexes()
[
  {
    "v" : 2,
    "key" : {
      "_id" : 1
    },
    "name" : "_id_",
    "ns" : "info.users"
  },
  {
    "v" : 2,
    "key" : {
      "name" : 1
    },
    "name" : "name_1",
    "ns" : "info.users"
  },
  {
    "v" : 2,
    "key" : {
      "name" : 1,
      "age" : -1
    }
  }
]
```

Рис.14.92. Перегляд інформації про індекси колекції методом getIndexes()

У даному прикладі для колекції users з БД info визначено 3 індекси: \_id\_, name\_1 та name\_1\_age\_-1. Поле key використовується для пошуку максимального і мінімального значень, а також для різних операцій, де треба застосовувати індекси.

Перший індекс `_id` був створений автоматично при створенні першого документа колекції і видаленню та модифікації не підлягає. Наступні два індекси створені у попередніх прикладах, їх також тепер не можна змінювати, а лише видаляти та створювати заново з іншою назвою або тією ж та заданим порядком сортування.

**Приклад 14.88.** Створити індекс за полем `languages` за зростанням з іменем `lang` (рис. 14.93):

```
> db.users.createIndex({languages: 1}, {name:"lang"})
{
  "createdCollectionAutomatically" : false,
  "numIndexesBefore" : 3,
  "numIndexesAfter" : 4,
  "ok" : 1
}
```

*Рис.14.93. Створення індексу з заданим іменем*

Тепер серед індексів колекції `users` буде також і створений індекс `lang` (рис. 14.94):

```
{
  "v" : 2,
  "key" : {
    "languages" : 1
  },
  "name" : "lang",
  "ns" : "info.users"
}
```

*Рис.14.94. Вивід інформації про індекси колекції методом `getIndexes()`*

Для видалення індексу з колекції використовується метод `dropIndex`:

**db . <колекція> . dropIndex(<ім'я\_індексу>)**

**Приклад 14.89.** Видалити створений індекс `lang` (рис. 14.95):

```
> db.users.dropIndex("lang")
{ "nIndexesWas" : 4, "ok" : 1 }
```

*Рис.14.95. Приклад видалення індексу*

## 14.10. Керування колекцією

### Явне створення колекції

У попередніх прикладах колекція створювалася неявно – автоматично при додаванні в неї перших даних. Але колекцію також можна створити явно, застосувавши метод

**db.createCollection(name, options),**

де name – назва колекції, а options – необов'язковий об'єкт для створення обмеженої колекції або для попереднього розподілу місця в новій колекції.

*Приклад 14.90.* Створити колекцію з іменем test:

```
> db.createCollection("test")
{"ok" : 1}
```

Таким чином, створюється колекція test.

### Перейменування колекції

В процесі роботи, можливо, буде потрібно змінити назву колекції. Наприклад, якщо при першому додаванні даних у її назві була помилка. І щоб не видаляти і потім перестворювати колекцію, слід використовувати функцію renameCollection:

```
> db.<колекція>.renameCollection("нова_назва")
```

*Приклад 14.91.* Перейменувати колекцію test на Proba:

```
> db.test.renameCollection("Proba")
{ "ok" : 1 }
>
```

*Рис.14.96. Приклад перейменування колекції*

### Обмеження на колекції

В MongoDB існує поняття обмеженої колекції (capped collection). Обмежені колекції мають фіксований розмір за:

- кількістю документів;
- за розміром.

Подібна колекція гарантує, що документи будуть розташовуватися в тому ж порядку, в якому вони додавалися в колекцію. І коли в колекції вже немає місця, найбільш старі документи видаляються, і в кінець додаються нові дані.

Обмеження задаються методом `createCollection` в аргументі `options`:

**`db.createCollection(name, options),`**

де аргумент `options` визначається:

`capped : < boolean > ,`

`size : < number > ,`

`max : < number >`

На відміну від звичайних колекцій, обмежені можна задавати явно.

**Приклад 14.92.** Створити обмежену колекцію з назвою `profile` і задати для неї розмір в 9500 байт:

```
> db.createCollection("profile", {capped:true, size:9500})
```

І після вдалого створення колекції консоль виведе:

```
{"ok":1}
```

**Приклад 14.93.** Також можна обмежити кількість документів в колекції, вказавши його в параметрі `max`:

```
> db.createCollection("profile", {capped:true, size:9500, max: 150})
```

Однак при такому способі створення колекції слід враховувати, що якщо все місце під колекцію заповнене (наприклад, виділені нами 9500 байтів), а кількість документів ще не досягло максимуму, в даному випадку 150, то в цьому випадку при додаванні нового документа найстаріший документ буде видалятися, а на його місце буде вставлятися новий документ.

При оновленні документів в таких колекціях треба пам'ятати, що документи не повинні рости в розмірах, інакше оновлення не вдасться зробити.

Також не можна видаляти документи з подібних колекцій, можна тільки видалити всю колекцію.

## Підколекції

Для спрощення організації даних в колекціях можна використовувати підколекції. Наприклад, дані по колекції `users` треба розмежувати на профілі і облікові дані. І в цьому випадку можна створити колекції `db.users.profiles` і

db.users.accounts. При цьому вони не будуть жодним чином пов'язані з колекцією users. Тобто в підсумку будуть три різні колекції, проте в плані логічної організації зберігання даних, можливо, для когось так буде простіше.

### Видалення колекції

Для видалення колекцій в MongoDB використовується наступна команда:

**db.<ім'я\_колекції>.drop()**

**Приклад 14.94.** Видалити колекцію Proba з поточної БД (рис. 14.97):

```
> db.test.drop()  
true  
> show collections  
companies  
persons  
users  
>
```

*Рис.14.97. Приклад видалення колекції*

## 14.11. Резервне копіювання і відновлення БД MongoDB

Резервне копіювання та відновлення MongoDB є важливою частиною для обробки бази даних. Цей функціонал має важливе значення для надійного забезпечення збереження даних в БД і їх відновлення у випадку пошкодження або випадкового видалення даних, пошкодження сервера, локального комп'ютера або будь-якої помилки, якою б вона не була.

### 14.11.1. Резервне копіювання БД

Для резервного копіювання бази даних MongoDB використовується утиліта **mongodump**, яка може створити резервну копію для всього сервера, конкретної бази даних, колекції або частини колекції.

Формат запису утиліти **mongodump**:

**>mongodump [-h dbhost][:port] [-d dbname] [-o dbdirectory],**

де:

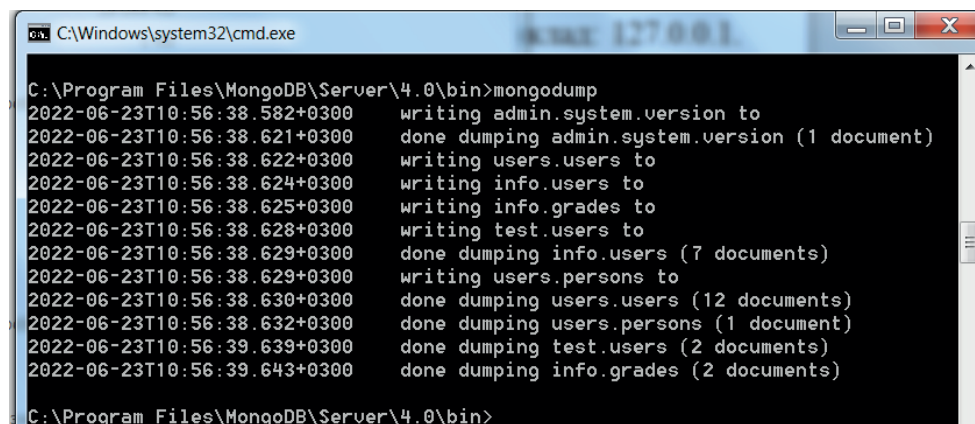
- **[-h dbhost][:port]** – адреса сервера, на якому знаходиться MongoDB, наприклад: 127.0.0.1. Можна також вказати номер порту: 127.0.0.1:27017;

- [-d dbname] – примірник бази даних для резервного копіювання, наприклад: test;
- [-o dbdirectory] – місце зберігання даних резервного копіювання, наприклад: c:\data\dump, звичайно, цей каталог необхідно попередньо створити.

**Увага!** Утиліта mongodump перезаписує файли копій, якщо вони вже існують.

Коли mongodump працює лише без будь-яких аргументів, команда підключається до екземпляра MongoDB у локальній системі (наприклад, localhost) через порт 27017 та створює резервну копію бази даних у поточному каталозі C:\Program Files\MongoDB\...\bin\dump\. Папку dump за вказаним шляхом попередньо треба створити.

**Приклад 14.95.** Створити резервну копію всієї локальної БД MongoDB, використавши утиліту mongodump без аргументів. Для цього за шляхом, де знаходиться сервер MongoDB, в даному прикладі – C:\Program Files\MongoDB\Server\4.0\bin\, необхідно створити папку dump, далі у командному рядку зайти у папку bin MongoDB та запустити на виконання утиліту mongodump (рис. 14.98).



```

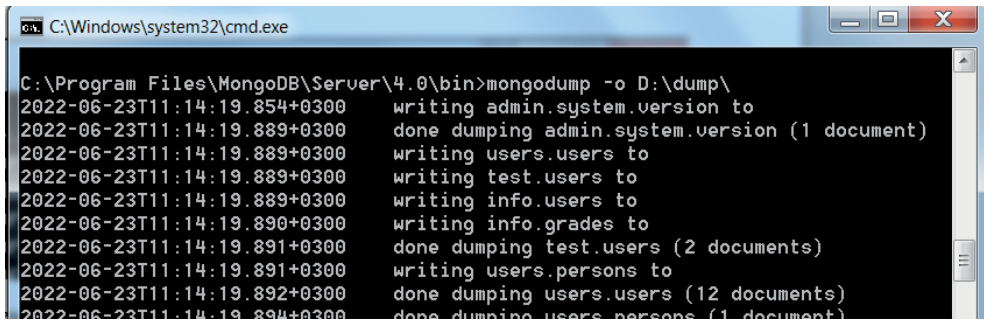
C:\Windows\system32\cmd.exe
C:\Program Files\MongoDB\Server\4.0\bin>mongodump
2022-06-23T10:56:38.582+0300   writing admin.system.version to
2022-06-23T10:56:38.621+0300   done dumping admin.system.version (1 document)
2022-06-23T10:56:38.622+0300   writing users.users to
2022-06-23T10:56:38.624+0300   writing info.users to
2022-06-23T10:56:38.625+0300   writing info.grades to
2022-06-23T10:56:38.628+0300   writing test.users to
2022-06-23T10:56:38.629+0300   done dumping info.users (7 documents)
2022-06-23T10:56:38.629+0300   writing users.persons to
2022-06-23T10:56:38.630+0300   done dumping users.users (12 documents)
2022-06-23T10:56:38.632+0300   done dumping users.persons (1 document)
2022-06-23T10:56:39.639+0300   done dumping test.users (2 documents)
2022-06-23T10:56:39.643+0300   done dumping info.grades (2 documents)
C:\Program Files\MongoDB\Server\4.0\bin>

```

Рис.14.98. Створення резервної копії БД MongoDB за замовчуванням

**Приклад 14.96.** Створити резервну копію всієї локальної БД MongoDB за шляхом D:\dump, використавши утиліту mongodump. Як і в попередньому прикладі, папка dump повинна існувати (рис. 14.99).

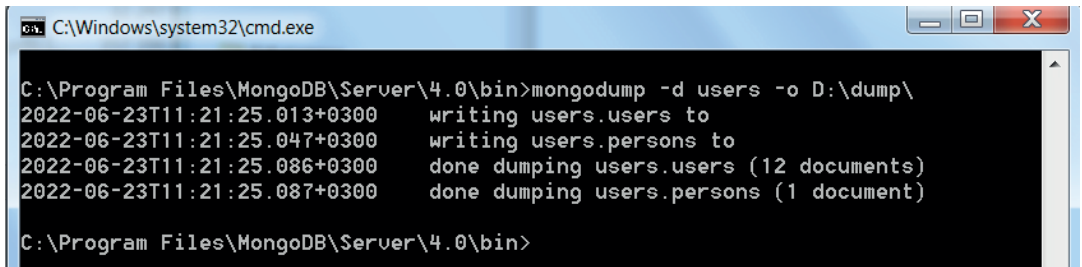




```
C:\Windows\system32\cmd.exe
C:\Program Files\MongoDB\Server\4.0\bin>mongodump -o D:\dump\
2022-06-23T11:14:19.854+0300   writing admin.system.version to
2022-06-23T11:14:19.889+0300   done dumping admin.system.version (1 document)
2022-06-23T11:14:19.889+0300   writing users.users to
2022-06-23T11:14:19.889+0300   writing test.users to
2022-06-23T11:14:19.889+0300   writing info.users to
2022-06-23T11:14:19.890+0300   writing info.grades to
2022-06-23T11:14:19.891+0300   done dumping test.users (2 documents)
2022-06-23T11:14:19.891+0300   writing users.persons to
2022-06-23T11:14:19.892+0300   done dumping users.users (12 documents)
2022-06-23T11:14:19.894+0300   done dumping users.persons (1 document)
```

Рис.14.99. Створення резервної копії БД MongoDB за вказаним шляхом

**Приклад 14.97.** Створити резервну копію локальної БД users MongoDB за шляхом D:\dump, використавши утиліту mongodump (рис. 14.100).



```
C:\Windows\system32\cmd.exe
C:\Program Files\MongoDB\Server\4.0\bin>mongodump -d users -o D:\dump\
2022-06-23T11:21:25.013+0300   writing users.users to
2022-06-23T11:21:25.047+0300   writing users.persons to
2022-06-23T11:21:25.086+0300   done dumping users.users (12 documents)
2022-06-23T11:21:25.087+0300   done dumping users.persons (1 document)
C:\Program Files\MongoDB\Server\4.0\bin>
```

Рис.14.100. Створення резервної копії БД users за вказаним шляхом

В результаті буде створено копію тільки БД users, яка містить 12 документів у колекції users та 1 документ колекції persons (рис. 14.101).

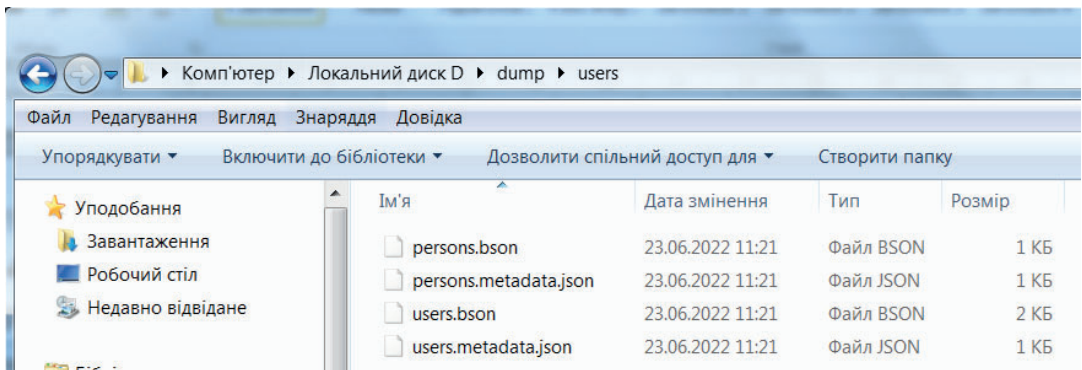
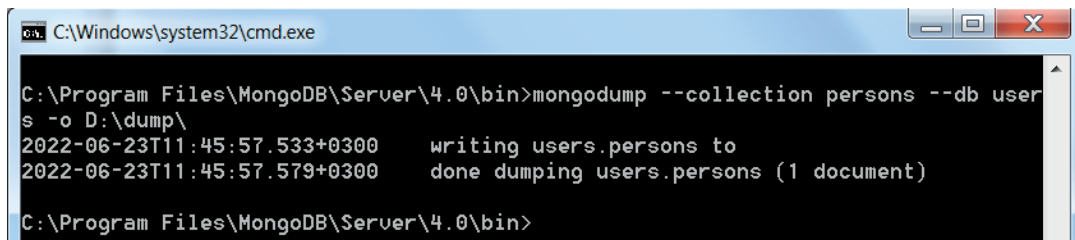


Рис.14.101. Результат створення резервної копії БД users за вказаним шляхом

**Приклад 14.98.** Створити резервну копію локальної БД users MongoDB тільки колекції persons за шляхом D:\dump, використавши утиліту mongodump (рис. 14.102).



```
C:\Windows\system32\cmd.exe

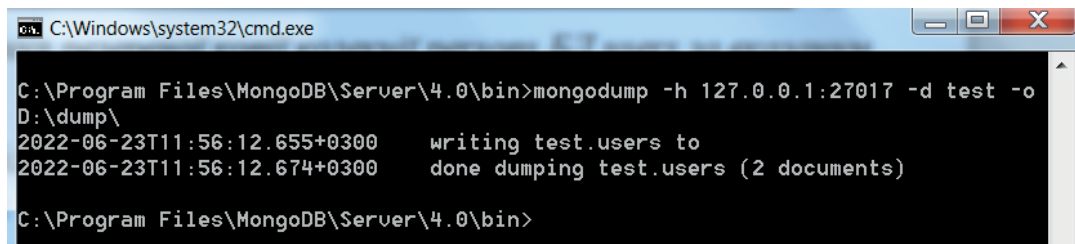
C:\Program Files\MongoDB\Server\4.0\bin>mongodump --collection persons --db users
s -o D:\dump\
2022-06-23T11:45:57.533+0300      writing users.persons to
2022-06-23T11:45:57.579+0300      done dumping users.persons (1 document)
C:\Program Files\MongoDB\Server\4.0\bin>
```

*Рис.14.102. Результат створення резервної копії колекції persons БД users за вказаним шляхом*

У даному прикладі необхідно звернути увагу на синтаксис: в якості параметрів колекції використовується запис `--collection <колекція>`, а БД `--db <база даних>`.

Звичайно, всі дії можна виконувати для БД за вказаними адресою сервера та портом, будь БД локальною чи віддаленою.

**Приклад 14.99.** Створити резервну копію БД test MongoDB з адресою сервера 127.0.0.1 та портом 27017 за шляхом D:\dump, використавши утиліту mongodump (рис. 14.103).



```
C:\Windows\system32\cmd.exe

C:\Program Files\MongoDB\Server\4.0\bin>mongodump -h 127.0.0.1:27017 -d test -o
D:\dump\
2022-06-23T11:56:12.655+0300      writing test.users to
2022-06-23T11:56:12.674+0300      done dumping test.users (2 documents)
C:\Program Files\MongoDB\Server\4.0\bin>
```

*Рис.14.103. Створення резервної копії БД за вказаними адресою сервера та портом*

### 14.11.2. Відновлення даних

Для відновлення БД MongoDB з резервних копій використовується утиліта mongorestore, яка відновлює резервну копію, створену mongodump. Mongorestore може відновити як повністю резервну копію бази даних, так і її частину.

Формат запису утиліти mongorestore:

`>mongodump [-h dbhost][:port] [-d dbname] [dbdirectory],`

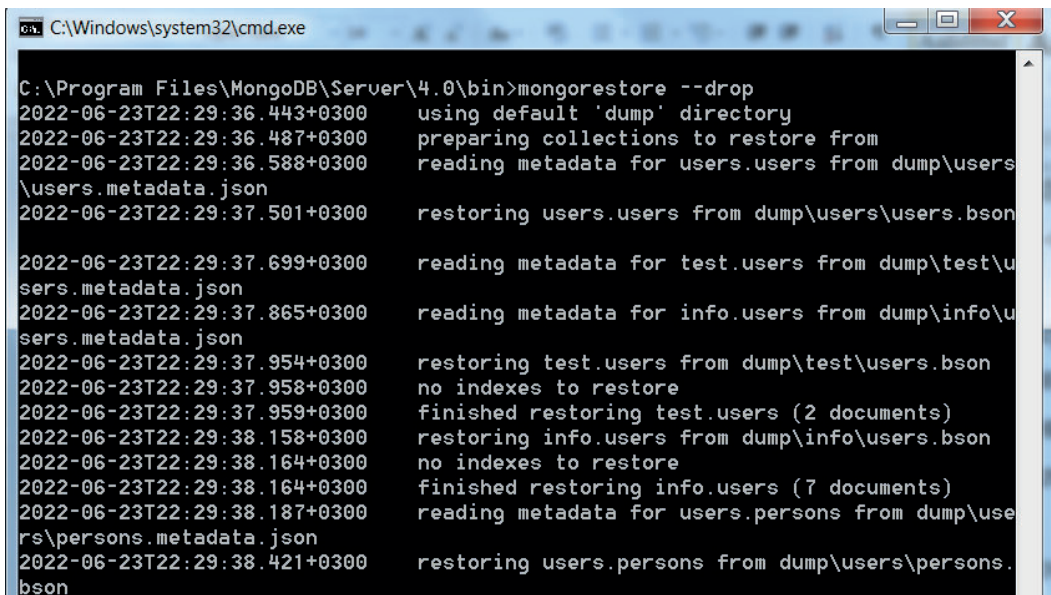
де:

- [-h dbhost][:port] – адреса сервера, на якому знаходиться MongoDB, наприклад: 127.0.0.1. Можна також вказати номер порту: 127.0.0.1:27017;
- [-d dbname] – ім'я БД, яку відновлюємо;
- [dbdirectory] – шлях, за яким знаходиться дамп БД.

За замовчуванням mongorestore не перезаписує і не видаляє будь-які наявні документи під час відновлення, тому необхідно додатково вказати опцію drop, наприклад:

```
mongorestore --drop my-backup
```

**Приклад 14.100.** Відновити всі БД з резервної копії за замовчуванням (рис. 14.104).

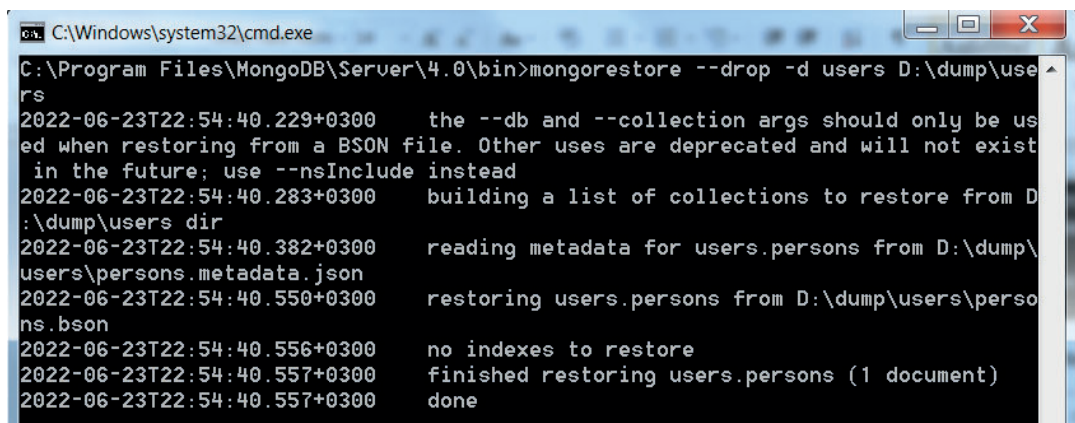


```
C:\Windows\system32\cmd.exe
C:\Program Files\MongoDB\Server\4.0\bin>mongorestore --drop
2022-06-23T22:29:36.443+0300 using default 'dump' directory
2022-06-23T22:29:36.487+0300 preparing collections to restore from
2022-06-23T22:29:36.588+0300 reading metadata for users.users from dump\users
\users.metadata.json
2022-06-23T22:29:37.501+0300 restoring users.users from dump\users\users.bson
2022-06-23T22:29:37.699+0300 reading metadata for test.users from dump\test\users
sers.metadata.json
2022-06-23T22:29:37.865+0300 reading metadata for info.users from dump\info\users
sers.metadata.json
2022-06-23T22:29:37.954+0300 restoring test.users from dump\test\users.bson
2022-06-23T22:29:37.958+0300 no indexes to restore
2022-06-23T22:29:37.959+0300 finished restoring test.users (2 documents)
2022-06-23T22:29:38.158+0300 restoring info.users from dump\info\users.bson
2022-06-23T22:29:38.164+0300 no indexes to restore
2022-06-23T22:29:38.164+0300 finished restoring info.users (7 documents)
2022-06-23T22:29:38.187+0300 reading metadata for users.persons from dump\users
rs\persons.metadata.json
2022-06-23T22:29:38.421+0300 restoring users.persons from dump\users\persons.
bson
```

*Рис.14.104. Результат відновлення всіх БД MongoDB з резервної копії*

Для відновлення конкретної БД, а тим паче колекції, необхідно вказувати, що відновлювати, і звідки відновлювати.

**Приклад 14.101.** Відновити БД users з резервної копії за шляхом D:\dump\users (рис. 14.105).



```
C:\Windows\system32\cmd.exe
C:\Program Files\MongoDB\Server\4.0\bin>mongorestore --drop -d users D:\dump\users
2022-06-23T22:54:40.229+0300 the --db and --collection args should only be used
when restoring from a BSON file. Other uses are deprecated and will not exist
in the future; use --nsInclude instead
2022-06-23T22:54:40.283+0300 building a list of collections to restore from D:\dump\users dir
2022-06-23T22:54:40.382+0300 reading metadata for users.persons from D:\dump\users\persons.metadata.json
2022-06-23T22:54:40.550+0300 restoring users.persons from D:\dump\users\persons.bson
2022-06-23T22:54:40.556+0300 no indexes to restore
2022-06-23T22:54:40.557+0300 finished restoring users.persons (1 document)
2022-06-23T22:54:40.557+0300 done
```

*Рис.14.105. Результат відновлення однієї БД users*

Утиліта `mongorestore` за замовчуванням підключається до екземпляра MongoDB, що працює на інтерфейсі `localhost` з портом за замовчуванням (27017). Щоб відновити резервну копію на інший хост або порт, використовуються опції `--host` та `--port`:

```
mongorestore --host [host name] --port [port name] --username [user]
               --password [password] [backup folder path of local machine]
```

Крім `mongodump` та `mongorestore` аналогічні за функціоналом в MongoDB є також утиліти `mongoimport` та `mongoexport`, які надають змогу працювати з форматами файлів дампу типу CSV та JSON.

## Контрольні запитання

1. Чи підтримує 32-розрядні ОС MongoDB?
2. Яку папку і за яким шляхом необхідно створити на жорсткому диску після установки MongoDB для зберігання баз даних за замовчуванням?
3. Як змінити шлях за замовчуванням до файлів БД?
4. Чи можна запустити консоль MongoDB без сервера?
5. Як встановити поточну БД в MongoDB? Якщо такої БД ще немає в MongoDB, що відбудеться?
6. Як вивести назви всіх БД на консоль?
7. Як переглянути список всіх колекцій в поточній БД?
8. Як представляється документ в MongoDB?

9. Які використовуються методи для додавання даних в колекцію MongoDB?
10. Яка команда показує вмісту документів колекції MongoDB?
11. В якому форматі записуються обмеження для функції find?
12. Чим відрізняється вибірка від проекції?
13. Який формат запису запиту find без обмежень?
14. Яка команда сортує дані в MongoDB? А яка в операторі SELECT мови SQL?
15. Для чого призначена функція limit в MongoDB?
16. Як у вибірці find пропустити певну кількість документів?
17. Що таке агрегація в MongoDB і які функції в ній застосовуються?
18. Які відмінності між умовними операторами в MongoDB та SQL?
19. Які оператори MongoDB призначені для пошуку в масивах?
20. В чому відмінності методу save та функції update в MongoDB?
21. Як в MongoDB оновити окреме поле?
22. Який передбачений метод для видалення документів в MongoDB?
23. Як видалити окреме поле документа?
24. Які конструкції використовуються для видалення елементів з масиву?
25. Яка функція видаляє колекцію або БД? Наведіть приклади.

# Тема 15. Графічний клієнт MongoDB Compass

## 15.1. Основні відомості про Compass

MongoDB Compass – це офіційний графічний інтерфейс для MongoDB, який допомагає користувачам приймати розумні рішення щодо структури даних, запитів, індексації та багатьох інших дій, які можна виконувати з базою даних.

MongoDB Compass може виконувати всі операції, які виконує Mongo Shell та багато іншого, включаючи:

- візуалізацію та дослідження даних, що зберігаються у базі даних;
- створення баз даних, додавання, оновлення та видалення даних у базі даних;
- отримання поточної статистики сервера в режимі реального часу;
- розуміння проблем продуктивності за допомогою візуальних планів пояснення;
- керування індексами;
- перевірка даних за допомогою правил перевірки схем JSON;
- розширення за допомогою плагінів.

Важливо зазначити, що MongoDB Compass випускається у трьох видах:

- **Compass** має всі можливості та можливості MongoDB Compass.
- **Compass** **лише** для читання обмежених операціями зчитування. У ньому не можна редагувати, додавати або видаляти.
- **Compass Isolated** не передбачає жодних запитів щодо мережі.

Здебільшого потрібна повна версія. MongoDB підтримує всі основні операційні системи, такі як:

- Windows;
- Ubuntu;
- OS X;
- RedHat.

Графічний клієнт Compass можна завантажити з офіційного сайту <https://www.mongodb.com/try/download/compass>, де попередньо необхідно обрати версію продукту, платформу (операційну систему) та тип архіву (рис. 15.1).

Завантажений архів у разі обрання типу архіву zip необхідно розпакувати в папку mongodb-compass за шляхом, де знаходиться власне сервер mongodb, у нашому випадку – C:\Program Files\MongoDB\.

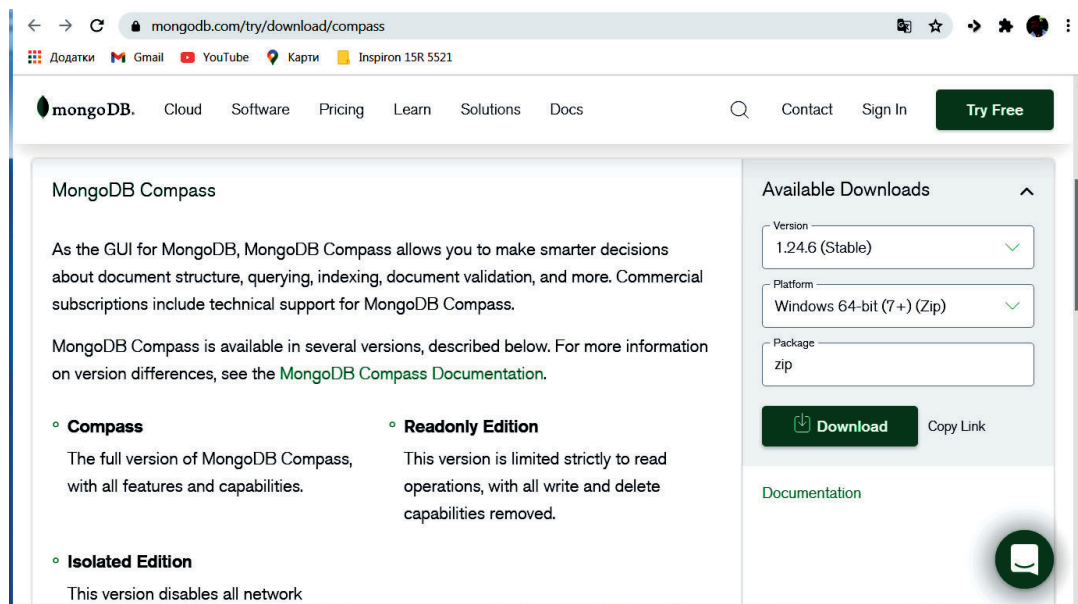


Рис. 15.1. Офіційний сайт MongoDB Compass

Для інсталяції графічного клієнта Compass необхідно запустити на виконання файл MongoDBCompass.exe з папки mongodb-compass та покроково, натискаючи Next, інсталювати його.

Під час запуску потрібно ввести ряд даних у рядок з'єднання для підключення до сервера:

- **Hostname:** хост сервера;
- **Port:** порт, по якому запущено сервер;
- **SRV Record:** специфікація, яка визначає розташування сервера (наприклад, адресу та порт);
- **Authentication:** тип аутентифікації. Тут є наступні варіанти:
  - **None;**
  - **Username / Password;**



- **X.509;**
- **Kerberos;**
- **LDAP;**
- **Replica Set Name:** назва MongoDB, до якої здійснюється підключення;
- **Read Preference:** визначає, яким чином Compass керує операціями читання. Може приймати наступні опції: Primary, Primary Preferred, Secondary, Secondary Preferred и Nearest;
- **SSL:** вказує, чи буде використовуватися захищене підключення;
- **SSH tunnel:** чи доцільно підключатися до кластера MongoDB через тунель SSH;
- **Favorite Name:** встановлює ім'я підключення.

## 15.2. Підключення Compass до сервера MongoDB

Для роботи з Compass MongoDB необхідне підключення до сервера.

Відповідно, сервер повинен бути *запущений*!

Рядок з'єднання з сервером повинен містити такі основні компоненти:

- ім'я хоста Hostname:порт Port;
- аутентифікація Authentication, включає назву БД та пароль;
- переваги читання Read Preference;
- використання SSH тунелю.

Для створення рядка з'єднання можна скористатись стандартними пропозиціями або записати власний рядок.

Якщо MongoDB встановлено на локальному комп'ютері з налаштуваннями за замовчуванням, то Hostname буде локальний сервер localhost, а порт 27017 (рис. 15.2).

Для створення нового рядка з'єднання необхідно у вкладці New Connection обрати команду Fill in connection fields individually (заповніть поля підключення окремо), та, власне, після їх заповнення, згенерувати новий рядок з'єднання (Paste connection string) у вкладках Hostname (рис. 15.3) та More Options (рис. 15.4).

Якщо вхід в MongoDB Compass не в перший раз, можна скористатися обраними з'єднаннями (Favorites) або останніми успішними (Recents).



Для прикладу, рядок підключення до локального сервера матиме вигляд:  
`mongodb://localhost:27017/?readPreference=primary&appname=MongoDB%20Compass&ssl=false`.

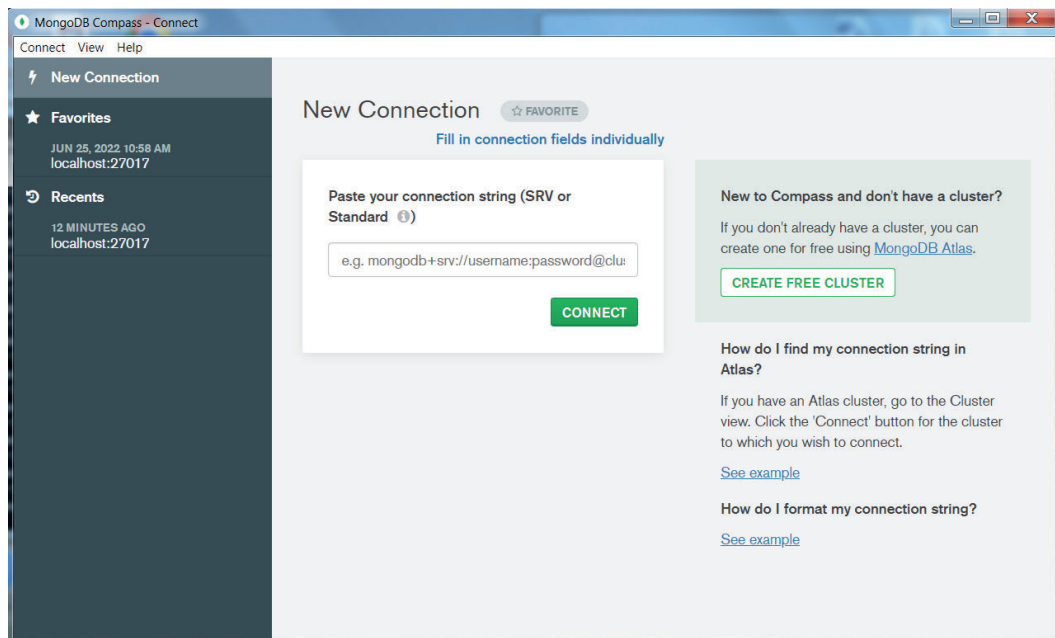


Рис. 15.2. Створення нового з'єднання з БД MongoDB у Compass

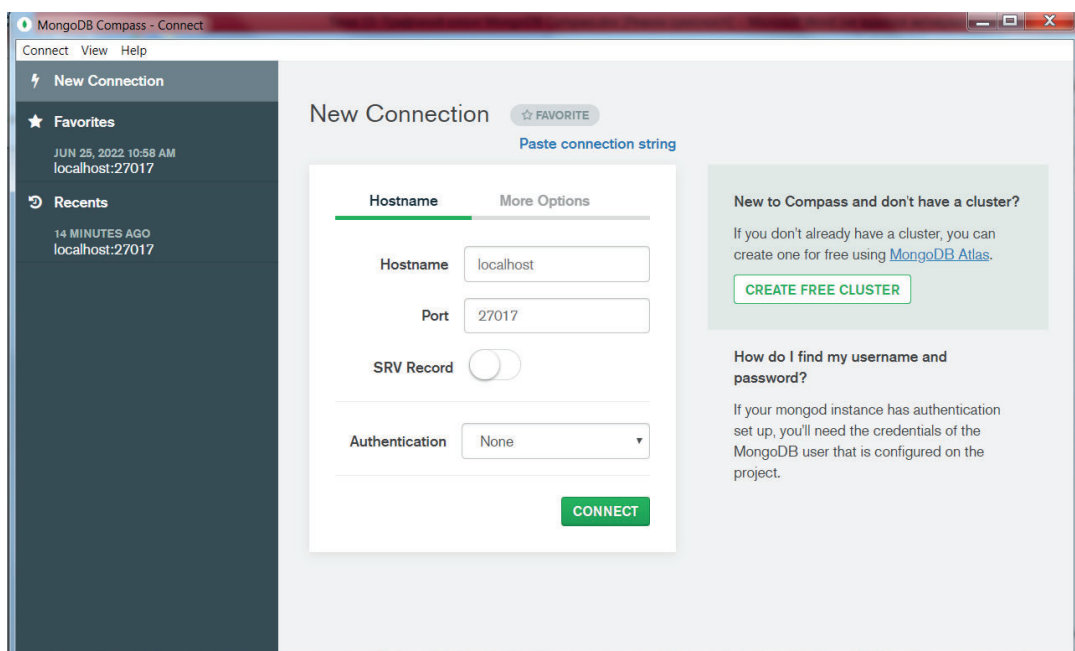


Рис. 15.3. Створення нового з'єднання – вкладка Hostname

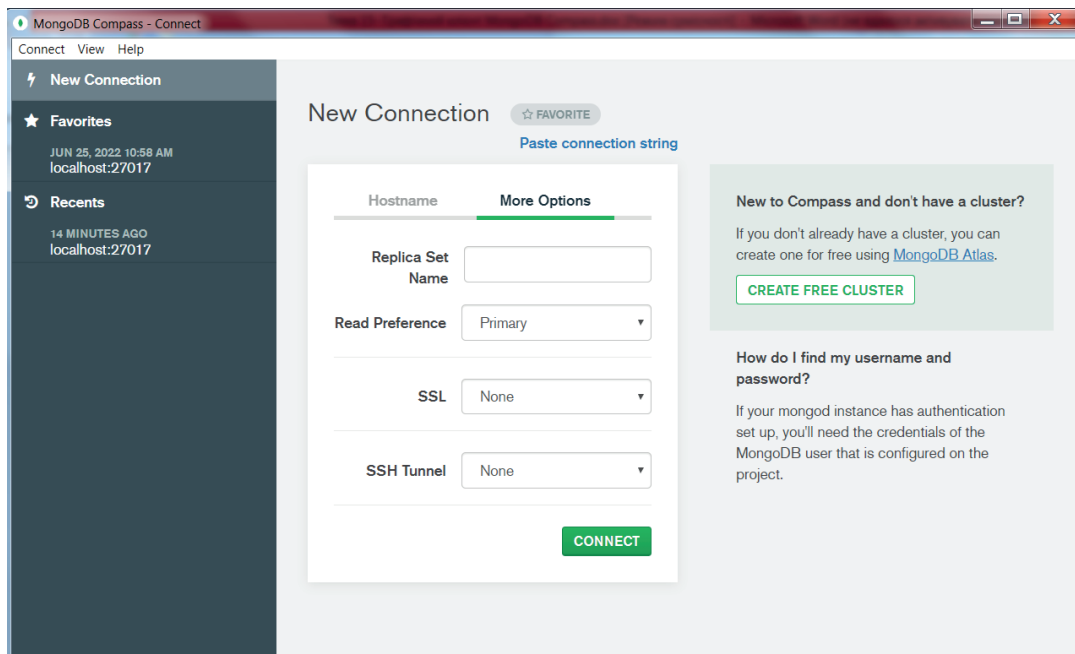


Рис. 15.4. Створення нового з'єднання – вкладка *More Options*

Після успішного підключення до БД MongoDB відкриється список баз даних, які є на сервері (рис. 15.5):

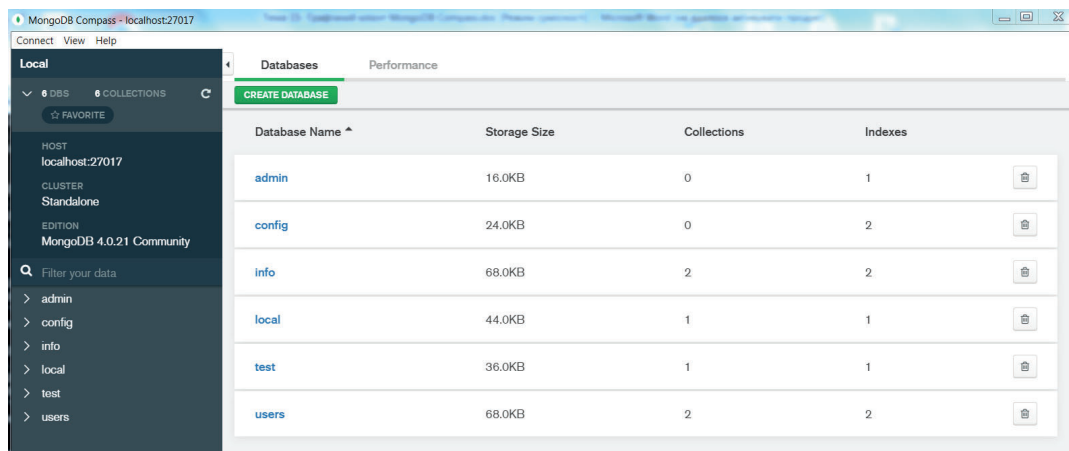


Рис. 15.5. Інтерфейс MongoDB Compass

Використовуючи графічний інтерфейс програми Compass, можна керувати даними – переглядати, додавати, змінювати та видаляти їх.

### 15.3. Створення баз даних та колекцій

Якщо після успішного підключення до БД MongoDB потрібної БД немає в наявності, її можна створити за допомогою кнопки CREATE DATABASE у вкладці Database (рис. 15.5).

Також можна створити нову БД за допомогою кнопки «+», як показано на рис. 15.6.

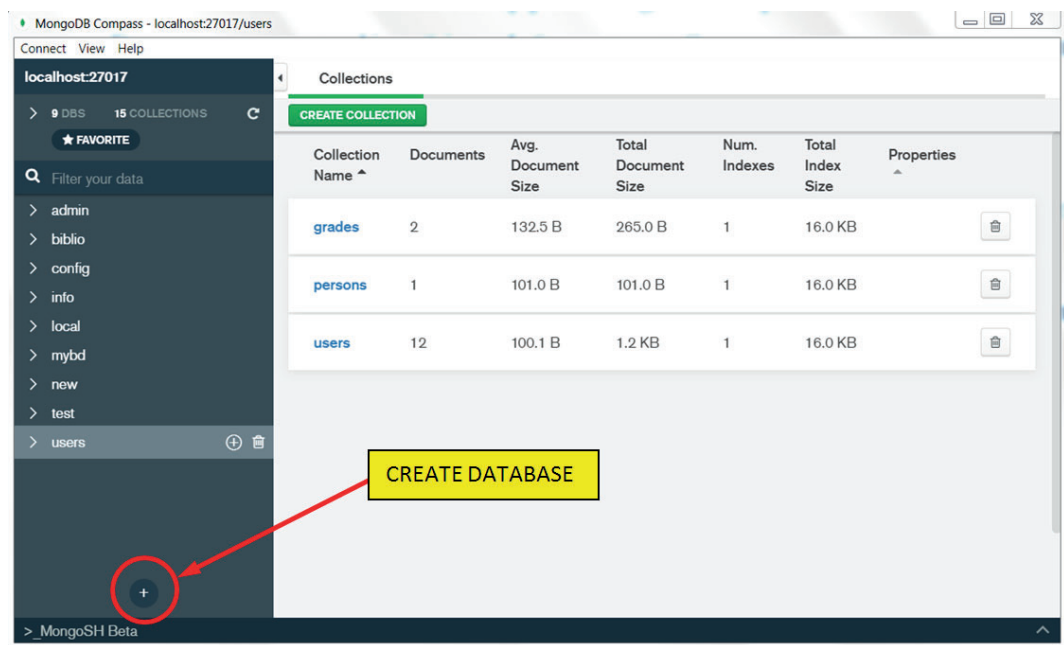


Рис. 15.6. Меню створення БД в MongoDB Compass

Відкриється вікно створення БД в MongoDB Compass, у відповідних полях якого необхідно вказати ім'я БД, що створюється, та ім'я колекції (рис. 15.7).

Під час створення БД можна додатково вказати обмеження на максимальний розмір майбутньої БД (Capped Collection), а також використання спеціального сортування (Use Custom Collation).

Так як в кожній БД може бути довільна кількість колекцій, у створеній, або в будь-якій іншій, в поточну БД можна додати в будь-який момент нову колекцію за допомогою інструменту CREATE COLLECTION (рис. 15.6).

Create Database

Database Name

Collection Name

☐ Capped Collection ⓘ

☐ Use Custom Collation ⓘ

Before MongoDB can save your new database, a collection name must also be specified at the time of creation. [More Information](#)

CANCEL CREATE DATABASE

*Рис. 15.7. Вікно для створення БД в MongoDB Compass*

При цьому відкриється діалогове вікно (рис. 15.8).

Create Collection

Collection Name

☐ Capped Collection ⓘ

☐ Use Custom Collation ⓘ

CANCEL CREATE COLLECTION

*Рис. 15.8. Вікно для колекцій БД в MongoDB Compass*

#### 15.4. Додавання документів

У MongoDB основними елементами даних є документи, тобто кожен запис у базі даних MongoDB є документом, який належить певній колекції.

Для додавання документу до відповідної колекції в MongoDB Compass необхідно цю колекцію зробити активною, і тоді у закладці Documents вибрати ADD DATA (рис. 15.9).

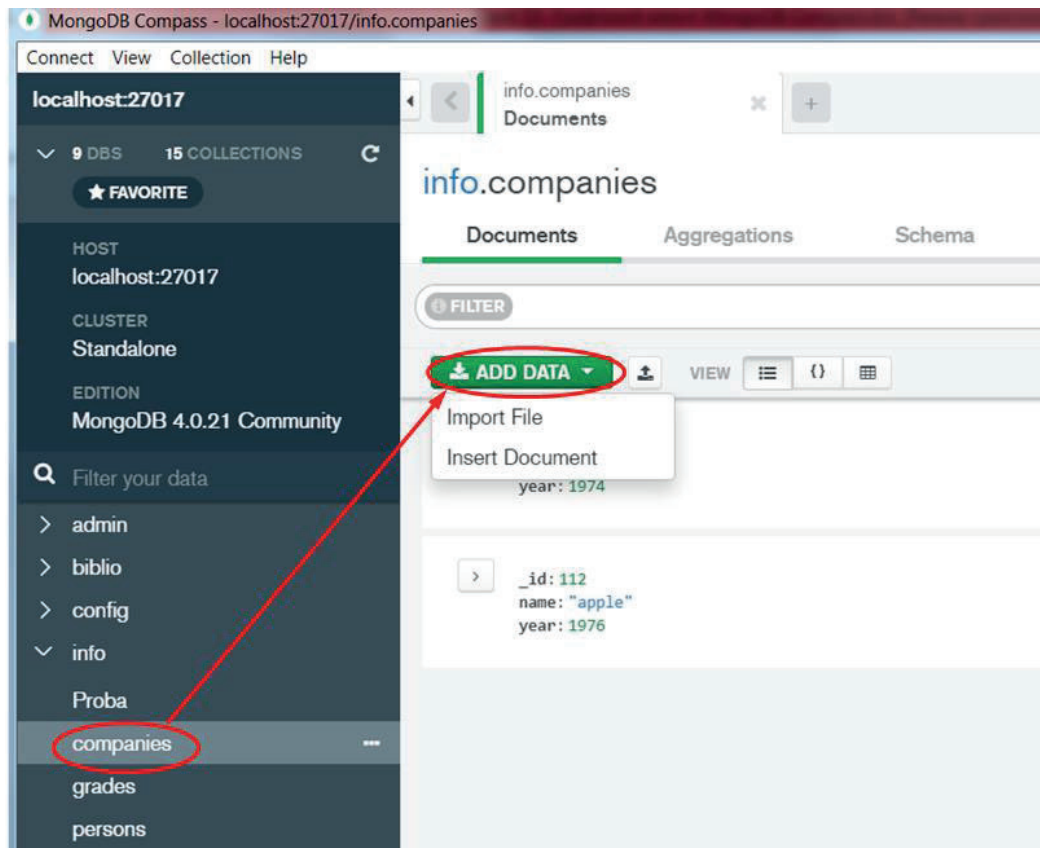


Рис. 15.9. Виклик функціоналу додавання документів

Команда ADD DATA здійснює додавання документів за допомогою:

- імпорту файлу (Import File);
- інтерактивного додавання документу (Insert Document).

#### 15.4.1. Імпорт файлу

Зазвичай документи в MongoDB відображаються в форматі JSON, але функціонал імпорту документів в MongoDB Compass надає змогу завантажувати в БД також і файли в форматі CSV.

Після вибору типу додавання документу до колекції Import File відкриється діалогове вікно (рис. 15.10), в якому треба вибрати шлях до файлу, а також відповідний тип файлу.

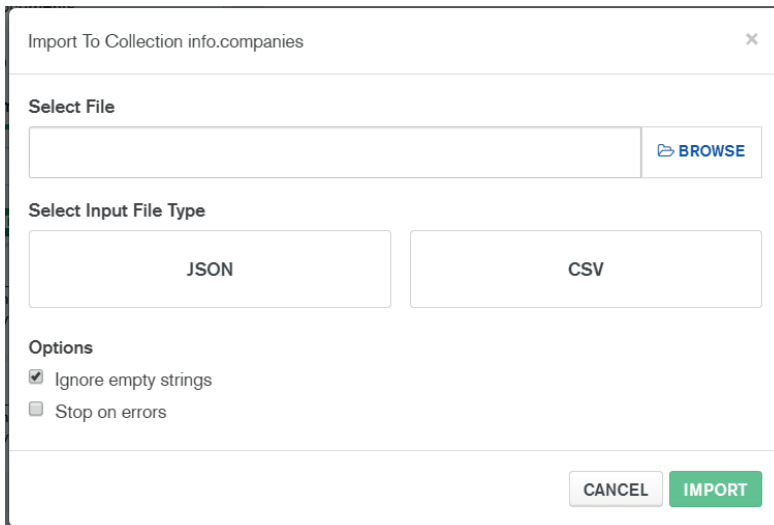


Рис. 15.10. Імпорт документів у поточну колекцію

#### 15.4.2. Інтерактивне додавання документу

Інтерактивне додавання документу Insert Document реалізує метод MongoDB `insert()` – об'єднання можливостей методів `insertOne` та `insertMany`, тобто надає змогу додавати як один документ, так і декілька документів одночасно.

Крім того, додавати документ(и) в MongoDB Compass можна у форматі JSON або пари ключ-значення (рис. 15.11).

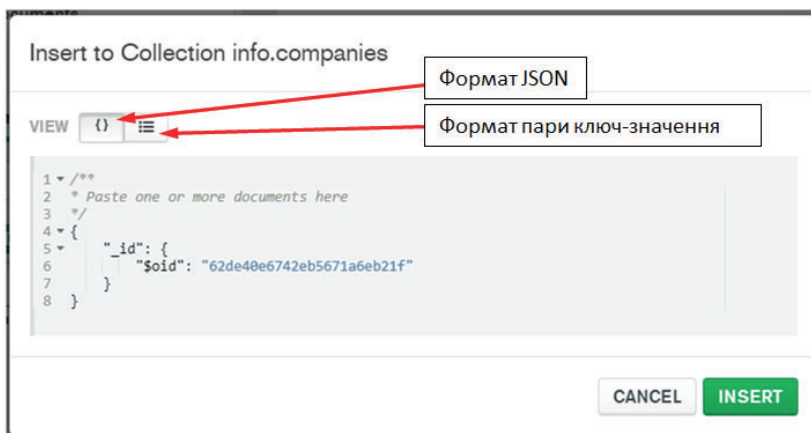


Рис. 15.11. Додавання документів у поточну колекцію

Як і у всіх попередніх прикладах додавання документів в консолі MongoDB, у графічному інтерфейсі Compass поле `_id` також автоматично генерується, якщо воно не вказано.

### 15.4.3. Клонування документів

Під час додавання великої кількості однотипних нових документів або складних за структурою вже наявних, MongoDB Compass надає змогу клонувати, редагувати та, відповідно, додавати змінені документи до поточної колекції.

Для використання цього функціоналу необхідно для відповідного документу поточної колекції обрати команду Clone Document (рис. 15.12).

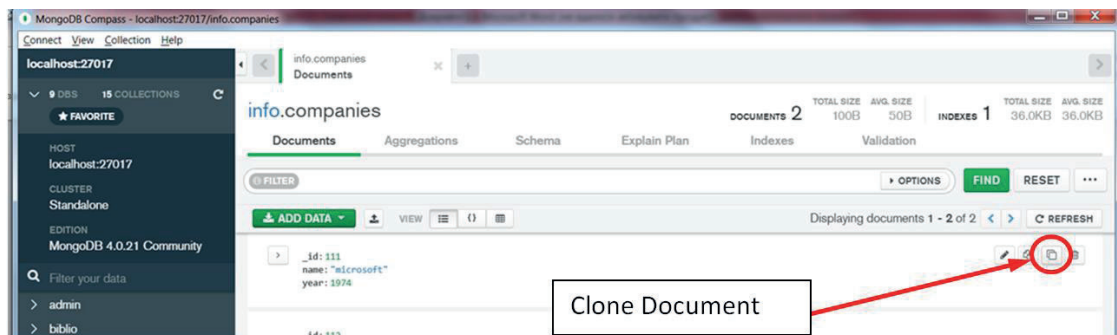


Рис. 15.12. Виклик функціоналу Clone Document

Команда Clone Document копіює документ у нове вікно Insert to Collection, у якому можна змінити значення даних полів, при цьому немає необхідності вказувати імена полів або типи даних знову.

При цьому поле `_id` не копіюється, а автоматично генерується, хоча за необхідності його можна вказати явно (рис. 15.13).



Рис. 15.13. Додавання документу методом Clone Document:

а) – клонований документ; б) – змінені доні для додавання у поточну колекцію

За допомогою функціоналу Clone Document можна продовжувати додавати будь-яку кількість даних до колекції, і ці дані відображатимуться в інтерфейсі Compass.

## 15.5. Перегляд даних (документів)

Компас дозволяє переглядати документи колекції у трьох режимах:

- Перегляд списку (рис. 15.14);
- Перегляд JSON (рис. 15.15);
- Вид таблиці (рис. 15.16).

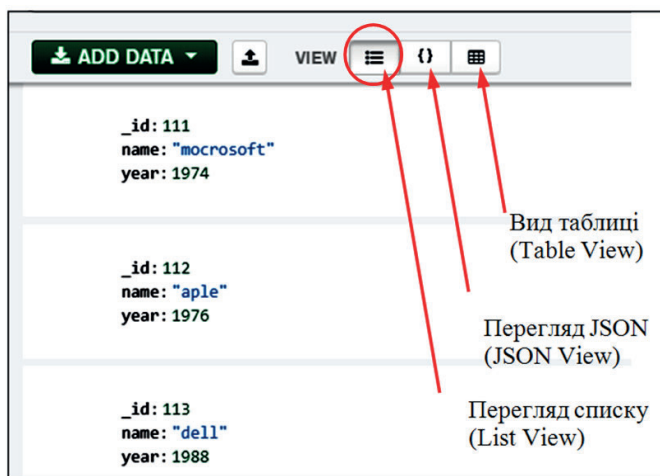


Рис. 15.14. Режими перегляду даних: Перегляд списку

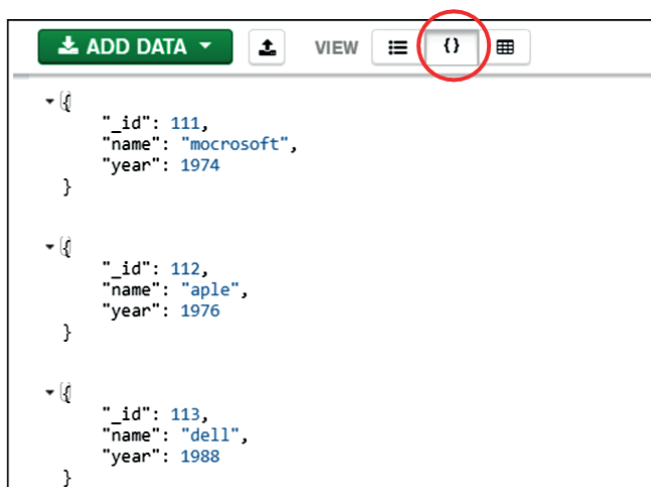


Рис. 15.15. Режими перегляду даних: Перегляд JSON



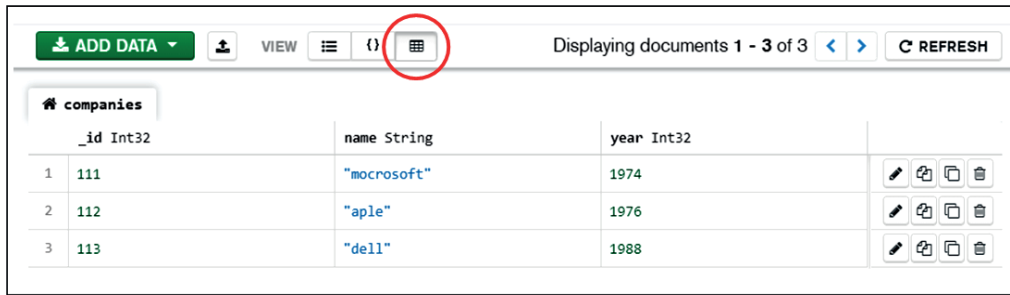


Рис. 15.16. Режими перегляду даних: Перегляд списку

Для фільтрування даних необхідно в полі FILTER ввести умову фільтра та виконати команду FIND (рис. 15.17).

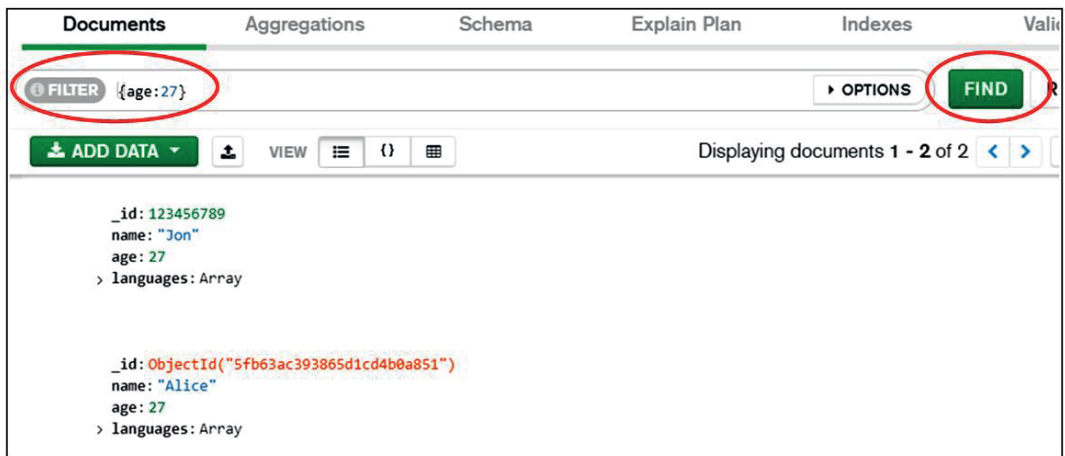


Рис. 15.17. Фільтр в MongoDB Compass

Відміна дії фільтра здійснюється командою RESET.

Під час перегляду даних у поточних документах також надається можливість додавання, редагування та видалення (рис. 15.18).

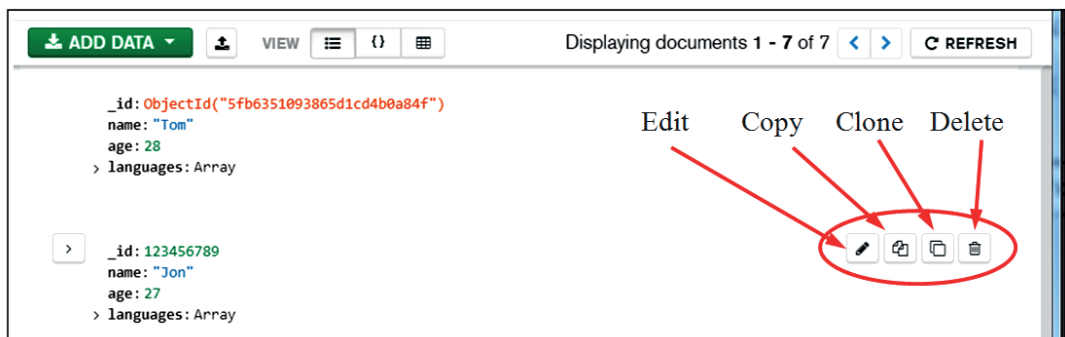


Рис. 15.18. Функціонал підтримки документа в актуальному стані

## 15.6. Редагування документів

Для редагування даних в поточному документі MongoDB Compass необхідно включити команду Edit (рис. 15.18).

Лише тоді графічний функціонал редагування стає доступний користувачу.

Оновлені дані можна як зберегти (команда UPDATE), так і відмовитись від зроблених оновлень (команда CANCEL) (рис. 15.19).

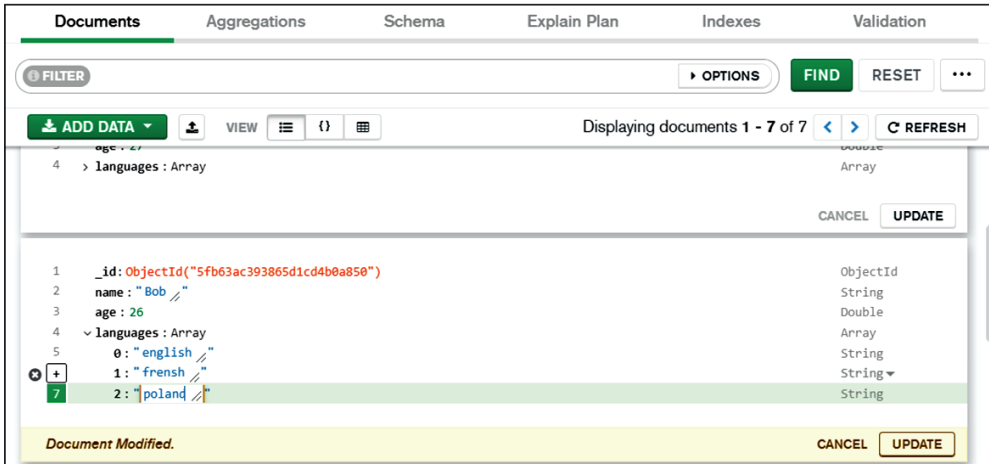


Рис. 15.19. Редагування документів

## 15.7. Видалення документів

Для видалення даних у MongoDB Compass необхідно зробити необхідний документ поточним (виділити) та у меню підтримки документа в актуальному стані вибрати команду Delete Document (рис.15.20).

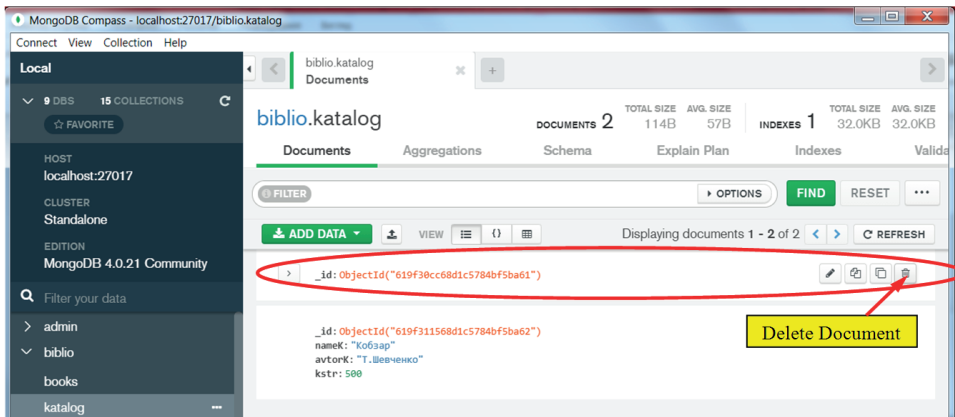


Рис. 15.20. Видалення документів

## 15.8. Видалення колекції та бази даних

Видалення колекцій та баз даних реалізується командою DROP.

Для видалення БД необхідно її виділити в провіднику Compass та виконати команду Drop database (рис. 15.21).

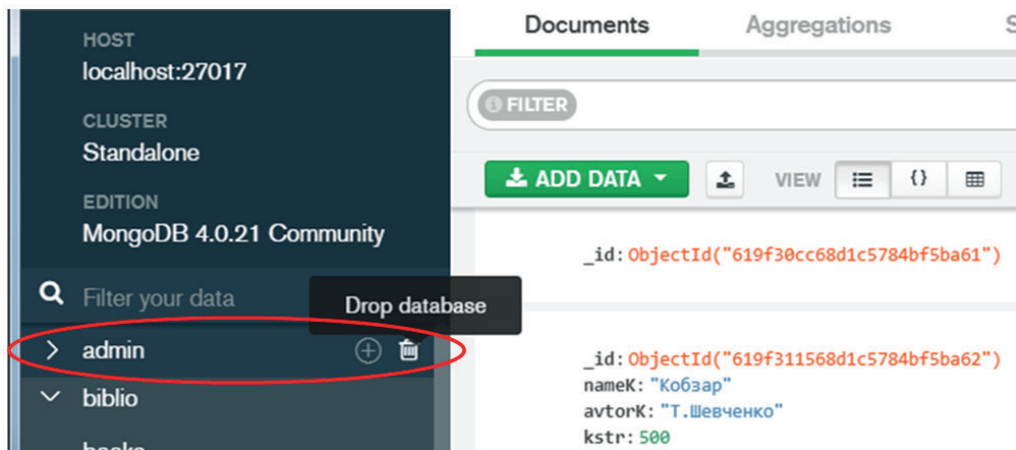


Рис. 15.21. Видалення поточної БД

Для видалення відповідної колекції БД необхідно її виділити в провіднику Compass та виконати команду Drop Collection (рис. 15.22).

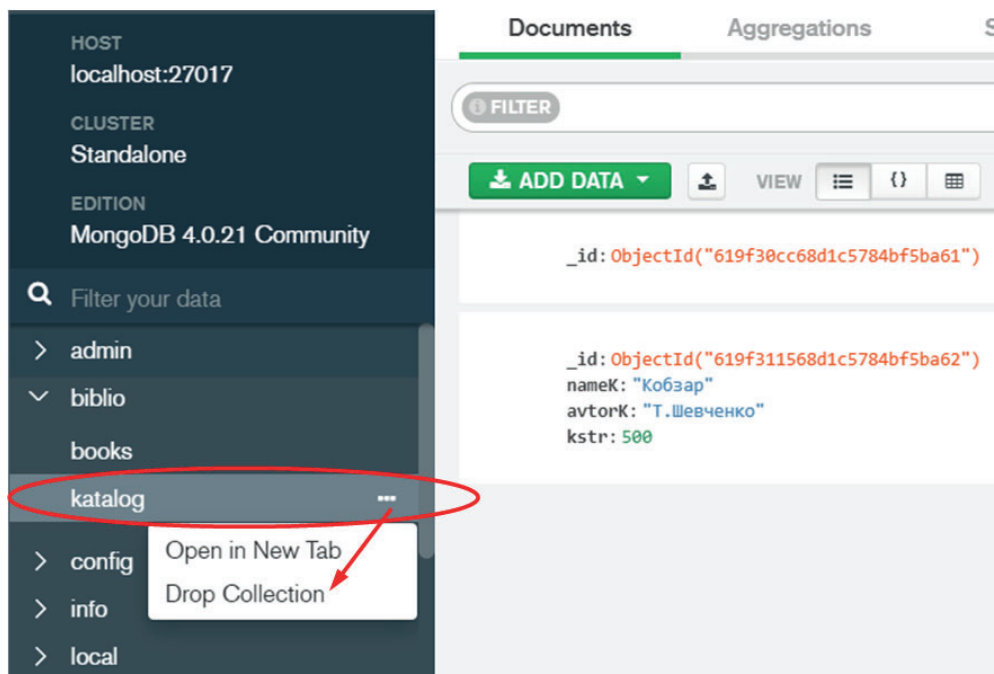


Рис. 15.22. Видалення поточної колекції katalog

## 15.9. Консольний клієнт Mongosh

Але не зважаючи на наявність графічних можливостей для управління даними, вони можуть виявитися недостатніми для більш складних сценаріїв роботи з даними. І для цього в MongoDB Compass є вбудований консольний клієнт Mongosh, в якому можна вводити ті ж команди для роботи з даними, що і в консольній оболонці mongo. Відкривається консольний клієнт в нижній частині вікна MongoDB Compass (рис. 15.23).

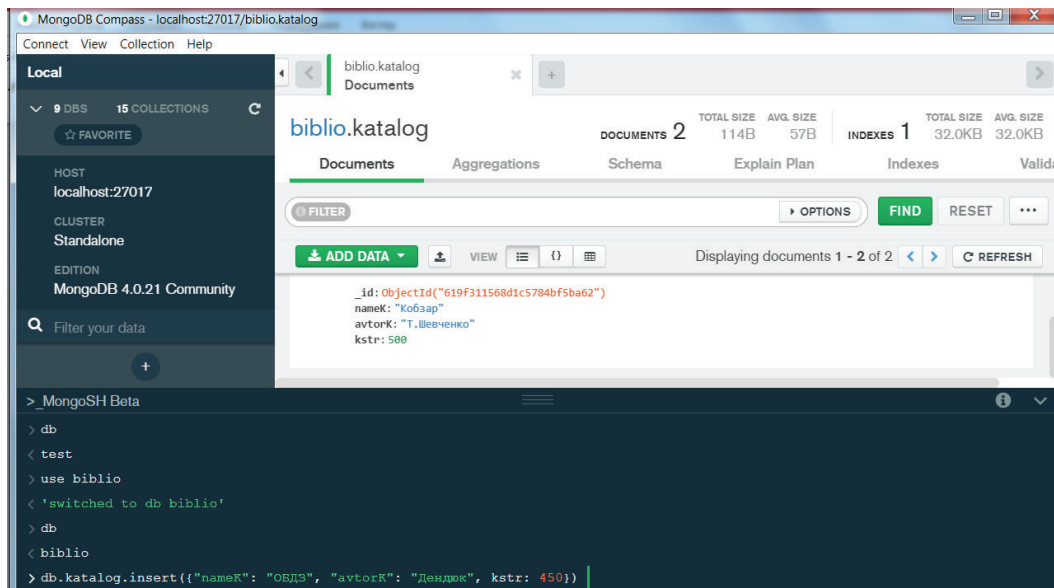


Рис. 15.23. Відкритий консольний клієнт Mongosh

Консольний клієнт Mongosh також повідомляє про застарілі функції. У прикладі рис. 15.23 використано застарілу функцію `insert`, яку консольний клієнт і виконав, але видав застереження (рис. 15.24).

Варто зазначити, що в консольній оболонці mongo виконання цієї функції не викликає жодних проблем.

Крім того, в консольній оболонці mongo є функціонал дозаповнення, що дещо спрощує написання запитів пропонуючи варіанти доступних функцій (рис. 15.25).

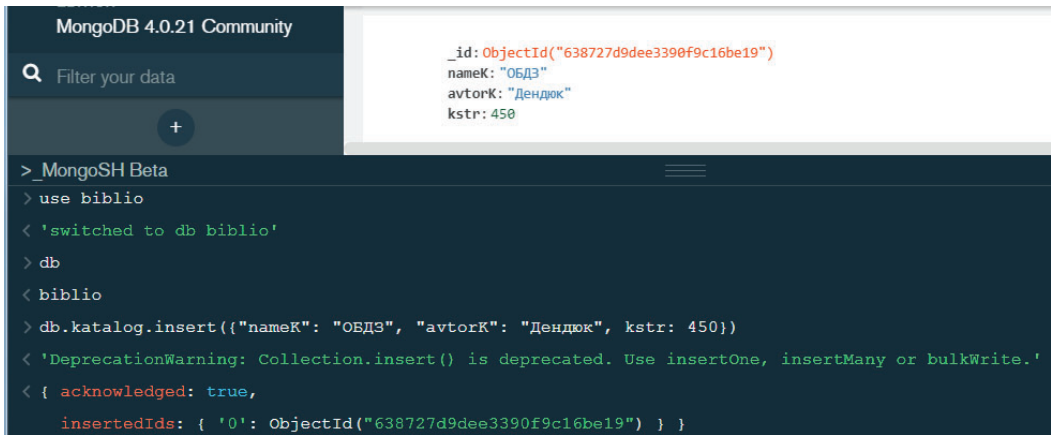


Рис. 15.24. Попередження про застарілу функцію

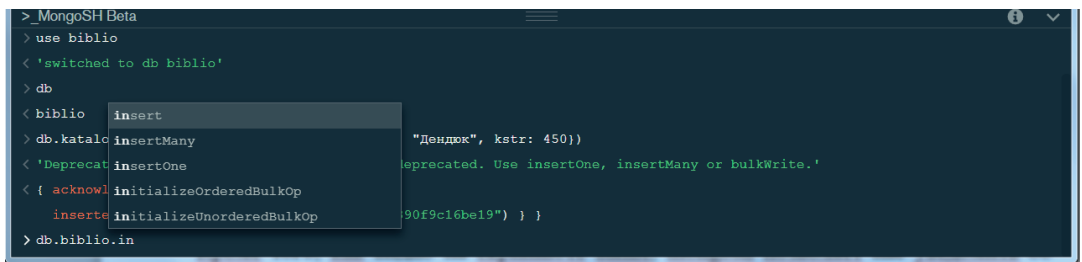


Рис. 15.25. Функціонал дозаповнення

І на закінчення хочеться додати, що в консольній оболонці mongo є можливість копіювати попередні запити, вставляти їх та редагувати, тобто як для програмування, доступні Ctrl+C та Ctrl+V.

## Контрольні запитання

1. Для чого призначений MongoDB Compass?
2. Чи працюватиме MongoDB Compass без сервера MongoDB?
3. Які основні компоненти повинен містити рядок з'єднання з сервером MongoDB?
4. Який порт використовується у рядку з'єднання MongoDB Compass з сервером MongoDB?
5. Які обмеження можна додатково вказати під час створення БД в MongoDB Compass?

6. Які способи надає функціонал MongoDB Compass для додавання документів у колекцію БД?
7. В яких форматах можна імпортувати документи в MongoDB Compass?
8. Чим відрізняються клонування та копіювання документів в MongoDB Compass?
9. Які режими надає MongoDB Compass для перегляду документів?
10. Як увійти в режим редагування в MongoDB Compass?
11. Чи можна в MongoDB Compass повернути операцію видалення документа?
12. Який функціонал дозволяє виконувати консольні команди в MongoDB Compass?

# ПРЕДМЕТНИЙ ПОКАЖЧИК

## A

ACID, 257, 260, 362  
ANSI, 20, 21, 140, 262

## B

Big Data, 272  
BigTable, 27, 364  
BSON, 359, 365, 381

## C

CASE засоби, 105, 112, 113  
cp1251, 136

## D

dataware, 17  
dbForge Studio for MySQL, 114  
DBTools Manager, 117  
DCL, 142, 320  
DDL, 141  
DML, 141, 305

## E

EMS SQL Manager for MySQL, 116  
Entity Relationship model, 95, 129  
ER діаграма, 99, 103, 129  
ER-модель, 95, 129

## H

Hadoop, 274, 275  
hardware, 17  
HeidiSQL, 115

## I

IDEF1X, 105  
ISO, 140

## J

JSON, 341, 343, 358, 365

## M

MapReduce, 274  
middleware, 17  
MOLAP, 289, 291  
MongoDB Compass, 442  
MySQL, 25  
MySQL Workbench, 113, 117

## N

Navicat, 114  
netware, 17  
NoSQL, 24, 357  
NULL-значення, 62, 171, 196

## O

OLAP, 50, 277  
OldSQL, 28  
OLTP, 255

## P

phpMyAdmin, 114, 325, 353

## R

R (мова програмування), 275  
ROLAP, 289, 291

## S

software, 17  
SQL Maestro для MySQL, 115  
SQL мова, 46, 140  
SQLyog, 116  
SSH, 131, 354, 444  
SSL, 131, 323, 444

## T

TCL, 142

## U

utf8, 136, 151

## V

VVV, 273

## Θ

θ-об'єднання, 73, 74

## A

агрегатні функції, 201  
агрегація гіперкуба, 281  
агрегація даних, 52, 281, 285, 402  
аномалія модифікації, 76  
архітектура баз даних, 20  
атомарність, 15, 257  
атрибути, 24, 45, 95

## Б

база даних, 17  
"ключ/значення", 26, 360  
NewSQL, 27  
NoSQL, 25, 275, 357  
багатокористувацька, 21  
багатокористувацька розподілена, 23  
багатокористувацька  
    централізована, 22  
графова, 26, 361  
документо-орієнтована, 26, 358  
однокористувацька (локальна), 22  
реляційна, 24, 28, 65  
стовпцева, 27, 363  
фізична, 40  
бекап, 330

## B

великі дані (Big Data), 272  
вибірка, 70, 164, 386  
вимір, 52  
висхідне проектування, 94  
відкат, 142, 260, 265  
відмовостійкість, 274

віднімання відношень, 69  
відношення, 32, 39, 65  
вітрина даних, 286  
внутрішній рівень, 20  
вторинний ключ (BK), 37, 45  
втрачені оновлення, 256

## Г

гіперкуб, 52, 278, 289  
гіперкубічна схема, 53  
глобальне сховище даних, 286  
горизонтальна масштабованість, 274  
групові функції, 201  
грязне читання, 256

## Д

декартовий добуток, 69, 180  
денормалізація, 90, 288  
деталізація гіперкуба, 281  
довговічність, 257, 258  
документ JSON, 357, 358, 366  
домен, 96, 146

## Ж

журнал транзакцій, 261, 289  
журналізація, 18, 260

## З

збережена підпрограма, 240  
збережена процедура, 240, 241  
збережена функція, 240, 253  
зв'язок, 39, 96  
змінна  
    глобальна, 196  
    локальна, 195, 243  
    сеансова, 196  
    системна, 196  
зовнішнє об'єднання, 176, 178  
зовнішнє подання, 21  
зовнішній запит, 228, 230, 237  
зовнішній ключ, 25, 46, 66, 128  
зовнішній рівень, 20, 21



## I

ідентифікатор, 95, 148, 225, 265, 359, 366, 367  
ізолюваність, 15, 257  
інкапсуляція, 57  
інтелектуальний аналіз даних, 15, 273, 276, 295  
інформаційна система, 17, 21  
історичність даних, 52

## K

клас належності зв'язку, 129  
колекція, 59, 366  
концептуальний рівень, 21  
корельований підзапит, 228, 237  
курсор, 308, 397

## L

ліве зовнішнє об'єднання, 178  
логічний вираз, 147, 410  
логічний зв'язок, 31, 71, 96  
логічний оператор, 70  
логічний рівень, 31  
логічні значення, 378  
логічні функції, 166  
локальний вузол, 24, 322  
локальний сервер, 132, 445  
локальні дані, 274

## M

математичні функції, 203  
менеджер завантаження, 288  
менеджер запитів, 288  
менеджер сховища, 288  
метадані, 285  
модель "сутність–зв'язок", 95, 99  
модель бази даних, 31  
дatalogічна, 31, 33  
концептуальна, 31  
логічна, 31, 33  
міфологічна, 31, 32  
фізична, 32, 33  
модель даних (Data Model), 35

багатовимірні, 35, 49  
дедуктивно-об'єктно-орієнтована, 36  
ієрархічна, 35, 38  
концептуальна, 32, 36  
мережна, 35, 42  
об'єктно-орієнтована, 35, 56  
об'єктно-реляційна, 36, 60  
постреляційна, 35, 48  
реляційна, 35, 44  
семантична, 36, 95  
файлова, 35, 36

## N

неузгоджена обробка, 256  
неузгодженість даних, 37  
низхідне проектування, 94  
нормалізація, 45, 76, 255  
нормальна форма (НФ), 76  
Бойса-Кодда (НБКФ), 82  
доменно-ключова (ДКНФ), 89  
друга (2НФ), 79  
неперша (NF2), 48  
перша (1НФ), 77  
п'ята (5НФ), 87  
третя (3НФ), 81  
четверта (4НФ), 85  
нотація, 99  
IDEF1X, 105  
Information Engineering (IE), 103  
UML, 109  
Баркера і Мартіна, 103  
Чена, 99

## O

обертання гіперкуба, 280  
об'єднання відношень, 68, 72, 171, 182  
обмежувач, 241  
обслуговуюча функція, 288  
оперативна аналітична обробка, 15, 273  
оператори керування, 198, 404

## П

первинний ключ (ПК), 25, 37, 66, 100  
перегляд (VIEW), 298, 300  
перетин відношень, 68, 177  
підзапит, 228, 305  
полікуб, 290  
полікубічна схема, 54  
поліморфізм, 57  
праве зовнішнє об'єднання, 179  
предметна область, 18, 93  
привілеї, 316  
природне об'єднання, 73  
прогнозованість даних, 52  
проекція, 71, 390

## Р

регулярний вираз, 168, 393  
реляція, 44  
рівень ізоляції транзакцій, 258  
рівень привілеїв, 321  
розподіл, 74  
рядкове блокування, 150, 267  
рядок-фантом, 256, 258

## С

система керування базою даних  
(СКБД), 18  
системні функції, 225  
статистичні функції, 201  
сторінкове блокування, 150, 267  
ступінь зв'язку, 96, 101, 129  
сутність, 95, 99  
схема  
"зірка", 291  
"сніжинка", 292

сховище даних, 20, 273, 277, 283

## Т

таблиця вимірів, 292  
таблиця фактів, 291  
табличне блокування, 267  
тип зв'язку, 39, 99, 129  
точка відкату, 265  
транзакція, 142, 255  
тригер, 311  
тризначна логіка, 63

## У

узгодженість, 257  
успадкування, 57

## Ф

формат  
CSV, 341  
JSON, 341  
функції  
властивостей рядка, 210  
дати, 205  
перетворення у рядку, 212  
повернення частини рядка, 220  
позиції входження, 218  
порівняння рядків, 222  
системні, 225  
часу, 207

## Ц

цілісність інформації, 61, 257

## Ш

шаблон пошуку, 167, 168, 223

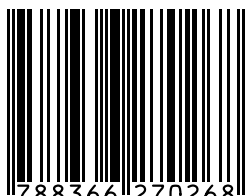
# ЛІТЕРАТУРА

1. Alvarez P. M. Main Memory Management on Relational Database Systems / P. M. Alvarez, M. L. Ayala, S. C. Cisneros. – Cham: Springer, 2022. – 115 p.
2. Karthik A. MySQL 8 Cookbook / Appigatla Karthik.- Packt Publishing, 2018. – 446 p.
3. Ardeleanu S. Two Styles of Database Development: Describe the opposition between an atomic, procedural and row-oriented style versus a holistic, SQL and dataset oriented style, within a relational database / Stefan Ardeleanu.- CreateSpace Independent Publishing Platform, 2015. – 184 p.
4. Aspin A. Querying MySQL: Make your MySQL Database Analytics Accessible with SQL Operations, Data extraction, and Custom Queries. / Aspin Adam.- BPB Online, 2022. – 674 p.
5. Atzeni P. Database Systems. Concepts, Languages & Architectures / Atzeni Paolo, Ceri Stefano, Paraboschi Stefano, Torlone Riccardo, - McGraw-Hill, 2003. – 612 p.
6. Ballard R. Relational Databases for Agile Developers / Ballard Ron. - CreateSpace, 2017. – 354 p.
7. Bechberger D. Graph Databases in Action: Examples in Gremlin / Bechberger Dave, Perryman Josh, - Manning Publications Co., 2020. – 375 p.
8. Bell C. Expert MySQL / Bell Charles, - Apress, 2013. – 640 p.
9. Bierer D. Learn MongoDB 4.x: A guide to understanding MongoDB development and administration for NoSQL developers / Bierer Doug, - Packt, 2020. – 610 p.
10. Boronczyk T. Jump Start MySQL / Boronczyk Timothy, - SitePoint, 2015. – 150 p.
11. Brodie M.L. Making Databases Work: The Pragmatic Wisdom of Michael Stonebraker / Michael L. Brodie, - ACM, 2019. – 730 p.
12. Campbell L. Database Reliability Engineering / Campbell Laine, Majors Charity, - O'Reilly Media, 2017. – 300 p.
13. Cheema MA. Databases Theory and Applications / Cheema Muhammad Aamir, Zhang Wenjie, Lijun Chang, - Springer, 2016. – 507 p.
14. Chellappan S. MongoDB Recipes: With Data Modeling and Query Building Strategies / Chellappan Subhashini, Ganesan Dharanitharan, -Apress, 2020. – 247 p.

15. Coronel C. Database Principles: Fundamentals of Design, Implementation, and Management / Coronel Carlos, – Cengage, 2020. – 960 p.
16. Dasadia C. MongoDB Cookbook / Dasadia Cyrus, Nayak Amol, – Packt Publishing, 2015. – 371 p.
17. Giamas A. Mastering MongoDB 6.x / Giamas Alex, – Packt, 2022. – 460 p.
18. Harrison G., Harrison M. MongoDB Performance Tuning: Optimizing MongoDB Databases and their Applications / Harrison Guy., Harrison Michael. - Apress, 2021. – 360 p.
19. Берко А. Організація баз даних: практичний курс / А.Берко, О.Верес. – Львів: НУЛП, 2003. – 152 с.
20. Буй Д. Б. Формалізація моделі сутність-зв'язок / Д. Б. Буй, Л. М. Сільвейструк. – Київ: ВПЦ Київський університет, 2010. – 195 с.
21. Гайдаржи В. І. Основи проектування та використання баз даних: навчальний посібник / В. І. Гайдаржи, О. А. Дацюк. – Київ: ІВЦ «Видавництво «Політехніка», ТОВ «Фірма «періодика»», 2004. – 256 с. – (2-е видання, перероблене та доповнене).
22. Гайна Г.А. Основи проектування баз даних: навчальний посібник / Г.А. Гайна, – Київ: КНУБА, 2005. – 204 с.
23. Демиденко М.А. Введення в сучасні бази даних / М.А. Демиденко, – Дніпро: Дніпровська політехніка, 2020. — 38 с.
24. Завадський І.О. Основи баз даних / І.О. Завадський, - Київ: ПП, 2011. – 192 с.
25. Зарицька О.Л. Бази даних та інформаційні системи: методичний посібник / О.Л. Зарицька, – Житомир: ЖДУ ім. І. Франка, 2009. – 132 с.
26. Корнієнко М.М., Іванова І.Д. Бази даних. Системи управління базами даних / М.М.Корнієнко, І.Д. Іванова, – Харків: Ранок, 2009. – 48 с.
27. Куваєв Я.Г. Організація реляційних баз даних: навчальний посібник / Я.Г. Куваєв, О.А. Жукова, І.А. Сечкін, Дніпро: ДВНЗ «Національний гірничий університет» (НГУ), 2017. – 157 с.
28. Морзе Н.В. Бази даних у навчальному процесі: навч.-метод. посібник / Н.В. Морзе, – Київ: ТОВ Редакція «Комп'ютер», 2007. – 120 с.

29. Мулеса О.Ю. Інформаційні системи та реляційні бази даних: навчальний посібник / О.Ю. Мулеса, - Ужгород: Електронне видання УжНУ, 2018. – 118 с.
30. Павельчак А.Г. Бази даних та знань: конспект лекцій / А.Г. Павельчак, В.В. Самотий, У.Ю. Дзелендзяк, – Львів: Львівська політехніка, 2011. – 115 с.
31. Пасічник В.В. Організація баз даних та знань: підручник / В.В. Пасічник, В.А. Резніченко, – Київ: Вид. група ВНУ, 2006. – 384 с.
32. Пасічник В. В. Сховища даних : навч. посібник / В. В. Пасічник, Н.Б. Шаховська ; за ред. В. В. Пасічника, - Львів : Магнолія-2006, 2008. - 492 с.
33. Ситник Н.В. Проектування баз і сховищ даних: навч. посібник / Н.В. Ситник, – Київ: КНЕУ, 2004. – 348 с.
34. Федоров Є.Є. Система управління базами даних MySQL 8.0 / Є.Є. Федоров, О.В. Нечипоренко, М.В. Чичужко, – Черкаси : ЧДТУ, 2019. – 371 с.
35. Харів Н.О. Бази даних та інформаційні системи: навчальний посібник / Н.О. Харів, – Рівне: Національний університет водного господарства та природокористування (НУВГП), 2017. – 127 с.
36. Швачич Г.Г. Системи управління базами даних: конспект лекцій. Частина 1 / Г.Г. Швачич, О.В. Овсянніков, В.В. Кузьменко, Н.І. Нечаєва, – Дніпропетровськ: НМетАУ, 2007. – 48 с.
37. Швачич Г.Г. Системи управління базами даних: конспект лекцій. Частина 2 / Г.Г. Швачич, О.В. Овсянніков, В.В. Кузьменко, Н.І. Нечаєва, – Дніпропетровськ: НМетАУ, 2007. – 35 с.
38. Юрчишин В.М. Організація баз даних: навчальний посібник / В.М. Юрчишин, Б.В. Клим, – Івано-Франківськ: Факел, 2010. – 224 с.
39. Ярцев В.П. Організація баз даних та знань: навчальний посібник / В.П. Ярцев, – Київ: Державний університет телекомунікацій, 2018. – 214 с.

WYDANIE  
ONLINE **PDF**



ISBN 978-83-66270-29-9  
Wydawnictwo GSW · 2023  
[www.gsw.gda.pl/wydawnictwo](http://www.gsw.gda.pl/wydawnictwo)  
Nr katalogowy: [ 129 ]



**GSW**

GDAŃSKA SZKOŁA WYŻSZA