**Міністерство освіти і науки України**

**Державний вищий навчальний заклад**
**«Національний гірничий університет»**


**Я.Г. Куваєв, О.А. Жукова, І.А. Сечкін**

# Організація реляційних баз даних

Навчальний посібник

Видання друге, доповнене та перероблене

Дніпро
НГУ
2017

**УДК 004.652.4(075.8)**
**К 64**

Рекомендовано вченою радою Державного ВНЗ «Національний гірничий університет» для студентів напрямів підготовки 125 Кібербезпека, 172 Телекомунікації (протокол № 1 від 20.09.2017).

***Рецензенти***:
**В.В. Ткачов** – д-р техн. наук, професор, завідувач кафедри автоматизації комп'ютерних систем Державного ВНЗ «Національний гірничий університет»;
**В.В. Слєсарєв** – д-р техн. наук, професор, завідувач кафедри системного аналіза і управління Державного ВНЗ «Національний гірничий університет».

**Куваєв Я.Г.**
К64 Організація реляційних баз даних : навч. посіб. / Я.Г. Куваєв, О.А. Жукова, І.А. Сечкін – 2-ге вид., допов. та переробл. – Дніпро : НГУ, 2017. – 157 с.

Висвітлено питання організації зберігання даних та методів доступу до них. Зокрема детально описано питання вибору тієї або іншої реляційної СУБД, розробки логічної та фізичної моделі поданих даних, забезпечення корпоративної цілісності даних тощо.

**УДК 004.652.4(075.8)**

# ЗМІСТ

# Передмова

Організація реляційних баз даних (БД) є невід'ємною областю знань сучасного фахівця, що здобуває освіту в технічній галузі. Це пов'язано насамперед з тим, що сьогодні приблизно у 80 % випадків, державні та комерційні структури всього світу використовують для своїх потреб БД, які базуються на реляційній моделі, що запропонована Едгаром Коддом у 1970 році. Концепція реляційної моделі була перевірена часом та виявилась на стільки вдалою, що системи управління базами даних (СУБД), які з'являлися до тепер на ринку та були створені на базі інших моделей, не змогли суттєво вплинути на розповсюдження СУБД, які базуються на реляційній моделі.

Курс організації реляційних БД може вивчатися як окрема дисципліна, або як складова частина більш загального курсу, яка розкриває питання організації зберігання даних та методів доступу до них, що використовуються основною дисципліною.

Основною метою курсу є здобуття теоретичних та практичних знань щодо організації БД, які базуються на концепції реляційної моделі даних. В зв'язку з цим розглянуто на прикладі процес побудови БД від фази логічного моделювання задля забезпечення корпоративних правил цілісності даних, які не може покрити реляційна модель в рамках цілісності сутностей та посилальної цілісності і до використання методів доступу до даних. Також розглянуто механізм трансакцій, який запобігає конфліктним ситуаціям при одночасному доступу до БД багатьох користувачів. Додатково розглянуті питання захисту інформації в СУБД від несанкціонованого доступу.

Автори не ставили перед собою завдання вивчення якої-небудь СУБД як програмного продукту. Навпаки, показано за якими критеріями потрібно обирати СУБД для реалізації певних проектів. Ідея полягає в тому, щоб отримані навики студент міг застосувати, використовуючи СУБД, що обрав самостійно.

Усі приклади та фізична реалізація БД, що використовується як приклад, відпрацьовані в СУБД Interbase. На час написання книги останньою версією є Interbase Усі приклади та фізична реалізація БД, що використовується як приклад, відпрацьовані в СУБД Interbase. На час написання книги останньою версією є Interbase XE (2010 р.) від Embarcadero.

# 1. Порівняння різних технологій СУБД

*Мета розділу – поняття інформаційної системи організації та її компонентів, стислий розгляд різних технологій СУБД, порівняння архітектур СУБД, ознайомлення з життєвим циклом бази даних.*

## 1.1. Інформаційна система

СУБД є невід'ємною частиною інформаційної системи організації [1-3].

**Інформаційна система** – ресурси, які дозволяють виконувати збір, корегування й поширення інформації усередині організації [4].

Типова комп'ютеризована інформаційна система включає наступні компоненти:

1) база даних;
2) програмне забезпечення БД;
3) прикладне програмне забезпечення;
4) апаратне забезпечення, у тому числі обладнання зберігання;
5) персонал, що експлуатує й розробляє цю систему.

## 1.2. Ручні картотеки – файлові системи – сучасні СУБД

Надамо характеристику методам зберігання й управління даними, які використовуються інформаційними системами організацій [4].

***Ручні картотеки*** дозволяють успішно справлятися з поставленими завданнями обробки інформації, якщо кількість об'єктів, що зберігаються, невелика. Вони цілком придатні для зберігання й здобуття великої кількості даних, але зовсім не підходять для тих випадків, коли необхідно встановити перехресні зв'язки або виконати обробку інформації. Якщо нам знадобиться якась інформація, то треба переглянути картотеку від початку й до кінця, щоб знайти потрібні дані. Для забезпечення пошуку потрібної інформації передбачається використання в такій системі деякого алгоритму індексування, що дозволяє прискорити пошук належних даних. Наприклад, можна використовувати спеціальні роздільники або окремі папки для різних типів об'єктів, що логічно пов’язані.

***Файлові системи*** – це набір програм, які виконують для користувачів деякі операції, наприклад, створення звітів. Кожна програма визначає свої власні дані й управляє ними [2, 4, 5].

Файлові системи були першою спробою комп'ютеризувати відомі всім ручні картотеки. Вони були розроблені у відповідь на потребу в одержанні більш ефективних засобів доступу до даних. Однак, замість організації централізованого сховища всіх даних організації, був використаний децентралізований підхід, при якому співробітники кожного відділу за допомогою фахівців з обробки даних працювали зі своїми власними даними й зберігали їх у своєму відділі.

Такого короткого опису файлових систем цілком достатньо для того, щоб зрозуміти їхні недоліки:

1) розподіл і ізоляція даних;

2) дублювання даних;

3) залежність від даних;

4) несумісність форматів файлів;

5) швидке збільшення кількості прикладних програм для здобуття інформації, необхідної користувачам.

***Розподіл та ізоляція даних*** приводить до істотних ускладнень, коли необхідно організувати обробку інформації у двох в більшій кількості файлів.

***Дублювання даних*** спричиняє неекономну витрату ресурсів, оскільки на введення надлишкових даних потрібно витрачати додатковий час та кошти. Воно може привести до порушення їх цілісності. Наприклад, у двох різних відділах організації можна одержати на одне й те саме питання суперечливі відповіді. У багатьох випадках дублювання даних можна уникнути спільним використанням файлів.

***Залежність від даних*** приводить до великих витрат при зміні фізичної структури файлів, яка відображена в програмному забезпеченні, що працює із ними. Наприклад, після деякого строку промислової експлуатації прикладної програми з'ясувалося, що за зміненим законодавством номер банківського рахунку буде збільшено з 10 до 16 знаків. Щоб вийти із ситуації, що створився, необхідно написати програму, яка відпрацює лише один раз. Вона повинна відкрити початковий файл, створити тимчасовий файл із новою структурою, зчитати інформацію з початкового файлу, перетворити дані в новий формат і записати їх у тимчасовий файл, вилучити первісний файл, привласнити тимчасовому файлу його ім'я.

***Несумісність форматів файлів*** ускладнює обробку інформації ще в більшому обсязі, ніж розподіл та ізоляція даних. Наприклад, вона може бути наслідком необхідності спільного використання даних двома локальними задачами. Для цього знадобиться створювати програмне забезпечення, яке конвертує дані в один загальний формат. Після чого можлива їхня спільна обробка.

***Швидке збільшення кількості прикладних програм для здобування інформації, яка необхідна користувачам файлових систем*** багато в чому залежать від програмістів, тому що всі потрібні запити й звіти повинні створювати саме вони. У результаті події розвивалися по одному з наступних сценаріїв. В організаціях, де програмісти були відсутні в штатному розкладі, типи запитів та звітів, що створювались, мали фіксовану форму, і не було ніяких інструментів виготовлення незапланованих або довільних запитів. В інших організаціях, де програмісти працювали, спостерігалося швидке збільшення кількості файлів та прикладних програм. В остаточному підсумку, настав момент, коли вони були просто не в змозі впоратися із задоволенням потреб користувачів в одержанні необхідної їм інформації.

Зазначені обмеження файлових систем є наслідком двох факторів.

1. Визначення даних відбувається усередині прикладних програм, а

не зберігається окремо й незалежно від них.

2. Крім прикладних програм не передбачене ніяких інших інструментів доступу до даних та їх обробки.

Для підвищення ефективності роботи стали використовувати новий підхід, а саме БД і СУБД.

***База даних* (БД)** – *це спільно використовуваний набір логічно зв'язаних даних і їх описів, призначений для задоволення інформаційних потреб організації* [4].

Щоб глибше вникнути в суть цього поняття, розглянемо його визначення більш уважно. БД – це єдине, велике сховище даних, яке одноразово визначається, а потім використовується одночасно багатьма користувачами з різних підрозділів. Замість розрізнених файлів, тут усі дані зібрані разом з мінімальною часткою надмірності. БД вже не належить якомусь єдиному відділу, а є загальним корпоративним ресурсом. Причому вона зберігає не тільки робочі дані, але і їх опис. Із цієї причини її ще називають набором інтегрованих записів із самоописом. У сукупності, опис даних називається системним каталогом (*system catalog*), або словником даних (*data-dictionary*), а самі елементи опису прийнято називати метаданими (*meta-data*), тобто "даними про дані". Саме наявність самоопису даних забезпечує незалежність між прикладними програмами й даними (*program-data independents*).

На сьогоднішній день визначено переваги й недоліки використання СУБД.

ПЕРЕВАГИ СУБД.

1. Контроль над надмірністю даних.
2. Виключення суперечливості даних.
3. Спільне використання даних.
4. Підтримка цілісності даних.
5. Підвищена безпека.
6. Спрощення масштабування системи.
7. Можливість знаходження компромісу при суперечливих вимогах.
8. Підвищення доступності даних і їх готовності до роботи.
9. Поліпшення показників продуктивності.
10. Спрощення супроводу системи за рахунок незалежності від даних.
11. Поліпшене керування паралельною обробкою даних.
12. Розвинені служби резервного копіювання й оновлення.

НЕДОЛІКИ СУБД.

1. Складність.
2. Розмір.
3. Вартість.
4. Додаткові витрати на апаратне забезпечення.
5. Витрати на конвертування даних.
6. Більш серйозні наслідки при виході системи з ладу.

## 1.3. Порівняння локальних СУБД із СУБД архітектури клієнт-сервер

### 1.3.1. Основні складові програмного забезпечення СУБД

**Сервер** надає сервіс клієнтові. Він, по суті, чекає, доки клієнт зробить запит, а потім обробляє його. Сервер повинен мати здатність обробляти одночасно кілька запитів від декількох клієнтів, а також уміти розподіляти ці запити по пріоритетах. Найчастіше серверна програма працює постійно задля забезпечення безперервного доступу до її послуг.

**Клієнти** являють собою прикладні програми, що забезпечують графічний або інший інтерфейс із користувачем. Прикладні програми клієнтів надають користувачеві інтерфейс для управління даними на сервері. Саме через прикладну програму користувач одержує доступ до функціональних можливостей сервера. Прикладом замовлених дій може бути одержання, додавання, корегування та вилучення інформації або друк звіту. У цьому випадку клієнт просто надсилає запит і надає необхідні для його виконання дані. Сервер відповідає за обробку запиту. Це не означає, що клієнт не може виконувати яких-небудь логічних дій самостійно. Цілком можливо, що клієнт реалізує більшу частину (якщо не всю) підтримки бізнес-правил.

**Бізнес-правила** — *це процедури керування, які визначають, як клієнт повинен одержувати доступ і маніпулювати даними на сервері*. Ці правила реалізуються програмним текстом клієнта, сервера або ними обома. На стороні сервера бізнес-правила реалізуються у вигляді збережених процедур, тригерів та інших об'єктів, які властиві серверним БД. Важливо розуміти, що бізнес-правила визначають поведінку всієї системи. При їх відсутності у вас є просто дані на одному комп'ютері й прикладна програма з інтерфейсом на іншому, однак немає методу їх з'єднання.

Якщо більша частина правил реалізована на сервері, то його називають "товстим сервером". Якщо правила реалізуються в основному на стороні клієнта, він називається "товстим клієнтом". Якщо правила реалізовані на середньому рівні, то сервер також називають "товстим". Те, на якій саме стороні реалізуються бізнес-правила, визначається обсягом і типом необхідних операцій управління даними.

### 1.3.2. Файл-серверна архітектура СУБД

БД на персональних комп'ютерах розвивалися за напрямом від настільних, або локальних прикладних програм, коли реально із БД могла працювати одна прикладна програма, до систем колективного доступу до даних [2…7].

Необхідність колективної роботи з тими самими даними спричинила перенесення БД на файловий сервер. Прикладна програма, що працює із БД, розташовувалася також на сервері. Менше поширення одержав інший спосіб, що полягав у зберіганні прикладної програми, яка звертався до БД, на комп'ютері користувача (клієнта). Були випущені нові версії локальних СКБД, які дозволяли створювати прикладні

програми, що одночасно працюють з однією БД на файловому сервері. Основною проблемою була явна або неявна обробка трансакцій та проблема забезпечення смислової та посилальної цілісності БД при одночасному корегуванні даних декількома клієнтами, що неминуче встає при колективному доступі.

У ході експлуатації локальних систем були виявлені наступні недоліки файл-серверного підходу при забезпеченні доступу багатьох користувачів до БД.

1. Увесь тягар обчислювального навантаження при доступі до БД лягає на прикладну програму клієнта, що є наслідком принципу обробки інформації в системах "файл-сервер": при видачі запиту на вибірку інформації з таблиці вся таблиця БД копіюється на комп'ютер клієнта, де здійснюється вибірка.

2. Не оптимально витрачаються ресурси комп'ютера клієнта й мережі. Наприклад, якщо в результаті запиту ми повинні одержати 2 записи з файлу обсягом 10 000 записів, усі 10 000 записів будуть скопійовані з файл-сервера на комп'ютер клієнта. У результаті зростає мережний трафік, і збільшуються вимоги до апаратних потужностей системи. Помітимо, що потреби в постійнім збільшенні обчислювальних потужностей комп'ютера клієнта обумовлюються не тільки розвитком програмного забезпечення, але й зростанням обсягів інформації, яка обробляється.

3. У БД на файл-сервері набагато простіше вносити зміни в окремі таблиці безпосередньо з інструментальних засобів, оминаючи прикладні програми клієнтів. Подібна можливість виникає у зв'язку з тим, що в локальних СУБД база даних – поняття більше логічне, ніж фізичне, тому що під БД розуміється набір окремих файлів, що перебувають у єдиному каталозі на диску. Усе це дозволяє говорити про низький рівень безпеки, як з погляду несанкціонованого доступу й нанесення шкоди, так і з погляду внесення помилкових змін.

4. Бізнес правила в системах файл-сервер реалізуються в прикладних програмах. Це дозволяє їм застосовувати взаємовиключні бізнес-правила. Смислова цілісність інформації при цьому може порушитися.

### 1.3.3. Архітектура клієнт-сервер

Наведені недоліки вирішуються при переході з архітектури "файл-сервер" [2, 4, 5] на архітектуру "клієнт-сервер", яка є наступним етапом у розвитку СУБД [2…7]. Характерна риса архітектури "клієнт-сервер" – перенесення обчислювального навантаження на сервер БД (SQL-сервер) і максимальне розвантаження прикладної програми клієнту від обчислювальної роботи, а також істотне підвищення безпеки даних – як від зловмисних, так і просто помилкових змін. БД у цьому випадку міститься на сервері, як і в архітектурі "файл-сервер", однак прямого доступу до БД із прикладних програм не відбувається. Функції прямого звертання до БД здійснює спеціальна керуюча програма – SQL-сервер, що є невід'ємною частиною СУБД.

Взаємодія сервера БД і прикладної програми-клієнта відбувається за такий спосіб: клієнт формує SQL-запит і відсилає його серверу. Сервер, який прийняв запит, виконує його, і результат повертає клієнтові. У прикладній програмі-клієнті в основному здійснюється інтерпретація отриманих від сервера даних, реалізація інтерфейсу з користувачем, введення даних, а також реалізація частини бізнес-правил.

Багато із вже існуючих систем "клієнт-сервер" розвивалися з прикладних програм, що використовували локальні БД, які використовувалися спільно та були розміщені у мережах персональних комп'ютерів на файлових серверах. Перенос на SQL-сервер СУБД файл-серверної архітектури здійснювався з метою підвищення ефективності їх роботи, захищеності й надійності БД.

Модель архітектури "клієнт-сервер" передбачає наявність декількох рівнів. Дворівнева модель, можливо, найбільш загальна, оскільки вона додержується схеми побудови локальних БД (рис. 1.1). У цій моделі дані постійно перебувають на сервері, а програмне забезпечення доступу до даних – на комп'ютері користувача. Бізнес-правила при цьому можуть розташовуватися на кожному з комп'ютерів (або навіть на обох одночасно).

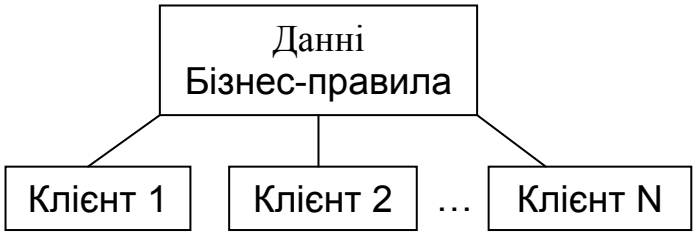



Рис. 1.1. Дворівнева модель "клієнт-сервер"

У трирівневій моделі архітектури "клієнт-сервер" (рис. 1.2) в ПЗ клієнта реалізований лише інтерфейс користувача, що дозволяє мати доступ до даних, які перебувають на сервері. Прикладна програма клієнта виконує запити для одержання доступу або зміни даних через сервер прикладних програм або сервер Remote Data Broker (Віддалений брокер-сервер даних) – RDB. Зазвичай бізнес-правила розташовуються саме на сервері RDB. Якщо клієнт, сервер і бізнес-правила розподілені по окремих комп'ютерах, розроблювач може оптимізувати доступ до даних і підтримувати їхню цілісність при звертанні до них з будь-яких прикладних програм у всій системі.

Архітектура "клієнт-сервер" має ряд переваг:

1. Більшість обчислювальних процесів відбувається на сервері, що знижує вимоги до обчислювальних потужностей комп'ютера клієнта.

2. Мінімізується мережний трафік за рахунок посилки сервером клієнтові тільки тих даних, які він запитував. Наприклад, якщо необхідно зробити з файлу обсягом 10 000 записів вибірку, результатом якої будуть усього 2 записи, сервер виконає запит і перешле клієнтові набір даних з 2 записів.

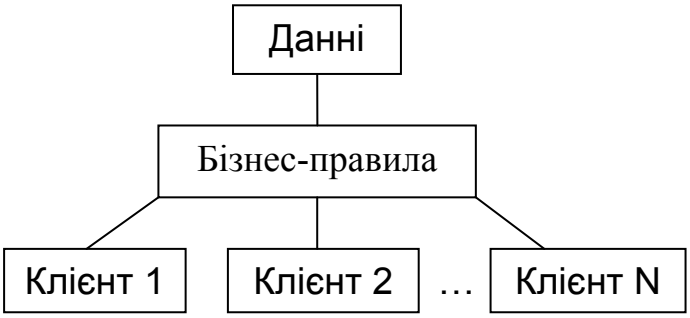



Рис. 1.2. Трирівнева модель "клієнт-сервер"

3. Спрощується нарощування обчислювальних потужностей в умовах розвитку програмного забезпечення й зростання обсягів даних, що оброблюються. Простіше й дешевше збільшити обчислювальну потужність сервера або замінити його на потужніший, ніж нарощувати потужності 100-500 комп'ютерів клієнтів.

4. База даних краще захищена від несанкціонованого доступу, тому що являє собою, як правило, єдиний файл, де утримуються таблиці, обмеження цілісності та інші компоненти. Зламати таку БД, навіть при наявності наміру, досить важко. Значно збільшується її захищеність від введення невірних значень, оскільки SQL-сервер проводить автоматичну перевірку їх відповідності накладеним обмеженням та автоматично виконує бізнес-правила. Окрім того, сервер відслідковує рівні доступу для кожного користувача й блокує здійснення спроб виконання недозволених для нього дій. Усе це дозволяє говорити про значно вищій рівень безпеки, що забезпечує посилальну й смислову цілісність інформації.

5. Сервер реалізує управління трансакціями й запобігає спробам одночасної зміни одних й тих самих даних. Різні рівні ізоляції трансакцій дозволяють визначити поведінку сервера при виникненні ситуацій одночасної зміни даних.

6. Безпека системи зростає за рахунок переносу більшої частини бізнес-правил на сервер. Падає питома вага взаємовиключних бізнес-правил у ПЗ клієнта, яке виконує різні операції над БД. У більшості випадків суперечливі дії будуть відхилені через автоматичне відстеження SQL-сервером посилальної цілісності даних.

### 1.4. Життєвий цикл бази даних

Життєвий цикл БД нерозривно пов'язаний з життєвим циклом інформаційної системи [1, 3, 4, 6, 8]. Він містить у собі такі етапи (рис. 1.3) [4]:

***1. Планування розробки БД.*** Планування найефективнішого способу реалізації етапів життєвого циклу системи.

***2. Визначення вимог до системи.*** Визначення діапазону дії й обмежень БД, складу його користувачів і областей застосування.

***3. Збір і аналіз вимог користувачів.*** На цьому етапі виконується збір і аналіз вимог користувачів із усіх можливих областей застосування.

***4. Проектування БД.*** Повний цикл розробки включає концептуальне, логічне й фізичне проектування БД, а також розробку прикладних програм клієнтів.

***5. Розробка прикладних програм.*** Визначення інтерфейсу користувача, переліку й функцій прикладних програм, які працюють із БД.

***6. Створення прототипів (необов'язково).*** Тут створюється робоча модель БД, яка дозволяє розробникам або користувачам представити й оцінити остаточний вид і способи функціонування системи.

***7. Реалізація.*** На цьому етапі в програмному коді реалізується концептуальна зовнішня й внутрішня функціональність БД і прикладного ПЗ, які визначені на попередніх етапах.

***8. Конвертування й завантаження даних*** полягає в перетворенні й завантаження даних (і прикладних програм) зі старої системи в нову.

***9. Тестування.*** БД тестується з метою виявлення помилок, а також її перевірки на відповідність усім вимогам, які висунуті користувачами.

***10. Експлуатація й супровід.*** На цьому етапі БД вважається повністю розробленим і реалізованим. Надалі уся система буде перебувати під постійним спостереженням і відповідним чином підтримуватися. При потребі у прикладні програми, що функціонують, можуть вноситися зміни, які відповідають новим вимогам. Вони реалізуються за допомогою повторного виконання деяких етапів життєвого циклу, які перераховані раніше.

Для малих БД з невеликою кількістю користувачів життєвий цикл може виявитися не дуже складним. Однак він може стати надзвичайно складним при проектуванні середовища великих БД, з десятками й навіть тисячами користувачів, сотнями запитів та прикладних програм.


Рис. 1.3. Життєвий цикл БД

Слід визнати, що наведені етапи не є строго послідовними, а включають деяку кількість повторів попередніх кроків у вигляді циклів зворотного зв'язку. Наприклад, при проектуванні БД можуть виникнути проблеми, для розв'язання яких буде потрібно повернутися до етапу збору й аналізу вимог. Цикли зворотного зв'язку можуть виникати майже між усіма етапами. Ми показали найбільш очевидні з них.

## 1.5. КОНТРОЛЬНІ ПИТАННЯ

1. Що таке інформаційна система організації та з яких компонентів вона складається?
2. Як ручні картотеки пристосовані для збереження та обробки даних?
3. Дайте визначення файлової системи з точки зору зберігання даних і управління ними.
4. До чого приводять відомі вам обмеження файлових систем як сховищ даних?
5. Що таке база даних? Сформулюйте визначення.
6. Які переваги та недоліки СУБД ви знаєте?
7. Порівняйте відомі вам архітектури СУБД.
8. Які недоліки СУБД архітектури "файл-сервер" вам відомі?
9. Які переваги СУБД архітектури "клієнт-сервер" ви знаєте?
10. Дайте характеристику моделям архітектури "клієнт-сервер.
11. Яка функціональна різниця між серверною та клієнтською частинами СУБД архітектури "клієнт-сервер"?
12. Що таке бізнес-правила? Де вони реалізуються?
13. Що ви знаєте про етапи життєвого циклу бази даних?

*Вивчення цього матеріалу розкриє поняття інформаційної системи організації, також допоможе студентові вибрати оптимальні засоби і методи зберігання і управління даними. Опис та порівняльний аналіз різних архітектур СУБД допоможе обрати більш ефективну, захищену та надійну СУБД.*

# 2. Концептуальне й логічне проектування БД

*Мета розділу – ознайомлення з концептуальним та логічним проектуванням БД, знайомство з історією створення реляційної моделі, структурою даних та описом відносин в реляційной моделі.*

## 2.1. Концептуальне проектування БД

Безпосереднє проектування БД (рис. 1.3) починається з концептуального моделювання. Це найбільш загальний опис, який називається ***схемою БД.*** Схема створюється за допомогою мови визначення даних СУБД, обраної для реалізації проекту. Вона не дає можливості представити дані так, щоб створену схему розуміли користувачі всіх категорій.

Для виходу із ситуації, що склалася, представлення даних здійснюють за допомогою ***моделі даних*** – інтегрованого набору понять для їхнього опису, зв'язків між ними й обмежень, що накладаються на них [1, 2, 4, 6, 9…11]. Це необхідно для того, щоб гарантувати однозначне розуміння всіма учасниками проекту набору правил, за якими може бути побудована БД, типи доступних операцій з даними, набір обмежень, який підтримує цілісність та гарантує коректність використовуваних даних.

***Концептуальне проектування БД*** полягає у виборі або створенні моделі представлення даних технологічного процесу організації, яка не залежить від будь-яких аспектів її фізичного розміщення, організації й обробки даних [4].

Вибір або створення моделі представлення даних базується на аналізі вимог користувачів кінцевої системи. Модель не залежить від типу обраної СУБД, набору прикладних програм, які створені або будуть створені, мов програмування, що використаються, типу обраної обчислювальної платформи і т.д. Модель представлення даних, яка отримана в результаті концептуального моделювання, є джерелом інформації для фази логічного проектування БД. Якщо жодна з існуючих моделей (реляційна, мережна, ієрархічна, об'єктно-орієнтована і т.д.) вам не підійшла, то у цьому випадку існує необхідність створювати власну модель представлення даних.

Далі будемо вважати, що за результатами концептуального проектування обрана реляційна модель виходячи з того, що 80% сучасних СУБД підтримують саме цю модель. А багато нереляційних систем тепер забезпечуються реляційним інтерфейсом користувача, незалежно від базової моделі даних, яка використовується.

## 2.2. Історія реляційної моделі

Реляційна модель уперше була опублікована Є.Ф. Коддом (E.F. Codd) в 1970 році в статті “Реляційна модель даних для великих банків даних, що використовуються спільно” [1, 4, 6, 7, 9, 10, 12]. Публікацію цієї статті прийнято вважати поворотним пунктом в історії розвитку систем БД, хоча слід зазначити, що ще раніше була запропонована модель, заснована на множені (Childs, 1968).

Мета створення реляційної моделі формулювалася так [4]:

1. Забезпечення максимально можливого ступеня незалежності прикладних програм від даних.

2. Створення міцного фундаменту для розв'язання семантичних питань, а також проблем несуперечності й надмірності даних.

3. Розширення мов управління даними за рахунок включення операцій над множинами.

Найбільш значні дослідження можливостей реляційної моделі були проведено у рамках трьох проектів.

Перший з них розроблявся наприкінці 70-х років у дослідницькій лабораторії корпорації IBM у місті Сан-Хосе, штат Каліфорнія, під керівництвом Астрахана (Astrahan), Його результатом стало створення системи за назвою “System R”, що була прототипом істинно реляційної СУБД [12]. Цей проект був задуманий з метою одержання реальних доказів практичності реляційної моделі. Він став найважливішим джерелом інформації про такі проблеми реалізації, як управління паралельністю, оптимізація запитів, управління трансакціями, забезпечення безпеки й цілісності даних, технологія оновлення, облік людського фактору й розробка інтерфейсу користувача.

Виконання проекту стимулювало публікацію багатьох науково-дослідницьких статей і створення інших прототипів реляційних СУБД, і все це в підсумку дозволило [4]:

по-перше, розробити мову управління даними SQL, або по літерах “S-Q-L” іноді за допомогою мнемонічного імені “See-Quell”, який з тих пір придбав статус формального стандарту ISO (International Organization for Standardization) і в цей час є фактичним галузевим стандартом мови реляційних СУБД.

по-друге, створити різні комерційні реляційні СУБД, які вперше з'явилися на ринку на початку 80-х років, таких як DB2 і SQL/DS корпорації IBM, а також ORACLE корпорації ORACLE Corporation.

Другим проектом, який зіграв помітну роль у розробці реляційної моделі даних, був проект INGRESS (Interactive Graphics Retrieval System), робота над яким проводилася у Каліфорнійському університеті (місто Берклі) майже в той самий час, що й над проектом System R [12]. Проект INGRESS включав розробку прототипу реляційної СУБД і переслідував ту ж саму мету, що й проект System R. Ці дослідження привели до появи академічної версії INGRESS, яка внесла істотний внесок у загальне визнання реляційної моделі даних. Пізніше від даного проекту відбрунькувалися комерційні продукти INGRESS фірми Relational Technology Inc. (тепер Ca-Open Ingres фірми Computer Associates) і Intelligent Database Machine фірми Britton Lee Inc.

Третім проектом була система Peterlee Relational Test Vehicle наукового центру корпорації IBM, розташованого у місті Петерли, Великобританія (Todd, 1976) [12]. Цей проект був більш теоретичний, ніж проекти System R і INGRES. Його результати мали дуже велике й навіть принципове значення, особливо в таких галузях, як обробка запитів і

оптимізація, а також функціональні розширення системи [4].

Крім того, пізніше були запропоновані деякі розширення реляційної моделі даних, призначені для найбільш повного й точного вираження змісту даних (Codd, 1979), для підтримки об'єктно-орієнтованих понять (Stonebraker and Rowe, 1986), а також для підтримки дедуктивних можливостей (Gardarin and Valduriez, 1989).

Таким чином, комерційні системи на основі реляційної моделі даних почали з'являтися наприкінці 70-х - початку 80-х років. У цей час існує кілька сотень типів різних реляційних СУБД як для мейнфреймів, так і для персональних комп'ютерів, хоча багато з них не повністю задовольняють точному визначенню реляційної моделі даних. Прикладами реляційних СУБД для персональних комп'ютерів є СУБД Oracle та MySQL від Oracle, Access і Foxpro від Microsoft, Paradox і Interbase фірми Borland, Rbase фірми Microrim і т.д.

## 2.3. Структура даних у реляційній моделі

### 2.3.1. Відношення

Реляційна модель заснована на математичному понятті відношення [4, 6, 12, 13]. Припустимо, що у нас є дві множини, $D_1$ і $D_2$, де $D_1=\{2,4\}$ й $D_2=\{1,3,5\}$. ***Декартовим добутком*** цих двох множин (позначається як $D_1 \times D_2$) називається набір з усіх можливих пар, у яких першим йде елемент множини $D_1$, а другим – елемент множини $D_2$. Альтернативний спосіб визначення цього добутку полягає у пошуку усіх комбінацій елементів, де першим йде елемент множини $D_2$, а другим – елемент множини $D_1$. У результаті одержимо:

D1 × D2={(2, 1), (2, 3), (2, 5),(4, 1), (4,3), ( 4, 5)}.

Будь-яка підмножина цього декартового добутку є відношенням. Наприклад, у ньому можна виділити відношення *R*:

R = {(2, 1), (4, 1)}.

Для визначення тих можливих пар, які будуть входити у відношення, можна задати деякі умови їх вибірки. Наприклад, якщо звернути увагу на те, що відношення R містить усі можливі пари, у яких другий елемент рівний 1, то визначення відношення R можна сформулювати у такий спосіб:

$R = \{(x, y),| \ x \in D_1, y \in D_2, y = 1\}$.

На основі тих же множин можна сформувати інше відношення – *S*, у якім перший елемент завжди повинен бути у два рази більше другого. Тоді визначення відношення *S* можна сформулювати так:

$S = \{(x, y),| \ x \in D_1, y \in D_2, x = 2 \cdot y\}$.

У даному прикладі тільки одна можлива пара даного декартового добутку відповідає цій умові:

S = {(2, 1)}.

Поняття відношення можна легко поширити й на три множини. Нехай є три множини: – $D_1$, $D_2$ і $D_3$. Декартів добуток $D_1 \times D_2 \times D_3$ цих трьох множин є набір, який складається з усіх можливих трійок елементів, у яких першим йде елемент множини $D_1$, другим – елемент множини $D_2$, а третім – елемент множини $D_3$. Будь-яка підмножина цього декартового добутку є відношенням. Обчислимо декартів добуток трьох множин $D_1=\{1,3\}$, $D_2=\{2,4\}$ і $D_3=\{5,6\}$:

$D_1 \times D_2 \times D_3$ = {(1, 2, 5), (1, 2, 6), (1, 4, 5), (1, 4, 6), (3, 2, 5), (3, 2, 6), (3, 4, 5), (3, 4, 6)}.

Будь-яка підмножина з наведених вище трійок елементів є відношенням. Збільшуючи кількість множин, можна дати узагальнене визначення відношення на $n$ доменах.

Нехай є $n$ множин $D_1, D_2, \ldots, D_n$. Декартовий добуток для цих $n$ множин можна визначити в такий спосіб:

D1 × D2 × … × Dn = {() | d1 є D1, d2 є D2, … dn є Dn}.

Звичайно цей вираз записують у такому символічному виді: $\overset{n}{\underset{i=1}{X}} D_i$.

Будь-яка множина *n-арних* кортежів цього декартового добутку є відношенням *n* множин. Зверніть увагу на те, що для визначення цих відношень необхідно вказати множини, або домени, з яких буде здійснюватись відбір значень.

Фізичним відображенням відношення є таблиця.

***Відношення*** *– це плоска таблиця, що складається зі стовпців і рядків.*

У реляційній моделі відношення використовуються для зберігання інформації про суб'єкти (сутностях), які представлені у БД.

### 2.3.2. Опис структури відношення

***Атрибут*** *– це пойменований стовпець відношення.*

Відношення звичайно має вигляд двовимірної таблиці, у якій рядки відповідають окремим записам, а стовпці – атрибутам. При цьому атрибути можуть розташовуватися у будь-якому порядку – незалежно від їх пере порядкування відношення буде залишатися тим самим, а тому мати той же зміст. Кожний атрибут реляційної БД визначається на деякому домені.

***Домен*** *– це набір припустимих значень для одного або декількох атрибутів.*

Домени це надзвичайно потужний компонент реляційної моделі. Вони можуть відрізнятися для кожного з атрибутів, але два й більш атрибутів можуть бути визначені на одному домені. За допомогою доменів користувач може централізовано розкривати зміст і джерело значень, які одержують атрибути. У результаті при виконанні реляційної операції системі доступно більше інформації. Це дозволяє їй уникнути семантично некоректних операцій. Наприклад, безглуздо порівнювати

прізвище автора з назвою книги, навіть якщо для обох цих атрибутів в якості домену використовуються рядки символів. З іншого боку, помісячна орендна плата об'єкта нерухомості й кількість місяців, протягом яких він здавався в оренду, належать різним доменам (перший атрибут має грошовий тип, а другий — цілочисловий). Однак множення значень із цих доменів є припустимою операцією. Як випливає із цих двох прикладів, забезпечити повну реалізацію поняття домена зовсім непросто, а тому в багатьох реляційних СУБД вони підтримуються не повністю, а лише частково.

Елементами відношення є кортежі, або рядки таблиці. Кортежі можуть розташовуватися у будь-якому порядку, при цьому відношення буде залишатися тим же самим, а відповідно, і мати однаковий зміст.

***Кортеж** – це рядок відношення.*

Опис структури відношення разом зі специфікацією доменів і будь-якими іншими обмеженнями можливих значень атрибутів іноді називають його заголовком (або змістом (intension)). Звичайно воно є фіксованим, доти, доки зміст відношення не змінюється за рахунок додавання в нього додаткових атрибутів. Кортежі називаються розширенням (extension), станом (state) або тілом відношення, яке постійно змінюється.

***Ступінь відношення** визначається кількістю атрибутів, яку воно містить.*

Відношення тільки з одним атрибутом має ступінь 1 і називається унарним (unary) відношенням (або 1-арним кортежем). Відношення із двома атрибутами називається бінарним (binary), відношення із трьома атрибутами – тернарним (ternary), а для відношення з більшою кількістю атрибутів використовується термін n-арний ( n-ary). Визначення ступеня відношення є частиною заголовка відношення.

***Кардинальність** – це кількість кортежів, яку містить відношення.*

Ця характеристика змінюється при кожному додаванні або вилученні кортежів. Кардинальність є властивістю тіла відношення й визначається поточним станом відношення в довільно взятий момент.

**2.3.3. *Реляційна схема** – це ім'я відношення, за яким йде множина пар імен атрибутів і доменів.*

Наприклад, для атрибутів *A1, A2, …, An* з доменами *D1, D2, …, Dn* реляційною схемою буде множина *{A1:D1, A2:D2, … An:Dn}*. Відношення *R*, задане реляційною схемою *S*, є множиною відображень імен атрибутів на відповідні їм домени. Таким чином, відношення *R* є безліччю таких *n*-арних кортежів *{A1:d1, A2:d2, … An:dn}*, де *{d1:D1, d2:D2, … dn:Dn}*.

Кожний елемент *n*-арного кортежу складається з атрибуту та його значення. Звичайно при записі відношення у вигляді таблиці імена атрибутів перелічуються у заголовках стовпців, а кортежі утворюють рядки формату $d_1, d_2, \ldots, d_n$, де кожне значення береться з відповідного домену. Таким чином, у реляційній моделі відношення можна представити як довільну підмножина декартового добутку доменів атрибутів, тоді як таблиця – це всього лише фізичне відображення такого відношення.

### 2.3.4. Властивості відношень

1. Відношення має унікальне ім'я в БД.
2. Кожна комірка відношення містить тільки неподільне значення.
3. Кожний атрибут має унікальне ім'я.
4. Значення атрибута беруться з одного домену.
5. Порядок розташування атрибутів не має ніякого значення.
6. Кожний кортеж відношення є унікальним.
7. Теоретично порядок розташування кортежів у відношенні не має ніякого значення. (Однак на практиці цей порядок може суттєво вплинути на ефективність доступу до даних.)

Більша частина властивостей відношень БД походить від властивостей математичних відношень.

1. Оскільки відношення є множиною, то порядок розташування в ньому елементів не має значення. Отже, порядок кортежів у відношенні несуттєвий.

2. У множині немає повторюваних елементів. Аналогічно, відношення не може містити кортежів-дублікатів.

3. При обчисленні декартового добутку множин із простими однозначними елементами (наприклад, цілочисловими значеннями), кожен елемент у кожному кортежі має єдине значення. Аналогічно, кожна комірка відношення містить тільки одне значення. Однак математичне відношення не має потреби в нормалізації. Кодд запропонував заборонити наявність груп, що повторюються, з метою спрощення реляційної моделі даних.

4. Набір можливих значень для даної позиції відношення визначається множиною, або доменом, на якім визначається ця позиція. Відносно БД усі значення у кожному атрибуті повинні походити від того домену, на якому визначено атрибут.

Однак у математичному відношенні порядок проходження елементів у кортежі має значення. Наприклад, припустима пара значень (1, 2) зовсім відмінна від припустимої пари (2, 1). Це твердження невірне для відношень у реляційній моделі, де спеціально обмовляється, що порядок атрибутів не є суттєвим. Однак, якщо структура відношення вже визначена, то порядок елементів у кортежах його тіла повинен відповідати порядку імен атрибутів.

**2.3.5.** ***Реляційні ключі*** необхідні для унікальної ідентифікації кожного окремого кортежу відношення за значеннями його атрибутів.

***Суперключ (super key)*** *– це атрибут або множина атрибутів, яка єдиним образом ідентифікує кортеж даного відношення.*

Оскільки суперключ може містити додаткові атрибути, які необов'язкові для унікальної ідентифікації кортежу, нас будуть цікавити суперключі, що складаються тільки з тих атрибутів, які дійсно необхідні для унікальної ідентифікації кортежів. Таким чином, до складу суперключа може входити декілька потенційних ключів

***Потенційний ключ*** *– це суперключ, який не містить підмножини, що також є суперключем даного відношення.* Відношення може мати

декілька потенційних ключів.

Якщо ключ складається з декількох атрибутів, то він називається складеним ключем.

Потенційний ключ для даного відношення володіє двома властивостями.

***Унікальність.*** У кожному кортежі відношення значення ключа єдиним образом ідентифікують цей кортеж.

***Мінімальність.*** Жоден з атрибутів не може бути виключений з ключа без порушення унікальності.

Зверніть увагу на те, що будь-який конкретний набір кортежів відношення не можна використовувати для доказу того, що якийсь атрибут або комбінація атрибутів є потенційним ключем. Той факт, що в деякий момент часу не існує значень-дублікатів, зовсім не означає, що їх не може бути взагалі. Однак наявність значень-дублікатів у конкретному існуючому наборі кортежів цілком може бути використане для демонстрації того, що деяка комбінація атрибутів не може бути потенційним ключем. Для ідентифікації потенційного ключа потрібно знати значення атрибутів, які використовуються, в "реальному світі". Тільки це дозволить обґрунтовано ухвалити рішення щодо можливості існування значень-дублікатів. Виходячи тільки з подібної семантичної інформації, можна гарантувати, що деяка комбінація атрибутів є потенційним ключем відношення.

***Первинний ключ*** *– це потенційний ключ, який обраний для унікальної ідентифікації кортежів усередині відношення.*

Оскільки відношення не містить кортежів-дублікатів, завжди можна унікальним образом ідентифікувати кожний його рядок. Це значить, що відношення завжди має первинний ключ. У найгіршому випадку вся множина атрибутів може використовуватися як первинний ключ, але звичайно, щоб розрізнити кортежі, досить використовувати трохи меншу підмножину атрибутів. Потенційні ключі, які не обрані в якості первинного ключа, називаються альтернативними ключами.

***Зовнішній ключ*** *– це атрибут або множина атрибутів усередині відношення, яке відповідає потенційному ключу якогось (може бути, того ж самого) відношення.*

Відношення, де перебуває потенційний ключ, називають базовим, а іноді цільовим або батьківським відношенням. Відношення із зовнішнім ключем називають дочірнім відношенням.

**2.3.6.** ***Реляційна БД*** *– набір нормалізованих відношень.*

Реляційна БД складається з відношень, структура яких визначається за допомогою особливих методів, які названі нормалізацією (normalization).

### 2.3.7. Опис структури реляційних даних на різних етапах проектування

На концептуальному рівні описується модель представлення даних у строгих математичних термінах реляційної алгебри, які визначають їхню структуру, методи управління й методи забезпечення цілісності даних. У

будь-який реляційній СУБД передбачається, що користувач сприймає БД як набір зв'язаних між собою таблиць. Подібне сприйняття не відноситься до фізичної структури БД, де мова йде про її реалізацію на вторинних носіях із зазначенням структур зберігання та методів доступу, які використовуються для організації їх ефективної обробки. На етапі фізичного проектування відношення називається ***файлом*** (file), кортежі – ***записами*** (records), а атрибути – ***полями*** (fields) і т.д. (табл. 2.1).

### 2.4. Логічне проектування БД

***Логічне проектування БД*** це процес створення схеми БД (логічної моделі БД) з урахуванням обраної моделі представлення даних, але незалежної від типу цільової СУБД та інших фізичних аспектів реалізації [4, 6].

Ціль логічного проектування полягає у створенні логічної моделі БД для частини підприємства, яка досліджується. Модель представлення даних, яка отримана на етапі концептуального проектування, є основою логічної моделі даних, що враховує особливості бізнес-процесів організації та їх реалізації в обраній СУБД. Однак, на цьому етапі ігноруються всі інші аспекти обраної СУБД – наприклад, будь-які особливості фізичної організації її структур зберігання даних та побудови індексів.

Таблиця 2.1

**Терміни, що описують структуру даних в реляційній моделі**

| Концептуальне проектування | Логічне проектування | Фізичне Проектування |
|---|---|---|
| Відношення | Таблиця | Файл |
| Кортеж | Рядок | Запис |
| Атрибут | Стовпець | Поле |
| Степінь відношення | Кількість стовпців | Кількість полів |
| Кардинальність | Кількість рядків | Кількість записів |
| Ключ | Ключ | Індекс |

Логічна модель даних є джерелом інформації для етапу фізичного проектування та забезпечує розроблювача фізичної БД засобами знаходження компромісів, які необхідні для досягнення поставлених цілей. Вона також відіграє важливу роль на етапі експлуатації та супроводу вже готової системи. При правильно організованому супроводі, логічна модель даних, яка підтримується в актуальному стані, дозволяє точно й наочно представити будь-які внесені в БД зміни, а також оцінити їхній вплив на прикладні програми й використання даних, які вже наявні у БД.

Логічна модель, що віддзеркалює особливості представлення про функціонування підприємства одночасно багатьох типів користувачів, називається ***глобальною логічною моделлю даних***. Існує два основні підходи до створення глобальної логічної моделі даних – це централізований підхід та підхід на основі інтеграції уявлень.

***Централізований підхід*** полягає у злитті вимог окремих користувачів, що виражені у вигляді їхніх різних уявлень, у єдиний набір

вимог усіх користувачів, який використовується для створення глобальної логічної моделі даних.

Характерною рисою централізованого підходу є те, що списки вимог складаються до створення глобальної логічної моделі даних. Цей підхід застосуємо тільки за умови, що БД не занадто велика або складна.

***Метод інтеграції уявлень*** полягає в злиття окремих локальних логічних моделей даних, що відображають уявлення різних груп користувачів, у єдину глобальну логічну модель даних.

Цей підхід більш керований, оскільки уся робота попередньо розподіляється на більш дрібні й легко контрольовані частини. Труднощі, звичайно, виникають лише при спробах злиття локальних моделей даних, що створені різними розроблювачами, які можуть використовувати різні терміни для однакових понять або, навпаки, використовувати один термін для різних понять.

Логічне проектування БД – найважливіші етап, що забезпечує загальний успіх розробки системи в цілому. Якщо створений проект не є точним віддзеркаленням методів роботи та структури підприємства, то буде дуже складно, якщо взагалі можливо, визначити усі уявлення (зовнішні схеми), які необхідні користувачам, або організувати підтримку цілісності БД. Крім того, можуть виникнути труднощі з фізичною реалізацією БД або забезпеченням прийнятної продуктивності системи. У той самий час здатність адаптації до змін є ознакою вдало спроектованої БД. Отже, завжди має сенс витратити необхідні час та енергію на створення найкращої з можливих логічної моделі.

### 2.5. Приклад логічної моделі

Припустимо, що вам поставлене завдання створити БД, яка дозволяє оперативно надавати інформацію про відвідування студентами занять. Необхідно, щоб було відомо по кожному студентові, скільки годин і по якому предмету він був відсутній.

Розділимо інформацію, яка міститься у БД на два типи: ***довідкову*** та ***оперативну***. Виходячи із цього, у нашій БД обліку відвідування занять буде дві довідкові таблиці: «Студенти» та «Предмети», і одна таблиця з оперативною інформацією «Відвідуваність», де будемо вести облік (мал. 2.1). Залишається додати, що спочатку необхідно вносити дані в таблиці, які відносяться до довідкової інформації, а потім – до оперативної.

Зверніть увагу на те, що у стовпці «№ зал.» таблиці «Відвідуваність» перебувають тільки значення, що містяться в стовпці «№ зал.» таблиці «Студенти». А в стовпці «Шифр предм.» таблиці «Відвідуваність» перебувають тільки значення, що містяться в стовпці «Шифр» таблиці «Предмети». Таким чином, можна за № залікової книжки з таблиці «Відвідуваність» довідатися прізвище студента, а за шифром предмету визначити його назву. З іншого боку, можна сказати,

хто зі студентів не пропустив жодного заняття, і який предмет відвідують усі студенти. Спробуйте назвати прізвище добросовісного студента самостійно. Назвіть предмет, який ніхто не прогулює.

При проектуванні завжди дивіться на БД, що створюєте, із двох сторін: з боку розроблювача та з боку користувача. Для розроблювача важливо без проблем вносити в БД інформацію та витягати її в тому вигляді, який влаштовує користувача. Для користувача важливо одержувати інформацію в зручній для себе формі. Дотримати баланс цих інтересів непросто. Особливо, коли є ліміт часу, за який користувач прагне одержати необхідні дані.

На рис. 2.1. представлен погляд з боку розроблювача БД. Спробуйте самостійно зобразити таблицю, у якій вам було б зручно побачити, скільки й по якому предмету було пропущено занять кожним із студентів. Цим питанням ми займемося в другій половині нашого курсу, коли будемо вивчати «Уявлення даних».

Така простота не повинна вас вводити в оману. Тільки досвід створення БД допоможе Вам виробити свої власні прийоми й підходи. У запропонованому курсі розглянуті концепції й прийоми проектування БД, з якими корисно ознайомитися та узяти їх за основу.

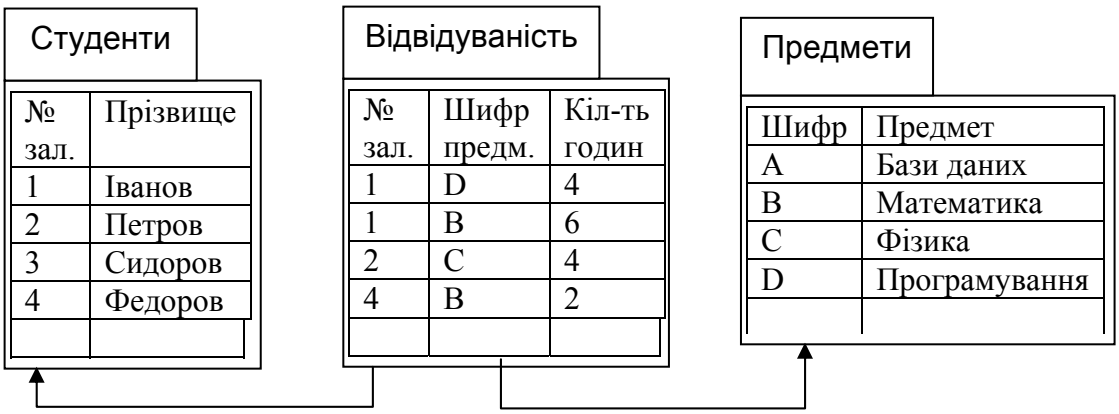



Рис. 2.1. БД обліку відвідуваності занять

## 2.6. КОНТРОЛЬНІ ПИТАННЯ

1. Ціль та суть концептуального проектування БД?
2. На яких складових базується реляційна структура даних?
3. Яким чином пов’язані декартів добуток та двомірна таблиця?
4. Що складає структуру відношення та які характеристики воно має?
5. Які властивості відношень Вам відомі?
6. Дайте визначення відомим Вам реляційним ключам?
7. Які властивості потенційного ключа Вам відомі?
8. Ціль та суть логічного проектування БД?
9. Порівняйте відомі вам підходи до створення логічної моделі БД?

*Знайомство з цим матеріалом допоможе студентові розібраться у питаннях концептуального та логічного проектування БД, та самостійно розробити логічну модель для запропонованого технологічного процесу.*

# 3. Нормалізація відношень при проектуванні БД

*Мета розділу – ознайомлення з процедурою нормалізації, її різними рівнями і критеріями визначення того або іншого рівня нормалізації.*

## 3.1. Процедура нормалізації

З одного боку, процес проектування структури БД є процесом творчим, неоднозначним, з іншого боку, вузлові його моменти можуть бути формалізовані [4…7, 10, 12]. У процесі розробки логічна модель даних постійно тестується й перевіряється на відповідність вимогам користувачів. Коректність логічної моделі даних забезпечується ***процедурою нормалізації***.

Процедура нормалізації БД полягає в усуненні ***надмірності даних*** та виявленні ***функціональних залежностей***. Усування ***надмірність даних*** гарантує компактність наборів даних за рахунок уникання їх зайвого дублювання та унеможливлює виникнення аномалій вставки, вилучення й оновлення кортежів після їхньої фізичної реалізації. ***Функціональна залежність*** – це залежність, що пов'язує атрибути в одному відношенні з єдиним значенням в іншім відношенні. Функціональну залежність для відношень ***A*** и ***B*** прийнято позначати як ***A→B***. Це поняття підводить “на один крок” до родинної концепції об'єднання відношень зв'язками типу один до одного (***1:1***) або один до багатьох (***1:M***).

Правила нормалізації або правила Кодда, як їх тепер прийнято називати, дуже прості й нечисленні, але досить суворі. У випадку застосування до відношень з даними кожне правило описує наступний рівень відповідності вимогам теорії реляційних БД і різні ступені нормалізації.

Існує п'ять різних рівнів нормалізації: перша нормальна форма (***1НФ***), ***2НФ***, ***3НФ***, нормальна форма ***Бойса-Кодда*** (***БКНФ***), ***4НФ***, ***5НФ***. Але дотепер жодна з реляційних СУБД не надає підтримки для всіх п'яти нормальних форм. Це відбувається через жорсткі вимоги до продуктивності. Суть справи полягає в тому, що в повністю нормалізованій БД для виконання запиту Вам буде потрібно з'єднати настільки багато таблиць, що продуктивність такої системи не зможе задовольнити при виконанні запитів користувачів. Тому на практиці використовують лише перші три – ***1НФ***, ***2НФ***, ***3НФ***.

## 3.2. Перша нормальна форма

Відношення представлене в ***першій нормальній формі*** (***1НФ***) тоді й тільки тоді, коли всі його атрибути містять тільки неподільні (атомарні) значення й у ньому відсутні групи атрибутів з однаковими за змістом значеннями, які повторюються у межах одного кортежу.

Неподільність значення атрибута говорить про те, що його не можна розділити на більш дрібні частини. Наприклад, якщо у полі

"Прізвище Ім'я По батькові" міститься прізвище, ім'я та по батькові читача, вимога неподільності не дотримується (рис. 3.1, *а*). Тут необхідно виділити в окремі атрибути ім'я та по батькові. У результаті вийде три атрибути відношення «Читачі»: «Прізвище», «Ім'я» і « По батькові» (рис. 3.1, *б*).

На більш дрібні частини можна розділити атрибути: «Місце народження» («Країна», «Адміністративне утворення», «Населений пункт»), «Місце видачі паспорта» («Країна», «Адміністративне утворення», «Населений пункт»), «Місце основної роботи» («Тип підприємства», «Назва підприємства»), «Місце проживання» («Країна», «Адміністративне утворення», «Населений пункт», «Житловий масив / Проспект / Вулиця / Провулок», «Будинок», «Корпус», «Квартира») (рис. 3.1, *б*).

*а*

| № п.п. | Атрибути | Ключ |
|---|---|---|
| 1. | № квитка читача | Перв. |
| 2. | Прізвище, Ім’я, По батькові | |
| 3. | Серія паспорта | По-тенц. |
| 4. | № паспорта | |
| 5. | Дата народження | |
| 6. | Місце народження | |
| 7. | Стать | |
| 8. | Місце видачі паспорта | |
| 9. | Дата видачі паспорта | |
| 10. | Місце проживання | |
| 11. | Місце основної роботи | |
| 12. | Посада | |

*б*

| № п.п. | Атрибути | Ключ |
|---|---|---|
| 1. | № квитка читача | Перв. |
| 2. | Прізвище | |
| 3. | Ім’я | |
| 4. | По батькові | |
| 5. | Серія паспорта | Потен-ц. |
| 6. | № паспорта | |
| 7. | Дата народження | |
| 8. | Країна місця народження | |
| 9. | Адмін. утв. місц. народж. | |
| 10. | Нас. пункт місц. народж. | |
| 11. | Стать | |
| 12. | Країна місц. вид. пасп. | |
| 13. | Адм.утв.місц.вид. пасп. | |
| 14. | Нас.пункт місц. вид. пасп. | |
| 15. | Дата видачі паспорта | |
| 16. | Країна місця проживання | |
| 17. | Адмін. утв. місц. прожив. | |
| 18. | Нас. пункт місц. прожив. | |
| 19. | Ж.м./Просп./Вул./Пров. | |
| 20. | Будинок | |
| 21. | Корпус | |
| 22. | Квартира | |
| 23. | Місце основної роботи | |
| 24. | Тип підприємства | |
| 25. | Назва підприємства | |
| 26. | Домашній телефон | |
| 27. | Робочий телефон | |
| 28. | Мобільний телефон | |

Рис. 3.1. Можливі специфікації ненормалізованого відношення «ЧИТАЧІ»

Не дотримується неподільність значення атрибута «Номер телефону», тому що в читача може бути кілька номерів або не бути телефону взагалі (рис. 3.1, *а*). На перший погляд цю проблему можна розв'язати так само, як і для прізвища, імені та по батькові, виділивши для найпоширеніших типів телефонів окремі атрибути (рис. 3.1, *б*). Однак, у цьому випадки, ми зіштовхнемося з групою атрибутів, що містять однакові за змістом значення в межах одного кортежу: «Домашній телефон», «Робочій телефон», «Мобільний телефон».

Щоб відношення «ЧИТАЧІ» (рис. 3.1, *б*) відповідало 1НФ, необхідно вилучити з нього групу атрибутів з номерами телефонів, які повторюються в в межах одного кортежу, в інше відношення разом з копією ключового атрибута «№ квитка читача» (рис. 3.2). Причому, виділимо для вказівки номера й типу телефону окремі атрибути. Таке рішення дозволяє: по-перше, ураховувати не тільки три зазначені типи телефону, але й додавати нові, по-друге, вказувати для кожного читача тільки ті типи телефонів, які в нього є, по-третє, можна вказати для будь-якого читача кілька однотипних телефонів або не вказати жодного номеру телефону.

### 3.3. Друга нормальна форма

Відношення представлене у ***другій нормальній формі (2НФ)*** тоді й тільки тоді, коли воно є в першій нормальній формі, і кожний неключовий атрибут повністю визначається первинним ключем, тобто щоб первинний ключ однозначно визначав кортеж і не був надлишковим (збігався із суперключем). Ті атрибути, які залежать тільки від частини суперключа, повинні бути виділені до окремих таблиць.

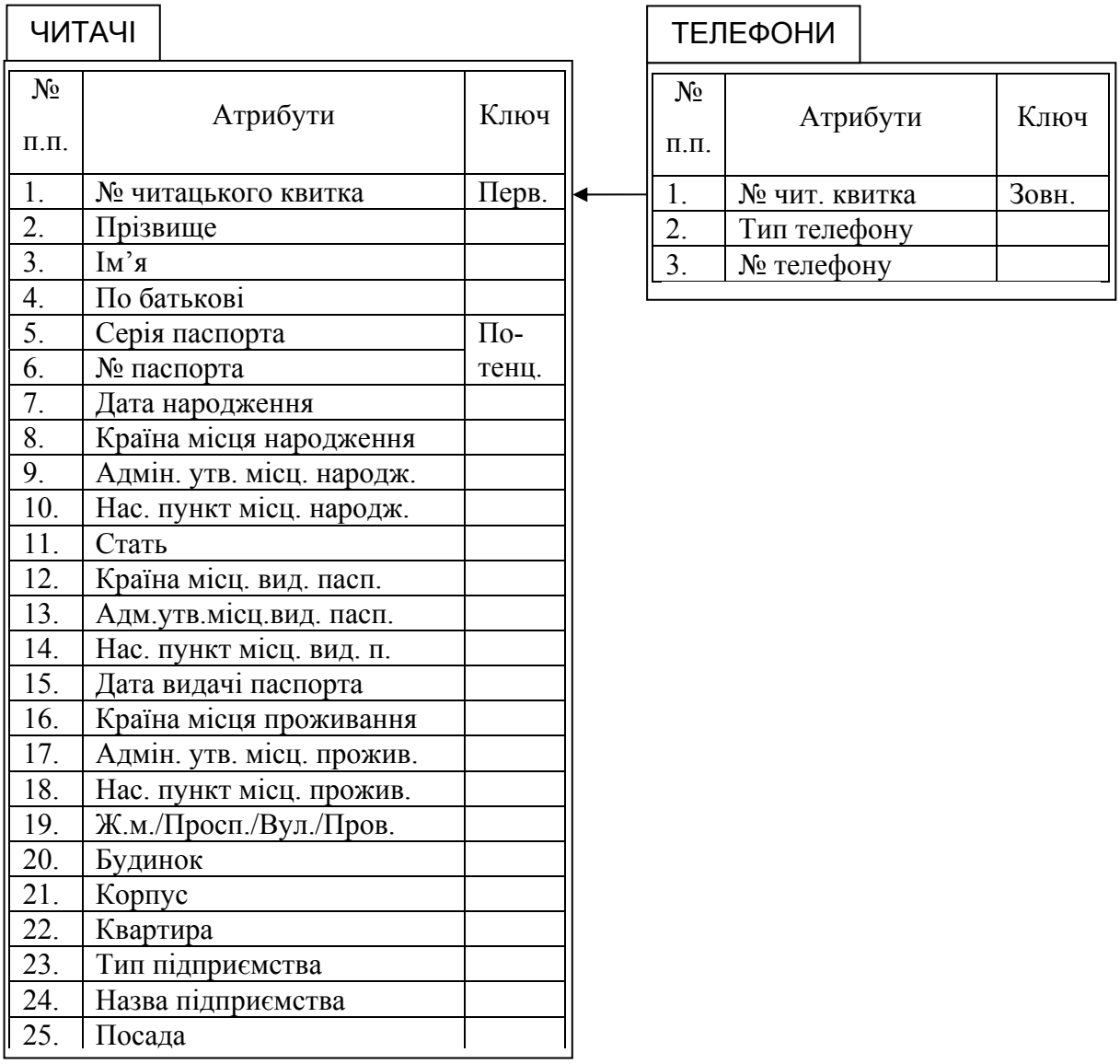



Рис. 3.2. Приведення відношення «ЧИТАЧІ» до 1НФ

Відношення «ЧИТАЧІ» (рис. 3.2) не є у 2НФ. У ньому кожен кортеж однозначно ідентифікується наступними атрибутами: «№ квитка читача», «Серія паспорта» і «№ паспорта». Сукупність цих атрибутів є ***суперключем*** цього відношення. Він складається із двох ***потенційних ключів,*** кожен з яких окремо може ідентифікувати кортежі відношення. Із двох потенційних ключів у якості первинного обраний той, довжина якого мінімальна. Наявність обох потенційних ключів обумовлена вимогами користувачів.

Привести відношення «ЧИТАЧІ» до 2НФ можна, якщо винести в окреме відношення атрибути 2 - 22, які стосуються паспортних даних, і копію первинного ключа «№ квитка читача» (рис. 3.3). Однак у результаті ми одержимо відношення «ПАСПОРТНІ ДАНІ», яке має такий же суперключ, як і у відношенні «ЧИТАЧІ», до приведення його до 2НФ. У нашім випадку подальша нормалізація відношення «ПАСПОРТНІ ДАНІ» неможлива.


Рис. 3.3. Відношення «ТЕЛЕФОНИ» і «ЧИТАЧІ» приведені до 2НФ

У відношенні «ТЕЛЕФОНИ» (рис. 3.3) на перший погляд здається, що якщо в описі логічної моделі обумовлено, що номер телефону читача вводиться повністю, включаючи код країни, оператора або населеного пункту, тоді потенційним є ключ, у складі якого всього один атрибут – «№ телефону». Однак, у двох різних читачів може бути однаковий домашній або робочій телефон. Отже, однозначно ідентифікувати кортежі можна по складеному суперключу, до якого входять атрибути «№ квитка читача» та «№ телефону». Ми бачимо, що суперключ не є надлишковим. Його склад збігається з первинним ключем. Значить відношення «ТЕЛЕФОНИ» презентовано у другій нормальній формі.

### 3.4. Третя нормальна форма

У загальному випадку 1НФ і 2НФ розглядаються як проміжні етапи в процесі нормалізації БД. Більша частина СУБД орієнтована на досягнення наступного ступеня нормалізації – третьою нормальною формою (3НФ). Це пов’язано з тим, що представлення відношень у 3НФ цілком достатньо майже для всіх практичних задач. При розробці винятково великих систем на надшвидкодіючих комп'ютерах, коли

необхідно забезпечити максимальне скорочення обсягів даних, бажане провести подальшу нормалізацію відношень.

***Відношення представлене у 3НФ*** тоді й тільки тоді, коли воно є у 2НФ й у ньому немає транзитивних залежностей між неключовими атрибутами, тобто значення будь-якого атрибута відношення, що не входить до первинного ключа, не залежить від значення іншого атрибута, що не входить до первинного ключа.

Це визначення – усього лише оригінальний спосіб виразити необхідність представлення системи пов'язаних відношень у такому виді, щоб значення атрибутів кожного відношення безпосередньо визначалися або ***суперключем***, або ***потенційним ключем*** цього відношення.

Існує метод розрахунків мінімального числа відношень, необхідних для представлення БД у 3НФ. У тому випадку, якщо ви склали список усіх функціональних залежностей для ваших даних, можна застосувати алгоритм Бернштейна, який описаний у будь-якому підручнику реляційної алгебри.

Візьмемо простий приклад з обліку товару на складі. Здається логічним, щоб у відношення з оперативною інформацією про товар на складі входили в числі багатьох три наступні атрибути: «Кількість товару», «Ціна за одиницю товару» та «Загальна вартість товару». Вони не є ключовими. Для одержання значення «Загальної вартості товару» необхідно перемножити значення, що перебувають у атрибутах: «Кількість товару» та «Ціна за одиницю товару». Тобто значення поля «Загальної вартості товару» залежить від двох атрибутів, що не входять до складу первинного ключа. Це суперечить визначенню 3НФ. Щоб дане відношення відповідало третій нормальній формі з нього необхідно виключити атрибут «Загальна вартість товару».

Відзначимо, що відношення «ТЕЛЕФОНИ» і «ЧИТАЧІ» (рис. 3.3) наведені до третьої нормальної форми. Це випливає з того, що вони представлені до другої нормальної форми й у них немає транзитивних залежностей.

Нижче наведене визначення варіанта 3НФ, яке називається нормальною формою ***Бойса-Кодда*** (Boyce – Codd) – ***БКНФ***, де встановлюються більш суворі вимоги.

***Відношення X представлено в нормальной формі Бойса-Кодда***, якщо в кожній нетривіальній функціональній залежності B→A B є суперключем.

### 3.5. Четверта та п'ята нормальні форми

Перш ніж закінчити розгляд правил Кодда, Вам буде запропонований короткий огляд двох останніх форм відношень реляційних БД. Вони призначені для усунення ще двох аномалій: багатозначна залежність та об'єднуюча залежність.

У відношенні X існує ***багатозначна залежність A→B,*** якщо в ньому можна виявити ситуації, де пара кортежів містить значення A, які дублюється, і одночасно існують інші пари кортежів, що отримані шляхом перестановки значень B, присутніх у першій парі.

Насамперед, для існування багатозначної залежності потрібне існування пар кортежів. A й B можуть бути як окремими атрибутами, так і об'єднанням деякого набору атрибутів. Тривіальна багатозначна залежність для A→B існує у випадку, коли B є підмножиною A або A поєднує B = XS (більше відношення містить вихідне відношення).

Існування багатозначної залежності породжує аномалію оновлення. 4НФ усуває нетривіальну багатозначну залежність у відношенні за допомогою створення менших відношень. Процес нормалізації полягає в створенні як можна більшого числа усе більш дрібних відношень з метою скорочення надмірності даних.

***Відношення X представлено в 4НФ*** тоді й тільки тоді, коли воно презентовано в БКНФ і для будь-якої багатозначної залежності A→B щодо цього можна сказати, що ця залежність або є тривіальної, або A є суперключем таблиці X.

***П'ята нормальна форма досягається в тому випадку***, коли відношення не може далі розбиватися на більш дрібні відношення за допомогою операції проектування.

Під операцією проектування розуміється декомпозиція, що не викликає втрати даних, і при якій відношення розбивається на частини таким чином, щоб для його одержання залишалася можливість об'єднання отриманих менших відношень.

### 3.6. Нормалізація – за й проти

Нормалізація відношень БД покликано усунути з них надлишкову інформацію. 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, заощаджується місце на носіях інформації. Однак у нормалізованій БД є й недоліки, насамперед практичного характеру.

Чим ширше число суб'єктів (сутностей), що охоплюються предметною областю, тим з більшого числа відношень буде полягати нормалізована БД. БД у складі великих систем, які керують життєдіяльністю організацій і підприємств, можуть містити сотні зв'язаних між собою відношень. Оскільки поріг людського сприйняття не дозволяє одночасно охоплювати велику кількість об'єктів з обліком їх взаємозв'язків, можна затверджувати, що зі збільшенням числа нормалізованих відношень зменшується цілісне сприйняття БД як системи взаємозалежних даних. Тому при розробці й експлуатації великих систем нерідкі ситуації, коли кожний співробітник уявляє собі

процеси, що протікають тільки в частині системи. Відомі випадки еволюційного створення таких систем, у яких принципи 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 приводить 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 представлені у 3НФ. Існують варіанти, коли має сенс розділити відношення на декілька дрібних, якщо частина представлених у ньому даних непостійна й часто оновлюється (оперативна інформація), а інші дані пасивні та змінюються в рідких випадках (довідкова інформація). Також є сенс об'єднати відношення при необхідності забезпечити високу швидкість реакції на запит. Можна навіть піти на дублювання даних у відношеннях, якщо це дозволить знизити витрати на обробку запитів, хоча формально не варто було б цього робити.

Необхідно також відзначити, що зв'язок між відношеннями доцільно здійснювати по атрибутах, повністю звільнених від семантичної залежності, яка породжується особливостями реалізації реальних процесів, що віддзеркалює БД. Для цього використовують спеціально виділені інкрементні атрибути однозначної ідентифікації кортежів усередині відношень. Такий прийом дозволяє звільнитися від необхідності перевизначення зв'язків між відношеннями при зміні в реальному житті правил обліку різних суб'єктів діяльності організації.

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

Якщо користувачеві немає необхідності деталізувати атрибути «Місце народження», «Місце видачі паспорта», «Місце основної роботи», «Місце проживання», то пошук компромісу між вимогами нормалізації й швидкодією для відношення «ЧИТАЧІ» (рис. 3.1, а) приводить до того, що, по суті, це відношення може не відповідати навіть 1НФ (рис. 3.4). Зверніть увагу, що паспортні дані читачів також не винесені в окреме відношення. Це дозволяє скоротити час виконання запиту, який видає всі наявні дані про читачів, за рахунок відсутності необхідності з'єднання відношень «ПАСПОРТНІ ДАНІ» та «ЧИТАЧІ».

На відміну від паспортних даних відомості про телефонні номери

читачів винесені в окреме відношення (рис. 3.4). Це пов'язане з тим, що у читача може бути як кілька контактних телефонів, так і не бути їх взагалі. Інакше кажучи, зв'язок 1:М між суб'єктами обліку та діяльності організації в переважній більшості випадків необхідно реалізувати за допомогою двох відношень. Зв'язок 1:1 у більшості випадків реалізується в одному відношенні.

На користь реалізації зв'язку 1:1 між суб'єктами обліку більш ніж в одному відношенні говорить наявність у вимогах користувачів необхідності одержання узагальнюючої інформації зі значення якого-небудь атрибута. Саме це виправдовує винесення даних про тип телефону й посади читача в окремі таблиці (рис. 3.4). У цьому випадку назва посади та типу телефону вказується один раз.

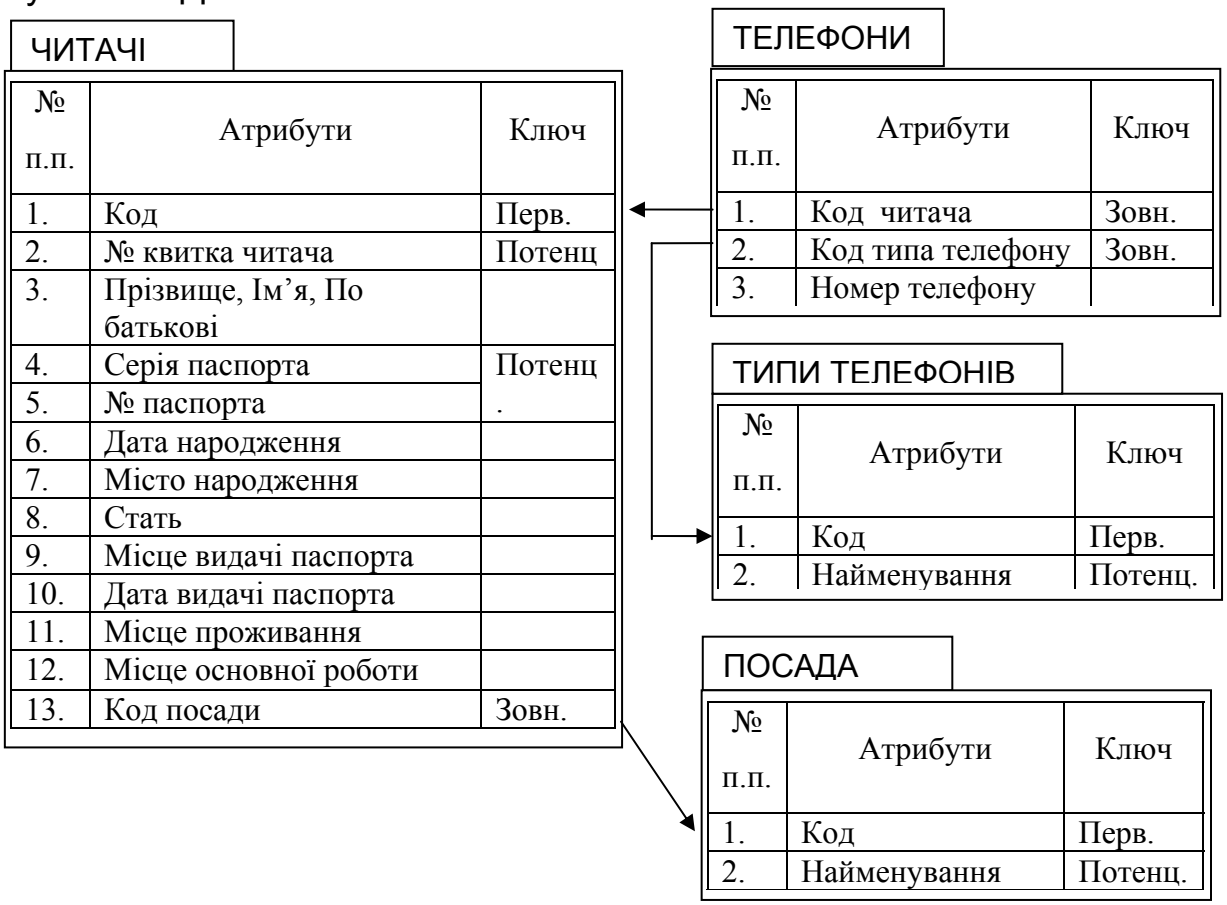
Якщо користувачі будуть вносити назву посади або тип телефону безпосередньо у відповідні атрибути відношень «ЧИТАЧІ» і «ТЕЛЕФОНИ» (рис. 3.3), то розроблювачі не зможуть гарантувати, що із БД запити будуть вибирати коректну інформацію. Це пов'язане з тим, що назву однієї й тієї ж посади або типу телефону оператор може внести із синтаксичною помилкою. У цьому випадку стандартні алгоритми обробки запитів користувачів до БД будуть інтерпретувати однакові за суттю значення відповідного атрибуту як різні.

Запит користувачів на одержання узагальнюючої інформації може бути сформульований так: «вивести на екран робочі телефони читачів із вказівкою їх прізвищ, імен, по батькові, місць роботи й посад». Або так: «вивести на екран контактні телефони всіх асистентів із вказівкою їх прізвищ, імен, по батькові».

Уведення інкрементних атрибутів «Код» дозволив повністю звільнитися від необхідності зміни властивостей ключових атрибутів, у зв'язку зі змінами правил обліку читачів у бібліотеці (рис. 3.4). Вони дозволили трохи компенсувати збільшення необхідного обсягу пам'яті для реалізації БД за рахунок скорочення місця для атрибута відношення, що пов'язує «ЧИТАЧІ» та «ТЕЛЕФОНИ» з відношеннями «ПОСАДА» та «ТИПИ ТЕЛЕФОНІВ» відповідно. Ці відношення можна було б зв'язати за значеннями атрибутів «Найменування посади» та «Найменування типу телефону» замість атрибутів «Код посади» та «Код типу телефону». Економія залежить від кількості знаків цих найменувань, яка закладена згідно з вимогами користувачів.Результат нормалізації відношення «ЧИТАЧІ» (рис. 3.4) може бути іншим. Він залежить від вимог користувачів, від досвіду та погляду розроблювача на процедуру нормалізації відношень, необхідних для розв'язання завдання, що поставлене. У нашім прикладі ми одержали з одного відношення «ЧИТАЧІ» (рис. 3.1, а) чотири відношення: «ЧИТАЧІ», «ТЕЛЕФОНИ», «ТИПИ ТЕЛЕФОНІВ» і «ПОСАДА» (рис. 3.4). Відношення «ЧИТАЧІ» ненормалізоване. Відношення «ТЕЛЕФОНИ» у 3НФ. Відношення «ТИПИ ТЕЛЕФОНІВ» і «ПОСАДА» представлені в 1НФ.

Одним з підсумків нормалізації відношень БД «БІБЛІОТЕКА» є діаграма зв'язків між ними (додаток А). Для компактності на ній

вказуються лише ключові атрибути відношень. Зв'язок між логічною й фізичною моделлю зручно показувати після етапу фізичного проектування БД.



ЧИТАЧІ

| № п.п. | Атрибути | Ключ |
|---|---|---|
| 1. | Код | Перв. |
| 2. | № квитка читача | Потенц |
| 3. | Прізвище, Ім’я, По батькові | |
| 4. | Серія паспорта | Потенц . |
| 5. | № паспорта | |
| 6. | Дата народження | |
| 7. | Місто народження | |
| 8. | Стать | |
| 9. | Місце видачі паспорта | |
| 10. | Дата видачі паспорта | |
| 11. | Місце проживання | |
| 12. | Місце основної роботи | |
| 13. | Код посади | Зовн. |

ТЕЛЕФОНИ

| № п.п. | Атрибути | Ключ |
|---|---|---|
| 1. | Код читача | Зовн. |
| 2. | Код типа телефону | Зовн. |
| 3. | Номер телефону | |

ТИПИ ТЕЛЕФОНІВ

| № п.п. | Атрибути | Ключ |
|---|---|---|
| 1. | Код | Перв. |
| 2. | Найменування | Потенц. |

ПОСАДА

| № п.п. | Атрибути | Ключ |
|---|---|---|
| 1. | Код | Перв. |
| 2. | Найменування | Потенц. |

Рис. 3.4. Результат нормалізації відношення «ЧИТАЧІ»

## 3.7. КОНТРОЛЬНІ ПИТАННЯ

1. Що забезпечує та гарантує процедура нормалізації?
2. Які нормальні форми відношень БД Вам відомі?
3. Коли відношення представлено в першій нормальній формі?
4. Коли відношення представлена в другій нормальній формі?
5. Коли відношення представлена в третій нормальній формі?
6. Які нормальні форми відношень використовують на практиці?
7. Які нормальні форми відношень рідко використовують на практиці?
8. В яких випадках в БД можна залишити відношення, що не є в 1НФ?
9. Як позбутися семантичної залежності в зв’язках між відношеннями?
10. Які особливості реалізації зв’язків 1:1 та 1:М в БД?

*Вивчення цього матеріалу допоможе студентові знаходити компроміс між вимогами нормалізації і необхідністю поліпшення швидкодії БД, що розробляється.*

# 4. Фізичне проектування структури даних

*Метою цього розділу є ознайомлення з фізичним проектуванням бази даних як способу фізичної реалізації логічного проекту, обґрунтування вибору тією або іншою цільової СУБД, а також знайомство з «еталонними» правилами, використовуючи які можна визначити приналежність СУБД до розряду дійсно реляційних систем.*

*Розділ містить коротку характеристику мови SQL та опис основних прийомів забезпечення цілістності реляційних даних.*

## 4.1. Завдання і цілі фізичного проектування

Під час логічного проектування була визначена структура БД (тобто набір її сутностей, атрибутів та зв'язків). Хоча логічна модель не залежить від конкретної цільовий СУБД, вона створювалася з урахуванням обраної моделі організації даних. Між логічним і фізичним проектуванням існує постійний зворотний зв'язок, тому що рішення прийняті на етапі фізичного проектування з метою підвищення продуктивності системи, здатні вплинути на структуру логічної моделі даних [4, 6].

***Фізичне проектування БД –*** *це процес створення та опису реалізації БД на вторинних запам'ятовувальних пристроях із вказівкою структур зберігання й методів доступу, які використовуються для організації ефективної обробки даних*.

Основною метою фізичного проектування є опис способу фізичної реалізації логічного проекту. У випадку реляційної моделі даних під цим мається на увазі наступне:

1) створення набору реляційних таблиць та обмежень для них на основі інформації, яка міститься в глобальній логічній моделі даних;

2) визначення конкретних структур зберігання даних і методів доступу до них, які забезпечують оптимальну продуктивність системи з БД;

3) розробка засобів захисту системи, що створюється.

В ідеалі, фазу логічного проектування великих систем слід відокремлювати від фази їхнього фізичного проектування. На це є ряд причин:

1) вони пов'язані із зовсім різними аспектами системи: що робити і як робити;

2) вони виконуються в різний час, оскільки зрозуміти, що треба зробити, необхідно перед тим, ніж вирішити, як це зробити;

3) вони вимагають зовсім різних навичок і вмінь, якими звичайно володіють різні люди.

## 4.2. Вибір СУБД

Приступаючи до фізичного проектування БД, насамперед, необхідно вибрати конкретну цільову СУБД. Назвемо наступні цільові етапи її вибору:

1. Визначення кола програмних продуктів, з якого буде проводитися вибір СУБД.

2. Скорочення списку претендентів до двох-трьох продуктів.

3. Оцінка продуктів.

4. Обґрунтування вибору єдиного продукту.

Розглянемо вибір продукту для вивчення організації реляційних БД.

Перший етап у цьому випадку зводитися до того, що вибір буде виконуватися із СУБД, що підтримують реляційну модель представлення даних і архітектуру клієнт-сервер. Це пояснюється тим, що приблизно 80% СУБД у світі використовують реляційну модель даних, а архітектура клієнт-сервер довела на практиці свою життєздатність та ефективність.

Книга не має на меті вивчення організації реляційних БД на базі певного програмного продукту або вивчення конкретної реалізації СУБД. Мета книги – дати загальну методологію й підхід. На їхній базі читач повинен виробити власні навички по створенню реляційних БД. Отже, обраний програмний продукт повинен бути найбільш простим і легким у вивченні. Для цього, на думку авторів, найбільше підходять Mysql і Interbase.

На третьому етапі спрацював суб'єктивний фактор. У зв'язку з тим, що автори книги програмували у середовищах розробки від фірми Borland, то Interbase на момент початку написання книги був для них найбільш прийнятним вибором із продуктів, які були визначені на другому етапі. Таким чином, завершений і четвертий етап, тому що на третьому етапі вибір СУБД обґрунтований.

Необхідно відзначити, що для реалізації практичних проектів на різних етапах вибору СУБД можуть впливати ряд факторів, які не представляється можливим строго формалізувати. Наприклад, наявність на підприємстві ліцензій на використання певної СУБД, досвід розроблювачів в реалізації проектів на базі певної СУБД, специфіка конкретного проекту, бюджет, який виділений на проект і т.д.

### 4.3. Правила, по яких СУБД може вважатися реляційною

#### 4.3.1. Правила Кодда – як еталон реляційної СУБД

На ринку присутні кілька сотень реляційних СУБД. Ось лише деякі з них: Firebird [7]; Interbase [10, 14, 15]; Microsoft Access [16]; Microsoft SQL Server [17]; MySQL [18]; Oracle [19]; Visual FoxPro [20] і т.д. На жаль, деякі з них, строго кажучи, не відповідають визначенню реляційної моделі. Ряд постачальників СУБД, які засновані на мережній або ієрархічній моделі даних, реалізують у своїх продуктах тільки деякі риси реляційних систем. Це не заважає їм заявляти про приналежність таких систем до реляційних. Стурбований тим, що потенційні можливості й зміст реляційного підходу спотворюються, Едгар Кодд у 1985 році запропонував 12 правил визначення реляційних систем (а точніше 13, якщо враховувати фундаментальне правило 0). Ці правила утворюють

свого роду еталон, по якому можна визначити приналежність СУБД до розряду дійсно реляційних систем.

Протягом багатьох років запропоновані Коддом правила викликали масу дорікань у зацікавлених осіб і фахівців. Одні визначали їх не більш ніж чисто теоретичними вправами. Інші заявили, що їх продукти вже задовольняють багатьом цим правилам, якщо не всім. Ця дискусія сприяла росту розуміння користувачами й співтовариством розроблювачів СУБД найважливіших властивостей дійсно реляційних СУБД. Щоб підкреслити особливе значення цих правил, їх поділили на п'ять функціональних груп [4].

1. Фундаментальні правила.
2. Структурні правила.
3. Правила цілісності.
4. Правила управління даними.
5. Правила незалежності від даних.

### 4.3.2. Фундаментальні правила (правила 0 і 12)

Образно виражаючись, правила 0 і 12 є «лакмусовим папірцем», який дозволяє визначити приналежність системи до реляційних СУБД. Якщо система не задовольняє цим правилам, то її не слід вважати реляційною.

***Правило 0 – фундаментальне правило.*** Будь-яка система, що рекламується або представляється як реляційна СУБД, повинна бути здатна управляти БД винятково за допомогою її реляційних функцій.

Це правило означає, що СУБД не повинна прибігати до будь-яких не реляційних операцій для виконання таких видів робіт, як визначення даних і маніпулювання ними.

***Правило 12 – правило заборони обхідних шляхів.*** Якщо реляційна система має мову низького рівня (з послідовною порядковою обробкою), вона не може бути використаною для скасування або обходу правил і обмежень цілісності, складених реляційною мовою більш високого рівня (з обробкою відразу декількох рядків).

Це правило гарантує, що всі спроби доступу до БД контролюються СУБД таким чином, що цілісність БД не може бути порушена без відома користувача або адміністратора. Це, однак, не виключає можливостей використання мови низького рівня з інтерфейсом послідовної порядкової обробки.

### 4.3.3. Структурні правила (правила 1 і 6)

Фундаментальним структурним поняттям реляційної моделі є відношення. Кодд стверджує, що реляційна СУБД повинна підтримувати роботу з декількома структурними елементами: відношення, домени, первинні та зовнішні ключі. Для кожного відношення БД повинен бути визначений первинний ключ.

***Правило 1 – подання інформації.*** Уся інформація в реляційній БД

подається в явному виді на логічному рівні й тільки одним способом – у вигляді значень у таблицях.

Згідно із цим правилом, уся інформація, навіть метадані в системному каталозі, повинна зберігатися у вигляді відношень і управлятися за допомогою тих же функцій, які використовуються для роботи з даними. Згадування в цьому правилі "логічного рівня" означає, що такі фізичні конструкції, як індекси, не повинні бути представлені в моделі й користувач не повинен явно посилатися на них в операціях вилучення даних, навіть у тих випадках, коли вони існують.

***Правило 6 – оновлення представлення.*** Усі представлення, які є теоретично оновлюваними, повинні бути оновлюваними й у даній системі.

Це правило стосується винятково представлень віртуальних таблиць. Більш докладне поняття представлення й умов його оновлення, які прийняті в мові SQL, будуть розглянуті пізніше. Дане правило говорить, що якщо представлення є теоретично оновлюваним, то СУБД повинна вміти виконувати подібні оновлення. Насправді жодна існуюча система не підтримує цю вимогу, тому що усе ще не визначені умови ідентифікації всіх теоретично оновлюваних представлень.

### 4.3.4. Правила цілісності (правила 3 і 10)

Кодд запропонував два правила підтримки цілісності даних. Підтримка цілісності даних є важливим критерієм оцінки придатності системи для практичних потреб. Чим більше обмежень цілісності підтримує сама СУБД, а не окремі її прикладні програми, тем вище гарантії якісності даних.

***Правило 3 – систематична обробка невизначених значень (NULL).*** Невизначені значення (що задаються за допомогою визначника NULL), тобто значення, відмінні від порожнього рядка або рядка порожніх символів, а також від нуля або будь-якого іншого конкретного значення, підтримуються для систематичного представлення відсутньої або неприйнятної інформації, причому незалежно від типу даних.

***Правило 10 – незалежність обмежень цілісності.*** Специфічні для даної реляційної СУБД обмеження цілісності повинні визначатися на підмові[1] реляційних даних і зберігатися в системному каталозі, а не в прикладних програмах.

Кодд особливо підкреслює, що відомості про встановлені обмеження цілісності даних повинні зберігатися в системному каталозі, а не інкапсулюватися в окремих прикладних програмах або інтерфейсах користувача. Зберігання обмежень цілісності в системному каталозі надає важливу перевагу централізованого управління та приведення їх у дію.

[1] Підмовою називається мова, яка не включає конструкції, що призначені для всіх обчислювальних потреб. Реляційна алгебра і реляційне обчислення є підмовами БД.

### 4.3.5. Правила управління даними (правила 2, 4, 5 та 7)

Ідеальна реляційна СУБД повинна підтримувати 18 функцій управління даними, які визначають повноту мови запитів (тут термін «запит» включає операції вставки, оновлення та вилучення). Правила маніпулювання даними визначають спосіб застосування 18 функцій управління даними. Суворе їх дотримання дозволяє ізолювати користувача та прикладні програми від фізичного та логічного механізмів реалізації засобів маніпулювання даними.

***Правило 2 – гарантований доступ.*** Для всіх і кожного елемента даних (тобто його атомарного значення) реляційної БД повинен бути гарантований логічний доступ на основі комбінації імені таблиці, значення первинного ключа та значення імені стовпця.

***Правило 4 – динамічний інтерактивний каталог, побудований за правилами реляційної моделі.*** Опис БД повинен представлятися на логічному рівні в такий же спосіб, як і звичайні дані, що дозволить санкціонованим користувачам застосовувати для звертань до цього опису тою же реляційною мовою, що й при звертанні до даних.

Це правило вказує на те, що повинна існувати тільки одна мова, яка призначена для маніпулювання, як метаданими, так і звичайними даними, причому в СУБД для організації зберігання системної інформації повинна використовуватися тільки одна логічна структура – відношення.

***Правило 5 – вичерпна підмова даних.*** Реляційна система може підтримувати кілька мов і різні режими роботи з терміналами (наприклад, режим заповнення форми – fill-in-the-blanks). Однак повинна існувати, принаймні, одна мова, оператори якої дозволяли б виражати всі наступні конструкції: 1) визначення даних; 2) визначення представлень; 3) команди маніпулювання даними (доступні як інтерактивно, так і з програм); 4) обмеження цілісності; 5) авторизація користувачів; 6) організація трансакцій (запуск, фіксація та скасування).

Слід зазначити, що новий стандарт *ISO* для мови *SQL* забезпечує виконання всіх цих функцій таким чином, що будь-яка мова, яка підтримує цей стандарт, автоматично буде задовольняти й цьому правилу.

***Правило 7 – високорівневі операції вставки, оновлення й вилучення.*** Здатність обробляти базові або похідні відношення (тобто представлення) як єдиний операнд повинна відноситися не тільки до процедур витягу даних, але й до операцій вставки, оновлення й вилучення даних.

### 4.3.6. Правила незалежності від даних (правила 8, 9 та 11)

Кодд визначає три правила незалежності даних від прикладних програм, які ці дані використовують. Суворе дотримання цих правил гарантує, що користувачі та розроблювачі будуть захищені від необхідності вносити тотальні зміни в прикладні програми при кожній

реорганізації БД на низькому рівні.

***Правило 8 – фізична незалежність від даних.*** Прикладні програми та засоби роботи з терміналами повинні залишатися логічно незачепленими при внесенні будь-яких змін у способи зберігання даних або методи доступу до них.

***Правило 9 – логічна незалежність від даних.*** Прикладні програми та засоби роботи з терміналами повинні залишатися логічно незачепленими при внесенні в базові таблиці будь-яких змін, що не міняють інформацію, які теоретично не повинні торкатися прикладного програмного забезпечення.

***Правило 11 – незалежність від розподілу даних.*** Підмова маніпулювання даними в реляційній СУБД повинна дозволяти прикладним програмам та запитам залишатися логічно незмінними, незалежно від того, як зберігаються дані фізично: централізовано або в розподіленому вигляді.

Незалежність від розподілу даних означає, що прикладна програма, що здійснює доступ до СУБД на окремому комп'ютері, повинна без жодних модифікацій продовжувати працювати й у тому випадку, коли дані в мережі будуть перенесені з одного комп'ютера на інший. Інакше кажучи, для кінцевого користувача повинна бути створена ілюзія того, що дані централізовано зберігаються на єдиному комп'ютері, а відповідальність за переміщення й пошук даних серед (можливо) декількох місць їх зберігання повинна покладати винятково на систему. Зверніть увагу, що тут не сказано про те, що реляційна СУБД повинна неодмінно підтримувати роботу з розподіленою БД. Тут просто мається на увазі, що мова запитів повинна залишатися незмінною і в тому випадку, коли можливість роботи з розподіленими даними реалізується в деякій СУБД.

## 4.4. Мова SQL

### 4.4.1. Коротка характеристика мови SQL

Мова *SQL* є першою й поки єдиною стандартною мовою роботи з БД, яка одержала досить широке поширення [4, 6…8]. Є ще одна стандартна мова роботи з БД – Network Data-base Language (*NDL*), – яка побудована на використанні мережної моделі *CODASYL*, але застосовується в далеко не в усіх розробках. Практично усі найбільші розроблювачі СУБД у цей час створюють свої продукти з використанням мови *SQL* або *SQL*-інтерфейсом.

В ідеалі, будь-яка мова роботи з БД повинна надавати користувачеві наступні можливості:

1) створювати БД і таблиці з повним описом їх структури;

2) виконувати основні операції маніпулювання даними, такі як вставка, модифікація й вилучення даних з таблиць;

3) виконувати прості та складні запити, здійснювати перетворення неопрацьованих даних у необхідну інформацію;

4) відповідати деякому визнаному стандарту, що дозволить використовувати той самий синтаксис і структуру команд при переході від однієї СУБД до іншої.

Крім того, мова роботи з БД повинна вирішувати всі зазначені вище завдання при мінімальних зусиллях з боку користувача, а структура й синтаксис його команд повинні бути досить прості й доступні для вивчення. Мова *SQL* задовольняє практично всім цим вимогам.

*SQL* є прикладом мови з орієнтацією на трансформацію. Або ж мови, яка призначена для роботи з таблицями з метою перетворення вхідних даних до необхідного вихідного виду. Мова *SQL*, має два основні компоненти:

1) мова *DDL* (*Data Definition Language*), призначена для визначення структур БД і управління доступом до даних;

2) мова *DML* (*Data Manipulation Language*), призначена для вибірки та оновлення даних.

Мова *SQL* включає тільки команди визначення й маніпулювання даними. У неї відсутні будь-які команди управління ходом обчислень. Інакше кажучи, у цій мові немає команд *IF ... THEN ... ELSE*, *GO TO*, *DO ... WHILE* і т.п., що призначені для управління ходом обчислювального процесу. Подібні задачі повинні вирішуватися за допомогою мов програмування або інтерактивно, як результат дій користувача. Через подібну незавершеність у плані організації обчислювального процесу мова *SQL* може використовуватися двома способами. Перший – передбачає інтерактивну роботу, що полягає в уведенні користувачем з термінала окремих *SQL*-операторів. Другий – полягає у впровадженні *SQL*-операторів у програми, що написані на процедурних мовах.

Мова *SQL* відносно проста у вивченні.

1. Це не процедурна мова, тому у ній необхідно вказувати, яка інформація повинна бути отримана, а не як її можна одержати. Інакше кажучи, *SQL* не вимагає опису методів доступу до даних.

2. Як і більшість сучасних мов, *SQL* підтримує вільний формат запису операторів. Це означає, що при введенні окремі елементи операторів не пов'язані з фіксованими позиціями екрана.

3. Структура команд задається набором ключових слів, що представляють собою звичайні слова та фрази англійської мови – такі, як *CREATE TABLE* (створити таблицю), *INSERT* (вставити), *SELECT* (вибрати) та ін.

4. Мова *SQL* може використовуватися широким колом фахівців, включаючи адміністраторів БД, керівний персонал, прикладних програмістів та безліч інших типів кінцевих користувачів.

На час написання книги для мови SQL існує міжнародний стандарт ISO/IEC 9075(1…4, 9…11, 13, 14):2008, що формально перетворює її у стандартну мову визначення й маніпулювання реляційними БД. Однак слід зазначити, що розробники СУБД відхиляються від стандартів. Або

підтримують стандарти, які були випущені значно раніше. Тому для фізичної реалізації БД потрібно використовувати відповідну технічну документацію розроблювача СУБД, що використовується. Усі приклади та фізична реалізація БД «БІБЛІОТЕКА» відпрацьовані в СУБД Interbase. На час написання книги останньою версією є Interbase XE (2010 р.) від Embarcadero.

### 4.4.2. Запис SQL-операторів

*SQL*-оператор складається із зарезервованих слів, а також зі слів, які визначаються користувачем [4, 6]. Зарезервовані слова є постійною частиною мови *SQL* і мають фіксоване значення. Їх слід записувати в точності так, як це встановлено, і не можна розбивати на частини для переносу з одного рядка в інший. Слова, які визначаються користувачем, задаються ним самим (відповідно до певних синтаксичних правил) і являють собою імена різних об'єктів БД: таблиць, стовпців, представлень, індексів і т.д. В *SQL*-операторі слова розміщаються відповідно до встановлених синтаксичних правил. Хоча в стандарті це не зазначено, багато діалектів мови *SQL* вимагають використання наприкінці оператора деякого символу, що позначає закінчення його тексту Як правило це «крапка з комою».

Більшість компонентів *SQL*-операторів не чутливі до регістру. Це означає, що можуть використовуватися будь-які літери – як рядкові, так і прописні. Одним важливим виключенням із цього правила є символьні літерали-дані, які не повинні відрізнятися від відповідних їм значень, що зберігаються в БД. Наприклад, якщо в БД зберігається значення прізвища '*ІВАНОВ*', а в умові пошуку прописаний символьний літерал '*Іванов*', то цей запис не буде знайдений.

Оскільки мова *SQL* має вільний формат, окремі *SQL*-оператори і їх послідовності будуть мати більш читабельний вигляд з використанням відступів та вирівнювання. Рекомендується дотримуватися наступних правил.

1. Кожна фраза оператору повинна починатися з нового рядка.
2. Початок кожної фрази повинне бути вирівняний з початком інших фраз оператора.
3. Якщо фраза має кілька частин, кожна з них повинна починатися с нового рядка з деяким відступом відносно початку фрази, що буде вказувати на їхню підпорядкованість.

Для визначення формату *SQL*-операторів ми будемо дотримуватись наступної форми їх запису у загальному виді.

1. Прописні літери будуть використовуватися для запису зарезервованих слів і повинні вказуватися в операторах точно так само, як це буде показано.
2. Малі літери будуть використовуватися для запису слів, які визначені користувачем.

3. Вертикальна риска ( | ) вказує на необхідність вибору одного з декількох наведених значень. Наприклад, a | b | c.

4. Фігурні дужки визначають обов'язковий елемент – наприклад, {a}.

5. Квадратні дужки визначають необов'язковий елемент наприклад, [a].

6. Три крапки (...) використовуються для вказівки необов'язкової можливості повторення конструкції, від нуля до декількох раз – наприклад, {a | b} [, c…]. Цей запис означає, що після a або b може слідувати від нуля до декількох повторень c, які розділені комами.

На практиці для визначення структури БД використовується *DDL*-оператори, а для заповнення цих відношень даними й вибірки з них інформації за допомогою запитів – *DML*-оператори.

### 4.5. SQL-оператори, що реалізовують структуру реляційних даних

На етапі вибору системи управління БД ми зупинилися на СУБД Interbase [10, 14, 15]. Відзначили, що в наші плани не входить вивчення усіх тонкощів роботи із цим програмним продуктом. Тих, кого зацікавлять подробиці створення й управління БД саме за допомогою цього продукту, традиційно запропонуємо ознайомитися з документацією, що поставляється розроблювачем.

СУБД Interbase дозволяє створювати БД як в інтерактивному режимі, по черзі виконуючи *SQL*-оператори, так і в режимі, інтерпретації SCRIPT-файлу. Практична частина курсу побудована на створенні SCRIPT-файлу і його послідовного наповнення командами. Рекомендовану послідовність команд в SCRIPT-файлі розглянемо на практичних заняттях. Тут обмежимося загальним оглядом *SQL*-операторів, що створюють на жорсткому диску файл БД і його структуру, що відповідаю логічній моделі.

Для створення файлу (або файлів) БД на жорсткому диску використовують оператор *CREATE DATABASE*. Він дає можливості:

1. Вказати ім'я файлу (або файлів) БД і його місце знаходження на локальному або мережному жорсткому диску.

2. Вказати ім'я й пароль користувача, який використовується при підключенні до БД.

3. Визначити набір символів, застосовуваний у БД за умовчуванням. Ми будемо використовувати набір WIN1251.

4. Для файлу БД указати довжину файлу в сторінках, довжину кожної сторінки в байтах, якщо БД розташовується в декількох файлах – указати з якої сторінки починається кожний із цих файлів.

Домени створюються оператором *CREATE DOMAIN*, у якому необхідно кожному домену дати унікальне ім'я. Він дозволяє:

1. Указати тип даних, на базі яких створюється домен.

2. Указати значення, прийняте за замовчуванням.

3. Визначити конструкції умов перевірки значення домену.

4. Визначити таблицю, по якою буде проводитися сортування значень домену.

Для створення таблиць та ключів застосовується оператор *CREATE TABLE*. Таблиця повинна мати унікальне ім'я, і в її межах імена стовпців не повинні повторюватися. За допомогою цього оператора ви можете:

1. Указати для кожного стовпця таблиці домен або тип даних і конструкції, що дозволяють визначити припустимі значення, які можна буде вводити в цей стовпець.

2. Визначити список стовпців, які входять у первинний ключ.

3. Визначити список стовпців потенційних ключів.

4. Визначити список стовпців, що входять у зовнішні ключі таблиці, і вказати імена таблиць, на первинні ключі яких посилаються значення зовнішніх ключів.

5. Визначити дії при каскадному оновленні або вилученні рядків.

6. Визначити умови перевірки значень стовпців при додаванні нового рядка в таблицю.

Порядок створення таблиць у БД має значення. Спочатку створюються ті таблиці, у яких відсутні зовнішні ключі. Потім створюються таблиці, у яких значення зовнішніх ключів мають зв'язок з первинними ключами вже створених таблиць. Як приклад приведемо порядок створення таблиць «ПОСАДА», «ЧИТАЧІ», «ТИПИ ТЕЛЕФОНІВ», «ТЕЛЕФОНИ» (рис. 3.4). Тут можна поміняти порядок створення таблиць «ЧИТАЧІ» і «ТИПИ ТЕЛЕФОНІВ». Однак, «ЧИТАЧІ» завжди повинні створюватися після таблиці «ПОСАДА», а «ТЕЛЕФОНИ» – після таблиць «ТИПИ ТЕЛЕФОНІВ» і «ЧИТАЧІ». А якщо ні, то СУБД видасть помилку.

Структура БД «БІБЛІОТЕКА», яка отримана після роботи відповідних операторів *CREATE TABLE*, показує зв'язок між логічною й фізичною моделями (додаток Б). У додатку Б літерою «*P*» позначені первинні ключі, а літерою «*F*» – зовнішні. Позначення первинних та зовнішніх ключів відповідають позначенням зв'язків між відношеннями, що наведені в додатку А. Стрілки вказують, що значення зовнішніх ключів у дочірніх відношеннях повинні обов'язково посилатися на значення первинних ключів батьківських відношень.

## 4.6. Додавання, оновлення й вилучення даних

### 4.6.1. Невідоме або неприйнятне значення

***Визначник NULL*** вказує, що значення атрибута в даний момент невідоме або неприйнятно для цього кортежу.

Визначник *NULL* слід сприймати як логічну величину "невідомо". Інакше кажучи, або це значення не входить в область визначення деякого кортежу, або ніяке значення ще не задане. Ключове слово *NULL* являє

собою спосіб обробки неповних або незвичайних даних. Однак визначник *NULL* не слід розуміти як нульове чисельне значення або заповнений пробілами текстовий рядок. Нулі й пробіли є значеннями, тоді як ключове слово *NULL* покликано позначати відсутність будь-якого значення. Деякі автори використовують термін "значення *NULL*", але насправді визначник *NULL* не є значенням, а лише позначає його відсутність, а тому термін "значення *NULL*" використовувати не рекомендується.

Застосування визначника *NULL* може викликати проблеми на етапі реалізації. Труднощі виникають через те, що реляційна модель заснована на вирахуванні предикатів першого порядку, яке має двозначну, або булеву логіку, тобто припустимими є тільки два значення: істина й хибність. Застосування визначника *NULL* означає, що потрібно вести роботу з логікою більш високого порядку, наприклад тризначної або навіть чотиризначної (Codd, 1986, 1987, 1990).

Використання поняття *NULL* у реляційній моделі є дискусійним питанням. Кодд (1990) розглядає поняття *NULL* як складову частину цієї моделі, а інші фахівці вважають цей підхід невірним. Вони визначають, що проблема відсутньої інформації ще не до кінця зрозуміла. Задовільне рішення для її розв'язання не знайдено, а тому включення визначника *NULL* у реляційну модель є передчасним (Date, 1995). Слід зазначити, що не у всіх реляційних системах підтримується робота з визначником *NULL*. Але в СУБД Interbase цей визначник присутній.

### 4.6.2. Забезпечення цілісності реляційних даних

Насамперед, цілісність забезпечується первинними (потенційними) ключами базових або батьківських відношень [2, 4, 6]. Тут батьківське відношення визначається як відношення, яке відповідає деякому суб'єктові (сутності) технологічного процесу в логічній моделі. Наприклад, відношення «ПОСАДИ» є батьківським для відношення «ЧИТАЧІ» (рис. 3.4). Відношення «ЧИТАЧІ» є батьківським для того відношення, де значення зовнішнього ключа посилаються на значення одного з потенційних ключів відношення «ЧИТАЧІ».

***Цілісність сутностей*** у базовому відношенні забезпечується тим, що жоден атрибут первинного ключа не може містити замість значення визначника *NULL*.

Припущення присутності визначника *NULL* у будь-якій частині потенційного ключа рівноцінно твердженню, що не всі його атрибути необхідні для унікальної ідентифікації кортежів. Наведене твердження суперечить визначенню потенційного ключа.

Цілісність реляційних даних забезпечується зовнішніми ключами в дочірніх відношеннях. Дочірніми називаються відношення, де присутні зовнішні ключі, значення яких повинні посилатися на значення потенційних ключів у батьківських відношеннях.

***Посилальна цілісність*** забезпечується, якщо значення зовнішнього

ключа дочірнього відношення посилається на існуюче значення потенційного ключа батьківського відношення, або задається за допомогою визначника *NULL*.

Допустимість присутності визначника *NULL* серед значень атрибутів, які використовуються у якості зовнішнього ключа, визначається вимогами користувачів. Наприклад, якщо користувач вважає, що інформація про посаду є необов'язкової, то в стовпці «Код посади» відношення «ЧИТАЧІ» може бути присутнім або визначник *NULL*, або значення первинного ключа, яке має місце в атрибуті «Код» відношення «ПОСАДА». Якщо користувач вважає, що інформація про посаду є обов'язкової, тоді в стовпці «Код посади» присутність визначника *NULL* неприпустимо.

### 4.6.3. Оператори для додавання, зміни й вилучення даних

Оператор *INSERT* дозволяє додати новий рядок у таблицю з іменем, яке ви вказуєте. Він дає можливість перелічити стовпці таблиці, у яких значення в новому рядку будуть відрізнятися від значень, які прийняті за замовчуванням. Якщо на рівні домену або в операторові *CREATE TABLE* значення за замовчуванням не визначені, тоді в комірках нового рядка, які розташовані в стовпцях, що не потрапили в список, буде записаний визначник *NULL*. Кількість стовпців і доданих значень, які перераховані в операторі, повинна збігатися. Якщо хоча б одне значення виходить за область його визначення, то дія оператору *INSERT* буде відхилена й СУБД видасть відповідне повідомлення.

Порядок додавання нових рядків у таблиці не є довільним. Це пов'язане з тим, що поки в батьківській таблиці не буде доданий рядок з деяким значенням потенційного ключа, в дочірню таблицю не можна буде додати рядок із значенням зовнішнього ключа, яке посилається на значення потенційного ключа в батьківській таблиці. Для виключення помилок додавайте нові рядки в таблиці, дотримуючи порядку їх створення в БД операторами *CREATE TABLE*.

Для визначення унікального значення первинного ключа в СУБД *Interbase* застосовується механізм генераторів. Для кожного атрибута, який передбачається використовувати в якості первинного ключа, визначається генератор. Для цього служить оператор *CREATE GENERATOR*. У ньому ви можете задати крок, на який нове значення потенційного ключа буде відрізнятися від попереднього. Оператор

**SET GENERATOR** {Ім'я} **TO** {Початкове значення}

дозволяє вказати значення, з якого почне свою роботу генератор. В інших СУБД механізм генераторів вбудований в оператор *CREATE TABLE*, де можна вказати інкрементний тип даних для стовпця таблиці.

Оператор *UPDATE* змінює значення в строках таблиці, ім'я якої обов'язково потрібно вказати. Він дозволяє змінити значення в будь-якій кількості стовпців таблиці. Також як і для оператора *INSERT*, при виході

нового значення за межі області визначення СУБД згенерує повідомлення про помилку, і дія оператора *UPDATE* буде скасовано.

Множина рядків таблиці, у яких буде працювати оператор *UPDATE*, визначається умовами пошуку. У більшості випадків в умові пошуку вказується область припустимих значень первинного або потенційного ключа таблиці. Дані оновляються в тих рядках, де логічне вираження пошуку істинно. Якщо ви не вкажете умови пошуку рядків для оновлення, тоді в усіх рядках таблиці дані будуть змінено.

Вилучення рядків з таблиці забезпечується оператором *DELETE*. Умова пошуку працює точно так само, як і в операторові *UPDATE*. Якщо умови пошуку не вказувати, то з таблиці будуть вилучені усі рядки.

### 4.6.4. Каскадне оновлення й вилучення

Цей механізм, що є одним з засобів забезпечення посилальної цілісності даних, реалізується в операторові *CREATE TABLE*. Він дозволяє розроблювачеві вказати для СУБД дію, яку необхідно виконати системі при виконанні операторів *UPDATE* або *DELETE*, що роблять спробу вилучити або оновити таке значення потенційного ключа в батьківській таблиці, на яке посилається один або кілька рядків дочірньої таблиці.

Мова *SQL* передбачає чотири варіанти реакції, як для оновлення, так і для вилучення значення потенційного ключа з батьківської таблиці, на яке є посилання значень зовнішнього ключа з рядків дочірньої таблиці. Варіант дії вказується окремо для оператора *UPDATE* або *DELETE*, при визначенні зовнішнього ключа в дочірній таблиці. Дії вказуються після ключових фраз *ON UPDATE* або *ON DELETE*. Вони визначаються виходячи з вимог користувачів, які були отримані на етапі логічного проектування БД.

Якщо після *ON UPDATE* указати слово *CASCADE*, то в цьому випадку виконується каскадне оновлення значень зовнішнього ключа в рядках дочірньої таблиці при зміні оператором *UPDATE* відповідного значення потенційного ключа в батьківській таблиці. Якщо *CASCADE* написати після *ON DELETE*, то виконання оператора *DELETE*, що вилучає рядок зі значенням потенційного ключа в батьківській таблиці, приведе до вилучення рядків у дочірній таблиці, у яких значення зовнішнього ключа збігається з значенням потенційного ключа, яка вилучається.

Фраза *SET NULL*, яка написана після *ON UPDATE* або *ON DELETE*, говорить системі, що після вилучення або зміни значення потенційного ключа в батьківській таблиці необхідно прописати визначник *NULL* у всіх рядках дочірньої таблиці, у яких значення зовнішніх ключів посилалися на вилучене або змінене значення потенційного ключа.

Наступним варіантом реакції СУБД на вилучення значення потенційного ключа з батьківської таблиці, є прописування у всіх атрибутах

зовнішнього ключа значень, визначених за замовчуванням. Для цього після фрази *ON DELETE* необхідно написати *SET DEFAULT*. Нагадаємо, що для кожного атрибута можна вказати значення, які в ньому прописуються за замовчуванням або при ініціалізації доменів, або безпосередньо в операторові *CREATE TABLE*.

Уведення інкрементних атрибутів «Код» дозволив повністю звільнитися від необхідності зміни властивостей ключових атрибутів, у зв'язку зі змінами правил обліку читачів у бібліотеці (рис. 3.4). Вони дозволили трохи компенсувати збільшення необхідного обсягу пам'яті для реалізації БД за рахунок скорочення місця для атрибута відношення, що пов'язує «ЧИТАЧІ» та «ТЕЛЕФОНИ» з відношеннями «ПОСАДА» та «ТИПИ ТЕЛЕФОНІВ» відповідно. Ці відношення можна було б зв'язати за значеннями атрибутів «Найменування посади» та «Найменування типу телефону» замість атрибутів «Код посади» та «Код типу телефону». Економія залежить від кількості знаків цих найменувань, яка закладена згідно з вимогами користувачів.

Якщо в операторові *CREATE TABLE*, що описує структуру дочірньої таблиці, не зазначені фрази *ON UPDATE* або *ON DELETE* у визначенні зовнішнього ключа, тоді механізм забезпечення посилальної цілісності при вилученні або зміні значення потенційного ключа в батьківській таблиці працює так само, як і при зазначені фрази *NO ACTION*.

Вміст БД «БІБЛІОТЕКА», яка буде використовуватися для вивчення прийомів вибірки інформації, наведено в додатку В.

## 4.7. КОНТРОЛЬНІ ПИТАННЯ

1. У чому полягає фізичне проектування БД?
2. Що впливає на вибір СУБД для реалізації фізичної моделі БД?
3. Які групи правил Вам відомі, за якими СУБД вважають реляційною?
4. Для чого використовують відомі вам компоненти мови SQL?
5. Яких правил треба дотримуватись для запису SQL-операторів?
6. Які характеристики має SQL-оператор, що створює БД?
7. Які можливості дає SQL-оператор що визначає домени?
8. Які структурні елементи таблиць визначаються SQL-оператором?
9. На що вказує і для чого використовується визначник NULL?
10. Як забезпечуються цілісність сутностей та посилальна цілісність?
11. Які SQL-оператори виконують маніпуляції даними у таблицях?
12. Що визначають умови пошуку для SQL-операторів, які оновлюють або вилучають дані з таблиці?
13. Які засоби Вам відомі для забезпечення унікальності значень первинних ключів?
14. Як СУБД реалізує каскадне оновлення або вилучення даних?

*Розглянутий навчальний матеріал дає студентові можливість забезпечувати цілісність сутностей та посилальну цілісність в розробляємой СУБД, виконувати каскадне оновлення і видалення записів в дочірніх таблицях тощо.*

# 5. Реляційна алгебра

*Мета розділу – надати загальні відомості про реляційну алгебру, як теоретичну мову операцій.*

## 5.1. Основні поняття

***Реляційна алгебра** – це теоретична мова операцій, яка на основі одного або декількох відношень дозволяє створити інше відношення без зміни вихідних даних* [1, 4, 6, 7, 9, 10, 12]. Обидва операнди й результат є відношеннями, а тому результати однієї операції можуть стати вихідними даними для іншої операції. Це дозволяє створювати вкладені вираження реляційної алгебри точно так само, як створюються вкладені арифметичні вираження. Ця властивість називається замкнутістю, тобто відношення покриваються реляційною алгеброю так само, як числа арифметичними операціями [4].

Реляційна алгебра є мовою послідовного використання відношень, у якої усі кортежі, навіть узяті із різних відношень, обробляються однією командою без організації циклів. Для команд реляційної алгебри запропоновано кілька варіантів синтаксису. Нижче ми скористаємося загальноприйнятими символічними позначеннями для цих команд і представимо їх у неформальному вигляді. Більш докладні відомості по цьому питанню зацікавлений читач зможе знайти у роботах Ульмана (Ullman, 1988).

Існує кілька варіантів вибору операцій, які включаються в реляційну алгебру. Спочатку Кодд запропонував вісім операторів, але згодом до них були додані й деякі інші. П'ять основних операцій реляційної алгебри (рис. 5.1): вибірка (*selection*), проекція (*projection*), декартів добуток (*Cartesian product*), об'єднання (*union*) і різниця (*set difference*), виконують більшість операцій витягу даних, які можуть представляти для нас інтерес. На підставі п'яти основних операцій можна також вивести додаткові операції (рис. 5.1): з'єднання (*join*), перетинання (*intersection*) і ділення (*division*).

Операції вибірки й проекції є унарними, оскільки вони працюють із одним відношенням. Інші операції працюють із парами відношень, і тому їх називають бінарними операціями. У наведених нижче формулюваннях $R$ і $S$ – це два відношення, які визначені на атрибутах $A = (a_1, a_2, \ldots, a_n)$ та $B = (b_1, b_2, \ldots, b_n)$ відповідно. Для ілюстрації результатів виконання операцій ми скористаємося відношеннями БД «БІБЛІОТЕКА» (додаток В).

## 5.2. Вибірка (або обмеження)

$\sigma_{предикат}(R)$ | Операція вибірки працює з одним відношенням $R$ і визначає результуюче відношення, яке містить тільки ті кортежі (рядка) відношення $R$, які задовольняють заданій умові (предикату).

ПРИКЛАД 5.1.

Складіть список бібліотекарів, у яких табельний номер перевищує 80.

$\sigma_{ClockNumber > 80}$ (Librarians)

Тут вихідним відношенням є відношення *Librarians*, а предикатом – вираз *ClockNumber>80*. Операція вибірки визначає нове відношення, що містить тільки ті кортежі відношення *Librarians*, де значення атрибута *ClockNumber* перевищує 80

(табл.5.1). Більш складні предикати можуть бути створені за допомогою логічних операторів *AND*, *OR* і *NOT*.

Таблиця 5.1

**Результат виконання операції вибірки**

| Code | ClockNumber | FamilyName | Name | Patronymic | PasportCode | Post | HomePhone | Note |
|---|---|---|---|---|---|---|---|---|
| 3 | 187 | Іноземцева | Іванна | Модестівна | 9 | Ст. бібліотекар | 775-34-00 | null |
| 4 | 83 | Мальцева | Діана | Петрівна | 12 | Бібліотекар | 29-06-15 | null |
| 6 | 100 | Ставка | Лілія | Іванівна | 7 | Бібліотекар | 22-00-01 | null |

## 5.3. Проекція

$П_{amp1,\ ...,\ amp\ n}(R)$

Операція проекції працює з одним відношенням $R$ і визначає нове відношення, яка містить вертикальну підмножину відношення $R$, що створена за допомогою витягу значень вказаних атрибутів та виключення з результату рядків-дублікатів.

ПРИКЛАД 5.2.

Складіть список бібліотекарів із зазначенням табельного номера (ClockNumber), прізвища (FamilyName), імені (Name), по батькові (Patronymic) та домашнього телефону (HomePhone).

$П_{ClockNumber,FamilyName,Patronymic,HomePhone}(Librarians)$

У цьому прикладі операція проекції визначає нове відношення, яке буде містити тільки атрибути ClockNumber, FamilyName, Name, Patronymic і HomePhone відношення Librarians, розміщені в зазначеному порядку (табл. 5.2).

Таблиця 5.2

**Проекція відношення Librarians**

| ClockNumber | FamilyName | Name | Patronymic | HomePhone |
|---|---|---|---|---|
| 28 | Іванова | Олена | Володимирівна | 52-07-75 |
| 12 | Ніколаєнко | Любов | Миколаївна | 46-32-19 |
| 187 | Іноземцева | Іванна | Модестівна | 775-34-00 |
| 83 | Мальцева | Діана | Петрівна | 29-06-15 |
| 10 | Сизранцева | Тетяна | Ігорівна | 370-98-22 |
| 100 | Ставка | Лілія | Іванівна | 22-00-01 |
| 50 | Лещенко | Алла | Федорівна | 722-36-36 |
| 36 | Сіра | Лідія | Іванівна | 254-78-02 |
| 45 | Прохіна | Тамара | Львівна | 63-05-01 |
| 78 | Самійленко | Вікторія | Ігорівна | 125-45-80 |
| 69 | Степанова | Олександра | Миколаївна | 445-45-65 |
| 17 | Петрова | Алина | Сергіївна | 999-12-05 |

## 5.4. Декартів добуток

$R \times S$

Операція декартового добутку визначає нове відношення, яке є результатом конкатенації (тобто зчеплення) кожного кортежу з відношення $R$ з кожним кортежем з відношення $S$.

Оператори вибірки та проекції роблять витяг інформації тільки з одного відношення. Очевидно, можливе виникнення таких ситуацій, коли потрібна деяка комбінація даних з декількох відношень. Оператор декартового добутку множить два відношення. В результаті утворюється інше відношення, у якому є усі можливі пари кортежів обох відношень. Отже, якщо одне відношення має *I* кортежів та N атрибутів, а інше – *J* кортежів і *M* атрибутів, то відношення з їх декартовим добутком буде містити ($I \times J$) кортежів та ($N + M$) атрибутів. Вихідні відношення можуть мати атрибути з однаковими іменами. У такому випадку імена атрибутів будуть містити назви відношень у вигляді префіксів. Це забезпечить унікальності імен атрибутів у результуючім відношенні.

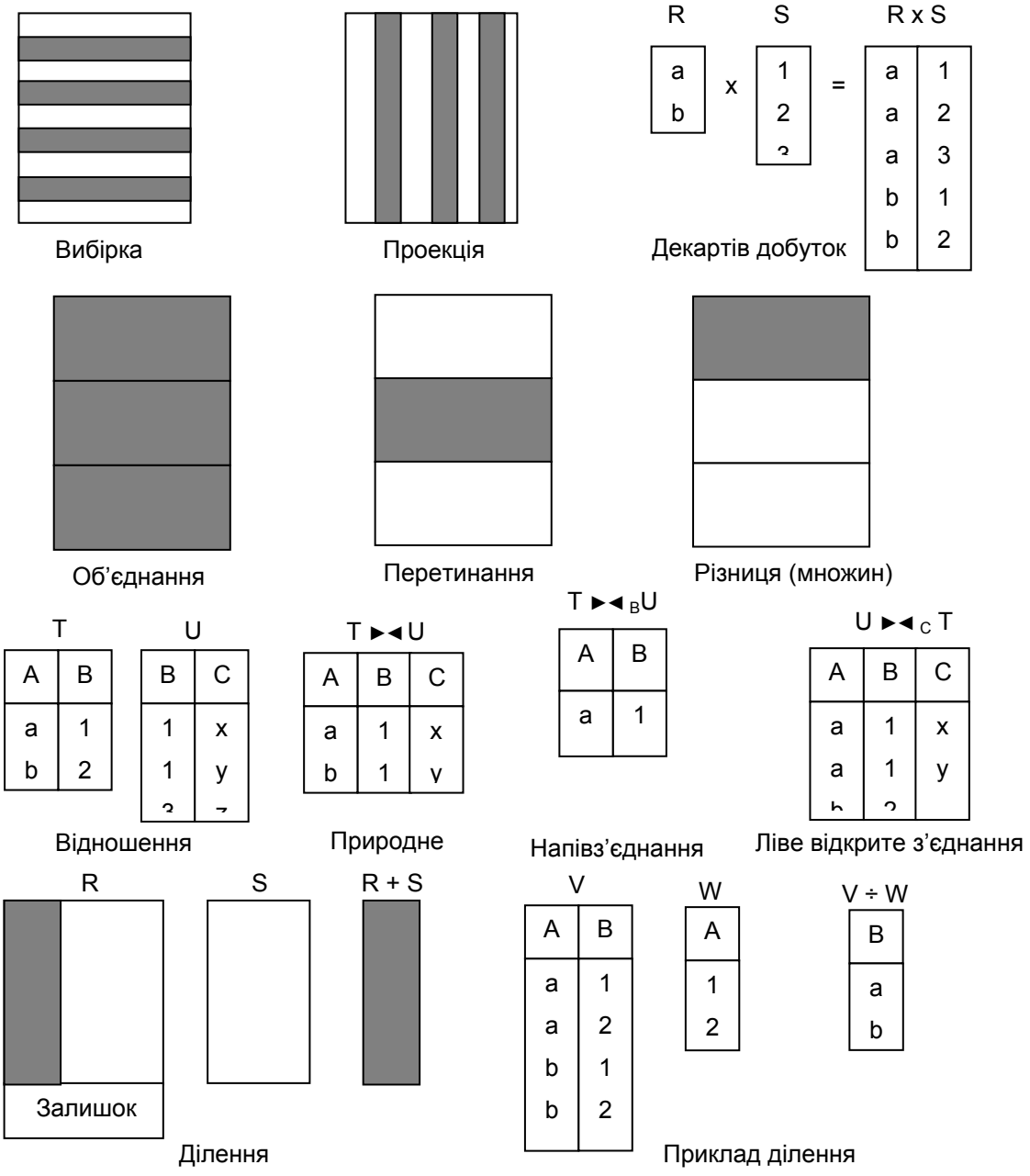



Рис. 5.1. Схематичне представлення функцій операторів реляційної алгебри

ПРИКЛАД 5.3.

Створіть список усіх читачів, які коли-небудь брали книги у бібліотеці з використанням наступних атрибутів відношення Readers: Code, FamilyName, Name та відношення BookGiveOutRecord: Code, ReaderCode, InventoryCode.

$(П_{Code, FamilyName, Name}(Readers)) \times (П_{Code, ReaderCode, InventoryCode}(BookGiveOutRecord))$

Результатом виконання такої операції буде відношення, що містить 120 кортежів та 6 атрибутів. Приводити його у повному обсязі не має сенсу. Подивиться на його перші 12 кортежів (табл. 5.3). Перші 10 – всі можливі комбінації першого кортежу відношення Readers з десятьма кортежами відношення BookGiveOutRecord. Останні два наведені для того щоб ви порівняли їх значення з першими двома. Це перші дві можливі комбінації другого кортежу відношення Readers з першими двома кортежами відношення BookGiveOutRecord. Наступним буде третя можлива комбінація другого кортежу відношення Readers з третім кортежем відношення BookGiveOutRecord. І так до стонадцятого кортежу.

Таблиця 5.3

**Декартів добуток а відношень Reader і BookGiveOutRecord**

| Readrs.Code | FamilyName | Name | BookGiveOutRecord.Code | ReaderCode | InventoryCode |
|---|---|---|---|---|---|
| 1 | Іванов | Петро | 1 | 2 | 6 |
| 1 | Іванов | Петро | 2 | 3 | 4 |
| 1 | Іванов | Петро | 3 | 6 | 3 |
| 1 | Іванов | Петро | 4 | 4 | 6 |
| 1 | Іванов | Петро | 5 | 7 | 7 |
| 1 | Іванов | Петро | 6 | 9 | 12 |
| 1 | Іванов | Петро | 7 | 11 | 10 |
| 1 | Іванов | Петро | 8 | 7 | 8 |
| 1 | Іванов | Петро | 9 | 9 | 12 |
| 1 | Іванов | Петро | 10 | 12 | 15 |
| 2 | Федорец | Ірина | 1 | 2 | 6 |
| 2 | Федорец | Ірина | 2 | 3 | 4 |

У такому виді (120 кортежів) це відношення містить більше інформації, ніж необхідно. Наприклад, перший кортеж цього відношення містить різні значення атрибутів Readers.Code і ReaderCode. Для одержання потрібного списку (табл. 5.4) необхідно для цього відношення зробити операцію вибірки з витягом тих кортежів, для яких виконується рівність Readers.Code = ReaderCode. Повністю ця операція виглядає так, як показано нижче:

$\sigma_{Readers.Code = ReaderCode}(П_{Code, FamilyName, Name}(Readers)) \times (П_{Code, ReaderCode, InventoryCode}(BookGiveOutRecord))$

Як ми незабаром побачимо, комбінація декартового добутку та вибірки може бути зведена до однієї операції: з'єднання.

## 5.5. Об'єднання

$R \cup S$ | Об'єднання відношень R і S з кортежами I та J відповідно можна одержати в результаті їх конкатенації з утворенням одного відношення з максимальною кількістю кортежів (I + J), якщо кортежі-дублікати виключені. При цьому відношення R і S повинні бути сумісні за об'єднанням.

Об'єднання відношень можливо тільки в тому випадку, якщо збігаються їхні схеми, тобто якщо вони мають однакову кількість атрибутів зі співпадаючими доменами (типами даних). Такі відношення є сумісними за об'єднанням. Відзначимо, що в деяких випадках для одержання двох сумісних за об'єднанням відношень може бути використана операція проекції.

Таблиця 5.4

**Обмежений декартів добуток**

| Readers.Code | FamilyName | Name | BookGiveOutRecord. Code | ReaderCode | InventoryCode |
|---|---|---|---|---|---|
| 2 | Іванов | Петро | 1 | 2 | 6 |
| 3 | Федорец | Ірина | 2 | 3 | 4 |
| 6 | Ільїн | Іван | 3 | 6 | 3 |
| 4 | Суренко | Дмитро | 4 | 4 | 6 |
| 7 | Брусов | Владимир | 5 | 7 | 7 |
| 9 | Левченко | Юлія | 6 | 9 | 12 |
| 11 | Щиглів | Петро | 7 | 11 | 10 |
| 7 | Брусов | Владимир | 8 | 7 | 8 |
| 9 | Левченко | Юлія | 9 | 9 | 12 |
| 12 | Кириленко | Віктор | 10 | 12 | 15 |

ПРИКЛАД 5.4.

Створіть список прізвищ усіх людей, інформація про яких є в БД.

$П_{FamilyName}(Readers) \cup П_{FamilyName}(Librarians)$

Для створення сумісних за об'єднанням відношень спочатку слід застосувати операцію проекції, щоб виділити з відношень Readers і Librarians стовпці з атрибутами FamilyName, виключаючи у разі потреби дублікати. Потім для комбінування отриманих проміжних відношень слід використовувати операцію об'єднання (табл. 5.5).

Таблиця 5.5

**Результат об'єднання двох відношень**

| FamilyName |
|---|
| Брусов |
| Іванов |
| Іванова |
| Ільїн |
| Іноземцева |
| Кириленко |
| Козирєв |
| Коршунова |

| FamilyName |
|---|
| Левченко |
| Лещенко |
| Мальцева |
| Ніколаєнко |
| Носенко |
| Петрова |
| Прохіна |
| Самійленко |

| FamilyName |
|---|
| Світла |
| Сіра |
| Ставка |
| Степанова |
| Суренко |
| Сизранцева |
| Федорец |
| Щиглів |

### 5.6. Різниця

| | |
|---|---|
| $R - S$ | Різниця двох відношень $R$ і $S$ складається з кортежів, які є у відношенні $R$, але відсутні у відношенні $S$. Причому відношення $R$ й $S$ повинні бути сумісні за об'єднанням. |

ПРИКЛАД 5.5.

Визначте індивідуальні коди читачів, які жодного разу не брали книги в бібліотеці.

$П_{Code}(Readers) – П_{ReaderCode}(BookGiveOutRecord)$

У цьому випадку аналогічно попередньому прикладу слід створити спільні за об'єднанням відношення Readers і BookGiveOutRecord, виконавши їх проекцію по атрибутах Code і ReaderCode. Потім слід використовувати операцію різниці (табл. 5.6).

Таблиця 5.6

**Різниця відношень Readers та BookGiveOutRecord**

| ReaderCode |
|---|
| 1 |
| 5 |
| 8 |
| 10 |

### 5.7. Операції з'єднання

Як правило, користувачів цікавить лише деяка частина всіх комбінацій кортежів декартового добутку, яка задовольняє заданим умовам. Тому замість декартового добутку зазвичай використовується одна з найважливіших операцій реляційної алгебри – ***операція з'єднання***. У результаті її виконання на базі двох вихідних відношень створюється нове відношення. Операція з'єднання є похідною від операції декартового добутку. Вона еквівалентна операції вибірки з декартового добутку двох відношень тих кортежів, які задовольняють умові, що зазначена в предикаті з'єднання. Предикат з'єднання є еквівалентом З погляду ефективної реалізації в реляційних СКБД, ця операція є однієї із самих важких і часто виявляється однієї з основних причин, що викликає проблеми із продуктивністю, властиві всім реляційним системам.

Розглянемо наступні види з'єднань:

1. Тета-з'єднання (*Θ-join*).
2. З'єднання по еквівалентності (*equi-join*) - окремий вид тета-з'єднання.
3. Природне з'єднання (*natural join*).
4. Зовнішнє з'єднання (*outer join*).
5. Напівз'єднання (*semi-join*).

### 5.7.1. Тета-з'єднання

$R \blacktriangleright\blacktriangleleft_F S$ | Операція тета-з'єднання визначає відношення, яке містить кортежі з декартового добутку відношень $R$ і $S$, що задовольняють предикату $F$. Предикат $F$ має вигляд $R.a_i \Theta S.b_i$, де замість $\Theta$ може бути зазначено один з операторів порівняння (<, <=, >, >=, =, ≈).

Позначення тета-з'єднання можна переписати на основі базових операцій вибірки й декартового добутку так, як показано нижче.

$R \blacktriangleright\blacktriangleleft_F S = \sigma_F (R \times S)$.

Так само, як і у випадку з декартовим добутком, ступенем тета-з'єднання називається сума ступенів операндів-відношень $R$ і $S$. Якщо предикат $F$ містить тільки оператор рівності (=), то це з'єднання по еквівалентності.

ПРИКЛАД 5.6.

Створіть список усіх читачів, які коли-або брали книги в бібліотеці (табл. 5.4).

У прикладі, що ілюструє декартів добуток, для одержання цього списку використовувалися операції декартового добутку й вибірки. Однак той же самий результат можна одержати за допомогою операції з'єднання по еквівалентності.

$(\Pi_{\text{Code, FamilyName, Name}}(\text{Readers})) \blacktriangleright\blacktriangleleft_{\text{Readers.Code = ReaderCode}} (\Pi_{\text{Code, ReaderCode, InventoryCode}}(\text{BookGiveOutRecord}))$

### 5.7.2. Природне з'єднання

$R \blacktriangleright\blacktriangleleft S$ | Природнім з'єднанням називається з'єднання по еквівалентності двох відношень $R$ і $S$, виконане по всіх загальних атрибутах $x$, з результатів якого виключається по одному екземпляру кожного спільного атрибута.

Степенем природного з'єднання є сума степенів операндів-відношень $R$ і $S$ мінус кількість атрибутів $x$. У прикладі тета-з'єднання для складання цього списку використовувалося з'єднання по еквівалентності. У ньому були присутні два атрибути Readers.Code і ReaderCode, що містять коди, читачів бібліотеки. Якби вони мали однакове ім'я, наприклад ReaderCode, то для усунення одного з них можна скористатися операцією природного з'єднання.

ПРИКЛАД 5.7.

Створіть список усіх читачів, які коли-небудь брали книги в бібліотеці (табл. 5.7).

$(\Pi_{\text{ReaderCode, FamilyName, Name}}(\text{Readers})) \blacktriangleright\blacktriangleleft (\Pi_{\text{Code, ReaderCode, InventoryCode}}(\text{BookGiveOutRecord}))$

Таблиця 5.7

**Природне з'єднання відношень Readers і BookGiveOutRecord**

| ReaderCode | FamilyName | Name | BookGiveOutRecord.Code | InventoryCode |
|---|---|---|---|---|
| 2 | Федорец | Ірина | 1 | 6 |
| 3 | Ільїн | Іван | 2 | 4 |
| 6 | Носенко | Олег | 3 | 3 |
| 4 | Суренко | Дмитро | 4 | 6 |
| 7 | Брусов | Владимир | 5 | 7 |
| 9 | Левченко | Юлія | 6 | 12 |
| 11 | Щиглів | Петро | 7 | 10 |
| 7 | Брусов | Владимир | 8 | 8 |
| 9 | Левченко | Юлія | 9 | 12 |
| 12 | Кириленко | Віктор | 10 | 15 |

### 5.7.3. Зовнішнє з'єднання

Найчастіше при з'єднанні двох відношень кортеж з одного відношення не знаходить відповідного кортежу в іншому відношенні. Інакше кажучи, у стовпцях з'єднання виявляються незбіжні значення. Може знадобитися, щоб рядок з одного відношення був представлений у результаті з'єднання, навіть якщо в іншому відношенні немає співпадаючого значення. Ця мета може бути досягнута за допомогою зовнішнього з'єднання.

Для позначення відсутніх значень у результуючому відношенні використовується визначник *NULL*. Перевагою зовнішнього з'єднання є зберігання вихідної інформації: кортежі, які втрачаються при використанні інших типів з'єднань.

$R \supset\!\blacktriangleleft S$ | Лівим зовнішнім з'єднанням є з'єднання, де кортежі відношення $R$, які не мають співпадаючих значень у спільних стовпцях з відношенням $S$, також включаються в результуюче відношення.

ПРИКЛАД 5.8.

Для усіх читачів вказати код, фамілію, ім’я, по батькові та інформацію з BookGiveOutRecord, для тих, що брали книги з бібліотеки (табл. 5.8).

$$(\Pi_{\text{Code, FamilyName, Name, Patronymic}}(\text{Readers})) \supset\!\blacktriangleleft (\text{BookGiveOutRecord})$$

Строго говорячи, у прикладі показане ***ліве (природне) зовнішнє з'єднання*****,** оскільки в результуючім відношенні втримуються всі кортежі лівого відношення. Існує також ***праве зовнішнє з'єднання*****,** назване так тому, що в результуючім відношенні втримуються всі кортежі правого відношення. Крім того, існує й ***повне зовнішнє з'єднання*****,** у результуюче відношення якого містяться всі кортежі з обох відношень, де для позначення незбіжних значень кортежів використовуються визначники *NULL*.

Таблиця 5.8

**Зовнішнє з'єднання проекції відношення Readers та BookGiveOutRecord**

| Reader.Code | FamilyName | Name | Patronymic | OutLibrarianCode | InventoryCode | IssueDate | ReturnDate | FactReturnDate | InLibrarianCde |
|---|---|---|---|---|---|---|---|---|---|
| 1 | Іванов | Петро | Іванович | NULL | NULL | NULL | NULL | NULL | NULL |
| 2 | Федорец | Ірина | Олегівна | 4 | 6 | 11.09.2004 | 25.09.2004 | 24.09.2004 | 3 |
| 3 | Ільїн | Іван | Петрович | 4 | 4 | 02.09.2004 | 16.09.2004 | 11.12.2004 | 3 |
| 4 | Суренко | Дмитро | Павлович | 3 | 6 | 30.10.2004 | 13.11.2004 | 10.01.2005 | 6 |
| 5 | Коршунова | Наталя | Юріївна | NULL | NULL | NULL | NULL | NULL | NULL |
| 6 | Носенко | Олег | Володимирович | 4 | 3 | 02.09.2004 | 16.09.2004 | 16.09.2004 | 1 |
| 7 | Брусов | Владимир | Михайлович | 10 | 7 | 10.11.2009 | 24.11.2009 | 24.11.2009 | 12 |
| 7 | Брусов | Владимир | Михайлович | 9 | 8 | 07.03.2009 | 21.03.2009 | 10.04.2009 | 10 |
| 8 | Козирєв | Олексій | Сергійович | NULL | NULL | NULL | NULL | NULL | NULL |
| 9 | Левченко | Юлія | Павлівна | 7 | 12 | 15.12.2009 | 29.12.2009 | NULL | 9 |
| 9 | Левченко | Юлія | Павлівна | 8 | 12 | 05.02.2010 | 29.02.2010 | NULL | 11 |
| 10 | Світла | Тетяна | Іванівна | NULL | NULL | NULL | NULL | NULL | NULL |
| 11 | Щиглів | Петро | Євгенович | 8 | 10 | 06.02.2009 | 20.02.2009 | 19.02.2009 | 7 |
| 12 | Кириленко | Віктор | Олександрович | 10 | 15 | 21.09.2010 | 05.10.2010 | 03.10.2010 | 9 |

### 5.7.4. Напівз'єднання

$R \blacktriangleright_F S$ | Операція напівз'єднання визначає відношення, яке містить ті кортежі відношення $R$, які входять у з'єднання відношень $R$ і $S$.

Перевага напівз'єднання полягає в тому, що воно дозволяє скоротити кількість кортежів, які потрібно обробити для одержання з'єднання. Це особливо корисно при обчисленні з'єднань у розподілених системах. Операцію напівз'єднання можна сформулювати й за допомогою операторів проекції й з'єднання:

$$R \not\in_F S = \Pi_A(R \blacktriangleright\!\blacktriangleleft_F S).$$

Тут $A$ – це набір усіх атрибутів у відношенні $R$. Насправді це напівтетаз'єднання, причому слід зазначити, що існують напівз'єднання по еквівалентності та напівприродне з'єднання.

ПРИКЛАД 5.9.

Створіть звіт, що містить повну інформацію про всіх читачів на ім'я «Дмитро», які коли-небудь брали книги в бібліотеці (табл. 5.9).

Readers ► Readers.Code = BookGiveOutRecord.ReaderCode AND Reader.Name = 'Дмитрий' BookGiveOutRecord

### 5.8. Перетинання

| | |
|---|---|
| $R \cap S$ | Операція перетинання визначає відношення, яке містить кортежі, що присутні, як у відношенні *R*, так і у відношенні *S*. Відношення *R* й *S* повинні бути сумісні за об'єднанням. |

Перетинання можна записати, використовуючи оператори різниці множин: $R \cap S = R - (R - S)$.

Таблиця 5.9

**Результат напівз'єднання відношень Reader і BookGiveOutRecord**

| Code | FamilyName | Name | Patronymic | ReaderCard Number | PasportCode | Job | Post | Note |
|---|---|---|---|---|---|---|---|---|
| 4 | Суренко | Дмитро | Павлович | 543 | 6 | ДГУ, каф. геофізики | Ст. викладач | null |

ПРИКЛАД 5.10.

Створить список бібліотекарів, які є одночасно читачами. Вкажіть їх код паспорту, прізвище, ім’я та по батькові.

$(\Pi_{\text{PasportCode, FamilyName, Name, Patronymic}}(\text{Readers})) \cap (\Pi_{\text{PasportCode, FamilyName, Name, Patronymic}}(\text{Librarians}))$

Для виконання умови сумісності за об’єднанням відношень, що перетинаються, були виконані відповідні операції проекції.

Данні, які внесені до відношень „ЧИТАЧИ” та „БІБЛІОТЕКАРІ” (додаток В) та бібліотекарі свідчать, що результатом їх перетинання є порожня множина, тобто жоден з бібліотекарів не є читачем.

### 5.9. Ділення

Оператор ділення може бути корисний у випадку запитів особливого типу, які досить часто виконуються до БД. Припустимо, що відношення *R* визначене на множині атрибутів *A*, а відношення *S* – на множині атрибутів *B*, причому $B \subseteq A$ (тобто *B* є підмножиною *A*). Нехай $C = A\text{-}B$, тобто *C* є множиною атрибутів відношення *R*, яка не є атрибутами відношення *S*. Тоді визначення оператора ділення буде виглядати в такий спосіб.

| | |
|---|---|
| $R \div S$ | Результатом оператора ділення є набір кортежів відношення *R*, певних на множині атрибутів *C*, які відповідають комбінації всіх кортежів відношення *S*. |

Цей оператор можна записати на базі інших основних операторів:

$T_1 = \Pi_C (R)$

$T_2 = \Pi_C ((S \times T_1) - R)$

$T = T1 - T2$

ПРИКЛАД 5.11.

Створіть список кодів усіх читачів, які народилися після 31 грудня 1960 року.

R = П Code, PassportCode (Readers) (табл. 5.10);

$S = П_{Code} (\sigma_{Birthday > 31.12.1960} (PasportData))$ (табл. 5.11);

$R \div S = (П_{Code,\ PassportCode} (Readers)) \div (П_{Code} (\sigma_{Birthday > 31.12.1960} (PasportData)))$ (табл. 5.12).

Для розв'язання поставленого завдання спочатку необхідно за допомогою проекції одержати тільки номери кодів читачів та кодів паспортів з відношення Readers (табл. 5.10). Потім за допомогою оператора вибірки виконати пошук у відношенні PasportData усіх кодів паспортів, у яких дата народження зазначена після 31 грудня 1960 року (табл. 5.11). Після цього потрібно застосувати оператор ділення й одержати нове відношення (табл. 5.12).

Таблиця 5.10 Таблиця 5.11 Таблиця 5.12

**Відношення R.**

| Code | PasportCode |
|---|---|
| 1 | 4 |
| 2 | 1 |
| 3 | 11 |
| 4 | 6 |
| 5 | 8 |
| 6 | 5 |
| 7 | 24 |
| 8 | 15 |
| 9 | 18 |
| 10 | 22 |
| 11 | 14 |
| 12 | 17 |

**Відношення S.**

| Code |
|---|
| 3 |
| 5 |
| 7 |
| 8 |
| 12 |
| 14 |
| 15 |
| 16 |
| 18 |
| 19 |
| 20 |
| 21 |
| 22 |
| 23 |
| 24 |

**Результат R ÷ S.**

| Code |
|---|
| 5 |
| 6 |
| 7 |
| 8 |
| 9 |
| 10 |
| 11 |

## 5.10. КОНТРОЛЬНІ ПИТАННЯ

1. Які особливості та властивості реляційної алгебри Вам відомі?
2. Які операції реляційної алгебри Вам відомі?
3. Яка різниця між операціями вибірки та проекції?
4. В якому із двох відношень кількість атрибутів та кортежів буде більшою, якщо перше є результатом декартового добутку відношень *R* з *I* кортежів та *N* атрибутів та *S* з *J* кортежів і *M* атрибутів, а друге – об'єднання тих самих відношень?
5. У чому полягає суть операцій реляційної алгебри, які потребують сумісних за об'єднанням вихідних відношень?
6. Які операції реляційної алгебри Вам відомі, що є похідними від декартового добутку?
7. Який з операторів порівняння перетворює тета-з'єднання у природне з'єднання?
8. В чому полягає різниця між зовнішнім та іншими типами з'єднань?
9. Як формується відношення, що є результатом роботи відомих Вам типів зовнішніх з'єднань?

*Розглянутий матеріал дає студентові поняття про основні операції реляційної алгебри .*

# 6. Оператор SELECT

*Цей розділ присвячений вивченню оператора SELECT - одного з найбільш важливих операторів SQL, приводиться його повний формат, розглядаються приклади.*

## 6.1. Формат оператора SELECT

Оператор *SELECT* – один з найбільш важливих операторів *SQL*, що використовується значно частіше інших операторів. Це дуже потужний оператор [4, 6]. Він дозволяє робити вибір необхідної інформації з відношень БД і перетворювати її до виду, який задовольняє користувачів. За його допомогою можна реалізувати досить складні й громіздкі умови вибору даних з різних взаємопов'язаних таблиць.

Оператор *SELECT* повністю абстрагований від питань представлення даних [4, 6 … 8]. Уся увага при його застосуванні сконцентрована на проблемах доступу до даних. Він має наступний формат:

**SELECT [DISTINCT | ALL] {* | <**значення **> [, <**вираз **> ...]}**
**FROM** <таблиця >[,<таблиця > ...]
[**WHERE** <умови пошуку >]
[**ORDER BY** <список стовпців >]
[**GROUP BY** стовпець [**COLLATE** collation]
[, стовпець] [**COLLATE** collation], ...]
[**HAVING** <умови пошуку >]
[**UNION** <оператор SELECT >]
**[PLAN <**план виконання запиту **>]**

Цей формат з першого погляду досить громіздкий і тому здасться складним. Але при поступовому вивченні можливостей оператору *SELECT* ви побачите наскільки він простий.

## 6.2. Найпростіший вид оператора SELECT (SELECT…FROM)

У найпростішому випадку оператор *SELECT* має вигляд:

**SELECT {*| <**значення > [,<значення >...]}
**FROM** <таблиця > [,<таблиця >...]

З точки зору реляційної алгебри даний оператор виконує операцію проекції. Після *FROM* наводиться список таблиць БД, з яких будемо вибирати інформацію. Після *SELECT* наводиться список значень. У більшості випадків це імена стовпців таблиць, перерахованих після пропозиції *FROM*. Зірочка вказує на те, що в результат виконання запиту потрібно включити всі стовпці таблиць.

ПРИКЛАД 6.1.

Видати набір даних, що вміщує усі атрибути та кортежі відношення Readers (табл. 6.1).

**SELECT** *
**FROM** Readers;

Це еквівалентно:

**SELECT** Code, FamilyName, Name, Patronymic, ReaderCardNumber,
PasportCode, Job, Post, Note
**FROM** Readers;

### 6.3. Розрахунок значень у стовпцях на основі арифметичних виразів

Для розрахунків значень у стовпцях після *SELECT* слід дотримуватися загальних правил формування арифметичних виразів, прийнятих в алгоритмічних мовах.

**SELECT** {* | <значення > [, <вираз >]}
**FROM** <таблиця > [,<таблиця > …]

ПРИКЛАД 6.2.

Для кожної книги з таблиці BookInventoryNumbers визначимо добуток її ціни на суму значень коду книги й фонду книги.

**SELECT** Code, InventoryNumber, (BookCode + FundCode) * Cost
**FROM** BookInventoryNumbers

Необхідно розуміти, що цей приклад не має логічного навантаження. Він наведений тільки за для демонстрації можливості оператору *SELECT*. Результат обчислення виразу *(BookCode + FundCode) · Cost* для кожного запису з таблиці BookInventoryNumbers записаний у стовпець, якому за замовчуванням СУБД привласнила ім'я Column3 (табл. 6.2).

Можливо перейменувати будь який стовпець таблиці, що є результатом роботи оператору *SELECT*. Задля цього після виразу або імені стовпця вихідної таблиці потрібно вказати пропозицію *AS* і далі нове ім'я:

**SELECT** ... {* | <значення > [, <вираз [**AS** <ім'я стовпця >]> ...]}
**FROM ...**

Таблиця 6.1

**Набір даних, що вміщує усі атрибути та кортежі відношення Readers**

| Code | FamilyName | Name | Patronymic | ReaderCard-Number | PasportCode | Job | Post | Note |
|---|---|---|---|---|---|---|---|---|
| 1 | Іванов | Петро | Іванович | 317 | 4 | НГУ, каф. ЭВТ | Асистент | null |
| 2 | Федорец | Ірина | Олегівна | 28 | 1 | НГУ, АХЧ | Вахтер | null |
| 3 | Ільїн | Іван | Петрович | 1345 | 11 | НГУ, каф. фізики | Доцент | null |
| 4 | Суренко | Дмитро | Павлович | 543 | 6 | НГУ, каф. геофізики | Ст. викладач | null |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 5 | Коршунова | Наталя | Юріївна | 128 | 8 | НГУ, каф. геоіформатики | Асистент | null |
| 6 | Носенко | Олег | Володимирович | 5672 | 5 | НГУ, ІКК | Інженер | null |
| 7 | Брусов | Владимир | Михайлович | 485 | 24 | НГУ, каф. геодезії | Лаборант | null |
| 8 | Козирєв | Олексій | Сергійович | 759 | 15 | НГУ, каф. кримінології | Професор | null |
| 9 | Левченко | Юлія | Павлівна | 146 | 18 | НГУ, каф. політичної теорії | Завідувач кафедри | null |
| 10 | Світла | Тетяна | Іванівна | 2021 | 22 | НГУ, каф. перекладу | Ст. викладач | null |
| 11 | Щиглів | Петро | Євгенович | 997 | 14 | НГУ, каф. електропостачання | Асистент | null |
| 12 | Кириленко | Віктор | Олександрович | 1010 | 17 | НГУ, каф. електроприводу | Заступник декана | null |

ПРИКЛАД 6.3.

Надайте ім'я „*Example*" стовпцю, що обчислюється за допомогою арифметичного виразу в попередньому прикладі.

```
SELECT Code, InventoryNumber,
        (BookCode + FundCode) * Cost AS Example
    FROM BookInventoryNumbers;
```

Таблиця 6.2

**Набір даних з таблиці BookInventoryNumbers**

| Code | InventoryNumber | Column3 |
|---|---|---|
| 1 | 4567890 | 31,12 |
| 2 | 4510000 | 66,99 |
| 3 | 4532477 | 136,04 |
| 4 | 4512890 | 64,95 |
| 5 | 4678532 | 397,46 |
| 6 | 4632112 | 80,80 |
| 7 | 7569832 | 661,5 |
| 8 | 5478956 | 405,9 |
| 9 | 2145876 | 592,5 |
| 10 | 5214786 | 360,5 |
| 11 | 5268933 | 816,2 |
| 12 | 7865890 | 277,16 |
| 13 | 6589321 | 476,97 |
| 14 | 7812639 | 673,82 |
| 15 | 7523690 | 419,85 |

## 6.4. Усунення записів, які повторюються

Записами, що повторюються вважаються ті, де містяться ідентичні значення у всіх стовпцях набору даних. Якщо результуючий набір даних не повинен їх вміщувати, тоді після *SELECT* вказують пропозицію *DISTINCT*. Якщо в результуючий набір даних необхідно включити всі записи, після *SELECT* вказують *ALL*. Ця пропозиція діє за замовчуванням:

```
SELECT [DISTINCT | ALL] {* | <значення > [, < вираз >]}
    FROM <таблиця > [, <таблиця > …]
```

ПРИКЛАД 6.4.

Виведіть коди бібліотекарів, які видавали книги (табл. 6.3).

```
SELECT  OutLibrarianCode
    FROM BookGiveOutRecord;
```

Зверніть увагу, що результат виконання запиту містить значення,

що дублюються, оскільки, на відміну від операції реляційної алгебри projection, оператор *SELECT* не виключає значень, що дублюються, при виконанні проекції одного або декількох стовпців. Для вилучення рядків, що дублюються, з результуючої таблиці використовується *DISTINCT* (табл. 6.4):

**SELECT DISTINCT** OutLibrarianCode
**FROM** BookGiveOutRecord;

### 6.5. Пропозиція ORDER BY

У відношення, що формує оператор *SELECT*, порядок розміщення кортежів невизначений. Це зручно далеко не завжди. Установити, за якими атрибутах необхідно відсортувати кортежі можна в пропозиції *ORDER BY*. Вона містить список імен стовпців, що розділені комами, по яких буде проводитися сортування. Перший стовпець у списку буде використаний для глобального сортування, другий стовпець – для сортування усередині групи, що визначена єдиним значенням першого стовпця, і т.д.:

**SELECT** [**DISTINCT | ALL**] {* | <значення > [, <вираз >]}
**FROM** <таблиця >[,<таблиця > ...]
**ORDER BY** <список стовпців >

### 6.6. Предикати пошуку в пропозиції WHERE оператора SELECT

У набір даних, що повертає оператор *SELECT*, можна включати тільки ті записи, які задовольняють деяким заданим умовам відбору або умовам пошуку, які вказують у пропозиції *WHERE* оператору *SELECT*:

**SELECT [DISTINCT| ALL]** {* | <стовпець > [,<вираз >]}
**FROM** <таблиця > [,<таблиця >…]
**ORDER BY** <список стовпців >
**WHERE <**умова пошуку **>**

Умови пошуку ділять на п'ять основних типів. Якщо використовувати термінологію *ISO*, то розрізняють п'ять основних предикатів: порівняння, влучення в діапазон, відповідність шаблону, приналежність до множини та наявність визначника *NULL*.

Таблиця 6.3

**Дубльовані значення з таблиці BookGiveOutRecord**

| OutLibrarianCode |
|---|
| 4 |
| 4 |
| 4 |
| 3 |
| 10 |
| 7 |
| 8 |
| 9 |
| 8 |
| 10 |

Таблиця 6.4

**Недубльовані значення з таблиці BookGiveOutRecord**

| OutLibrarianCode |
|---|
| 3 |
| 4 |
| 7 |
| 8 |
| 9 |
| 10 |

ПРИКЛАД 6.5.
Видати список читачів бібліотеки за абеткою (табл. 6.5).

```
SELECT *
    FROM Readers
     ORDER BY FamilyName, Name, Patronymic
```

Порівняння дозволяє порівнювати результати обчислення одного виразу з результатами обчислення іншого виразу. Як вирази можуть виступати: константи, значення атрибутів, арифметичні вирази, скалярні оператори *SELECT*. Між виразами можуть бути використані наступні оператори порівняння: = (дорівнює); < (менше); > (більше); <= (менше або дорівнює); >= (більше або дорівнює); !< (не менше, тобто більше або дорівнює); !> (не більше, тобто менше або дорівнює); <> (не дорівнює); != (не дорівнює).

Результати порівняння декількох виразів можуть бути об'єднані операторами *OR* і *AND*. Використання оператора *NOT* дозволяє одержати інверсію результату порівняння виразів. Перевірку входження в діапазон можна організувати за допомогою об'єднання оператором *AND* двох результатів порівняння виразів. Також у мові *SQL* є спеціальна функція *BETWEEN*, яка здійснює перевірку входження результатів обчислення виразу в заданий діапазон. Для перевірки відповідності виразу деякому шаблону в мові *SQL* передбачені наступні оператори: *LIKE*, *CONTAINING*, *STARTING*. Входження результатів обчислення виразу у множину перевіряється за допомогою оператора *IN*. Перевірка наявності невизначеного значення в атрибуті відношення здійснюється оператором *IS NULL*.

Таблиця 6.5

**Список читачів бібіліотеки за абеткою**

| Code | FamilyName | Name | Patronymic | RederCardNumber | PasportCode | Job | Post | Note |
|---|---|---|---|---|---|---|---|---|
| 7 | Брусов | Владимир | Михайлович | 485 | 24 | НГУ, каф. геодезії | Лаборант | null |
| 1 | Іванов | Петро | Іванович | 317 | 4 | ДГУ, каф. ЭВТ | Асистент | null |
| 3 | Ільїн | Іван | Петрович | 1345 | 11 | ДГУ, каф. фізики | Доцент | null |
| 12 | Кириленко | Віктор | Олександрович | 1010 | 17 | НГУ, каф. електроприводу | Заступник декану | null |
| 8 | Козирєв | Олексій | Сергійович | 759 | 15 | НГУ, каф. кримінології | Професор | null |
| 5 | Коршунова | Наталя | Юріївна | 128 | 8 | ДГУ, каф. геоінформатики | Асистент | null |
| 9 | Левченко | Юлія | Павлівна | 146 | 18 | НГУ, каф. політичної теорії | Завідувач кафедри | null |
| 6 | Носенко | Олег | Володимирович | 5672 | 5 | ДГУ, ІКК | Інженер | null |
| 10 | Світла | Тетяна | Іванівна | 2021 | 22 | НГУ, каф. перекладу | Ст. викладач | null |
| 4 | Суренко | Дмитро | Павлович | 543 | 6 | ДГУ, каф. геофізики | Ст. викладач | null |
| 2 | Федорец | Ірина | Олегівна | 28 | 1 | ДГУ,АХЧ | Вахтер | null |
| 11 | Щиглів | Петро | Євгенійович | 997 | 14 | НГУ, каф. електропостачання | Асистент | null |

## 6.7. Порівняння результатів обчислення виразів

**6.7.1. Порівняння константи зі значенням в атрибуті** відношення є найбільш простою умовою пошуку, яка використовується у пропозиції *WHERE* оператора *SELECT*.

ПРИКЛАД 6.6.

Складіть список бібліотекарів з табельним номером більше 80.

$$_{ClockNum}\sigma_{ber > 80}\ (Librarians)$$

```
SELECT *
    FROM Librarians
     WHERE ClockNumber > 80
```

Тут *Librarians* – є вихідним відношенням, а вираз *ClockNumber > 80* – предикатом. Операція вибірки визначає нове відношення, що містить тільки ті кортежі вихідного відношення, у яких значення атрибута *ClockNumber* перевищує 80 (табл. 5.1).

**6.7.2. Порівняння значень атрибутів відношень** у більшості випадків використовується для реалізації їх внутрішнього з'єднання.

ПРИКЛАД 6.7.

Створіть список усіх читачів, які коли-небудь брали книги в бібліотеці.

$$(\Pi_{Code,\ FamilyName,\ Name}\ (Readers)) \blacktriangleright\blacktriangleleft_{Readers.Code\ =\ ReaderCode} (\Pi_{Code,\ ReaderCode,\ InventoryCode}\ (BookGiveOutRecord))$$

У прикладі тета-з'єднання для складання цього списку використовувалося з'єднання по еквівалентності. У ньому були присутні два атрибути *Readers.Code* і *ReaderCode*. Якби вони мали однакове ім'я, наприклад *ReaderCode*, то для усунення в результуючім відношенні одного з них можна було б скористатися операцією природного з'єднання:

$$(\Pi_{ReaderCode,\ FamilyName,\ Name}\ (Readers)) \blacktriangleright\blacktriangleleft (\Pi_{Code,\ ReaderCode,\ InventoryCode}\ (BookGiveOutRecord))$$

Оператор *SELECT* буде мати такий вигляд:

```
SELECT ReaderCode, FamilyName, Name, BookGiveOutRecord.Code,
            InventoryCode
      FROM BookGiveOutRecord, Readers
       WHERE Readers.Code = BookGiveOutRecord. ReaderCode
```

При виконанні оператора *SELECT* для кожного кортежу з відношення *BookGiveOutRecord* шукається рядок у таблиці *Readers*, де значення атрибута *ReaderCode* збігається зі значенням атрибута *Code* поточного кортежу відношення *Readers*. При цьому байдуже, у якому порядку ставити стовпці таблиць в умові пошуку: *Readers.Code = BookGiveOutRecord.ReaderCode* ідентично

*BookGiveOutRecord.ReaderCode = Readers.Code*.

Для тета-з'єднання з'єднанні двох відношень *R* і *S* логічний порядок формування результуючого відношення можна визначити у такий спосіб.

1. З атрибутів, які зазначені після пропозиції *SELECT*, складається декартів добуток шляхом зчеплення результуючих атрибутів кожного кортежу з відношення *R* й результуючих атрибутів кортежів з відношення *S*.

2. З відношення, що є результатом попереднього пункту, відкидаються усі кортежі, що не задовольняють умовам пошуку в пропозиції *WHERE*.

ЗАУВАЖЕННЯ. Визначення "логічний порядок формування результуючого відношення" вжите не випадково. У проектуванні даних завжди розрізняють два рівні – логічний та фізичний. Логічний рівень – це часто досить абстрактний рівень: так нам легше розуміти процеси. Фізичний рівень визначає процеси, що не лежать на поверхні, які дійсно протікають та у більшості випадків сховані від погляду. Фізичні процеси, що реально протікають при виконанні запиту, можуть не збігатися з логічним розумінням про них.

*SQL*-сервер при виконанні запиту завжди прагне його оптимізувати, тобто виконати максимально швидко при мінімальних витратах ресурсів. Зокрема, оптимізація запитів в *InterBase* являє собою "чорний ящик", тобто не можна сказати, як саме буде виконуватися конкретний запит, оскільки, крім іншого, при оптимізації не останню роль відіграє поточний стан БД. Оптимізація запитів будуть розглянута нижче.

**6.7.3. Порівняння значення стовпця з результатом обчислення виразу** найчастіше застосовується при використанні механізму вкладених підзапитів (операторів *SELECT*, що вкладені один в один), мова про які піде нижче. Спочатку розглянемо випадок, коли результат обчислення виразу порівнюється із вмістом атрибута відношення.

ПРИКЛАД 6.8.

Для кожної книги визначимо добуток її ціни на суму значень коду книги й фонду книги. Виберемо тільки ті, у яких результат, отриманий при обчисленні даного виразу більший 120 (табл.6.6).

```
SELECT Code, InventoryNumber, (BookCode + FundCode) * Cost AS
      Example
     FROM BookInventoryNumbers
      WHERE ((BookCode + FundCode) * Cost) > 120
```

Таблиця 6.6

**Набір даних з таблиці BookInventoryNumbers**

| Code | InventoryNumber | Example |
|---|---|---|
| 3 | 4532477 | 136,04 |
| 5 | 4678532 | 397,46 |
| 7 | 7569832 | 661,5 |
| 8 | 5478956 | 405,9 |

| | | |
|---|---|---|
| 9 | 2145876 | 592,5 |
| 10 | 5214786 | 360,5 |
| 11 | 5268933 | 816,2 |
| 12 | 7865890 | 277,16 |
| 13 | 6589321 | 476,97 |
| 14 | 7812639 | 673,82 |
| 15 | 7523690 | 419,85 |

**6.7.4. Використання операторів AND, OR і NOT** дозволяє будувати більш складні умови пошуку. Для усунення будь-якої можливої неоднозначності рекомендується використовувати дужки. Обчислення виразів виконується за наступними правилами:

вираз обчислюється зліва направо;
першими обчислюються підвираз в дужках;
оператори *NOT* виконуються до операторів *AND* і *OR*;
оператори *AND* виконуються до операторів *OR*;

ПРИКЛАД 6.9.

Перерахуйте прізвища, імена та по батькові читачів, що займають посади доцентів і асистентів (табл. 6.7).

**SELECT** Code, FamilyName, Name, Patronymic, Job, Post
FROM Readers
**WHERE** Post = ‘Доцент’ **OR** Post = ’Асистент’;

У цьому прикладі для вибірки відомостей про читачів, що займають посади доцентів і асистентів у пропозиції *WHERE* використовується логічний оператор *OR*: Post = ‘Доцент’ або Post = ’Асистент’.

## 6.8. Псевдоніми таблиць

У прикладі 6.7 у переліку стовпців після пропозиції *SELECT* і в умовах пошуку після *WHERE* перед іменем стовпця через крапку написана назва таблиці. У ряді випадків вказівка імені таблиці перед іменем стовпця є вкрай необхідною, оскільки в різних таблицях можуть міститися однойменні стовпці (як у нашому прикладі), і *SQL*-сервер повинен знати, зі стовпцем якої таблиці він має справу.

Таблиця 6.7

**Вибірка відомостей про читачів, що займають посади доцентів і асистентів**

| Code | FamilyName | Name | Patronymic | Job | Post |
|---|---|---|---|---|---|
| 1 | Іванов | Петро | Іванович | ДГУ, каф. ЕВТ | Асистент |
| 3 | Ільїн | Іван | Петрович | ДГУ, каф. фізики | Доцент |
| 5 | Коршунова | Наталя | Юріївна | ДГУ, каф. геоінформатики | Асистент |
| 11 | Щиглів | Петро | Євгенійович | НГУ, каф. електропостачання | Асистент |

Використання імен таблиць для ідентифікації стовпців незручно через свою громіздкість. Псевдоніми, які вказуються через пробіл після імені таблиці у пропозиції *FROM*, роблять запис оператора *SELECT* компактнішим:

**SELECT [DISTINCT | ALL]** {* | <стовпець > [, <вираз >]}
**FROM** <таблиця псевдонім > [, <таблиця псевдонім > …]
**ORDER BY** <список стовпців >
**WHERE <**умова пошуку **>**

Наприклад, запит, який наведений у ПРИКЛАДІ 6.7, після введення в нього псевдонімів таблиць є набагато компактнішим:

**SELECT** ReaderCode, FamilyName, Name, B.Code, InventoryCode
**FROM** BookGiveOutRecord B, Readers R
**WHERE** R.Code = B.ReaderCode

### 6.9. Перевірка входження значення виразу в заданий діапазон

Використання оператора порівняння *BETWEEN* дозволяє в умові пошуку вказати, що деяке значення повинне перебувати в інтервалі між двома іншими. Зарезервоване слово *NOT* інвертує умову.

<значення **> [NOT] BETWEEN** <значення > **AND** <значення >

ПРИКЛАД 6.10.

Перелічіть інвентарні номери книг, ціна яких лежить у діапазоні від 20 до 60 гривень включно (табл. 6.8).

**SELECT** InventoryNumber, Cost
**FROM** BookInventoryNumbers
**WHERE** Cost **BETWEEN** 20 **AND** 60**;**

Наведений вище запит можна переписати у такий спосіб:

**SELECT** InventoryNumber, Cost
**FROM** BookInventoryNumbers
**WHERE** Cost >= 20 **AND** Cost <=60;

Багато хто вважає, що перевірка входження у діапазон за допомогою оператора *BETWEEN* є більш простим способом запису, ніж використання звичайної перевірки, що наведена в останньому операторі *SELECT*.

Таблиця 6.8

**Перелік інвентарних номерів книг, ціна яких лежить у діапазоні від 20 до 60 гривень**

| InventoryNumber | Cost |
|---|---|
| 510000 | 34.009998 |
| 4678532 | 45.099998 |
| 2145876 | 36.049999 |
| 7865890 | 36.689999 |
| 7812639 | 27.99 |
| 510000 | 34.009998 |
| 4678532 | 45.099998 |
| 2145876 | 36.049999 |
| 7865890 | 36.689999 |
| 7812639 | 27.99 |

## 6.10. Перевірка відповідності значення виразу заданому шаблону

**6.10.1. Оператор LIKE** задає шаблони порівняння строкових значень. Він чутливий до регістру. Якщо необхідно перевірити задовольняє чи ні значення стовпця або результат обчислення строкового виразу шаблону (<*значення* > після пропозиції *LIKE*), в умові пошуку необхідно вказати:

<значення > **[NOT] LIKE** < значення > [**ESCAPE** <символ >]

У шаблоні використовуються спеціальні символи – "%" та "_". Символ "%" (відсоток) – для вказівки будь-якого значення будь-якої довжини, а символ "_" (підкреслення) – для вказівки будь-якого одиничного символу. Наприклад:

IssuePlace *LIKE* ‘M%’ – цей шаблон означає, що перший символ значення обов'язково повинен бути символом M, а всі інші символи не представляють інтересу й не перевіряються;

IssuePlace *LIKE* ‘M_’ – цей шаблон означає, що значення повинне мати довжину, яка дорівнює строго двом символам, причому першим символом обов'язково повинен бути символ ‘M';

IssuePlace *LIKE* ‘%e' – цей шаблон визначає будь-яку послідовність символів довжиною не менше одного символу, причому останнім символом обов'язково повинен бути символ ‘e’;

IssuePlace *LIKE* ‘%Дніпропетровська область%’ – цей шаблон означає, що нас цікавить будь-яка послідовність символів, що включає підрядок ‘Дніпропетровська область’;

IssuePlace *NOT LIKE* ‘M%’ – цей шаблон вказує, що потрібні будь-які рядки, які не починаються із символу ‘M'.

Якщо необхідний рядок повинен включати також службовий символ, який використовується у якості символу підстановки, то слід застосувати «escape»-символ, помістивши його перед символом підстановки. Наприклад, для перевірки значень на відповідність ‘15%’ можна скористатися такою умовою пошуку:

**LIKE** ‘15#%**’ ESCAPE** ‘**#'**

ПРИКЛАД 6.11.

За допомогою механізму пошуку за шаблоном вивести код, серію та номер паспорта для осіб, які проживають у Дніпропетровській області (табл. 6.9).

```
SELECT Code, Series, Number, IssuePlace
    FROM PasportData
    WHERE IssuePlace LIKE ‘%Дніпропетровська область%’;
```

Таблиця 6.9.

**Набір даних з таблиці PasportData**

| Code | Series | Number | IssuePlace |
|---|---|---|---|
| 3 | АБ | 87134 | Дніпропетровська область, село Солоне |
| 7 | АЗ | 43188 | Дніпропетровська область, м. Дніпродзержинськ |
| 12 | ІК | 45190 | Дніпропетровська область, село Петропавлівка |

**6.10.2. Оператор STARTING** чутливий до регістру. Він використовується, якщо в умові пошуку необхідно щоб значення якого-небудь символьного стовпця або виразу починалося з певної послідовності символів:

<значення > [**NOT**] **STARTING** [**WITH**] <значення >

ПРИКЛАД 6.12.

Виберемо коди, прізвища, імена та по батькові читачів, що починаються з букви «І» (табл. 6.10).

**SELECT** Code, FamilyName, Name, Patronymic
**FROM** Readers
**WHERE** FamilyName **STARTING WITH** ‘І’;

**6.10.3. Оператор CONTAINING** не чутливий до регістру. Він використовується, якщо необхідно, щоб значення якого-небудь стовпця або виразу містило в собі (неважливо з якої позиції) деяку послідовність символів:

<значення > **[NOT] CONTAINING** <значення >

ПРИКЛАД 6.13.

Виберемо коди, прізвища, імена та по батькові читачів, у прізвищі яких міститься послідовність букв «ко» (табл. 6.11).

**SELECT** Code, FamilyName, Name, Patronymic
**FROM** Readers
**WHERE** FamilyName **CONTAINING** ‘ко’;

Таблиця 6.10

**Вібірка кодів, прізвищ, імен та по батькові читачів, що починаються з букви «І»**

| Code | FamilyName | Name | Patronymic |
|---|---|---|---|
| 1 | Іванов | Петро | Іванович |
| 3 | Ільїн | Іван | Петрович |

Таблиця 6.11

**Вібірка кодів, прізвищ, імен та по батькові читачів, у прізвищах яких міститься послідовність букв «ко»**

| Code | FamilyName | Name | Patronymic |
|---|---|---|---|
| 4 | Суренко | Дмитро | Павлович |
| 5 | Коршунова | Наталя | Юріївна |
| 6 | Носенко | Олег | Володимирович |
| 8 | Козирєв | Олексій | Сергійович |
| 9 | Левченко | Юлія | Павлівна |
| 12 | Кириленко | Віктор | Олександрович |

**6.10.4. Функція UPPER(<*значення* >)** використовується для перетворення букв символьних значень (змісту стовпця, результату обчислення виразу) до заголовних букв. Функція *UPPER* може фігурувати як у списку стовпців результуючого набору даних (після пропозиції *SELECT*), так і в умові пошуку в пропозиції *WHERE*. Вона використовується в умовах пошуку, коли необхідно ігнорувати регістр букв.

ПРИКЛАД 6.14.

Виберемо коди, прізвища, імена та по батькові читачів, у прізвищі яких міститься послідовність букв «ко» незалежно від того заголовні вони або прописні. Прізвище вивести заголовними буквами (табл. 6.12).

**SELECT** Code, UPPER(FamilyName), Name, Patronymic
**FROM** Readers
**WHERE** FamilyName **CONTAINING** ‘ко’;

Таблиця 6.12

**Набір даних з таблиці Readers**

| Code | FamilyName | Name | Patronymic |
|---|---|---|---|
| 4 | СУРЕНКО | Дмитро | Павлович |
| 5 | КОРШУНОВА | Наталя | Юріївна |
| 6 | НОСЕНКО | Олег | Володимирович |
| 8 | КОЗИРЄВ | Олексій | Сергійович |
| 9 | ЛЕВЧЕНКО | Юлія | Павлівна |
| 12 | КИРИЛЕНКО | Віктор | Олександрович |

**6.10.5. Використання функції CAST** необхідно у випадку, коли виникає потреба трактувати значення одного типу як значення іншого типу. Наприклад, використовувати числове значення як символьний рядок або навпаки. У цьому випадку застосовують функцію

**CAST** (<значення > **AS** <тип даних >)

Функція *CAST* робить копію значення, перетворюючи його до зазначеного типу даних. При цьому не слід забувати про безліч типів даних, у яке може бути перетворене дане значення:

| Тип даних | Можна привести до типів |
|---|---|
| NUMERIC | CHARACTER, DATE |
| CHARACTER | NUMERIC, DATE |
| DATE | CHARACTER, NUMERIC |

ПРИКЛАД 6.15.

Вибрати коди, серії й номера паспортів, у номерах яких зустрічається число 84 (табл. 6.13).

**SELECT** Code, Series, Number
**FROM** PasportData
**WHERE CAST(**Number **AS** CHAR(5)) **LIKE** ‘%84%’;

Таблиця 6.13.

**Набір даних з таблиці PasportData**

| Code | Series | Number |
|---|---|---|
| 9 | АС | 90843 |
| 11 | ІК | 10842 |

### 6.11. Перевірка входження значення до множини

Така перевірка здійснюється за допомогою оператору *IN*:

<значення > **[NOT] IN** (<значення > [, <значення > ...])

Множина значень може бути сформована за допомогою оператора *SELECT*. Цей випадок буде розглянутий нижче при вивченні вкладених підзапитів, що повертають множину значень.

ПРИКЛАД 6.16.

Використовуючи таблицю PasportData, виведіть код, серію й номер паспорта для осіб, що проживають у Донецьку або Києві (табл. 6.14).

**SELECT** Code, Series, Number, IssuePlace
**FROM** PasportData
**WHERE** IssuePlace **IN** (‘Донецьк’, ‘Київ’);

Як і у випадку оператора *BETWEEN*, використання ключового слова *IN* являє собою більш ефективний спосіб запису умов пошуку, особливо якщо список припустимих значень досить великий. Той же самий запит може бути написаний так:

**SELECT** Code, Series, Number, IssuePlace
**FROM** PasportData
**WHERE** IssuePlace = ‘Донецьк’ **OR** IssuePlace = ‘Київ’;

### 6.12. Перевірка наявності визначника NULL

Присутність визначника *NULL* у рядках таблиці перевіряється за допомогою наступної конструкції:

<значення > **IS [NOT] NULL**

ПРИКЛАД 6.17.

Вибрати коди й назви книг, у яких зазначені УДК (табл. 6.15).

**SELECT** Code, Name
**FROM** Books
**WHERE** UDK **IS NOT NULL**;

Таблиця 6.14

**Набір даних з таблиці PasportData**

| Code | Series | Number | IssuePlace |
|---|---|---|---|
| 4 | АЄ | 12300 | Донецьк |
| 6 | АЖ | 01568 | Київ |
| 15 | АК | 89125 | Київ |
| 16 | АК | 55706 | Донецьк |
| 20 | АК | 12578 | Київ |

Таблиця 6.15

**Вібірка кодів й назв книг, у яких зазначені УДК**

| Code | Name |
|---|---|
| 1 | Автоматизація виробничих процесів на збагачувальній фабриці |
| 2 | Розв'язок завдань по автоматизації процесів збагачення й металургії |
| 3 | Асимптотичні методи оптимального керування |
| 4 | Синтез оптимальних автоматичних систем |
| 5 | Методи оптимізації стохастичних систем |
| 6 | Автоматизовані системи керування технологічним процесом збагачення руди |
| 7 | C/C++ Програмування мовою високого рівня |
| 8 | Комп'ютерні мережі. Принципи, технології, протоколи |
| 9 | Довідник з диференційних рівнянь з частками похідними першого порядку |
| 10 | Теорія ймовірностей й математична статистика |
| 11 | C#. Програмування мовою високого рівня |
| 12 | Теорія ймовірностей й математична статистика |
| 13 | Теорія ймовірностей й математична статистика |
| 14 | Дискретно - групові методи інтегрування звичайних диференційних рівнянь |

## 6.13. Агрегатні функції

**6.13.1. Розрахунки підсумкових значень в операторі SELECT** здійснюється за допомогою агрегатних або статистичних функцій: *COUNT()*, *SUM()*, *AVG()*, *MAX()*, *MIN()*.

Функція *COUNT*(<*вираз* >) підраховує число входжень значення виразу в усі записи результуючого набору даних. Вона працює з виразами будь-якого типу: числові, символьні або типу дата/час.

ПРИКЛАД 6.18.

Визначте, вартість скількох книг перевищує 25 гривень (табл. 6.16).

```
SELECT COUNT (*) AS cCount
    FROM BookInventoryNumbers WHERE Cost > 25;
```

Якщо із групи однакових записів при підрахунку необхідно враховувати тільки одну, перед виразом у дужках включають пропозицію *DISTINCT.* Найчастіше в якості виразу виступають імена стовпців. Вираз може обчислюватися й за значеннями декількох таблиць:

```
COUNT (DISTINCT <вираз >)
```

ПРИКЛАД 6.19.

Визначте, скільки різних бібліотекарів видавали книги (табл. 6.17).

```
SELECT COUNT (DISTINCT OutLibrarianCode) AS cCount
    FROM BookGiveOutRecord;
```

Загальна кількість записів, що задовольняє зазначеній умові, може бути визначена за допомогою функції *COUNT* без фрази *DISTINCT*. Однак, той самий бібліотекар може видати кілька книг різним читачам. Пропозиція *DISTINCT* дозволяє виключити з розрахунку значення, що дублюються.

Функція *SUM*(<вираз >) обчислює суму всіх значень стовпця або виразу. При цьому стовпець повинен мати числовий тип даних (містити цілі

числа, числа із плаваючою комою або грошові величини). Результат, що повертається цією функцією, має той же тип даних, що й стовпець або вираз, однак точність результату може бути вище. Наприклад, якщо застосувати функцію *SUM*() до стовпця, що містить 16-розрядні цілі числа, вона може повернути в якості результату 32-розрядне ціле число.

ПРИКЛАД 6.20.

Визначте загальну вартість книг бібліотечного фонду (табл. 6.18.)

**SELECT SUM**(Cost) **AS** cSum
**FROM** BookInventoryNumbers;

Таблиця 6.16

| cCount |
|---|
| 10 |

Таблиця 6.17

| cCount |
|---|
| 6 |

Таблиця 6.18

| cSum |
|---|
| 574 |

Функції *MIN*(<вираз >), *MAX*(<вираз >) і *AVG*(<вираз >) дозволяють знайти відповідно найменше, найбільше й середнє значення в стовпці або виразі. *MIN*(), *MAX*() працюють із числовими, строковими й даними типу дата/час. *AVG*() працює тільки з числовими типами даних. Результат, що вертається цими функціями, має точно такий же тип даних, як і вираз.

ПРИКЛАД 6.21.

Визначте мінімальну, максимальну й середню вартість книг (табл. 6.19).

**SELECT MIN(**Cost**) AS** Mini**, MAX(**Cost**) AS** Maxi**, AVG** (Cost) **AS** Avgi
**FROM** BookInventoryNumbers;

**6.13.2. Угруповання** записів використовується, якщо необхідно видати агреговані значення не по всім відношенням, а по кожній із вхідних у нього груп кортежів, які характеризуються загальною ознакою. Найчастіше це однакові значення в якому-небудь атрибуті або групі атрибутів. У цьому випадку в операторові *SELECT* використовують пропозицію

**GROUP BY** стовпець [,стовпець ...]

При цьому необхідно, щоб один зі стовпців результуючого набору даних був представлений агрегатною функцією.

ПРИКЛАД 6.22.

Визначте кількість книг, що зберігаються в різних бібліотечних фондах, а також їх сумарну вартість (табл. 6.20).

**SELECT** FundCode**, COUNT (**BookCode**) AS** cCount**, SUM(**Cost**) AS** cSum
**FROM** BookInventoryNumbers
**GROUP BY** FundCode
**ORDER BY** FundCode

Немає необхідності включати імена стовпців BookCode і Cost у список

фрази *GROUP BY*, оскільки у списку пропозиції *SELECT* вони використовуються тільки в узагальнюючих функціях. У той же час, стовпець FundCode у списку пропозиції *SELECT* не зв'язаний з жодною узагальнюючою функцією й із цієї причини обов'язково повинен бути зазначений у фразі *GROUP BY.*

Таблиця 6.19

| Mini | Maxi | Avgi |
|---|---|---|
| 10.10 | 74.2 | 38.27 |

Таблиця 6.20

| FundCode | cCount | cSum |
|---|---|---|
| 1 | 9 | 307.95 |
| 2 | 6 | 266.05 |

При обробці цього запиту виконується ряд логічних дій:

| FundCode | BookCode | Cost |
|---|---|---|
| 1 | 1 | 15.56 |
| 1 | 2 | 22.33 |
| 1 | 3 | 34.01 |
| 1 | 4 | 12.99 |
| 1 | 9 | 36.05 |
| 1 | 10 | 74.2 |
| 1 | 12 | 36.69 |
| 1 | 13 | 48.13 |
| 1 | 14 | 27.99 |
| 2 | 5 | 56.78 |
| 2 | 6 | 10.10 |
| 2 | 7 | 73.50 |
| 2 | 7 | 45.10 |
| 2 | 8 | 59.25 |
| 2 | 11 | 21.32 |

| COUNT(BookCode) | SUM(Cost) |
|---|---|
| 9 | 307.95 |
| 6 | 266.05 |

Рядки таблиці BookInventoryNumbers розподіляються в групи у відповідності зі значеннями у стовпці номера коду фонду (FundCode). У межах кожної із груп виявляються дані про коди книг одного з фондів. У нашому прикладі буде створено дві групи. Для кожної із груп обчислюється загальна кількість рядків, яка дорівнює кількості кодів книг, а також сума значень у стовпці Cost, яка і є сумою вартості книг кожного фонду, що нас цікавить. Потім генерується єдиний зведений рядок для всієї групи вихідних рядків. Нарешті, отримані рядки результуючої таблиці перегруповуються в порядку зростання номера коду фонду, який зазначений в стовпці FundCode.

### 6.13.3. Умова відбору згрупованих записів вказується в пропозиції

**HAVING** < умови пошуку >

Тут умови пошуку вказуються по тим же правилам, що й у *WHERE*. Однак *HAVING* перевіряє відповідність групи з результатами агрегування значень вказаним умовам пошуку, а *WHERE* виключає з розрахунку агрегатних значень записи, що не задовольняють відповідним умовам. В умовах пошуку *WHERE* вживати агрегатні функції не можна.

Стандарт *ISO* вимагає, щоб імена стовпців, які використовуються у пропозиції *HAVING*, обов'язково були присутні в списку *GROUP BY* або застосовувалися в агрегатних функціях. На практиці умови пошуку у фразі *HAVING* завжди включають, щонайменше, одну агрегатну функцію.

Умови пошуку, що не містять агрегатних функцій, повинні бути поміщені в *WHERE* і застосовуватися для відбору окремих рядків.

ПРИКЛАД 6.23.

Для кожного бібліотечного фонду, де зберігається більше шести книг, визначте кількість книг, а також їх сумарну вартість (табл. 6.21).

```
SELECT FundCode, COUNT (BookCode) AS cCount, SUM(Cost) AS cSum
        FROM BookInventoryNumbers
        GROUP BY FundCode
        HAVING COUNT (BookCode) > 6
        ORDER BY FundCode;
```

Цей приклад аналогічний попередньому, але тут використовуються додаткові обмеження, що вказують на те, що нас цікавлять відомості тільки про ті фонди, у яких зберігається більше шести книг. Подібна вимога накладається на групи, тому в запиті слід використовувати фразу *HAVING*.

Таблиця 6.21

**Набір даних з таблиці Books**

| FundCode | cCount | cSum |
|---|---|---|
| 1 | 9 | 307.95 |

## 6.14. Використання підзапитів в умовах пошуку

**6.14.1. Підзапит –** це інструмент створення тимчасової таблиці, вміст якої обробляється зовнішнім оператором. Внутрішні запити можуть бути використані для обчислення значень відразу після пропозиції *SELECT*, поміщені в *WHERE* і *HAVING* зовнішнього оператора *SELECT* безпосередньо після операторів порівняння (тобто операторів =, <, >, >=, <=, <>) або оператора *IN*. Крім того, підзапити можуть застосовуватися в операторах *INSERT*, *UPDATE* і *DELETE*. Текст підзапита повинен бути взятий у дужки.

Існує три типи підзапитів:

Скалярний підзапит повертає таблицю, що складається з одного стовпця й одного рядка, тобто одного значення. Він може використовуватися скрізь, де потрібно вказати одне значення.

Строковий підзапит повертає значення декількох стовпців таблиці, але у вигляді єдиного рядка. Він може використовуватися скрізь, де застосовується конструктор строкових значень.

Табличний підзапит повертає значення з одного або більше стовпців таблиці, розміщених в декількох рядках. Він може використовуватися скрізь, де допускається вказувати множини значень. Наприклад, як операнд предикату *IN*.

До підзапитів застосовуються наступні правила й обмеження:

1. У підзапитах не повинна використовуватися фраза *ORDER BY*, хоча вона може бути присутня у зовнішньому запиті.

2. Список у пропозиції *SELECT* повинен складатися з імен окремих стовпців або складених з них виразів. За винятком випадку, коли підзапит є операндом оператору *EXISTS*.

3. За замовчуванням імена стовпців у підзапиті відносяться до таблиці, ім'я якої зазначено в пропозиції *FROM*. Однак допускається посилатися й на стовпці таблиці, зазначеної у *FROM* зовнішнього запиту, для чого використовуються кваліфіковані імена стовпців. Такі підзапити називаються співвіднесеними.

4. Якщо підзапит є одним із двох операндів, що брав участь в операції порівняння, то він повинен вказуватися в правій частині цієї операції.

Коли неможливо обійтися одним підзапитом, тоді в ньому використовують вкладений підзапит й т.д.

**6.14.2. Скалярні підзапити** повертають таблицю, що складається з одного стовпця й одного рядка. Таку таблицю можна одержати, якщо після *SELECT* вказати один стовпець, а в *WHERE* умову пошуку організувати по потенційному ключу.

ПРИКЛАД 6.24.

Виведіть на екран усі дати повернення книг для читача з номером читацького квитка 28 (табл. 6.22).

```
SELECT ReturnDate, FactreturnDate
    FROM BookGiveOutRecord
    WHERE ReaderCode = (SELECT Code
                        FROM Readers
                        WHERE ReaderCardNumber = 28);
```

Внутрішній оператор *SELECT* (*SELECT* Code *FROM* Readers *WHERE* ReaderCardNumber = 28) призначений для визначення коду читача з номером квитка 28. Після цього працює зовнішній підзапит. Інакше кажучи, внутрішній оператор *SELECT* повертає таблицю, що має єдине значення – Code = 2. У результаті зовнішній оператор *SELECT* здобуває наступний вигляд:

```
SELECT ReturnDate, FactreturnDate
    FROM BookGiveOutRecord
    WHERE ReaderCode = 2;
```

Таблицю з одним стовпцем і одним рядком можна одержати, якщо після *SELECT* вказати одну агрегатну функцію без використання угруповання записів у пропозиції *GROUP BY*. У цьому випадку не мають значення умови відбору, які зазначені в *WHERE* і *HAVING*.

ПРИКЛАД 6.25.

Складіть список інвентарних номерів книг, ціна яких вище середньої. Вкажіть, наскільки їх ціна перевищує середню ціну книг по бібліотеці (табл. 6.23).

```
SELECT InventoryNumber,
        Cost – (SELECT AVG (Cost)
                FROM BookInventoryNumbers) AS Cost_diff
    FROM BookInventoryNumbers
    WHERE Cost > (SELECT AVG (Cost)
                  FROM BookInventoryNumbers);
```

Таблиця 6.22

| ReturnDate | FactreturnDate |
|---|---|
| 25-SEP-04 | 24-sep-04 |

Таблиця 6.23

| InventoryNumber | Cost_diff |
|---|---|
| 4678532 | 18.51 |
| 7569832 | 35.23 |
| 5478956 | 6.83 |
| 2145876 | 20.98 |
| 5268933 | 35.93 |
| 7812639 | 9.86 |

Необхідно відзначити, що не можна використовувати '*WHERE* Cost > *AVG* (Cost)', оскільки застосовувати агрегатні функції в *WHERE* заборонено. Для досягнення бажаного результату слід створити підзапит, що обчислює середнє значення ціни книг, а потім використовувати його в зовнішньому операторі *SELECT*, який призначений для вибірки відомостей про ті книги, ціна яких перевищує це середнє значення. Інакше кажучи, підзапит повертає значення середньої ціни книг по бібліотеці, яке дорівнює 38.27 гривень. Результат виконання цього скалярного підзапита використовується в зовнішньому операторові *SELECT* як для обчислення відхилення ціни від середнього рівня, так і для відбору відомостей про книги. Тому зовнішній оператор *SELECT* має наступний вид:

```
SELECT InventoryNumber, Cost – 38.27 AS Cost_diff
    FROM BookInventoryNumbers
    WHERE Cost > 38.27
```

**6.14.3. Підзапити, що повертають множину значень** використовуються як операнд предиката *IN.* Наприклад, табличний підзапит можна використовувати замість внутрішнього з'єднання таблиць. Для ілюстрації цього повернемося до ПРИКЛАДУ 6.7: «Створіть список усіх читачів, які коли небудь брали книги в бібліотеці». З використанням табличного підзапита й псевдонімів таблиць оператор *SELECT* буде мати такий вигляд:

```
SELECT ReaderCode, FamilyName, Name, B.Code, InventoryCode
    FROM  Readers R
    WHERE R.Code IN (SELECT  ReaderCode
                     FROM BookGiveOutRecord B);
```

Усі читачі, які колись брали книги в бібліотеці, зареєстровані в таблиці BookGiveOutRecord. Отже, нам необхідно вивести з таблиці Readers прізвища тільки тих читачів, коди яких містяться в множині значень, що вертаються підзапитом

```
SELECT ReaderCode
    FROM  BookGiveOutRecord B.
```

### 6.15. Оператори, що використовуються тільки з підзапитами

**6.15.1. Оператор EXISTS** застосовується, коли в умовах пошуку необхідно вказати, що з таблиці потрібно відібрати тільки ті записи, для яких підзапит повертає одне або більш значень:

**[NOT] EXISTS** (<підзапит >).

Отже, оператор *EXISTS* використовується тільки разом з підзапитами. Він повертає *TRUE*, якщо в результуючим наборі даних підзапита є одна або більше записів. В іншому випадку він повертає *FALSE*. Для *NOT EXISTS* правила зворотні.

ПРИКЛАД 6.26.

Вибрати коди й назви книги студентського фонду бібліотеки, які коли-небудь видавалися на руки читачам (табл. 6.24).

```
SELECT Code, Name
  FROM Books
  WHERE Code IN (SELECT BookCode
                    FROM BookInventoryNumbers
                    WHERE FundCode
                        = (SELECT Code FROM BookFunds
                          WHERE Name ='Студентський')
AND
                    EXISTS (SELECT Code
                        FROM BookGiveOutRecord
                        WHERE InventoryCode =
                            BookInventoryNumbers.Code));
```

**6.15.2. Оператор SINGULAR** застосовується, якщо в умовах пошуку необхідно вказати, що з таблиці потрібно вибрати лише ті записи, для яких підзапит повертає тільки одне значення:

[**NOT**] **SINGULAR** (<підзапит >)

Таблиця 6.24

**Коди й назви книг студентського фонду бібліотеки**

| Code | Name |
|---|---|
| 6 | Автоматизовані системи керування технологічним процесом збагачення руди |
| 7 | C/C++ Програмування мовою високого рівня |
| 11 | C#. Програмування мовою високого рівня |

ПРИКЛАД 6.27.

Вибрати коди й назви книг, які видавалися на руки читачам тільки один раз (табл. 6.25).

```
SELECT B.Code, B.Name
  FROM Books B
```

```
WHERE B.Code IN (SELECT BIN.BookCode
                 FROM BookInventoryNumbers BIN
                 WHERE SINGULAR
                       (SELECT BG.InventoryCode
                        FROM BookGiveOutRecord BG
                        WHERE BIN.Code =
BG.InventoryCode));
```

**6.15.3. Оператори ALL, SOME, ANY** використовуються, якщо в умовах пошуку необхідно вказати, що значення (значення стовпця, результат обчислення виразу), яке порівнюється, повинне перебувати в певних відношеннях з усіма значеннями з множини, що повертається підзапитом:

<знач.> [**NOT**] <оператор порівн.> {**ALL | SOME | ANY**} (<підзапит>).

Таблиця 6.25

**Коди й назви книг, які видавалися на руки читачам тільки один раз**

| Code | Name |
|---|---|
| 3 | Асимптотичні методи оптимального керування |
| 4 | Синтез оптимальних автоматичних систем |
| 7 | C/C++ Програмування мовою високого рівня |
| 9 | Довідник з диференційних рівнянь з частками похідними першого порядку |
| 14 | Дискретно-групові методи інтегрування звичайних диференційних рівнянь |

Тут підзапит може повертати більше одного значення. Оператор визначає операцію порівняння (>, >=, < і т.д.). Відношення між значенням, що порівнюється та значеннями, що вертаються підзапитом, установлюється словами *ALL* та *SOME* (*ANY*):

**WHERE** STOLBEZ > **ALL** (**SELECT** POLE **FROM** TABLIZA)

поверне *TRUE*, якщо поточне значення атрибута «STOLBEZ» буде більше всіх значень в атрибуті «POLE» відношення «TABLIZA»;

**WHERE** STOLBEZ > **SOME** (**SELECT** POLE **FROM** TABLIZA)

поверне *TRUE*, якщо поточне значення атрибута «STOLBEZ» буде більше хоча б одного значення в атрибуті «POLE» відношення «TABLIZA».

ПРИКЛАД 6.28.

Знайдіть інвентарні номери всіх книг, чия вартість перевищує вартість хоча б однієї книги книжкового фонду під номером «1» (табл. 6.26).

```
SELECT BookCode, InventoryNumber, Cost
    FROM BookInventoryNumbers
    WHERE Cost > SOME (SELECT Cost
                       FROM BookInventoryNumbers
                       WHERE FundCode = '1');
```

У цьому випадку внутрішній підзапит поверне наступні значення {15.56, 22.33, 34.01, 12.99, 36.05, 74.20, 36.69, 48.13, 27.99}, а зовнішній запит вибирає відомості про ті книги, ціна яких більше хоча б одного зі значень у цьому наборі (фактично, більше мінімального значення – 12.99).

ПРИКЛАД 6.29.

Знайдіть інвентарні номери усіх книг, у яких вартість перевищує вартість будь-якої книги книжкового фонду під номером «1» (табл. 6.27).

```
SELECT BookCode, InventoryNumber, Cost
    FROM BookInventoryNumbers
    WHERE Cost >= ALL (SELECT Cost
                       FROM BookInventoryNumbers
```

Тут зовнішній запит вибирає відомості про книги, ціна яких більше максимального значення множини {15.56, 22.33, 34.01, 12.99, 36.05, 74.20, 36.69, 48.13, 27.99}.

Таблиця 6.26

**Інвентарні номери книг**

| BookCode | InventoryNumber | Cost |
|---|---|---|
| 1 | 4567890 | 15.56 |
| 2 | 4510000 | 22.33 |
| 3 | 4532477 | 34.01 |
| 5 | 4678532 | 56.78 |
| 7 | 7569832 | 73,50 |
| 7 | 5478956 | 45,10 |
| 8 | 2145876 | 59,25 |
| 9 | 5214786 | 36,05 |
| 10 | 5268933 | 74,20 |
| 11 | 7865890 | 21,32 |
| 12 | 6589321 | 36,69 |
| 13 | 7812639 | 48,13 |
| 14 | 7523690 | 27,99 |

Пропозиція **HAVING** використовується, якщо в умовах пошуку для вкладеного підзапиту необхідно вказати агрегатну функцію.

ПРИКЛАД 6.30.

У бібліотеку екземпляри однієї й тієї ж книги можуть надходити у різний час і мати різну вартість. Необхідно визначити код та середню вартість екземплярів книг, у яких середня вартість екземпляра книги більше середньої вартості екземплярів інших книг (табл. 6.28).

```
SELECT BIN1.Bookcode, InventoryNumber, AVG(BIN1.Cost) AS AvgCost
    FROM BookInventoryNumbers BIN1
    GROUP BY BIN1.Bookcode
    HAVING AVG(Cost) >= ALL (SELECT AVG(BIN2.Cost)
                             FROM BookInventoryNumbers BIN2
                             GROUP BY BIN2.Bookcode);
```

Спочатку з таблиці BookInventoryNumbers вибирається значення середньої вартості екземплярів усіх книг (вкладений підзапит). Потім із цієї ж таблиці вибирається код і середня вартість екземпляра тієї книги, яка є найбільшою.

Таблиця 6.27

**Результат запиту (приклад 6.29)**

| BookCode | InventoryNumber | Cost |
|---|---|---|
| 10 | 5268933 | 74,20 |

Таблиця 6.28

**Результат запиту (приклад 6.30)**

| BookCode | InventoryNumber | Cost |
|---|---|---|
| 10 | 5268933 | 74,20 |

Зверніть увагу на те, що в цьому випадку псевдоніми таблиць використовуються не стільки для скорочення записів, скільки для визначення які атрибути брати для роботи зовнішнього й внутрішнього операторів *SELECT*.

### 6.16. Зовнішні з'єднання

Зовнішнє з'єднання визначається в пропозиції *FROM* оператора *SELECT*. Воно записується в наступному форматі:

**SELECT {* | <**значення **> [, <**значення **> ...]}**

**FROM <**табл. **> <**вид з'єднання **> JOIN <**табл**. > ON <**умова пошуку **>**

У результуючий набір даних включаються усі записи провідної таблиці. Яка з таблиць буде провідної, визначає вид з'єднання:

*LEFT* – (ліве зовнішнє з'єднання), коли провідною є таблиця, яка розташована ліворуч від виду з'єднання;

*RIGHT* – (праве зовнішнє з'єднання), коли провідною є таблиця, яка розташована праворуч від виду з'єднання;

*FULL* – у результуючий набір даних входять усі рядки таблиці, які розташованої ліворуч від типу з'єднання та таблиці, що розташована праворуч від типу з'єднання, навіть якщо умови пошуку для них не виконуються.

ПРИКЛАД 6.31.

Виконати зовнішнє ліве й праве з'єднання табл. 6.28. і табл. 6.29. по атрибутах P2 і P1 відповідно.

Таблиця 6.28

| Стовпець P1 | Стовпець P2 | Стовпець P3 |
|---|---|---|
| A | x | 400 |
| B | x | 200 |
| C | y | 500 |
| D | NULL | NULL |

Таблиця 6.29

| Стовпець P1 | Стовпець P2 |
|---|---|
| X | 1 |
| Y | 2 |
| Z | 2 |

Запишемо оператор *SELECT* для лівого з'єднання, позначивши табл. 6.28 і 6.29 через A и B відповідно:

**SELECT** A.P1, A.P2, B.P2

**FROM** A **LEFT JOIN** B **ON** A.P2 = B.P1

Сірим кольором показані стовпці провідної таблиці A. Як видно, для запису таблиці A, де стовпець A.P1 має значення '*d*', немає парних записів у таблиці B, для яких задовольнялося б умова пошуку A.P2 =

B.P1. Тому даний запис таблиці A показаний в з'єднанні з порожнім записом (табл. 6.30).

Запишемо оператор *SELECT* для правого з'єднання:

```
SELECT A.PI, A.P2, B.P2
    FROM A RIGHT JOIN B ON A.P2 = B.P1
```

Сірим кольором показані стовпці провідної таблиці B. Як видно, для запису таблиці B, де стовпець B.P1 має значення '*z*' і стовпець B.P2 має значення '2', немає парних записів у таблиці A, для яких задовольнялася б умова пошуку A.P2 = B.P1. Тому даний запис таблиці B показаний в з'єднанні з порожнім записом (табл. 6.31).

Таблиця 6.30

| Стовпець A.P1 | Стовпець A.P2 | Стовпець B.P2 |
|---|---|---|
| a | x | 1 |
| b | x | 1 |
| c | y | 2 |
| d | NULL | NULL |

Таблиця 6.31

| Стовпець A.P1 | Стовпець A.P2 | Стовпець B.P2 |
|---|---|---|
| A | x | 1 |
| B | x | 1 |
| C | y | 2 |
| NULL | NULL | 2 |

## 6.17. UNION - об'єднання результатів виконання декількох операторів SELECT

Іноді є необхідність об'єднання двох або більше результуючих наборів даних, які повертають окремі оператори *SELECT*. Для цього результуючі набори даних повинні мати однакову структуру, тобто однакову кількість і тип стовпців, що вертаються. *Такі набори даних називають сумісними по об'єднанню.* Записи, що дублюються, з об'єднаного набору даних виключаються.

ПРИКЛАД 6.23.

З'єднати результати виконання трьох запитів:

1) Паспорта із серією AA (табл. 6.32);

```
SELECT *
FROM PasportData
        WHERE Series = 'AA';
```

Таблиця 6.32

**Дані для паспортів із серією 'AA'**

| Code | Series | Number | Birthday | Birthplace | Sex | IssuePlace | IssueDate | Note |
|---|---|---|---|---|---|---|---|---|
| 1 | AA | 45003 | 30.05.1930 | Росія, м. Опочка | Жін. | Дніпропетровськ | 12.01.1995 | null |
| 2 | AA | 15700 | 23.02.1930 | Росія. м. Володимир | Жін. | м. Житомир | 16.03.2000 | null |

2) вибрати дані для паспортів, виданих в 2000 році (табл. 6.33);

```
SELECT *
    FROM PasportData
    WHERE IssueDate BETWEEN '01/01/2000' AND '12/31/2000'
```

3) Паспорта, номера яких починаються на 4 (табл. 6.34);

```
SELECT *
    FROM PasportData
    WHERE Number LIKE '4%'
```

Об'єднаємо три отримані набори даних: (табл. 6.35).

```
SELECT *
     FROM PasportData  WHERE Series = 'AA'
UNION
SELECT *
     FROM PasportData   WHERE Number LIKE '4%'
UNION
SELECT *
     FROM PasportData
          WHERE IssueDate BETWEEN '01/01/2000' AND '12/31/2000'
```

### 6.18. Операція зчеплення рядків

Операція || з'єднує два строкові значення, які можуть бути представлені виразами: <строковий вираз1> || <строковий вираз2>

Операцію || можна використовувати як після пропозиції *SELECT* для вказівки значень, що вертаються, так і в *WHERE*.

Таблиця 6.33

**Дані для паспортів, виданих в 2000 році**

| Code | Series | Number | Birthday | Birthplace | Sex | IssuePlace | IssueDate | Note |
|---|---|---|---|---|---|---|---|---|
| 2 | АА | 15700 | 23.02.1930 | Росія. м. Володимир | Жін. | М. Житомир | 16.03.2000 | null |
| 8 | АК | 23490 | 05.01.1961 | Росія, місто Самара | Жін. | Дніпропетровськ | 13.09.2000 | null |
| 20 | АК | 12578 | 11.11.1987 | Донецьк, Краматорськ | Жін. | Київ | 26.01.2000 | null |

Таблиця 6.34

**Дані для паспортів, номера яких починаються на 4**

| Code | Series | Number | Birthday | Birthplace | Sex | IssuePlace | IssueDate | Note |
|---|---|---|---|---|---|---|---|---|
| 1 | АА | 45003 | 30.05.1930 | Росія, м. Опочка | Жін. | Дніпропетровськ | 12.01.1995 | null |
| 7 | АЗ | 43188 | 13.11.1970 | Дніпропетровська область, м. Дніпродзержинськ | Жін. | Дніпропетровська область, м. Дніпродзержинськ | 15.05.1998 | null |
| 12 | ІК | 45190 | 18.07.1983 | Дніпропетровська область, село Петропавліка | Жін. | Дніпропетровська область, село Петропавліка | 20.09.1999 | null |
| 18 | АЖ | 45879 | 4.02.1961 | Дніпропетровськ | Жін. | Дніпродзержинськ | 14.03.1980 | null |
| 22 | АЯ | 45789 | 7.8.1972 | Угорщина | Жін. | Івано-Франківськ | 3.10.1988 | null |

Таблиця 6.35

**Набір об'єднаних даних**

| Code | Series | Number | Birthday | Birthplace | Sex | IssuePlace | IssueDate | Note |
|---|---|---|---|---|---|---|---|---|
| 1 | АА | 45003 | 30.05.1930 | Росія, м. Опочка | Жін. | Дніпропетровськ | 12.01.1995 | null |
| 2 | АА | 15700 | 23.02.1930 | Росія. м. Володимир | Жін. | М. Житомир | 16.03.2000 | null |
| 7 | АЗ | 43188 | 13.11.1970 | Дніпропетровська область, м. Дніпродзержинськ | Жін. | Дніпропетровська область, м. Дніпродзержинськ | 15.05.1998 | null |
| 8 | АК | 23490 | 05.01.1961 | Росія, місто Самара | Жін. | Дніпропетровськ | 13.09.2000 | null |
| 12 | ІК | 45190 | 18.07.1983 | Дніпропетровська область, село Петропавлівка | Жін. | Дніпропетровська область, село Петропавліка | 20.09.1999 | null |
| 18 | АЖ | 45879 | 4.02.1961 | Дніпропетровськ | Жін. | Дніпродзержинськ | 14.03.1980 | null |
| 20 | АК | 12578 | 11.11.1987 | Донецьк, Краматорськ | Жін. | Київ | 26.01.2000 | null |
| 22 | АЯ | 45789 | 7.8.1972 | Угорщина | Жін. | Івано-Франківськ | 3.10.1988 | null |

ПРИКЛАД 6.33.

Помістити прізвище, ім'я, по батькові та місце роботи читачів в одну колонку, причому місце роботи узяти в круглі дужки (табл. 6.36).

```
SELECT FamilyName ||' ' Name || Patronymic || '(' || Job || ')'
        FROM Readers;
```

### 6.19. Робота з різними базами даних в одному запиті

В одному запиті можна використовувати таблиці з різних БД. У цьому випадку в СУБД Interbase ім'я таблиці вказується у форматі

:Псевдонім БД :Ім'я таблиці

Під псевдонімом БД розуміється псевдонім, визначений в утиліті BDE Administrator. Нижче наведений приклад звернення в одному запиті до таблиць БД InterBase (псевдонім ' MONITOR') і Oracle (псевдонім 'DWH'):

```
SELECT U.*
    FROM ":MONITOR:NLS" N, ":DWH:OLAP_UPE" U
    WHERE U.SC_CODE = N.COD_SCENARIO
    ORDER BY U.SC_CODE;
```

Таблиця 6.36

**Дані читачів виведені в одну колонку**

| COLUMN1 |
|---|
| Іванов Петро Іванович (ДГУ, каф. ЕВТ) |
| Федорец Ірина Олегівна (ДГУ, АХЧ) |
| Ільїн Іван Петрович (ДГУ, каф. фізики) |
| Суренко Дмитро Павлович (ДГУ, каф. геофізики) |
| Коршунова Наталя Юріївна (ДГУ, каф. геоінформатики) |
| Носенко Олег Володимирович (ДГУ, ИКК) |
| Брусов Володимир Михайлович (НГУ, каф. геодезії) |
| Козирєв Олексій Сергійович (НГУ, каф. кримінології) |

| |
|---|
| Левченко Юлія Павлівна (НГУ, каф. політичної теорії) |
| Світла Тетяна Іванівна (НГУ, каф. перекладу) |
| Щиглів Петро Євгенович (НГУ, каф. електропостачання) |
| Кириленко Віктор Олександрович (НГУ, каф. електропривода) |

Формат оператора *SELECT* необхідно уточнювати в відповідній документації, яка надається до обраної Вами СУБД.

## 6.20. КОНТРОЛЬНІ ПИТАННЯ

1. Які службові слова (пропозиції) Вам відомі, що входять до складу оператору *SELECT*?
2. Яку операцію реляційної алгебри виконує оператор *SELECT*, що має найбільш простий вигляд?
3. Як виконати перейменування стовпців результуючого набору даних оператору *SELECT*?
4. Як усунути вміст в результуючому наборі даних оператору *SELECT* записів, що містять ідентичні записи у всіх стовпцях?
5. Як працює механізм сортування результату роботи оператору *SELECT*?
6. Які предикати пошуку Вам відомі?
7. Що спільного та яка різниця між порівнянням константи та результатом обчислення виразу зі значенням в атрибуті відношення?
8. Які операції реляційної алгебри реалізує використання предикату порівняння?
9. Яка різниця між логічним та фізичним рівнями формування результатів роботи оператору *SELECT*?
10. Як реалізувати складні умови пошуку в операторі *SELECT*?
11. Коли та що вказують перед ім'ям стовпця в операторі *SELECT*?
12. Як перевірити входження значення або виразу в заданий діапазон?
13. Який механізм перевірки відповідності значення виразу заданому шаблону?
14. Що спільного та відмінного між операторами, які визначають відповідність значення виразу заданому шаблону?
15. Де і для чого використовуються службові символи, що формують шаблон для визначення відповідності йому значень виразів?
16. Як досягти ігнорування регістру строкових літералів в умовах пошуку?
17. Як можливо трактувати значення одного типу даних як значення іншого типу?
18. Які особливості оператору, що відповідає за перевірку входження результату обчислення виразу у множену, Вам відомі?
19. Яка конструкція відповідає за перевірку наявності визначника *NULL*?
20. Що спільного та яка різниця між відомими Вам функціями, які розраховують підсумкові значення в операторі *SELECT*?
21. Для чого використовується пропозиція *GROUP BY* оператору *SELECT*?
22. Яка різниця між умовами пошуку, що містять у пропозиціях *HAVING* та *WHERE*?
23. Як і де можливо застосовувати підзапити?
24. Чим відрізняються результуючі набори даних, що повертаються підзапитами відомих Вам типів?
25. Які правила та обмеження, щодо формування підзапитів, Вам відомі?
26. Яким чином підзапит дає можливість використання агрегатних

функції в умовах пошуку пропозиції *WHERE*?

27. Які значення та при яких умовах повертають оператори *EXISTS* та *SUNGULAR*?

28. Яким чином в умовах пошуку можна визначити відношення скалярного значення з усіма значеннями множини, що повертається підзапитом?

29. Яка різниця між відомими Вам типами зовнішніх з'єднань відношень?

30. Яка різниця між зовнішніми та внутрішніми з'єднаннями відношень?

31. Яким умовам повинні відповідати результуючі набори даних операторів *SELECT*, що об’єднуються за допомогою оператору *UNION*?

32. Як в результуючому наборі даних оператору *SELECT* об'єднати в одному стовпці данні, що розташовані в різних стовпцях вихідних таблиць?

33. Чи є можливість в межах одного оператору *SELECT* працювати з таблицями, що розташовані у різних базах даних?

*Знайомство з цим матеріалом допоможе студентові самостійно працювати з оператором SELECT, розраховувати необхідні значення, використовуючи агрегатні функції і арифметичні вирази, складні умови пошуку пропозиції WHERE оператора SELECT, а також працювати з підзапитами, що повертають одиничні і множинні значення.*

# 7. Корпоративні обмеження цілісності

*Мета розділу – надати загальні відомості про корпоративні обмеження цілісності та основні «механізми» їх реалізації.*

## 7.1. Класифікація корпоративних обмежень цілісності

Забезпечення цілісності реляційних даних (п. 4.6.2) за рахунок використання первинних і зовнішніх ключів доповнюється корпоративними обмеженнями цілісності [2, 4, 6].

***Корпоративні обмеження цілісності*** *– це правила, які визначають, як клієнт повинен і може маніпулювати даними на сервері.*

Під правилами маніпулювання даними мається на увазі наступні дії, які обумовлені особливостями ведення обліку та бізнес-процесів усередині організації: додавання, вилучення рядків у таблицях, а також оновлення інформації в рядках, що існують. Вони відображають, по-перше, специфіку взаємодії організації зі своїми зовнішніми клієнтами та структурами, для яких вона виступає як клієнт, а по-друге, взаємодію суб'єктів в середині організації. У переважній більшості випадків зовнішні взаємодії регулюються законодавством різного рівня та договорами. Усі ці особливості ведення обліку та бізнесу неможливо втримати усередині рамок реляційної моделі.

Корпоративні обмеження цілісності визначаються на етапі логічного проектування, а реалізуються під час фізичного проектування БД. Їх можна умовно поділити на наступні групи:

1. Підтримка унікальності значень потенційних ключів, які не використовуються для зв'язків між відношеннями усередині БД.
2. Підтримка області припустимих значень в атрибутах відношень.
3. Підтримка обмежень кількості кортежів у відношеннях, викликане чинним законодавством і (або) особливостями ведення обліку усередині організації.
4. Підтримка каскадного оновлення й вилучення інформації у відношеннях БД.

Усі потенційні ключі визначаються на етапі нормалізації відношень. У БД «Бібліотека» необхідно підтримувати унікальність значень наступних потенційних ключів, які не використовуються для зв'язків між відношеннями:

*ReaderCardNumber* (№ квитка читача),
*PasportCode* (код паспорта) у відношенні *Readers* (читачі);
*ClockNumber* (табельний номер),
*PasportCode* (код паспорта) у *Librarians* (бібліотекарі);
*Name, IssueYear* (назва, рік видання) у *Books* (книги);
*Series, Number* (серія, номер) у *PasportData* (паспортні дані);
*ReaderCode*, *PhoneNumber* (код читача, телефонний номер) у *Phones* (телефони);

*InventoryNumber* (інвентарний номер) у *BookInventoryNumbers* (інвентарні номери книг);

*Name* (назва книжкового фонду) у *BookFunds* (книжкові фонди);

*Name* (назва типу телефону) у *PhoneTypes* (типи телефонів);

*Code* (номер операції) у *BookGiveOutRecord* (облік видачі книг).

Необхідно визначити область припустимих значень, насамперед, для атрибутів відношень, які входять до складу потенційних ключів, і не беруть участь у зв'язках між відношеннями в БД. Наприклад, аналіз перерахованих вище атрибутів, які входять до складу потенційних ключів, і не беруть участь у зв'язках між відношеннями показує, що в них не повинно бути визначника *NULL*. Є ряд очевидних додаткових обмежень. В атрибутах типу *INTEGER* – *ReaderCardNumber* (№ квитка читача), *ClockNumber* (табельний номер), *Number* (номер паспорта), InventoryNumber (інвентарний номер) – значення повинні бути > 0. В атрибутах типу *CHAR* – *Name* (назва книги), *Series* (серія паспорта), *PhoneNumber* (номер телефону читача), *Name* (назва книжкового фонду), *Name* (назва типу телефону) – не повинно бути порожніх рядків або рядків, що складаються з одних пробілів.

Для інших не ключових атрибутів відношень визначення області припустимих значень також може мати зміст. Ця область залежить від вимог користувачів, що обумовлені на етапі логічного проектування БД. Наприклад, якщо обов'язково необхідно вказати основне місце роботи читачів і їх посаду, то в атрибутах *Job* (місце основної роботи) і *Post* (посада) не повинно бути визначника *NULL*, порожніх рядків або рядків, що складаються з одних пробілів.

Прикладом підтримки обмеження кількості кортежів у відношенні *BookGiveOutRecord* (облік видачі книг) БД «Бібліотека» може служити забезпечення видачі на руки заданої кількості книг одному читачеві. Іншим очевидним обмеженням може бути заборона на видачу одночасно декількох екземплярів однієї й тієї ж книги. Наприклад, якщо є правило, що один читач може взяти не більше п'яти книг, тоді у відношенні *BookGiveOutRecord* може бути лише 5 кортежів з однаковими значеннями в атрибуті *ReaderCode* (код читача) у яких значення *FactReturnDate* (фактична дата повернення) містить визначник *NULL*. Додатково необхідно перевіряти у відношенні *BookGiveOutRecord* (облік видачі книг), щоб значення атрибутів *BookGiveOutRecord.InventoryCode* (інвентарний номер книги) для зазначених кортежів відповідали різним значенням *BookGiveOutRecord.BookCode* (код книги).

У СУБД архітектури клієнт-сервер механізми підтримки корпоративних обмежень цілісності можуть розміщатися як на стороні клієнта, так і на стороні сервера (див. п. 1.3.3). Ми розглянемо тільки механізми, що реалізовані з боку сервера: *SQL*-оператори, збережувана процедури та тригери. Для кожного конкретного випадку необхідно вибирати найпростіше рішення, що дозволяє виконати поставлене завдання по підтримці корпоративних правил обмеження цілісності.

### 7.2. Використання SQL операторів для підтримки корпоративних обмежень цілісності

Оператори *CREATE DOMAIN*, *CREATE INDEX* та *CREATE TABLE* можуть використовуватися для підтримки корпоративних обмежень цілісності. Вони дозволяють забезпечувати унікальність значень потенційних ключів, які не використовуються для зв'язків між відношеннями усередині БД та визначати область припустимих значень атрибутів відношень.

Формат оператора

**CREATE DOMAIN** Ім'я **[AS] <**Тип даних **>**
**[DEFAULT** {Константа **| NULL** | USER}]
**[NOT NULL] [CHECK** (<Область визначення >)]
**[COLLATE** Набір символів для сортування**]**

дозволяє в пропозиції *CHECK* вказати область визначення значень потенційного ключа та виключити з неї визначник *NULL,* вказавши *NOT NULL* при створенні домену. Для опису області визначення значень потенційного ключа використовуються такі самі предикати та правила побудови логічних виразів, що й для вибірки даних за умовою в пропозиції *WHERE* оператору *SELECT*:

<Область визначення > =
**VALUE** <оператор > <значення >
| **VALUE** [**NOT**] **BETWEEN** <значення > **AND** <значення >
**| VALUE** [**NOT**] **LIKE** <значення > [**ESCAPE** <значення >]
| **VALUE** [**NOT**] **IN** (<значення > [, <значення > ...])
| **VALUE IS** [**NOT**] **NULL**
| **VALUE** [**NOT**] **CONTAINING** <значення >
| **VALUE** [**NOT**] **STARTING** [**WITH**] <значення >
| (<Область визначення >)
| **NOT** <Область визначення >
| <Область визначення > **OR** <Область визначення >
| <Область визначення > **AND** <Область визначення >
<оператор > = {= | < | > | <= | >= | !< | !> | <> | !=}.

Формат оператору *CREATE INDEX* дозволяє виключити дублювання значень у ключових атрибутах, вказавши фразу *UNIQUE*:

**CREATE [UNIQUE] [ASC[ENDING] | DESC[ENDING]]**
**INDEX** Ім'я **ON** Ім'я таблиці (атрибут [, атрибут...]).

Оператор *CREATE TABLE* містить у собі всі можливості забезпечення підтримки корпоративних обмежень цілісності, закладені в операторах *CREATE DOMAIN* та *CREATE INDEX*:

**CREATE TABLE** Ім'я **[EXTERNAL [FILE]**'Шлях та ім'я файлу **']**
(<Опис. атр. > [,<Опис. атр. > | <Обмеження цілісності >…])

Опис атрибуту дозволяє визначити колонку на базі раніше

створеного домену, вказати область визначення значень потенційного ключа й виключити визначник *NULL* з неї, вказавши *NOT NULL*:

**<**Опис. атр. > Ім'я
{Тип даних | **COMPUTED [BY]** (<Вираз >)| Домен}
[**DEFAULT** { *Константа* | **NULL** | **USER**}]
[**NOT NULL**] [<Обмеження цілісності >]
**[COLLATE** Набір символів для сортування**].**

Пропозиція *COMPUTED [BY]* (<Вираз >) дозволяє обчислювати значення атрибуту. <Вираз > повинен повертати скалярне значення. Наприклад, оператор

**CREATE TABLE** SomeTable
(CostFloat NOT NULL,
Number INTEGER NOT NULL,
TotalCost COMPUTED BY (Cost * Number))

створює три стовпці: Cost (вартість), Number (кількість), TotalCost (загальна вартість). В останньому стовпці значення розраховується як добуток значень у двох попередніх.

На рівні опису атрибута обмеження цілісності мають наступний формат:

**<**Обмеження цілісності **> = [CONSTRAINT** Ім'я**]** <Опис. обмеж. ціл. >**,** де

<Опис обмеження цілісності > **= UNIQUE | PRIMERY KEY**
**| CHECK** (<Область визначення >)
**| REFERENCES** Ім'я ін. табл. [( Ім'я ін. атр. [, Ім'я ін. атр. …]]
**[ON DELETE {NO ACTION | CASCADE | SET DEFAULT | SET NULL}]**
**[ON UPDATE {NO ACTION | CASCADE | SET DEFAULT | SET NULL}]**

Ви можете в <*Описі обмеження цілісності*> вказати, що значення в атрибуті повинні бути унікальні (пропозиція *UNIQUE* або *PRIMERY KEY*); можете визначити область визначення значень атрибуту (пропозиція *CHECK*); маєте можливість організувати каскадне оновлення, вилучення значень та кортежів відношення, указавши зв'язок зі значеннями атрибутів іншої таблиці (пропозиція *REFERENCES*…).

Опис області визначення значень атрибуту відношення, відрізняється від області значень домену тим, що в ньому можна використовувати оператори *SELECT*:

< Область визначення > **=**
<знач. > <оператор > {<знач. > | (<**SELECT**: 1 знач. >) }
| <знач. **> [NOT] BETWEEN** <знач. >**AND** <знач. >
| <знач. **> [NOT] LIKE** <знач. > **[ESCAPE** <знач. >]
| <знач. > **[NOT] IN** (<знач. >[, <знач. >…] | (<**SELECT**: список >)
| <*знач.* **> IS [NOT] NULL**
| <*знач.* > **{[NOT] {=| < | > | >= | <=|}**

**{ALL | SOME | ANY } (<*SELECT:* список >)**
**| EXISTS (<SELECT >)**
**| SINGULAR (<SELECT >)**
| <знач. **> [NOT] CONTAINING** <знач. >
| <знач. > **[NOT] STARTING [WITH]** <знач. >
| (<Область визначення >)
**| NOT** < Область визначення >
| <Область визначення > **OR** <Область визначення >
**|** <Область визначення > **AND <Область визначення >**

### 7.3. Збережувана процедури

#### 7.3.1. Визначення й класифікація збережуваних процедур

***Збережувана процедура*** – це модуль, що написаний процедурною мовою СУБД та зберігається в БД як метадані ( тобто як дані про дані).

Збережувану процедуру можна викликати з прикладної програми клієнта. Існує два різновиди збережуваних процедур: процедури вибору та процедури дії [2, 4, 6].

***Процедури вибору*** можуть повертати більше одного значення. У прикладній програмі її ім'я може підставлятися в оператор *SELECT* замість імені таблиці.

***Процедури дії*** взагалі можуть не повертати даних і використовуються для реалізації будь-яких дій.

Збережуваним процедурам можна передавати параметри. Вони можуть повертати значення параметрів, змінені відповідно до закладених алгоритмів.

Переваги використання збережуваних процедур наступні:

1) здатність однієї процедури, що розташована на сервері, спільно використовуватись багатьма прикладними програмами клієнта;

2) спрощення прикладних програм клієнтів за рахунок переносу частини коду на сервер;

3) при зміні збережуваної процедури на сервері всі зміни негайно стають доступними для прикладних програм клієнтів;

4) зменшення мережного трафіку.

#### 7.3.2. Створення збереженої процедури виконується оператором

**CREATE PROCEDURE** Ім'я
[(<вхідні парам. > <тип даних > [, < вхідні парам. > <тип даних >... ])]
**[RETURNS**
(<вхідні парам. > <тип даних > [, < вхідні парам. > <тип даних >... ])]
**AS**
<Тіло процедури >

***Вхідні параметри*** служать для передачі в процедуру значень із прикладної програми клієнта [10, 14, 15]. Змінювати значення вхідних параметрів у тілі процедури безглуздо: ці зміни будуть забуті після закінчення роботи процедури. ***Вихідні параметри*** служать для повернення результуючих значень. Значення вихідних параметрів установлюються в тілі процедури, і після закінчення її роботи передаються в прикладну програму клієнта. І вхідні, і вихідні параметри можуть бути опущені, якщо в них немає необхідності.

Формат тіла процедури наступний:

[<оголошення локальних змінних процедури >]
**BEGIN**
<оператор >;
[<оператор >; ...]
**END**

### 7.3.3. Алгоритмічна мова для написання тіла тригерів і збережених процедур

**7.3.3.1. Локальні змінні,** якщо вони визначені в процедурі, мають строк життя від початку виконання процедури й до її закінчення. Поза процедурою вони невідомі й спроба звертання до них викличе помилку. Локальні змінні використовують для зберігання проміжних значень.

Оголошення локальних змінних має наступний формат:

**DECLARE VARIABLE** <ім'я > <тип >;

Тут використовуватися стандартні типи СУБД Interbase.

**7.3.3.2. Оператор присвоювання** служить для занесення значень у змінні. Його формат:

Ім'я змінної = вираз,

де в якості виразу можуть виступати константи, змінні, арифметичні й строкові вирази, у яких можна використовувати вбудовані функції, функції, визначені користувачем, а також генератори. Наприклад:

POK = "Покупець не визначений";

Тут змінній з ім’ям POK присвоюється строкова константа.

**7.3.3.3. Операторні дужки BEGIN ... END,** по-перше, обмежують тіло процедури, а по-друге, можуть використовуватися для вказівки меж складеного оператора. Під простим оператором розуміється одинична дозволена дія (див. приклад вище). Під складеним оператором розуміється група простих або складених операторів, що укладена в операторні дужки *BEGIN ... END*.

**7.3.3.4.Оператор IF ... THEN ... ELSE** у загальному виді записується так:

**IF** )(<умова > **THEN** <оператор >[**ELSE** <оператор >]

Якщо умова достеменна, то виконується оператор, після пропозиції *THEN*, якщо ні – оператор після *ELSE.*

**7.3.3.5. Оператор SELECT** використовується в збережуваній процедурі для видачі одиничної строки. У нього додана пропозиція:

**INTO** змінна [, змінна...]

Вона служить для вказівки змінних або вихідних параметрів, у які повинні бути записані значення, що повертаються оператором *SELECT* (ті результуючі значення, які перелічуються після пропозиції *SELECT*).

**7.3.3.6. Оператор SUSPEND** припиняє виконання збережуваної процедури, і в прикладну програму, з якої її викликали, передаються значення параметрів, що перераховані після пропозиції *RETURNS*. Наприклад, збережувана процедура FindAutorName повертає у вихідному параметрі InAutorName ім'я автора, що відповідає його коду – InAuthorCode:

```
CREATE PROCEDURE FindAuthorName
        (InAuthorCode INTEGER)
RETURNS (InAuthorName CHAR (30)) AS
BEGIN
        SELECT Name
                FROM BookAuthors
                WHERE Code = :InAuthorCode
                INTO :InAuthorName;
        SUSPEND;
END
```

**7.3.3.7. Оператор FOR SELECT ... DO** має наступний формат:

```
FOR
        <оператор SELECT>
DO
        < оператор >;
```

Після пропозиції *FOR* оператор *SELECT* повертає набір кортежів. Потім для кожного кортежу виконується оператор або послідовність операторів, яка є наступною за пропозицією *DO*. Іншими словами це цикл по рядках таблиці, яка сформовано оператором *SELECT.*

**7.3.3.8. Оператор WHILE ... DO** утворює конструкцію виду

```
WHILE (<умова >)
DO <оператор >
```

Він організує циклічне виконання оператора або послідовності операторів після пропозиції *DO*, до тих пір, поки умова після *WHILE* достеменна.

**7.3.3.9. Оператор EXIT** ініціює припинення виконання процедури й повертає управління прикладну програму, яка її викликала.

**7.3.3.10. Оператор EXECUTE PROCEDURE** має такий вигляд:

**EXECUTE PROCEDURE** Ім'я [параметр [, параметр ...]];
**[RETURNING_VALUES** параметр [, параметр ... ]];

З його допомогою в будь-якім місці тіла процедури можна виконати іншу збережувану процедуру, яка є у БД. Після *EXECUTE PROCEDURE* обов'язково вказується її ім'я. Далі, якщо це необхідно, вказуються параметри, які передаються процедурі. Пропозиція *RETURNING_VALUES* дозволяє одержати результати виконання процедури, що викликається, й записати їх у змінні.

**7.3.3.11. Оператор POST_EVENT** застосовується для посилки зі збережуваної процедури у прикладні програми-клієнти повідомлення про виникнення якої-небудь ситуації, пов'язаної з іменем події. Він має формат:

**POST_EVENT** "Ім'я події".

**7.3.3.12. Оператор CREATE EXCEPTION** визначає виняткову ситуацію із заданим іменем і повідомленням:

**CREATE EXCEPTION** Ім'я “ повідомлення ”.

Для генерації виняткової ситуації усередині збереженої процедури використовується наступний синтаксис:

**EXCEPTION** Ім'я**.**

При виникненні виняткової ситуації в прикладну програму, що викликається вертається повідомлення про помилку й виконання збереженої процедури завершується. Однак виняткову ситуацію можна обробити й усередині збереженої процедури, після чого продовжити виконання збереженої процедури.

**7.3.4. Підтримка корпоративних обмежень цілісності даних** за допомогою збережуваних процедур є найбільш потужним інструментом. Його слід застосовувати у тому випадку, коли за допомогою SQL операторів на етапі створення структури БД у вас немає можливості врахувати всі корпоративні й законодавчі правила, що впливають на особливості ведення обліку усередині організації. Збережувані процедури застосовують при необхідності генерації повідомлень для користувача, що описують виняткову ситуацію, яка виникає.

**CREATE EXCEPTION** AuthorNotFound "Автор не знайдений.";
**CREATE PROCEDURE** FindAuthor
(AuthorCode INTEGER, BookCode INTEGER)

```
AS
DECLARE VARIABLE FindAuthorCode INTEGER;
/* У змінну поміщаємо результат пошуку коду автора в таблиці
   BookAuthors./*
BEGIN
    SELECT ba.Code
        FROM BookAuthors ba
        WHERE ba.Code = AuthorCode
        INTO : FindAuthorCode;
/* У змінну поміщаємо результат пошуку коду автора в таблиці BookAuthors./*
    IF (FindAuthorCode IS NULL) THEN
/* У таблиці BookAuthors заданого коду автора книги не виявлено./*
        EXCEPTION AuthorNotFound;
    ELSE
/* У таблиці BookAuthors виявлений заданий код автора книги./*
        UPDATE Books
            SET AuthorCode = FindAuthorCode
            WHERE Code = :BookCode
END
```

Наприклад, вище показаний текст збережуваної процедури, що змінює код автора книги у відношенні Books. Процедура необхідна при помилковому вводі автора книги. Вона містить усі необхідні перевірки допустимості цієї операції для БД „БИБЛІОТЕКА”. Якщо операція неприпустима, після генерації відповідної виняткової ситуації виконання процедури припиняється. Якщо з'ясується, що задля збереження цілісності даних при зміні коду автора необхідно виконати ще якісь дії або перевірки, тоді всі вони повинні виконуватися усередині даної збережуваної процедури. У результаті, будь-яка, прикладна програма клієнта, що її використовує, буде дотримуватися однакових обмежень цілісності даних.

### 7.4. Тригери

***Тригер*** *– це процедура, яка автоматично викликається SQL-сервером при оновленні, вилученні або додаванні даних у відношення* [10, 14, 15]. Безпосередньо із програм до тригерів звернутися неможливо. Неможливо передавати їм вхідні параметри та одержувати від них значення вихідних параметрів. Тригери завжди реалізують дію. Вони обробляють наступні події: додавання нового запису, зміна значень атрибутів в існуючому записі або вилучення запису. Стосовно події тригери розрізняються на ті, що виконуються до події та після її. Оператором *CREATE TRIGGER* має наступний формат:

```
CREATE TRIGGER Ім'я
    FOR Ім'я таблиці [ACTIVE | INACTIVE]
```

**{BEFORE | AFTER} {DELETE | INSERT | UPDATE}**
[**POSITION** номер]
**AS**
<тіло тригеру >.

Після пропозиції *CREATE TRIGGER* обов'язково необхідно вказати унікальне ім'я тригера. Потім у *FOR* визначити ім'я таблиці, для якої створюється тригер. За замовчуванням тригер активний (*ACTIVE*), але якщо вказати *INACTIVE*, то тригер не буде обробляти подію. Пропозиція *BEFORE* і *AFTER* вказують, коли буде виконуватись тригер – до або після події вилучення (*DELETE*), додавання (*INSERT*) запису або зміни значень атрибутів у ній (*UPDATE*). Пропозиція *POSITION* дозволяє для кожної події призначити декілька тригерів. Їхній порядок виконання відповідає номеру.

Структура тіла тригера:

[<оголошення локальних змінних >]
**BEGIN**
<оператор >
**END**

Для визначення тіла тригера використовується процедурна мова, що розглянута у розділі, який присвячений збережуваним процедурам. До неї додається можливість доступу до старого та нового значень стовпців, які розташовані у рядку, що змінюється, *OLD* і *NEW* відповідно. Ця можливість, недоступна при визначенні тіла збережуваних процедур. Значення *OLD*.Ім'я_стовпця дозволяє звернутися до стану стовпця, що мав місце до внесення можливих змін, а значення *NEW*.Ім'я_стовпця – до стану стовпця після внесення можливих змін. У тому випадку, якщо значення в стовпці не змінилося, *OLD*.Ім'я_стовпця дорівнює *NEW*.Ім'я_стовпця.

Тригери активно використовуються для реалізації корпоративних обмежень цілісності. Зокрема, це може бути установка за допомогою генераторів унікальних значень індексних полів, накопичування статистики в інших таблицях та багато чого іншого. До складностей, що виникають при реалізації бізнес-правил за допомогою тригерів, слід віднести нерозвиненість засобів налагодження логіки коду, що становить тіло тригерів.

Тригер може забезпечити виконання каскадних впливів у дочірньому відношенні при зміні, вилученні запису в батьківському відношенні. Це доцільно, якщо такі впливи не можна забезпечити механізмом зовнішніх ключів. У цьому випадку розроблювачеві немає необхідності опікуватися про програмну реалізацію каскадних оновлень у прикладної програмі клієнта. Такий підхід знижує мережний трафік за рахунок відсутності необхідності пересилати додаткову інформацію прикладним програмам клієнтів для обробки подій, що пов'язані з маніпулюванням даними.

Не рекомендується одночасно забезпечувати виконання каскадних впливів за допомогою механізму зовнішніх ключів і тригерів. Для реалізації автоматичного виконання каскадних відновлень і змін

необхідно, по-перше, вилучити зовнішні ключі, що блокують такі зміни в БД і, по-друге, визначити самі тригери для відповідних таблиць.

Наприклад, для БД «Бібліотека» на етапі створення відношень був визначений зовнішній ключ, що не дозволяє вводити в атрибут Book.AuthorCode (код автора книги) таке значення, яке відсутнє в атрибуті BookAuthors.Code відношення, у якому організований довідник авторів книг:

```
CREATE TABLE BookAuthors
(Code INTEGER NOT NULL,
…
…
 PRIMARY KEY(Code));
CREATE TABLE Books
(Code INTEGER NOT NULL,
AuthorCode INTEGER NOT NULL,
…
…
PRIMARY KEY(Code),
FOREIGN KEY(AuthorCode) REFERENCES BookAuthors Code);
```

Для реалізації автоматичного виконання каскадного вилучення запису й зміни коду автора за допомогою тригерів необхідно, по-перше, вилучити визначення зовнішнього ключа з відношення BOOKS, а по-друге, визначити тригери, які будуть виконувати каскадну зміну інформації.

Тригер, що реалізує каскадне оновлення, може виглядати таким чином

```
CREATE TRIGER UpAuthorCodeBooks FOR BookAuthors
        ACTIVE
        BEFORE UPDATE AS
        BEGIN
                IF (OLD.Code <> NEW.Code) THEN
                        UPDATE Books
                                SET AuthorCode = NEW.Code
                                WHERE AuthorCode = OLD.Code
                END
```

Тригер, що реалізує каскадне вилучення, може виглядати так.

```
CREATE TRIGER DelAuthorCodeBooks FOR BookAuthors
        ACTIVE
        AFTER DELETE AS
        BEGIN
                DELETE FROM Books
                        WHERE AuthorCode = OLD.Code
        END
```

За допомогою тригерів можна вести журнал спостережень за зміною інформації в БД. Журнал змін являє собою таблицю, у якій фіксуються дії над усією БД або окремими її таблицями. У системах, де багато користувачів, ведення такого журналу дозволяє визначити джерело недостовірних або перекручених даних.

## 7.5. КОНТРОЛЬНІ ПИТАННЯ

1. Які складові забезпечення цілісності реляційних даних вам відомі?
2. Які дії регулюють корпоративні обмеження цілісності?
3. На які групи можливо поділити корпоративні обмеження цілісності?
4. Як обрати місце розташування корпоративних обмежень цілісності?
5. Які SQL-оператори реалізують корпоративні обмеження цілісності?
6. Які види корпоративних обмежень цілісності можливо реалізувати за допомогою SQL-операторів?
7. Що спільного та відмінного у відомих Вам різновидів збережуваних процедур?
8. Які переваги має використання збережуваних процедур для реалізації корпоративних обмежень цілісності?
9. Які можливості алгоритмічної мови збережуваних процедур Вам відомі?
10. Коли виправдане використання збережуваних процедур задля реалізації корпоративних обмежень цілісності?
11. Що спільного та відмінного між збережуваними процедурами та тригерами?
12. Які події можливо обробити за допомогою тригерів?
13. Які додаткові можливості має процедурна мова тригерів?
14. Які особливості використання тригерів для забезпечення корпоративних обмежень цілісності Вам відомі?

*Матеріал цього розділу дає поняття про основні складові забезпечення цілісності реляційних даних, про основні прийоми регулювання корпоративних обмежень цілістності даних. На прикладах показано, як забезпечити цілісність даних з використанням SQL-операторов, а також трігерів і збережуваних процедур.*

# 8. Представлення даних

*Цей розділ присвячен представленню даних, використання яких не лише спрощює структуру запитів до БД. У випадку, коли СУБД обслуговує запити багатьох користувачів у мережі, представлення відіграють ключову роль у визначенні структури БД та організації захисту інформації.*

## 8.1. Визначення, створення та класифікація представлень даних

***Представлення*** – це динамічний результат однієї або декількох реляційних операцій, виконаних над відношеннями БД з метою одержання нового відношення [4].

Представлення створюється динамічно [4, 6, 8, 12]. Воно є віртуальним відношенням, яке реально в БД не існує й створюється в результаті виконання запиту користувача. Зіштовхнувшись із посиланням на представлення, система управління знаходить відповідне визначення, і перетворює вихідний запит до представлення в еквівалентний запит до відношень, які використовуються у визначенні представлення, після чого модифікований запит виконується. Цей процес називається ***розв'язання представлення***.

З погляду користувача, представлення виглядає як таблиця з даними, що складається з стовпців та рядків. Воно дозволяє досягнути більш високої захищеності даних, а також надає проектувальникові засоби настроювання моделі користувача. Однією з найважливіших причин використання представлень є прагнення до спрощення багатотабличних запитів. Після визначення представлення із з'єднанням декількох таблиць можна буде використовувати до нього прості однотабличні запити, замість запитів з виконанням того ж самого багато табличного з'єднання.

Порядок формування записів у представленні даних визначається оператором *SELECT*. Для створення представлення застосовується конструкція:

**CREATE VIEW** Ім'я [(стовпець [,стовпець ...])
**AS <**оператор **SELECT> [WITH CHECK OPTION];**

Тут обов'язково повинно бути визначене *ім'я* представлення та оператор *SELECT*. Додатково є можливість вказати імена для кожного стовпця представлення – [(стовпець [,стовпець ...])], а для представлень, що оновлюються, заборонити введення рядків, що не задовольняють умові його формування в пропозиції *WHERE* оператора *SELECT*– [*WITH CHECK OPTION*].

Список імен стовпців повинен складатися з кількості елементів, яка дорівнює кількості стовпців у результуючому відношенні, що формує оператор *SELECT*. Якщо список імен стовпців опущений, кожний стовпець представлення буде мати ім'я відповідного стовпця результуючого відношення. Список імен стовпців повинен обов'язково задаватися, якщо в іменах стовпців результуючого відношення має місце неоднозначність. Така ситуація виникає, коли в операторі *SELECT* мають місце стовпці, що

обчислюються, або коли результуюче відношення створюється за допомогою операції з'єднання й включає атрибути з однаковими іменами.

Заданий оператором *SELECT* запит прийнято називати ***визначальним***. Залежно від його структури представлення може бути таким, що оновлюється чи ні. В представленнях, які оновлюються, можна додавати й вилучати рядки, змінювати значення. Усі ці маніпуляції з даними можуть бути збережені у відношеннях БД. Представлення, які не оновлюються, можуть тільки відображати інформацію. СУБД зберігає їх визначення в БД.

Для того щоб представлення було таким, що оновлюється, СУБД повинна мати можливість однозначно відобразити будь-який його рядок або стовпець на відповідний рядок або стовпець вихідного відношення. Усі оновлення, виконані у відношенні БД, негайно відображаються у всіх представленнях, що включають звернення до цього відношення. Якщо дані були змінені в представленні, що оновлюється, то ця зміна буде відображена й у тому відношенні, на базі якого воно побудоване.

Згідно зі стандартом *ISO* представлення може бути таким, що оновлюється, у тому й тільки в тому випадку, якщо:

По-перше, у його визначенні не використовується пропозиція *DISTINCT*, тобто із результатів визначального запиту не виключаються рядки, що дублюються;

По-друге, кожний елемент у списку пропозиції *SELECT* визначального запиту являє собою ім'я стовпця, а не константу, вираз або узагальнюючу функцію. Причому ім'я кожного зі стовпців у цьому списку згадується не більше одного разу;

По-третє, у пропозиції *FROM* повинно бути зазначене тільки одне відношення. Тобто представлення повинно бути створене на базі одного відношення, до якого користувач повинен мати необхідні права доступу. Якщо вихідне відношення саме є представленням, то воно також повинне відповідати зазначеним умовам. Дана вимога виключає можливість оновлення будь-яких представлень, що побудовані на базі з'єднання, об'єднання (*UNION*), перетинанням (*INTERSECT*) або різниці (*EXEPT*) відношень;

По-четверте, пропозиція *WHERE* не повинна включати ніяких вкладених запитів, які посилаються на відношення, що зазначені в пропозиції *FROM*;

По-п'яте, визначальний запит не повинен містити пропозиції *GROUP BY* або *HAVING*.

Крім того, будь-який рядок даних, який додається через представлення, не повинен порушувати вимоги підтримки цілісності даних, що встановлені для вихідного відношення. Наприклад, при додаванні через представлення нового рядка відношення в усі стовпці, які відсутні в ньому, будуть уведені значення *NULL*. Однак при цьому повинні дотримуватися усі вимоги щодо пропозиції *NOT NULL*, яка зазначена в описі вихідного відношення.

Основну концепцію, що використовувалась при формулюванні обмежень, можна виразити так. В оновлене представлення поміщаються

тільки ті рядки, які задовольняють умові *WHERE* у визначальному запиті. Якщо рядок у представленні буде змінено таким чином, що він перестане задовольняти цій умові, то він повинен з нього зникнути. Аналогічним чином, у представленні будуть з'являтися нові рядки щораз, коли вставка або оновлення даних у представленні приведе до того, що нові рядки будуть задовольняти умові *WHERE*. Рядки, які додаються або виключаються з представлення, прийнято називати мігруючими.

Фразу *WITH CHECK OPTION* можна вказувати тільки для представлень, що відновлюються. Вона гарантує, що рядок даних не задовольняючий умові, яка визначена в *WHERE*, не буде доданий у відношення, зазначене у фразі *FROM* оператора *SELECT*. У загальному випадку *WITH CHECK OPTION* в операторі *CREATE VIEW* використовується для запобігання міграції рядків з представлення. Якщо для представлення, що оновлюється, зазначити параметр *WITH CHECK OPTION*, то будуть відкидатися всі спроби додавання нових або зміни існуючих записів таким чином, щоб порушувалася умова в *WHERE* визначального оператору *SELECT*.

Ця функція може виявитися настільки корисною, що працювати з представленнями виявиться зручніше, ніж з відношеннями БД. У тому випадку, коли оператор *INSERT* або *UPDATE* порушує умови, зазначені в *WHERE* визначального запиту, виконання операції відміняється. У результаті з'являється можливість реалізувати в БД додаткові обмеження, які спрямовані на збереження коректності й цілісності даних.

Для вилучення представлення використовується оператор **DROP VIEW** Ім'я.

### 8.2. Способи й обмеження формування представлень даних

Хоча усі представлення створюються за допомогою одного методу, на практиці для різних цілей використовуються різні способи формування представлень: вертикальний зріз таблиці, горизонтальний зріз таблиці, вертикально-горизонтальний зріз таблиці, підмножина рядків та стовпців з'єднання різних таблиць [4, 6].

**8.2.1. Представлення даних формується як вертикальний зріз таблиці,** якщо оператор *SELECT* повертає значення не всіх стовпців таблиці, яка зазначена у *FROM*, і не визначене обмеження на кортежі, що увійдуть до представлення даних. Таке представлення даних звичайно використовується в тих випадках, коли дані з відношення обробляються різними користувачами або групами користувачів. За допомогою вертикальних представлень у їхнє розпорядження надаються віртуальні відношення, що мають тільки ті атрибути, які їм необхідні.

ПРИКЛАД 8.1.

Надати користувачам БД «БІБЛІОТЕКА» можливість перегляду наступних відомостей про читачів: № квитка читача, прізвище, ім'я, по батькові, місце роботи та посада (табл. 8.1).

**CREATE VIEW** ReadersView
**AS**
**SELECT** ReaderCardNumber, FamilyName, Name, Patronymic, Job, Post
**FROM** Readers

**8.2.2. Представлення даних формується як горизонтальний зріз таблиці,** якщо оператор *SELECT* повертає значення із усіх стовпців, але на кортежі, які увійдуть у представлення даних, накладене обмеження.

ПРИКЛАД 8.2.

Надати користувачам БД «БІБЛІОТЕКА» відомості про книги, видані після 1984 року (табл. 8.2).

**CREATE VIEW** AllBookAfter1984
**AS**
**SELECT** * FROM Books
**WHERE** IssueYear >= ’01.01.1984’

**8.2.3. Представлення даних формується як вертикально-горизонтальний зріз таблиці,** якщо оператор *SELECT* повертає значення не із усіх атрибутів відношення й на кортежі, які увійдуть у представлення даних, накладені обмеження.

Таблиця 8.1

**Результат роботи представлення даних(вертикальний зріз таблиці)**

| FamilyName | Name | Patronymic | Reader-Card-Number | Job | Post |
|---|---|---|---|---|---|
| Іванов | Петро | Іванович | 317 | НГУ, каф. ЕВТ | Асистент |
| Федорец | Ірина | Олегівна | 28 | НГУ, АХЧ | Вахтер |
| Ільїн | Іван | Петрович | 1345 | НГУ, каф. фізики | Доцент |
| Суренко | Дмитро | Павлович | 543 | НГУ, каф. геофізики | Ст. викладач |
| Коршунова | Наталя | Юріївна | 128 | НГУ, каф. гєоінформатики | Асистент |
| Носенко | Олег | Володимирович | 5672 | НГУ, ІКК | Інженер |
| Брусов | Володимир | Михайлович | 485 | НГУ, каф. геодезії | Лаборант |
| Козирєв | Олексій | Сергійович | 759 | НГУ, каф. кримінології | Професор |
| Левченко | Юлія | Павлівна | 146 | НГУ, каф. політичної теорії | Завідувач кафедри |
| Світла | Тетяна | Іванівна | 2021 | НГУ, каф. перекладу | Ст. викладач |
| Щиглів | Петро | Євгенович | 997 | НГУ, каф. електропостачання | Асистент |
| Кириленко | Віктор | Олександрович | 1010 | НГУ, каф. електропривода | Заступник декана |

ПРИКЛАД 8.3.

Надати користувачам БД «БІБЛІОТЕКА» відомості про книги, що видані після 1984 року. У представлення даних включити назву книги, рік, місце видання та шифр (табл. 8.3).

**CREATE VIEW** Bookafter1984
**AS**
**SELECT** Name, IssueYear, Drawing, Cipher
**FROM** Books
**WHERE** IssueYear >= ’01.01.1984’

**8.2.4. Представлення даних формується як підмножина рядків та стовпців,** якщо в операторові *SELECT* використовується внутрішні або зовнішні з'єднання відношень.

ПРИКЛАД 8.4.

Представлення даних повинно містити: номер читацького квитка, прізвище й ім'я читача, назву, інвентарний номер і вартість книги, дату видачі, повернення й фактичного повернення книги (табл. 8.4).

**CREATE VIEW** Debtors

(ReaderCardNumber, ReaderFamilyName, ReaderName, BookName,
BookInventoryNumber, BookCost, IssueDate, ReturnDate, FactReturnDate) **AS**
**SELECT** R.ReaderCardNumber, R.FamilyName, R.Name, B.Name,
I.InventoryNumber, I.Cost, G.IssueDate, G.ReturnDate, G.FactReturnDate
**FROM** R, Books B, BookInventoryNumbers I, BookGiveOutRecord G
**WHERE** G.ReturnDate < G.FactReturnDate ANDG.ReaderCode = R.Code
**AND** G.InventoryCode = I.Code AND I.BookCode = B.Code

**8.2.5. Обмеження на створення й використання представлень** включені в стандарт ISO. Ось деякі з них.

Якщо стовпець у представленні створюється з використанням агрегатної функції, то він може вказуватися в пропозиціях *SELECT* і *ORDER BY* тих запитів, які звертаються до даного представлення. Однак подібний стовпець не може використовуватися в *WHERE*, а також не може бути аргументом в узагальнюючій функції кожного із запитів, що звертаються до даної представлення.

Згруповане представлення ніколи не повинно з'єднуватися з таблицями БД або іншими представленнями.

Таблиця 8.2

**Результат роботи представлення даних (горизонтальний зріз таблиці)**

| Code | Name | Issue Year | Drawing | UDK | Cipher | Note |
|---|---|---|---|---|---|---|
| 1 | Автоматизація виробничих процесів на збагачувальній фабриці | 1985 | «Надра» | null | 622.7-52/Т | null |
| 3 | Асимптотичні методи оптимального управління | 1987 | «Автомат» | null | 681.513.5:/А | null |
| 4 | Синтез оптимальних автоматичних систем | 1984 | «Автомат» | null | 681.513.5:/ ДО | null |
| 5 | Методи оптимізації стохастичних систем | 1987 | «Матстат» | null | 681.513.5:/ ДО | null |
| 6 | Автоматизовані системи керування технологічним процесом збагачення руди | 1987 | «Автомат» | null | 622.7-52/П | null |
| 7 | С/С++ Програмування мовою високого рівня | 2007 | «Пітер» | null | 681.3.06(075) | null |
| 8 | Комп'ютерні мережі. Принципи, технології, протоколи. | 2006 | «Пітер» | null | 004.72(075) | null |

| | | | | | | |
|---|---|---|---|---|---|---|
| 9 | Довідник з диференційних рівняннях з частинними похідними першого порядку | 2003 | «ФІЗМАТ ЛІТ» | null | 517.9 | null |
| 10 | Теорія вірогідності і математична статистика | 2004 | «Пітер» | null | 519.2 | null |
| 11 | C#. Програмування мовою високого рівня | 2009 | «Пітер» | null | 004.43 | null |
| 12 | Теорія вірогідності і математична статистика | 2005 | «Вища школа» | null | 519.2 | null |
| 13 | Теорія вірогідності і математична статистика | 2002 | «ФІЗМАТ ЛІТ» | null | 519.2 | null |
| 14 | Дискретно-групові методи інтегрування звичайних диференційних рівнянь | 1991 | «ЛИИАН» | null | 517.9-37 | null |

## 8.3. Переваги та недоліки представлень

**8.3.1. Переваги.** У випадку роботи СУБД на персональному комп'ютері, що стоїть окремо, використання представлень зазвичай має на меті лише спрощення структури запитів до БД. Однак у випадку, коли СУБД обслуговує запити багатьох користувачів у мережі, представлення відіграють ключову роль у визначенні структури БД та організації захисту інформації. Основні переваги використання представлень у подібнім середовищі полягає у наступному [4].

**8.3.1.1. Незалежність від даних** для користувача полягає в стабілізації структури БД, яка буде залишатися незмінної навіть у випадку перегляду структури вихідних відношень. Наприклад, додавання або вилучення стовпців, зміни зв'язків, поділу відношень, їх реструктуризації або перейменування. Якщо у відношення додаються або з нього віддаляються атрибути, не використовувані у представленні, то змінювати визначення цього представлення не буде потрібно. Якщо структура вихідного відношення переупорядковується або воно розділяється, то можна буде створити представлення, що дозволяє користувачам працювати з віртуальним відношенням попереднього формату. У випадку поділу вихідного відношення на декілька частин (відношень), колишній формат може бути віртуально відновлений за допомогою представлення, побудованого на основі з'єднання цих відношень – звичайно, якщо це буде можливо. Можливість можна забезпечити за допомогою переміщення в усі нові відношення первинного ключа колишнього відношення.

Таблиця 8.3

**Результат роботи представлення даних (вертикально-горизонтальний зріз таблиці)**

| | | | |
|---|---|---|---|
| Автоматизація виробничих процесів на збагачувальній фабриці | 1985 | «Надра» | 622.7-52/Т |
| Асимптотичні методи оптимального керування | 1987 | «Автомат» | 681.513.5:/А |
| Синтез оптимальних автоматичних систем | 1984 | «Автомат» | 681.513.5:/ ДО |
| Методи оптимізації стохастичних систем | 1987 | «Матстат» | 681.513.5:/ ДО |
| Автоматизовані системи керування технологічним процесом збагачення руди | 1987 | «Автомат» | 622.7-52/П |
| C/C++ Програмування мовою високого рівня | 2007 | «Пітер» | 681.3.06(075) |

| | | | |
|---|---|---|---|
| Комп'ютерні мережі. Принципи, технології, протоколи. | 2006 | «Пітер» | 004.72(075) |
| Довідник з диференційних рівняннях з частинними похідними першого порядку | 2003 | «ФІЗМАТЛІТ» | 517.9 |
| Теорія вірогідності і математична статистика | 2004 | «Пітер» | 519.2 |
| C#. Програмування мовою високого рівня | 2009 | «Пітер» | 004.43 |
| Теорія вірогідності і математична статистика | 2005 | «Вища школа» | 519.2 |
| Теорія вірогідності і математична статистик | 2002 | «ФІЗМАТЛІТ» | 519.2 |
| Дискретно-групові методи інтегрування звичайних диференційних рівнянь | 1991 | «ЛПАН» | 517.9-37 |

Таблиця 8.4

**Результат роботи представлення даних (підмножина рядків та стовпців)**

| ReaderCard-Number | ReaderFamily-Name | ReaderName | BookName | BookInventory Number | BookCost | IssueDate | ReturnDate | FactReturnDate |
|---|---|---|---|---|---|---|---|---|
| 1345 | Ільїн | Іван | Синтез оптимальних автоматичних систем | 4512890 | 12.99 | 02.09.2004 | 16.09.2004 | 11.12.2004 |
| 543 | Суренко | Дмитро | Автоматизовані системи керування технологічним процесом збагачення руди | 4632112 | 10.1 | 30.10.2004 | 13.11.2004 | 10.01.2005 |
| 485 | Брусов | Володимир | C/C++ Програмування мовою високого рівня | 5478956 | 45.10 | 07.03.2009 | 21.03.2009 | 10.04.2009 |

**8.3.1.2. Актуальність** даних забезпечується негайним відображенням у представленні змін, що відбулись у будь-якому відношенні БД, яке зазначено у визначальному запиті.

**8.3.1.3. Підвищення захищеності даних** відбувається за рахунок того, що кожному користувачеві права доступу до даних можуть бути надані винятково через обмежений набір представлень, що включають тільки ту підмножину даних, з якою йому необхідно працювати. Подібний підхід дозволяє суттєво посилити контроль над доступом окремих категорій груп і окремих користувачів до інформації в БД.

**8.3.1.4. Додаткові зручності й можливості настроювання** моделі користувача даних полягають у максимальному її спрощенні. У результаті однакові відношення можуть бути показані різними користувачами в залежності від їх прав доступу та привілеїв. Це дозволить працювати тільки з тією частиною даних, яка дійсно їм необхідна. Так можна добитися максимального спрощення моделі даних, з якої буде працювати кінцевий користувач.

**8.3.1.5. Зниження складності** відображення даних є наслідком спрощення структури запитів, що поєднують дані з декількох відношень у єдине віртуальне відношення. У результаті багатотабличні запити зводяться до простих запитів, які працюють із одним представленням даних.

**8.3.1.6. Забезпечення цілісності даних** здійснюється за рахунок контролю над тим, щоб у вихідні відношення БД не був уведений жоден кортеж, що не задовольняє умовам відбору у визначальному запиті. Це

забезпечує пропозиція *WITH CHECK OPTION* оператора *CREATE VIEW*.

**8.3.2. Недоліки.** Використання представлень дозволяє досягнути істотних переваг, але необхідно звернути увагу на наступні недоліки [4].

**8.3.2.1. Обмежені можливості оновлення** полягають у неможливості корегування даних у представлені, що побудоване з декількох відношень.

**8.3.2.2. Структурні обмеження** стосуються структури представлення, яка встановлюється в момент його створення. Якщо визначальний запит представлений у форматі *SELECT * FROM…,* то символ * посилається на всі стовпці, що існують у вихідному відношенні на момент його створення. Якщо згодом у вихідне відношення БД будуть додані нові атрибути, то вони не з'являться в представлені доти, доки воно не буде вилучене і знову створене.

**8.3.2.3. Зниження продуктивності** СУБД пов'язане з використанням додаткових обчислювальних ресурсів для дозволу представлень. В одних випадках вплив цього фактора буде абсолютно незначним, тоді як в інших він може послужити джерелом істотних проблем. Наприклад, представлення, визначене за допомогою складного багатотабличного запиту може зажадати значних витрат часу на обробку, за рахунок виконання з'єднання відношень щоразу, коли знадобиться доступ до даного представлення.

## 8.4. КОНТРОЛЬНІ ПИТАННЯ

1. Яка різниця та що спільного між представленням даних та таблицею, яка визначена за допомогою оператору CREATE TABLE?
2. Яким чином проходить процес розв’язання представлення даних?
3. Які обов’язкові та додаткові можливості має оператор CREATE VIEW щодо визначення представлення даних?
4. Які особливості роботи визначального оператору SELECT Вам відомі?
5. Коли в CREATE VIEW список імен стовпців задається обов'язково?
6. Яка різниця між представленням даних, яке оновлюється, та представленням, що не оновлюється?
7. Які обмеження накладаються на представлення даних, що оновлюються?
8. Які особливості використання пропозиції WITH CHECK OPTION?
9. Чим відрізняються відомі Вам способи формування представлень даних?
10. Які обмеження накладаються на створення й використання представлень даних?
11. Які є переваги використання представлень даних?
12. Які недоліки в використанні представлень даних Вам відомі?

*Розглянутий матеріал допоможе студентові формувати представлення даних.*

# 9. Трансакції

*У цьому розділі розглядаються питання забезпечення цілісності даних, використовуючи «механізм» трансакцій.*

## 9.1. Проблеми, що виникають при внесенні змін у БД

Для локальних СУБД (Paradox, dbase і т.д.) характерний підхід негайного відображення змін. Відмовитися від модифікації даних у цьому випадку неможливо – адже зміни вже фізично внесені в БД. Правда, вилучений запис можна ввести заново вручну. Або інший варіант: вилучені записи можна зберігати в деякому проміжному (тимчасовому) відношенні БД. Якщо виникла потреба відмовитися від вилучення записів, то їх можна перемістити з тимчасового сховища назад у те відношення, з якого вони були вилучені.

Відмова від кожного модифікування даних в одному або декількох відношеннях називають відкотом до останнього стану БД, у якім забезпечуються всі види цілісності даних – посилальна цілісність, цілісність сутностей та корпоративні обмеження цілісності. Таким чином, будь-яка модифікація інформації в БД повинна гарантувати її перехід з одного цілісного стану в інший.

Класичним прикладом переходу БД є бухгалтерська проводка: деяка сума S повинна бути списана з рахунку К та зарахована на рахунок D. Тільки успішне виконання цих двох операцій гарантує цілісність інформації в БД. Але цілісність буде порушена, якщо в результаті збою сума S буде списана з рахунку К, але не буде зарахована на рахунок D або, навпаки, зарахована на D, але не списана з К. Тому у випадку помилки списання/зарахування суми результати попередньої операції зарахування/списання повинні бути скасовані. Інакше кажучи, бухгалтерська проводка складається із групи наступних елементарних операцій: списання з рахунку К суми S; зарахування суми S на рахунок D.

Існують механізми відкату змін у БД у випадку невиконання умови успішного завершення всіх елементарних операцій, які входять до складу групи. Основним механізмів є обробка трансакцій [2, 4, 6, 7]. Звичайно обробка трансакцій реалізується в промислових БД. Однак ряд засобів розробки прикладних програм [5] дозволяють управляти трансакціями й для таблиць локальних СУБД (Paradox, dbase). Крім того, як для промислових, так і локальних у цих засобах розробки надається додатковий механізм управління відкотами змін у БД – так звані кешовані зміни (cached updates, що часто переводять також як буферизовані зміни).

## 9.2. Визначення й властивості трансакції

***Трансакція*** *– це дія або серія дій, що виконуються одним користувачем або прикладною програмою*, які здійснюють доступ до даних або зміну вмісту БД [4].

Трансакція є логічною одиницею роботи, що виконується в БД. Вона може бути представлена окремою програмою, бути частиною алгоритму програми або навіть окремою командою (наприклад, *INSERT* або *UPDATE*) і включати довільну кількість операцій, що виконуються у БД. З погляду БД, виконання прикладної програми може розцінюватися як серія трансакцій, у проміжках між якими виконується обробка даних, здійснювана поза БД.

Будь-яка трансакція завжди повинна переводити БД з одного погодженого стану в інший, хоча допускається, що погодженість її стану буде порушуватися в ході виконання трансакції. Тому, будь-яка трансакція завершується одним із двох можливих способів. У випадку успішного завершення результати трансакції фіксуються (*commit*) у БД, і вона переходить у новий погоджений стан. Якщо виконання трансакції не є успішним, вона відміняється (*rollback*). У цьому випадку у БД повинен бути відновлений той погоджений стан, у якім вона перебувала до початку даної трансакції. Цей процес називається відкотом (*rollback*) трансакції. Зафіксована трансакція не може бути скасована. Якщо виявляється, що зафіксована трансакція була помилковою, буде потрібно виконати іншу трансакцію, що скасовує її дії. Таку трансакцію називають ***компенсуючою***. Слід зазначити, що скасована трансакція може бути ще раз запущена пізніше й, залежно від причин попередньої відмови, цілком успішно завершена й зафіксована в БД.

Ніяка СУБД не має внутрішньої можливості встановити, які саме зміни повинні бути сприйняті як єдине ціле, що утворює одну логічну трансакцію. Тому існує метод, що дозволяє вказувати межі кожної із трансакцій ззовні, з боку користувача. У більшості мов маніпулювання даними для визначення меж окремих трансакцій використовуються оператори *BEGIN TRANSACTION*, *COMMIT* і *ROLLBACK* (або їх еквіваленти). Якщо вони не були використані, усі дії програми, що виконувала зміни в БД, розцінюються як єдина трансакція. СУБД автоматично виконає команду *COMMIT*, якщо програма мала успішне завершення. В іншому випадку СУБД автоматично виконує команду *ROLLBACK*.

Існують чотири основні властивості трансакції (ACID – абревіатура, складена з перших букв їх англійських назв).

***Атомарність***. Ця властивість типу "усі або нічого". Будь-яка трансакція являє собою неподільну одиницю роботи, яка може бути або виконана вся цілком, або не виконана зовсім.

***Погодженість***. Кожна трансакція повинна переводити БД з одного погодженого стану в інший погоджений стан.

***Ізольованість***. Усі трансакції виконуються незалежно одна від іншої. Інакше кажучи, проміжні результати незавершеної трансакції не повинні бути доступні іншим трансакціям.

***Тривалість***. Результати успішно завершеної (зафіксованої) трансакції повинні зберігатися в базі даних постійно й не повинні бути загублені в результаті наступних збоїв.

### 9.3. Рівні ізоляції трансакцій: прикладна програма клієнта

**9.3.1. Проблеми одночасної зміни даних** виникають при одночасній роботі декількох клієнтів з одним відношенням. Нехай користувач А одержав дані з відношення Books і згодом змінив їх. У цей час, з тим же записом відношення Books, працює користувач В. Він також модифікував дані в тому ж записі, що й А, і намагається підтвердити ці зміни. Користувач С працює з відношенням Books у режимі читання. Відразу ж виникає низка питань: дозволяти або не дозволяти В змінювати запис, якщо А ще не підтвердив свої зміни? Чи дозволяти С бачити зміни, внесені А та В? Чи може А бачити зміни, внесені В, та навпаки?

Для вирішення зазначених проблем існує кілька рівнів ізоляції (розмежування) трансакцій: *Dirty Read*, *Read Committed*, *Repeatable Read*. Рівень ізоляції трансакції визначає:

чи можуть інші (конкуруючі) трансакції вносити зміни в дані, змінені поточної трансакцією;

чи може поточна трансакція бачити зміни, зроблені конкуруючими трансакціями, та навпаки.

**9.3.2. Рівень ізоляції трансакції *Dirty Read*** дає можливість конкуруючим трансакціям бачити зміни, які внесені але непідтверджені поточною трансакцією. Якщо поточна трансакція скасує зроблені зміни, інші трансакції будуть бачити недостовірні дані. Цей рівень ізоляції може привести до серйозних помилок і застосовується рідко**.**

**9.3.3. Рівень ізоляції трансакції *Read Committed*** дозволяє конкуруючим трансакціям оперувати тільки підтвердженими змінами, які зроблені у поточній трансакції.

**9.3.3.1. Читання даних.** Нехай трансакції А и В виконуються клієнтами, які корегують або читають данні одного відношення БД. Нехай трансакція А змінила дані, але не підтвердила зміни. Нехай конкуруюча трансакція В намагається зчитати ці дані. Тоді вона одержує їх у тому стані, у якім вони перебували до старту трансакції А. Іншими словами, трансакція В не бачить у відношенні змін, внесених але не підтверджених трансакцією А.

Можливий і інший варіант. Нехай трансакція А вносить зміни в дані деякого відношення й не підтверджує їх. У цей час стартує трансакція В у прикладній програмі, де не відкрите відношення, у якому трансакція А змінює данні. Тоді спроба ініціалізації трансакції В буде відхилена. Відкриття набору даних клієнтом стане можливим лише після того, як А підтвердить зроблені зміни.

**9.3.3.2. Зміна даних**. Нехай трансакції А и В започатковуються прикладними програмами, у кожному з яких відкритий набір даних, що пов'язаний з однаковим відношенням БД. Нехай трансакція А змінила

дані, але не підтвердила зміни. Нехай конкуруюча трансакція В також внесла зміни в ті самі дані. Тоді спроба трансакції В підтвердити внесені зміни буде відхилена.

**9.3.4. Рівень ізоляції трансакції *Repeatable Read*** забезпечує ситуацію, при якій поточна трансакція завжди бачить дані в тому стані, у якім вони перебували на момент її старту.

**9.3.4.1. Читання даних**. Нехай трансакція А відкрила набір даних. Після цього трансакція В внесла в ті ж дані зміни й не підтвердила їх. Тоді при повторному відкритті набору даних трансакція А одержить дані у тому стані, в якім вони перебували на момент її старту. Однак «свої» зміни А бачити буде. Нехай трансакція В підтвердила зроблені нею зміни, а трансакція А знову відкрила набір даних. І в цьому випадку трансакція А одержить дані в тому стані, у якім вони перебували на момент її старту.

**9.3.4.2. Зміна даних.** Нехай трансакція А внесла зміни в дані й не підтвердила їх. Трансакція В після цього також внесла зміни в ті ж дані. Тоді спроба трансакції В підтвердити внесені нею зміни буде відхилена.

**9.3.5. Визначення рівня ізоляції трансакцій** у прикладній програмі клієнта залежить від середовища розробки. Наприклад, прикладні програми написані в середовищі Delphi або C-Bilder рівень ізоляції трансакцій визначається властивістю компонента Tdatabase – property TransIsolation: TtransIsolation. Можливі значення: tiDirtyRead, tiReadCommitted, tiRepeatableRead. Різні сервери БД порізному інтерпретують рівні ізоляції трансакцій, що визначені у властивості TransIsolation (табл. 9.1).

Таблиця 9.1

**Інтерпретація рівнів ізоляції трансакцій, що визначені у властивості TransIsolation**

| Сервер | Рівень трансакції | Інтерпретується як |
|---|---|---|
| Oracle | Tidirtyread<br>tireadcommitted<br>tirepeatableread | tireadcommitted<br>tireadcommitted<br>tirepeatableread (Тільки для читання) |
| Sybase, MS-SQL | Tidirtyread<br>tireadcommitted<br>tirepeatableread | tireadcommitted<br>tireadcommitted<br>Не підтримується |
| DB2, Informix, Interbase | tidirtyread<br>tireadcommitted<br>tirepeatableread | tidirtyread<br>tireadcommitted<br>tirepeatableread |
| Paradox, dbase | tidirtyread<br>tireadcommitted<br>tirepeatableread | tidirtyread<br>Не підтримується<br>Не підтримується |

## 9.4. Управління трансакціями на стороні SQL-серверу InterBase

InterBase управляє трансакціями за допомогою SQL-операторів *SET TRANSACTION* (почати трансакцію), *COMMIT* (підтвердити трансакцію) і *ROLLBACK*, (відкотити трансакцію). *SET TRANSACTION* має наступній формат [10, 14, 15]:

**SET TRANSACTION [READ WRITE | READ ONLY]**
**[WAIT | NO WAIT]**
**[[ISOLATION LEVEL] {SNAPSHOT [TABLE STABILITY]**
**| READ COMMITTED [[NO] RECORD_VERSION]}]**
**[RESERVING** <список таблиць>
**[FOR [SHARKD | PROTECTED] [READ | WRITE]],** [<список таблиць> ...].

*READ WRITE* | *READ ONLY* установлює рівень доступу до даних (за замовчуванням *READ WRITE*);

*WAIT* | *NO WAIT* визначає поведінку при виникненні конфлікту по відновленню запису даної трансакції з іншою трансакцією, що раніше зробила зміну в тому ж записі: *WAIT* (за замовчуванням) спонукає дану трансакцію очікувати завершення конкуруючої трансакції; *NO WAIT* визначає аварійне завершення даної трансакції;

*ISOLATION LEVEL* визначає рівні ізоляції трансакцій на сервері (за замовчуванням *SNAPSHOT*);

*RESERVING* у рамках даної трансакції забезпечує монопольний доступ до таблиць, що наведені в одному або декількох списках. В останньому випадку кожному елементу списку таблиць ставляться у відповідність параметри:

*PROTECTED READ* – конкуруючі трансакції можуть читати дані, але не можуть змінювати;

*PROTECTED WRITE* – читати дані можуть тільки трансакції з рівнями *SNAPSHOT* або *READ COMMITTED* і ніяка конкуруюча трансакція не може їх змінювати.

Стандарт *ISO* включає визначення моделі трансакцій, побудованої на використанні двох спеціальні операторів – *COMMIT* і *ROLLBACK*. Більшість комерційних реалізацій мови *SQL*, у тому числі й InterBase, підтримує цю модель, яка вперше була реалізовано в СУБД DB2 компанії IBM. У стандарті вказується, що в мові *SQL* трансакція автоматично запускається будь-яким *SQL*-оператором, що виконується користувачем, або програмою (наприклад, *SELECT*, *INSERT* або *UPDATE*). Зміни, що внесені в БД у ході виконання даної трансакції, не будуть доступні будь-яким іншим трансакціям, які виконуються паралельно, доти, доки ця трансакція не буде явно завершена. Завершення трансакції може бути виконано одним з наступних способів.

1. Уведення оператору *COMMIT* означає успішне завершення трансакції. Після його виконання, зміни, які внесені до БД, здобувають постійний характер. Після обробки оператора *COMMIT* уведення будь-якого ініціюючого трансакцію оператора автоматично викличе запуск нової трансакції.

2. Уведення оператору *ROLLBACK* означає відмову від завершення трансакції, у результаті чого виконується відкіт усіх змін, які внесені у БД при виконанні цієї трансакції. Після обробки оператора *ROLLBACK* уведення

будь-якого ініціюючого трансакцію оператора автоматично викличе запуск нової трансакції.

3. При використанні *SQL*-операторів у тексті програми успішне закінчення її роботи автоматично викличе завершення останньою трансакції, що запущена програмою до виконання, навіть якщо оператор *COMMIT* для неї не був уведений явно.

4. При використанні *SQL*-операторів у текст програми аварійне закінчення її роботи автоматично викличе відмову останньої трансакції, що виконувалась в межах цією програми.

## 9.5. КОНТРОЛЬНІ ПИТАННЯ

1. Які проблеми спонукали створенню механізму трансакцій у СУБД?
2. Що таке трансакція?
3. Як механізм трансакцій забезпечує цілісність даних?
4. Які властивості трансакцій Вам відомі?
5. Які рівні ізоляції трансакцій Вам відомі та що вони визначають?
6. В чому полягає основний недолік рівня ізоляції трансакції Dirty Read?
7. Як працює рівень ізоляції трансакції Read Committed при читання та зміні даних?
8. Як працює рівень ізоляції трансакції Repeatable Read при читання та зміні даних?
9. Які основні можливості закладені в оператор SET TRANSACTION?
10. Які Вам відомі способи завершення трансакції?

*Розглянутий матеріал допоможе студентові, використовуючи потрібний рівень ізоляції трансакцій, на практиці забезпечити цілісність даних.*

# 10. Оптимізація роботи БД

*Мета розділу – розглянути основні напрямки оптимізації роботи БД.*

## 10.1. Напрямки оптимізації роботи БД

Під оптимальною роботою БД зазвичай розуміють створення таких умов для доступу й маніпулювання даними, що забезпечують максимальну швидкодію при мінімальних витратах ресурсів. На такий важливий ресурс, як продуктивність сервера, розроблювачі БД можуть впливати не часто. Тому, декларувавши відомий факт, що чим потужніше апаратна частина, тим краще, розглянемо такі напрямки, які піддаються впливу з боку розроблювачів і адміністраторів БД:

1) оптимізація структури БД;
2) оптимізація запитів;
3) оптимізація прикладної програми клієнта.

У свою чергу, оптимізацію структури БД ведуть по трьом напрямкам:

1) нормалізація відношень;
2) методи доступу до даних;
3) налаштування внутрішніх механізмів СУБД.

Оптимізація запитів містить у собі:

1) побудова структури БД адекватної запитам;
2) оптимізацію структури індексів таблиць БД;
3) оптимізацію текстів запитів.

Оптимізацію прикладних програм клієнтів будемо розглядати лише з погляду організації ефективного доступу до БД.

## 10.2. Оптимізація структури БД

**10.2.1. Нормалізація відношень** впливає на швидкодію операцій вибірки даних [4…7, 10, 12]. Часто нормалізація погіршує швидкодію. З погляду теорії, вона приводить до найбільш ясного відображення сутностей предметної області. Усуваючи надмірність даних, вона тим самим суттєво заощаджує дисковий простір. З погляду практики, і саме на великих обсягах даних, оптимізація може суттєво зменшити швидкодію. При звертанні до однієї, нехай великої, таблиці БД, витрачається менше часу, чим при звертанні до декількох, більш дрібних таблиць із застосуванням операцій з'єднання.

Крім того, високий ступінь нормалізації приводить до збільшення кількості відношень у БД. У результаті цього структура БД не сприймається розроблювачем та користувачами цілісно. Спрацьовує так званий "людський фактор". Це здатне внести серйозні помилки в структуру БД уже на стадії логічного проектування, що надалі може мати самі серйозні – і завжди негативні – наслідки.

Тому при проектуванні структури БД слід враховувати як негативні, так і позитивні сторони нормалізації відношень. Звичайно жертвують додатковою витратою дискової пам'яті для зберігання не повністю

нормалізованих таблиць, прагнучи забезпечити максимальну швидкодію, що актуально при зростанні числа користувачів, що одночасно працюють із системою.

**10.2.2. Методи доступу до даних.** Хоча з теорії відомо, що структура даних повинна бути незалежна від способів доступу до них, на практиці це не так. Наприклад, виникають ситуації забезпечення доступу й корегування даних, які беруть участь у досить складних обчисленнях тих або інших значень, декількома користувачами. У таких ситуаціях іноді неможливо обійтися без додаткового поля, яке визначає поточний статус цієї сукупності даних. Його значення дозволяє читати, коректувати або переглядати дані, задіяні в розрахунках при звертанні до них групи користувачів з різних терміналів. Таким чином, проектуючи логічну структуру БД, неможливо абстрагуватися від того, яким чином дані будуть оброблятися на сервері та у прикладній програмі клієнта.

Відмітимо, що залежність між текстами запитів до БД, структурою й складом індексів таблиць також свідчить про зв'язок між структурою даних і методами доступу до них.

**10.2.3. Налаштування внутрішніх механізмів СУБД. Д**ля швидкого доступу до таблиці БД необхідно, щоб вона фізично займала безперервний блок сторінок. Відомо, що при виділенні нових сторінок у СУБД InterBase не робить ніяких спроб виділяти суміжні сторінки для зберігання однієї й тієї ж таблиці [14, 15]. Тому дані, що відносяться до однієї сторінки, можуть бути інформацією з декількох таблиць, тобто фрагментовані. Зберігання множинних поколінь записів також приводить до сильного "забруднення" БД і сповільнює роботу з нею. При всякій зміні запису фактично створюється її нова версія, яка змінена або додана трансакціями, що згодом відмінялися та які з БД не вилучаються. Крім того, при вилученні записів не відбувається переміщення їх версій, які залишилися, для того, щоб вилучити "дірки", які утворювалися на сторінках БД.

Вирішенням зазначених проблем є періодичне створення резервної копії й оновлення з неї БД. У цьому випадку: збирається "сміття", тобто версії записів, які далі не будуть використовуватись. Усуваються "дірки" на сторінках БД, що утворювалися після вилучення записів. Кожна таблиця розміщається в безперервному блоці сторінок.

Розмір самої сторінки БД має значення. Читання-запис у СУБД InterBase здійснюються сторінками. Тому запис таблиці по можливості повинен розміщатися в межах однієї сторінки. Якщо розмір сторінки малий для зберігання одного запису й вона розташовується на декількох сторінках, для читання такого запису потрібно виконати кілька фізичних операцій читання. З іншого боку, розмір сторінки не повинен бути занадто великий, оскільки в цьому випадку будуть зчитуватися непотрібні записи.

Розмір буфера вводу-виводу також здатний впливати на швидкодію. Для БД, до яких частіше застосовуються операції читання,

рекомендується збільшити розмір буфера вводу-виводу. Для БД, у яких частіше виконуються операції запису, розмір буфера рекомендується зменшити.

### 10.3. Індексація таблиць

Наявність індексу може суттєво підвищити швидкість виконання деяких запитів. Однак, оскільки індекси повинні оновлюватися системою при кожнім внесенні змін у базову таблицю, вони створюють додаткове навантаження на систему. Індекси зазвичай створюються з метою задоволення певних критеріїв пошуку, після того як таблиця вже перебувала якийсь час у роботі й збільшилася в розмірах. Вони можуть створюватися тільки для таблиць БД, але не для представлень.

***Індекс*** *– це структура даних, що дозволяє скоротити час доступу до окремих рядків таблиць*, незалежно від їхнього фізичного розміщення. Він аналогічний предметному покажчику, що наведений наприкінці книги. Це структура, яка пов'язана з файлом і призначена для пошуку інформації за тим принципом, що й предметний покажчик у книзі. Індекс дозволяє уникнути проведення послідовного або покрокового сканування файлу у пошуках потрібних даних. При його використанні об'єктом пошуку може бути один або кілька записів файлу. Як і предметний покажчик книги, індекс упорядкований і кожний його елемент містить назву об'єкту пошуку, а також один або кілька покажчиків (ідентифікаторів записів), що мають посилання на місце його розташування.

В InterBase розділені поняття ключів та індексів на логічному рівні [14, 15]. Первинний (*PRIMARY KEY*) і зовнішній (*FOREIGN KEY*) ключі будуються для забезпечення посилальної цілісності реляційно-пов'язаних таблиць. Первинний ключ, крім цього, виконує функції підтримки унікальності своїх значень, що обумовлене його основним призначенням – однозначно характеризувати запис у таблиці БД. Для таких же цілей може використовуватися й просто унікальний ключ (*UNIQUE*). Однак вони використовуються операторами *SELECT* для прискорення доступу до даних. "Звичайні" індекси на відміну від ключів, служать лише для оптимізації доступу до даних. З фізичної точки зору, те, що логічно при створенні таблиць поділялося на ключі та індекси, на фізичному рівні перетвориться на індекси. Однак, якщо "звичайним" індексам можна привласнити ім'я, то фізичні індекси, реалізовані на основі визначень ключів, будуються й іменуються системою автоматично.

Більшість діалектів, у тому числі InterBase, підтримують наступний оператор, що дозволяє створювати індекси для таблиць БД:

**CREATE [UNIQUE] [ASC[ENDING] | DESC[ENDING]]**
**INDEX** Ім'я **ON** Ім'я таблиці (атрибут [, атрибут ...])**.**

Тут обов'язково необхідно вказати Ім'я індексу, Ім'я таблиці й перерахувати імена атрибутів, по яких необхідно побудувати індекс. Нагадаємо, що формат оператора *CREATE INDEX* дозволяє виключити дублювання значень у ключових

атрибутах, вказавши пропозицію *UNIQUE*. У цьому випадку унікальність значень потенційного ключа буде автоматично підтримуватися системою. За замовчуванням сортування значень індексних полів ведеться по зростанню. Це можна вказати явно, використовуючи пропозицію *ASC[ENDING]*. Якщо необхідно забезпечити зворотний порядок сортування – по зменшенню, тоді використовують пропозицію *DESC[ENDING]*.

Створення індексів можливо в будь-який момент, навіть для вже заповненої даними таблиці. Однак можуть виникнути проблеми, що пов'язані з дублюванням значень у різних рядках, якщо в операторі використовується пропозиція *UNIQUE*. Тому має сенс створювати унікальні індекси безпосередньо при створенні таблиці, перед тим як у неї почали заносити інформацію. У результаті система відразу ж візьме на себе контроль над унікальністю значень у відповідних атрибутах.

Для вилучення індексу використовується оператор

**DROP INDEX** Ім'я;

Їм не можна вилучити індекс, створений в операторові *CREATE TABLE* для визначення первинного й зовнішнього ключів, а також унікальності значень атрибута або групи атрибутів, що утворюють потенційний ключ (*PRIMARY KEY*, *FOREIGN KEY*, *UNIQUE*). Для цієї мети слід застосовувати оператор *ALTER TABLE*. Не можна також вилучити індекс, що використовується та може мати місце при виконанні іншими користувачами запитів до БД. Крім цього, для вилучення індексу потрібно мати відповідні привілеї доступу до БД.

### 10.4. Оптимізація структури індексів

**10.4.1. Загальні рекомендації.** Від структури індексів у величезному ступені залежить ефективність виконання запитів [4, 6]. При виконанні запитів СУБД спочатку переглядає список індексів, які визначені для таблиць, що беруть участь у запиті. Потім вибирається одна із двох схем виконання запиту – використовувати наявні індекси або послідовно переглянути таблиці. Оптимізатор СУБД прагне виконати запит з максимальною швидкодією й з мінімальними накладними витратами. Перед першим викликом СУБД завжди оптимізує виконання запиту, ґрунтуючись на поточному стані БД. Повторно запити, у яких змінюються тільки значення параметрів, не оптимізуються. Відбувається лише попереднє зв'язування формальних та фактичних параметрів, після чого запит виконується.

Повністю передбачити за якою схемою оптимізатор СУБД буде виконувати запит неможливо, тому що стан БД може змінюватися. Але існують загальні положення, які слід враховувати при проектуванні запитів і структури індексів. Індекси необхідно створювати у випадку, коли по стовпцю або групі стовпців часто:

1) проводиться пошук у БД: атрибут або група атрибутів часто перелічуються в пропозиції *WHERE* оператора *SELECT*;

2) будуються об'єднання таблиць;

3) проводиться сортування результатів у відношеннях, що вертаються в якості, запитів до БД: атрибут або група атрибутів часто використовуються в пропозиції *ORDER BY* оператора *SELECT*.

Не рекомендується будувати індекси по стовпцях або групах стовпців, які:

1) рідко використовуються для пошуку, об'єднання й сортування результатів запитів;

2) часто міняють значення, що приводить до необхідності частого оновлення індексів, що здатне суттєво сповільнити швидкість роботи із БД;

3) містять невелике число варіантів значення.

У випадку, коли при виконанні запитів використовується сортування по тих самих стовпцях, необхідно пам'ятати наступне: створення єдиного індексу здатне прискорити виконання запитів. Якщо в умові пошуку або в пропозиції *ORDER BY* використовуються не всі стовпці із цього індексу, то для оптимізації запиту застосовують тільки безперервну послідовність стовпців.

**10.4.2. Часткове використання складеного індексу** дозволяє суттєво скоротити витрати ресурсів СУБД [2, 4, 12]. Для цього складений індекс слід будувати так, щоб стовпці, по яких найбільш часто ведеться пошук, перебували на початку списку. Тоді для пошуку може бути використана частина індексу. Наприклад, нехай часто виконуються запити

```
SELECT  *
        FROM SOMETABLE
        WHERE A = : ParamA AND B = :ParamB;
SELECT *
        FROM SOMETABLE
        WHERE A = : ParamA AND B = :ParamB AND C = : ParamC;
SELECT *
        FROM SOMETABLE
        WHERE A = : ParamA AND B = :ParamB AND
                C = :ParamC AND D = :ParamD.
```

Тоді, побудувавши складений індекс по стовпцях A, B, C, D, ми можемо затверджувати, що даний індекс буде використаний при оптимізації всіх трьох запитів. У першому випадку буде використана підмножина індексу, тобто значення A, B; у другому - значення A, B, C, у третьому - A, B, C, D ( тобто всі значення індексу).

Порядок проходження умов по стовпцях у пропозиції *WHERE* оператора *SELECT* не важливий, якщо умови об'єднані за допомогою *AND*. Наприклад, для виконання наступних запитів може використовуватися індекс, що використовувався в попередньому випадку:

```
SELECT  *
        FROM SOMETABLE
        WHERE A = 100 AND B = 200;
SELECT *
        FROM SOMETABLE
        WHERE B = 200 AND A = 100.
```

Для пропозиції *ORDER BY* порядок перерахування стовпців у складеному індексі має значення. Наприклад, для оптимізації виконання наступних запитів необхідно використовувати інший індекс:

```
SELECT *
        FROM SOMETABLE
        ORDER BY A, B, C;
SELECT *
        FROM SOMETABLE
        ORDER BY A, C, B.
```

Слід пам'ятати, що в пропозиції *ORDER BY* можна використовувати тільки безперервну послідовність значень. Наприклад, попередній індекс не може використовуватися для виконання запитів.

```
SELECT  *
        FROM SOMETABLE
        ORDER BY A, C;
SELECT *
        FROM SOMETABLE
        ORDER BY B, D.
```

Але він буде корисним для запиту виду:

```
SELECT *FROM SOMETABLE ORDER BY A, B.
```

**10.4.3. Багатопоточність пошуку по OR та IN.** При частому використанні в умовах пошуку пропозиції *WHERE* декількох стовпців, які пов'язані між собою операцією "або" (*OR*) замість індексу по стовпцях A, B, C створюють кілька індексів, що побудовані по кожному стовпцю окремо, оскільки інакше буде здійснений послідовний перегляд усієї таблиці. Це пов'язане з тим, що індексно-послідовний доступ для індексу по атрибутах A, B, C може бути здійснений тільки для стовпця A; значення стовпців B и C у цьому випадку розкидані по індексу. Важливо пам'ятати, що при використанні оператора *OR* кожна частина умов пошуку спричиняє окреме сканування таблиць, що беруть участь у запиті. Так, наприклад, при виконанні оператора *SELECT*

```
SELECT *
        FROM SOMETABLE
        WHERE A = 100 OR B = 200 OR C = 300
```

буде здійснено три окремі сканування таблиці для пошуку значень, що задовольняють умовам A = 100; B = 200; C = 300.

Окремий потік пошуку породжує й кожний елемент у списку *IN*. Наприклад, *WHERE* A *IN* (100, 200, 300) інтерпретується як *WHERE* A = 100 OR A = 200 OR A = 300. Однак при вказівці діапазону *BETWEEN WHERE* A *BETWEEN* 100 *AND* 300 цього не відбувається. Тому там, де можливо, слід заміняти *IN* на *BETWEEN*.

**10.4.4. Зменшення загальної кількості індексів** здійснюється за рахунок раціонального використання складених індексів, а також забезпечення посилальної цілісності за допомогою тригерів. При великій кількості індексів знижується швидкість додавання, зміни й вилучення записів у таблицях БД. Нам відомо, що в БД визначається два види індексів: одні для прискорення доступу до даних, інші – для забезпечення посилальної цілісності. Якщо останні не беруть участь в оптимізації доступу до даних, тоді їх доцільно вилучати, а посилальну цілісність забезпечувати за допомогою тригерів.

### 10.5. Поліпшення продуктивності роботи індексу

**10.5.1. Розбалансування індексу** приводить до того, що його глибина (depth) зростає понад критичне значення (2). *Глибина індексу* – параметр, що показує максимальне число операцій, необхідних для знаходження шуканого значення в таблиці БД з використанням даного індексу. Індекси таблиці можуть бути розбалансовані після багаторазового внесення змін. У випадку розбалансування цінність індексу при виконанні запитів знижується через збільшення часу, затрачуваного на вибірку даних**.** Тому час від часу необхідно робити одну з перерахованих нижче дій:

1) виконувати перебудову індексу;
2) перерахувати показник "вибираємості" індексу;
3) знищити індекс і заново його створити.

**10.5.2. Перебудова індексу** полягає в перестворенні й збалансуванні індексу, що наступає після деактивізації індексу оператором

**ALTER INDEX** Ім'я **INACTIVATE**

і наступної його активізації оператором

**ALTER INDEX** Ім'я **ACTIVATE**.

Деактивізація індексу корисна також у тому випадку, коли має місце вставка в таблицю БД великої кількості записів. У звичайному режимі додавання записів при активному індексі зміни вносяться в індекс у міру додавання записів у таблицю, що може його розбалансувати. Слід пам'ятати, що:

1) не можна перебудувати індекс, використовуваний у цей момент іншими користувачами запитів до БД;
2) не можна перебудувати індекс, створений у результаті визначення первинного й зовнішнього ключів, а також унікальності значень стовпця або групи стовпців (*PRIMARY KEY*, *FOREIGN KEY*,

*UNIQUE*). Для цієї мети слід застосовувати оператор *ALTER TABLE*;

3) для виконання оператора *ALTER INDEX* потрібно мати відповідні привілеї доступу до БД.

**10.5.3. Показник "корисності" індексу** заснований на відомостях про число повторюваних рядків індексу. Він використовується СУБД при доступі до таблиці для вироблення оптимального плану виконання запиту. Ефективність використання індексу при пошуку інформації в таблиці БД сильно залежить від того, чи побудований індекс за унікальними значенням і, якщо ні, наскільки відрізняються дані, за якими він побудований.

Показник корисності індексу розраховується при його створенні як число значень, що різняться, індексних полів усередині індексу, яке віднесене до середньої кількості записів. Після внесення змін у таблицю змінюється ступінь відмінності значень стовпців, по яких побудований індекс. Тому розрахований показник корисності може не відображати реального стану індексу й значення показника рекомендується примусово перераховувати: час від часу – при внесенні невеликих змін і завжди – при внесенні істотних змін. Перерахування реалізується оператором

**SET STATISTICS INDEX** Ім'я.

Для участі у виконанні запиту вибираються індекси з максимальним показником корисності. Такі індекси забезпечують більш швидкий пошук. Максимальний показник корисності мають унікальні індекси.

Середня кількість записів - показник, який розраховується щораз при оптимізації запиту як кількість сторінок БД, зайнятих цією таблицею, поділене на максимальне число записів на сторінці. Зменшення числа сторінок, зайнятих БД, і знищення на них "дірок" ведуть до зменшення показника середнього числа записів і, як наслідок, до підвищення показника корисності індексів. Це ще один аргумент на користь періодичної оптимізації БД шляхом створення резервної копії та її оновлення.

### 10.6. Перегляд і складання плану виконання запитів

Утиліті WISQL можна встановити режим показу плану виконання запиту (вибравши елемент меню Session | Basic Settings і відзначивши режим Display Query Plan). Тоді при виконанні запитів буде виводитися разом з результатом план їх виконання. Під планом виконання запиту розуміється перелік індексів, які використовуються СУБД для вибірки даних з таблиць.

Для примусового виконання запиту по тому або іншому плану, необхідно в операторі *SELECT* указати наступну пропозицію:

**PLAN** <план виконання запиту>, де <план виконання запиту> =
**[JOIN | [SORT] MERGE]** ({елемент плану | план виконання запиту}
[, {елемент плану | план виконання запиту} ...]),
де
<елемент плану> = {таблиця | псевдонім}
**NATURAL | INDEX** (<*індекс*> [, <*індекс*> ...]) **| ORDER** <індекс>.

Синтаксис пропозиції <план виконання запиту> дозволяє з'єднати між собою кілька таблиць (пропозиція *JOIN*). У випадку відсутності індексів, по яких записи таблиці можуть бути впорядковані, застосовують примусове сортування (пропозиція [*SORT*] *MERGE*). У пропозиції <елемент плану> – вказують таблицю, у якій проводиться пошук даних. Якщо таблиця кілька раз бере участь у запиті, для скорочення тексту плану корисно застосовувати її псевдонім. Пропозиція *NATURAL*, вказує, що для пошуку записів застосовується послідовний доступ. Це єдиний спосіб пошуку в тому випадку, якщо немає відповідних індексів. Можна вказати один або кілька індексів, які повинні використовуватися для пошуку записів, що задовольняють умові запиту (пропозиція *INDEX*). А пропозиція *ORDER* дозволяє визначити індекс, що необхідний для сортування таблиці. Приклади використання пропозиції *PLAN* рекомендується подивитися у відповідній документації до СУБД.

## 10.7. Оптимізація прикладних програм клієнтів

**10.7.1. Мінімізація кількості з'єднань із БД** дозволяє заощаджувати ресурси системи [3, 5, 13, 18]. Витрата системних ресурсів може позначитися на ефективності доступу до даних. Рекомендується зводити кількість з'єднань до мінімуму, а в ідеалі використовувати тільки одне з'єднання для кожного прикладної програми клієнта.

Методи мінімізації з'єднань із БД залежать від того середовища розробки, у якім написана прикладна програма-клієнт. Тому ви повинні ознайомитися з відповідної технічною документацією. Наприклад, для з'єднання з віддаленою БД у клієнтських додатках написаних в Delphi або C-Bilder використовується компонент TDatabase. Він служить для:

1) створення постійного з'єднання з БД;
2) створення локального псевдоніма БД;
3) зміни параметрів з'єднання, встановлених для псевдоніма БД;
4) управління трансакціями.

Якщо не використовувати компонент TDatabase, то з'єднання з БД буде встановлювати кожний компонент типу "набір даних" – Ttable (таблиця), TQuery (SQL – запит), TStoredProc (збережена процедура). Крім того, при з'єднанні з віддаленою БД "безпосередньо", з компонентів типу "набір даних", неможливо змінювати предустановлені параметри з'єднання.

**10.7.2. Зниження мережного трафіку** досягається за рахунок доставки в прикладні програми клієнтів мінімальної можливої кількості інформації із БД, якої достатньо для проведення поточних маніпуляцій з даними. Наприклад, із двох наборів даних, що реалізовані компонентами TTable і TQuery, для маніпулювання даними в прикладної програмі-клієнті рекомендується використовувати останній компонент. По-перше, при доступі до табличних даних компонент TTable зчитує всі записи віддаленої таблиці, у той час як TQuery – рівно стільки, скільки необхідно для поточних

цілей візуалізації, наприклад, для заповнення сітки TDBGrid. По-друге, при доступі до таблиць великого обсягу використання TTable може привести до істотних тимчасових затримок.

Компоненти TTable і TQuery мають різну природу: TTable орієнтований на навігаційний метод доступу до даних, що більш характерно для роботи з локальними СУБД, TQuery орієнтований на роботу з безліччю записів, що характерно при доступі до віддалених БД архітектури "клієнт-сервер". TTable дозволяє звернутися до одній таблиці БД, TQuery – до результатів виконання запиту одночасно з декількох таблиць БД. Відповідно, підтвердження змін даних в TTable здійснюється для кожного запису, що суттєво збільшує мережний трафік. Зміна даних при використанні TQuery може проводитися відразу над множиною записів операторами *INSERT*, *UPDATE*, *DELETE*.

Компонент TStoredProc використовується тільки для роботи з процедурами, що викликаються, й не застосовується для роботи із процедурами вибору, які також можуть повертати набори даних. Для роботи із процедурами вибору використовується компонент TQuery.

**10.7.3. Перенос ваги обчислювальної роботи на сервер,** по-перше, прискорює роботу прикладної програми-клієнта, а по-друге, мінімізує можливість виникнення помилок. От деякі рекомендації, що дозволяють це реалізувати.

1. Намагайтесь в додатку клієнта реалізовувати лише інтерфейс користувача, відсилання запитів до сервера і інтерпретацію отриманих даних.
2. Не звертайтеся до сервера із запитом необґрунтовано великого обсягу даних, на які доводиться накладати фільтри в прикладній програмі клієнта.
3. Максимально використовуйте можливості сервера, пам’ятаючи про те, що він може виконувати більшість вимог ніж прикладні програми, швидше й оптимальніше й, крім того, серверу не потрібно пересилати дані самому собі по мережі.
4. Реалізуйте обмеження на значення, що вводяться користувачем даних за допомогою апарата обмежень БД, а посилальну цілісність – за допомогою тригерів.
5. Запити, що вимагають для свого виконання алгоритмів, які розгалужуються або потребують циклічних дій, а також обчислення значень, що засновані на поточних даних із БД, реалізуйте за допомогою збережених процедур.
6. Бізнес-правила, пов'язані із трансакційними змінами таблиць, реалізовуйте за допомогою тригерів.
7. Одержуйте унікальні значення числових полів за допомогою генераторів.
8. Дії, які повторюються та можуть розділятися різними прикладними програмами й використовуватися в *SQL*-операторах, реалізуйте за допомогою функцій, що визначаються користувачем (UDF).

## 10.8. КОНТРОЛЬНІ ПИТАННЯ

1. Які напрямки оптимізації роботи з БД Вам відомі?
2. Як нормалізація відношень впливає на роботу БД?
3. У чому різниця теорії та практики взаємовпливу методів доступу до даних та їх структури?
4. Яким чином налаштування внутрішніх механізмів СУБД впливає на оптимізацію її роботи?
5. Дайте визначення та характеристику індексу як структури даних.
6. Яка різниця та що спільного між ключами та індексами?
7. Які можливості мають більшість діалектів SQL для створення та вилучення індексів з БД?
8. Які загальні дії виконує СУБД задля оптимізації виконання запитів?
9. У яких випадках рекомендується створювати індекси, а в яких ні?
10. Які особливості часткового використання складеного індексу?
11. Коли можна уникнути багатопотокового пошуку замінивши в умовах пошуку предикату приналежності до множини на влучання в діапазон?
12. Яким чином зменшують загальну кількість індексів у БД?
13. Що Вам відомо про разбалансування індексу?
14. У чому полягає перебудова індексу?
15. Що Вам відомо про показник „корисності” індексу?
16. Які можливості дає пропозиція PLAN оператору SELECT?
17. В чому полягають відомі Вам напрями оптимізації роботи прикладних програм-клієнтів?

*Вивчення цього матеріала допоможе студентові знаходити необхідні напрямки оптимізації роботи з БД.*

# 11. Захист БД від несанкціонованого доступу

*Цей розділ присвячен проблемам захисту інформації в БД. Детально розглядаються комп'ютерні і некомп'ютерні засоби контролю для захисту даних.*

## 11.1. Проблеми захисту БД

Поняття захисту торкаються не тільки даних, що зберігаються в БД [4]. Вразливість системи захисту може виникати і в інших її частинах, що, у свою чергу, наражає на загрозу й БД. Отже, захист повинен охоплювати увесь об'єкт інформаційної діяльності (організації) в цілому: устаткування, програмне забезпечення, що використовується, персонал і дані [2, 4, 6, 7, 21].

***Захист БД*** *– це відвертання будь-яких навмисних або ненавмисних загроз за допомогою різних комп'ютерних і некомп'ютерних засобів* [4].

***Загроза*** *– будь-які обставини або події, що можуть бути причиною порушення політики безпеки інформації і/або нанесення збитків організації* [22].

Загроза може бути викликана ситуацією або подією, що здатна принести шкоду організації, причиною якої може служити людина, подія або збіг обставин. Навмисні загрози завжди здійснюються людьми, що є як авторизованими, так і неавторизованими користувачами, які можуть не бути співробітниками організації. Будь-яка загроза повинна розглядатися як потенційна можливість порушення системи захисту, яка у випадку своєї реалізації може здійснити той або інший негативний вплив (табл. 11.1). Тривалість періоду бездіяльності й швидкість оновлення БД залежать від ряду факторів:

1. Чи є в системі резервне устаткування з розгорнутим програмним забезпеченням.
2. Коли востаннє виконувалося резервне копіювання системи.
3. Час, необхідний на оновлення системи.
4. Чи існує можливість оновлення й уведення в експлуатацію даних, які були зруйновані.

БД є найважливішим корпоративним ресурсом, який повинен бути належним чином захищений за допомогою відповідних засобів контролю, що визначаються конкретними вимогами, які випливають із особливостей експлуатування системи. Ми обговоримо проблеми захисту БД із погляду таких потенційних загроз:

1) викрадення і фальсифікація даних;
2) втрата конфіденційності (порушення таємниці);
3) порушення недоторканності особистих даних;
4) втрата цілісності;
5) втрата доступності.

Ці потенційні загрози визначають основні напрямки, у яких керівництво повинно вживати заходів, щодо зниження ступеню ризику,

тобто потенційну можливість втрати або ушкодження даних. У деяких ситуаціях усі відзначені аспекти ушкодження даних тісно пов'язані між собою. Так що дії, які спрямовані на порушення захищеності системи в одному напрямку, часто приводять до зниження її захищеності у всіх інших. Крім того, деякі події, наприклад, порушення недоторканності особистих даних або фальсифікація інформації, можуть виникнути внаслідок як навмисних, так і ненавмисних дій. І зовсім необов'язково будуть супроводжуватися якими-небудь змінами, що виявляються яким-небудь способом, в БД або системі.

***Викрадення й фальсифікація*** даних може відбуватися не тільки в середовищі БД – уся організація піддана цьому ризику. Однак дії по викраденню або фальсифікації інформації завжди організується людьми, тому основна увага повинна бути зосереджена на скороченні загальної кількості, зручних ситуацій для виконання таких дій. Викрадення й фальсифікація зовсім необов'язково пов'язані зі зміною яких-небудь даних, що справедливо й відносно втрати конфіденційності або порушення недоторканності особистих даних.

***Втрата конфіденційності й порушення недоторканності особистих даних*** приводить до втрати позицій у конкурентній боротьбі і юридичним наслідкам, що є результатами позовів на організацію, які подали постраждалі фізичні особи. Як правило, конфіденційними вважаються ті дані, які є критичними для всієї організації, тоді як поняття недоторканності даних стосується вимоги захисту інформації про окремих працівників.

Таблиця 11.1

**Деякі загрози, що можуть загрожувати БД.**

| Загроза | Викрадення і фальсифікація даних | Втрата конфіденційності | Порушення недоторканності особистих даних | Втрата цілісності | Втрата доступності |
|---|---|---|---|---|---|
| Використання прав доступу іншої людини | √ | √ | √ | | |
| Несанкціонована зміна або копіювання даних | √ | | | √ | |
| Зміна програм | √ | | | √ | √ |
| Необдумані методики й процедури, що допускають змішування конфіденційних і звичайних даних в одному документі | √ | √ | √ | | |
| Підключення до кабельних мереж | √ | √ | √ | | |
| Уведення хакерами некоректних даних | √ | √ | √ | | |
| Шантаж | √ | √ | √ | | |
| Створення «лазівок» у систему | √ | √ | √ | | |
| Викрадення даних, програм і устаткування | √ | √ | √ | | √ |
| Відмова системи захисту, що викликає перевищення припустимого рівня доступу | √ | √ | √ | | |
| Нестача персоналу й страйки | | | | √ | √ |
| Недостатня кваліфікація персоналу | | √ | √ | √ | √ |
| Перегляд і розкриття засекречених даних | √ | √ | √ | | |
| Електронні наведення й радіація | | | | √ | √ |

| | | | | | |
|---|---|---|---|---|---|
| Руйнування даних у результаті відключення або перенапруги в мережі електроживлення | | | | √ | √ |
| Пожежі (через короткі замикання, удари блискавки, підпали), повені, диверсії | | | | √ | √ |
| Фізичне ушкодження устаткування | | | | √ | √ |
| Обрив або від'єднання кабелів | | | | √ | √ |
| Впровадження комп'ютерних вірусів | | | | √ | √ |

***Втрата цілісності й втрата доступності даних*** приведе до викривлення або руйнування даних, що може мати самі серйозні наслідки для подальшої роботи організації.

***Втрата доступності даних*** – це коли дані або система або вони одночасно стануть недоступними користувачам, що може наразити на загрозу подальше існування організації. У деяких випадках ті події, які послужили причиною переходу системи в недоступний стан, можуть одночасно викликати й руйнування БД. На тепер безліч організацій функціонує в безперервному режимі, надаючи свої послуги клієнтам по 24 години на добу й по 7 днів на тиждень. Тому втрата доступності даних призводить до чималих збитків за рахунок відтоку платежеспроможних клієнтів.

## 11.2. Контрзаходи – комп'ютерні засоби контролю

Відносно загроз, що загрожують комп'ютерним системам, можуть бути прийняті контрзаходи різних типів, починаючи від фізичного спостереження й закінчуючи адміністративно-організаційними процедурами. Незважаючи на широкий діапазон комп'ютерних засобів контролю, що доступні у цей час на ринку, загальний рівень захищеності СУБД визначається можливостями операційної системи, яка використовується, оскільки робота цих двох компонентів тісно пов'язана між собою. Зазвичай застосовуються наступні засоби контролю: авторизація й аутентифікація користувачів, представлення даних, підтримка цілісності, резервне копіювання й оновлення БД, шифрування даних, допоміжні процедури.

**11.2.1. Авторизація –** *це надання повноважень, що дозволяють їхньому власникові мати законний доступ* до системи або до її об'єктів [4]; встановлення відповідності між повідомленням і його джерелом [22].

За надання користувачам доступу до комп'ютерної системи зазвичай відповідає системний адміністратор, в обов'язки якого входить створення їх облікових записів. Кожному користувачеві привласнюється унікальний ідентифікатор, який використовується операційною системою для визначення його власника. З кожним ідентифікатором пов'язується певний пароль, що обирається користувачем і відомий в операційній системі.

Засоби авторизації користувачів можуть бути вбудовані безпосередньо в програмне забезпечення та управляти не тільки правами доступу до системи або об'єктів, що надані користувачам, але й набором операцій, які користувачі можуть виконувати з кожним доступним їм

об'єктом. Із цієї причини механізм авторизації часто інакше називають підсистемою управління доступом. Термін "власник" у визначенні може представляти користувача-людину або програму. Термін "об'єкт" може представляти таблицю даних, представлення, прикладну програму, процедуру або будь-який інший об'єкт, що може бути створений у рамках системи.

Ще одним аспектом авторизації є розробка процедур, за допомогою яких конкретним користувачам надаються права доступу до різних об'єктів БД. Дуже важливо зберігати історичну інформацію про надання прав користувачам, особливо якщо їх службові обов'язки змінюються, і вони перестають потребувати доступу до певних масивів даних. Зокрема, якщо користувач звільняється, надзвичайно важливо негайно вилучити його обліковий запис і анулювати всі надані йому права доступу, що дозволить вчасно припинити будь-які можливі спроби порушення захисту системи.

**11.2.2 Аутентифікація –** процедура перевірки відповідності наданого ідентифікатора об’єкта комп’ютерної системи на предмет належності його цьому об’єкту; встановлення або підтвердження автентичності [22].

При реєстрації користувач повинен надавати системі свій пароль для виконання перевірки (аутентифікації) того, чи є він тим, за кого себе видає. Використання паролів є найпоширенішим методом аутентифікації користувачів. Однак цей підхід не дає абсолютної гарантії, що даний користувач є саме тим, за кого себе видає.

З погляду забезпечення необхідного рівня захисту дуже важливо, щоб усі паролі, що використовуються, трималися в секреті. У ході процедури реєстрації в системі значення пароля не повинно відображатися на екрані, а списки ідентифікаторів користувачів і їх паролів повинні зберігатися в системі в зашифрованому виді. Крім того, в організації повинен використовуватися деякий стандарт для вибору припустимих значень пароля. Наприклад, усі паролі повинні бути не менше встановленої довжини, обов'язково містити цифри й службові символи, а також підлягати заміні через установлений інтервал часу (скажемо, п'ять тижнів). Для виявлення слабких паролів у системі слід використовувати спеціальне програмне забезпечення. Наприклад, у якості пароля можуть використовуватися реальні імена або адреси користувачів, що неприпустимо. Крім того, спеціалізовані програми можуть застосовуватися для виявлення застарілих паролів.

**11.2.3. Механізм надання даних** являє собою потужний і гнучкий інструмент організації їх захисту. У результаті користувачі не будуть мати ніяких відомостей про існування будь-яких атрибутів або рядків даних, які недоступні через представлення, що перебувають у їхньому розпорядженні. Представлення може бути визначено на базі декількох

таблиць, після чого користувачеві будуть делеговані необхідні привілеї доступу до цього представлення, але не до базових таблиць. У цьому випадку використання представлення являє собою більш твердий механізм контролю доступу, ніж звичайне надання користувачеві тих або інших прав доступу до базових таблиць.

**11.2.4. Засоби підтримки цілісності даних** вносять певний вклад у загальну захищеність БД, оскільки запобігають переходу даних у неузгоджений стан і одержанню помилкових або некоректних результатів розрахунків.

**11.2.5. Резервне копіювання –** це процедура, яка періодично виконується для одержання копії БД, її файлу журналу та програм на носії, що зберігаються окремо від системи [4].

Визначено наступні види резервного копіювання:

1) повне резервування (*full backup*);

2) диференційоване резервування (*differential backup*);

3) додаткове резервування (*incremental backup*);

4) пофайлове резервування;

5) блочне інкрементальне копіювання (*block level incremental*).

Повне резервне копіювання рекомендується виконувати для БД об’єм інформації в яких незначний. Після того, як час на резервне копіювання досягає критичної величини, що робить неможливим його оперативне виконання використовують диференційоване або додаткове резервування. Різниця між ними полягає в тому, що диференційне резервування виконуються лише для частин БД, які змінювались з моменту останнього повного копіювання. Додаткове резервування виконуються для тих частин БД, що змінилися з моменту останнього резервування будь якого виду. Пофайлове копіювання БД, яка розглядається як набір файлів, виконуються засобами операційної системи. При блочному інкрементальному копіюванні виконуються посекторне копіювання простору жорсткого диску, де розмішені файли БД.

Будь-яка сучасна СУБД повинна надавати засоби резервного копіювання, що дозволяють відновлювати БД. Крім того, рекомендується створювати резервні копії БД та її файлу журналу з деякою встановленою періодичністю. Великі БД можуть повністю копіюватися раз у тиждень або навіть раз на місяць, але при цьому слід організувати обов'язкове інкрементне копіювання, яке потрібно виконувати з більш високою частотою. День і година його виконання повинні встановлюватися відповідальними особами. У випадку відмови, у результаті якого БД стає непридатною для подальшої експлуатації, резервна копія й зафіксована у файлі журналу оперативна інформація використовується для її оновлення до останнього погодженого стану.

Залежно від частоти внесення в систему змін, протягом однієї доби може виконуватися кілька копіювань. Створені копії повинні поміщатися в безпечне місце. Місце зберігання останніх копій повинно бути обладнане

як мінімум шафами, що не згорають. Крім того, бажано використовувати деяке зовнішнє сховище, у яке буде поміщатися другий екземпляр створених копій. Усі згадані деталі повинні знайти своє чітке відбиття в розроблених процедурах резервного копіювання, які повинні неухильно виконуватися обслуговуючим персоналом.

Процедури, що регламентують процеси створення резервних копій, визначаються типом і розмірами БД, а також тим набором відповідних інструментів, який надається СУБД. Ці процедури повинні включати необхідні етапи, на яких буде безпосередньо виконуватися створення резервної копії. У процедурах копіювання також може вказуватися, які ще частини системи (наприклад, прикладні програми), крім самих даних, повинні підлягати копіюванню.

***Ведення журналу*** *– це процедура створення й обслуговування файлу журналу, що містить відомості про всі зміни, внесені в БД із моменту створення останньої резервної копії, і призначеного для забезпечення ефективного оновлення системи у випадку її відмови* [4].

СУБД повинна представляти засоби ведення системного журналу, у якому будуть фіксуватися відомості про всі зміни стану БД у ході виконання поточних трансакцій, що необхідно для ефективного її оновлення у випадку відмови. Перевага використання подібного журналу полягає в тому, що у випадку порушення роботи або відмови СКБД БД можна буде відновити до останнього відомого погодженого стану, скориставшись її останньою резервною копією й оперативною інформацією, що втримується в журналі. Якщо в системі, що відмовила, функція ведення системного журналу не використовувалася, БД можна буде відновити тільки до того стану, який був зафіксоване в останній створеній резервній копії. Усі зміни, які були внесені в неї після створення останньої резервної копії, виявляться загубленими.

***Контрольна точка*** – це момент синхронізації між станом БД та станом журналу виконання трансакцій. У цей момент усі буфери примусово вивантажуються на обладнання вторинної пам'яті [4].

Сучасні СУБД, як правило, надають засоби створення контрольних точок, що дозволяють зафіксувати в БД серію змін, що виконані в останній момент. Механізм створення контрольних точок може використовуватися разом з веденням системного журналу, що дозволить підвищити ефективність процесу оновлення. У момент створення контрольної точки СУБД виконує дії, що забезпечують запис на диск усіх даних, які зберігалися в основній пам'яті машини, а також поміщає у файл журналу спеціальний запис щодо контрольної точки.

**11.2.6. Оновлення.** Як і процедури копіювання, процедури оновлення також повинні бути ретельно продумані й пророблені. Те, які саме процедури оновлення будуть виконуватися, повинно визначатися типом відкату, який мав місце (руйнування носія, відмова програмного забезпечення або відмова устаткування системи). Крім того, процедурами повинні враховуватися особливості методів оновлення, що

прийняті СУБД, яка використовується. У кожному разі розроблені процедури оновлення повинні бути ретельно протестовані, оскільки необхідно одержати повну гарантію того, що вони працюють вірно, ще до того, як відбудеться реальна відмова системи. В ідеалі, процедури оновлення повинні регулярно тестуватися з деяким інтервалом.

**11.2.7. Шифрування –** *це кодування даних з використанням спеціального алгоритму*, у результаті чого дані стають недоступними для читання будь-якою програмою, що не має ключа дешифрування [4].

Якщо в системі із БД утримується досить важлива конфіденційна інформація, то має сенс закодувати її з метою попередження можливої загрози несанкціонованого доступу із зовнішньої сторони (відносно СУБД). Деякі СУБД включають засоби шифрування, призначені для використання в подібних цілях. Підпрограми таких СУБД забезпечують санкціонований доступ до даних (після їхнього декодування), хоча це буде пов'язане з деяким зниженням продуктивності, що викликана необхідністю перекодування.

**11.2.8. Допоміжні процедури** повинні використовуватися разом з іншими механізмами захисту.

**11.2.8.1. Аудит** – *це допоміжна процедура, яка призначена для перевірки повноти залучення передбачених засобів управління й відповідності рівня захищеності БД вимогам, що встановлені* [4]. У ході виконання інспекції аудитори можуть ознайомитися з ручними процедурами, що використовуються, обстежити комп'ютерні системи й перевірити стан усієї наявної документації на дану систему. Зокрема, аудиторська перевірка передбачає контроль наступних використовуваних процедур і механізмів управління:

1. Підтримка точності даних, що вводяться.
2. Підтримка точності процедур обробки даних.
3. Запобігання появи й своєчасне виявлення помилок у процесі виконання програм.
4. Коректне тестування, документування й супровід програмних засобів, що розроблені.
5. Попередження несанкціонованої зміни програм.
6. Надання прав доступу і контроль над використанням програм.
7. Підтримка документації в актуальному стані.

Усі згадані процедури й засоби контролю повинні бути досить ефективними, а якщо ні, то вони повинні бути переглянуті. Для визначення активності використання БД аналізуються її файли журналу. Ці ж джерела можуть використовуватися для виявлення будь-яких незвичайних дій у системі. Регулярне проведення аудиторських перевірок, що доповнене постійним контролем умісту файлів журналу з метою з'ясування ненормальної активності в системі, дуже часто дозволяє вчасно виявити й припинити будь-які спроби порушення захисту.

**11.2.8.2. Установка нового прикладного програмного забезпечення** повинна супроводжуватися рядом регламентованих заходів. Нові прикладні програми, які розроблені власними силами або сторонніми організаціями, обов'язково слід ретельно протестувати, перш ніж ухвалювати рішення про їхнє розгортання й передачу до експлуатації. Якщо рівень тестування буде недостатнім, суттєво зростає ризик руйнування БД. Слід вважати гарною практикою виконання резервного копіювання БД безпосередньо перед здачею нового програмного забезпечення в експлуатацію. Крім того, у перший період експлуатації нової прикладної програми обов'язково слід організувати ретельне спостереження за функціонуванням системи. Окремим питанням, яке повинно бути погоджено зі сторонніми розроблювачами програм, є право власності на програмне забезпечення. Дана проблема повинна бути вирішена ще до початку розробки, причому це особливо важливо в тих випадках, коли існує імовірність, що згодом організації обов'язково буде потрібно вносити зміни в ці програми. Ризик, пов'язаний з подібною ситуацією, полягає в тому, що організація не буде мати юридичного права використовувати самостійно модернізоване програмне забезпечення, що створене сторонніми розроблювачами.

**11.2.8.3. Установка або модернізація системного програмного забезпечення** при вступі від розроблювача чергових пакетів змін входить в обов'язку адміністратора БД. У деяких випадках внесені зміни виявляються зовсім незначними і стосуються тільки невеликої частини модулів системи. Однак можливі ситуації, коли буде потрібна повна ревізія всієї встановленої системи. Як правило, кожний пакет змін, що поступає супроводжується друкованою або інтерактивною документацією, що містить докладні відомості про суть змін і їх призначення.

Результатом ознайомлення із супровідною документацією пакета повинен стати план дій по його установці. У цьому плані повинні бути відбиті будь-які зміни, які можуть вплинути на БД і прикладні програми, а також запропоновані шляхи по їхній реалізації. Однак які б зміни не потрібні були, усі вони повинні бути враховані й для кожного з них повинна бути дана оцінка часу виконання з урахуванням обсягу всього наявного програмного забезпечення. Головне завдання адміністратора БД – забезпечити плавний перехід від старої версії системи до нової.

У функціонуючій системі, яка повинна бути постійно доступна протягом усього робочого часу, установка будь-яких пакетів модернізації звичайно виконується у неробочий час. Наприклад, у вихідні дні. Безпосередньо перед виконанням модернізації повинна бути зроблена повна резервна копія існуючої системи на випадок можливої відмови. Потім виконується установка пакета модернізації, а в дані програми вносяться всі необхідні зміни, супроводжувані процедурами тестування. Тільки після повного завершення зазначених процедур система може бути запущена в роботу з використанням реальних даних.

### 11.3. Контрзаходи – некомп'ютерні засоби контролю

**11.3.1. Класифікація методів некомп'ютерного контролю** ділить їх на зовнішні й внутрішні. Некомп'ютерні засоби контролю включають такі методи, як вироблення обмежень, угод і інших адміністративних заходів, не пов'язаних з комп'ютерною підтримкою:

1) заходи забезпечення безпеки й планування захисту від непередбачених обставин;
2) контроль за персоналом;
3) захист приміщень і сховищ;
4) гарантійні угоди;
5) договори про супровід;
6) контроль над фізичним доступом.

**11.3.2. Заходи забезпечення безпеки й планування захисту** від непередбачених обставин є різними речами. Перше припускає вичерпне визначення засобів, за допомогою яких забезпечується захист обчислювальної системи даної організації. Друге полягає у визначенні методів, за допомогою яких буде підтримуватися функціонування організації у випадку виникнення будь-якої аварійної ситуації. Кожна організація повинна підготувати й реалізувати як перелік заходів забезпечення безпеки, так і план захисту від непередбачуваних обставин.

У документі по заходах безпеки рекомендується визначати наступне:

1) область ділових процесів організації, для якої вони встановлюються;
2) відповідальність і обов'язки окремих працівників;
3) дисциплінарні заходи, прийняті у випадку виявлення порушення встановлених обмежень;
4) процедури, які повинні обов'язково виконуватися.

План захисту від непередбачених обставин розробляється з метою докладного визначення послідовності дій, необхідних для виходу з різних незвичайних ситуацій, які не передбачені процедурами нормального функціонування системи, наприклад, у випадку пожежі або диверсії. У системі може існувати єдиний план захисту від непередбачених обставин, а може бути декілька – кожний по окремому напрямкові. Типовий план захисту від непередбачених обставин повинен включати такі елементи:

1) відомості про те, хто є головною відповідальною особою і як можна встановити з нею контакт;
2) інформація про те, хто й на якій підставі ухвалює рішення про виникнення надзвичайної ситуації;
3) технічні вимоги до передачі керування резервним службам.

У ці вимоги входять:

1) відомості про розташування альтернативних сайтів;

2) відомості про необхідне додаткове устаткування;

3) відомості про необхідність додаткових ліній зв'язку.

4) організаційні вимоги відносно персоналу, який здійснює передачу керування резервним службам;

5) відомості про наявність страховки на випадок даної конкретної ситуації.

Будь-який план захисту від невизначених обставин повинен періодично переглядатися й тестуватися на предмет його здійсненності.

**11.3.3. Захист приміщень, сховищ і контроль над персоналом** дозволяє мінімізувати ризик фізичного проникнення людей, що не працюють в організації, а також максимально звузити доступ до устаткування і даних співробітників самої організації до того мінімуму, який необхідний для нормального обслуговування СУБД і апаратного забезпечення сервера. Розроблювачі комерційних СУБД покладають усю відповідальність за ефективність керування системою на користувачів. Тому з погляду захисту системи винятково важливу роль відіграє відношення до справи і дії людей, які безпосередньо залучені у ці процеси.

Основне устаткування системи, включаючи принтери, якщо вони використовуються для друку конфіденційної інформації, повинні розміщатися в приміщенні, що замикається, з обмеженим доступом, який повинен бути наданий тільки основним фахівцям. Усе інше устаткування, особливо переносне, повинно бути надійно закріплене в місці розміщення й забезпечено сигналізацією. Однак умова тримати приміщення постійно замкненим може виявитися небажаної або нездійсненою, оскільки працівникам може бути потрібний постійний доступ до встановленого там устаткування.

Вище, при обслуговуванні питань резервного копіювання, вже згадувалося про необхідність мати захищене приміщення, призначене для зберігання носіїв інформації. Для будь-якої організації життєво необхідно мати подібне, надійно захищене місце, у якому буде зберігатися копії програм, резервні копії системи, інші архівні матеріали і документація. Бажано, щоб це приміщення перебувало на майданчику, який знаходиться за межами місця розташування основного устаткування системи. Усі носії інформації, включаючи документацію, диски й магнітні стрічки, повинні зберігатися у вогнетривких сейфах. Усе, що зберігається в такому приміщенні, повинно бути зареєстроване в спеціальному каталозі, із вказівкою дати поміщення носія або документа на зберігання. Періодичність, з якою новий матеріал буде поміщатися в подібний архів, повинна регламентуватися розробленими процедурами копіювання. Крім того, організації можуть використовувати й зовнішні

сховища, періодично доставляючи туди знову створені резервні копії й інші архівні матеріали, причому частота доставки також повинна визначатися встановленими нормами й процедурами. Існують незалежні компанії, що спеціалізуються на організації зовнішніх сховищ інформації. Вони надають свої послуги численним компаніям-клієнтам.

**11.3.4. Контроль над фізичним доступом** є невід'ємним методом досягнення захисту приміщень, сховищ і контролю над персоналом. Він ділиться на внутрішній і зовнішній контроль.

***Внутрішній контроль*** використовується усередині окремих будинків і призначений для керування тим, хто буде мати доступ у певні приміщення. Наприклад, найбільш «відповідальні» приміщення (скажемо, ті, у яких установлені основні комп'ютери) можуть бути обладнані системами вхідного контролю. У подібних системах можуть використовуватися всілякі принципи – наприклад, спеціальні ключі, картки, кодові замки або засоби вказівки пароля. Найбільш складні комплекси припускають використання відбитків пальців, знімків райдужної оболонки ока, фіксації особливостей голосу або почерку. Однак у цей час далеко не всі комерційні організації застосовують настільки витончені методи контролю – в основному через їхню високу вартість.

***Зовнішній контроль*** застосовується поза будовами й призначений для обмеження доступу на майданчик або в окремі будівлі. Для спостереження за територією може використовуватися спеціальна охорона, в обов'язки якої входить контроль входу/виходу персоналу й відвідувачів організації. Слід зазначити, що основним завданням будь-яких механізмів контролю над фізичним доступом є забезпечення їх ефективності, однак необхідна ефективність повинна досягатися без створення додаткових перешкод персоналу у виконанні їх службових обов'язків. А якщо ні, то зростає загроза пошуку персоналом усіляких шляхів обходу встановлених вимог.

**11.3.5. Гарантійні договори** - це юридичні угоди відносно програмного забезпечення, що вкладені між розроблювачами програм та клієнтами, на підставі яких деяка третя фірма забезпечує зберігання вихідного тексту програм, що розроблені для клієнта. Це одна з форм страховки клієнта на випадок, якщо компанія-розроблювач відійде від справ. У цьому випадку клієнт одержить право забрати вихідні тексти програм у третьої фірми, замість того щоб залишитися з прикладною програмою, що позбавлена усякого супроводу. Дана область юриспруденції вважається однією з тих, у яких дуже часто допускаються помилки й різні недооцінки. На думку деяких авторів, 95% усіх гарантійних договорів по програмному забезпеченню не досягає своєї початкової мети. Щоб уникнути помилок, при укладенні подібних угод необхідно ретельно продумувати наступне:

1) тип матеріалів, що поміщаються на зберігання;

2) процедури оновлення й підтримки актуальності цих матеріалів;

3) відомості про використання програмного забезпечення від сторонніх виробників;

4) необхідність проведення верифікації матеріалів, що зберігаються;

5) умови, при яких матеріали можуть бути передані клієнтові;

6) чітка регламентація процесу передачі матеріалів, що зберігаються.

**11.3.6. Договори про супровід** повинні бути укладені для всього використовуваного організацією устаткування й програмного забезпечення сторонньої розробки або виготовлення. Установлений інтервал очікування відповіді у випадку будь-якої відмови або помилки залежить від важливості елемента, що відмовив, для нормального функціонування усієї системи. Наприклад, якщо відбудеться відмова сервера БД, бажана негайна реакція обслуговуючої організації з метою якнайшвидшого повернення системи в працездатний стан. Однак у випадку відмови принтера договір цілком може передбачати усунення несправності протягом робочого дня або навіть двох – передбачається, що завжди знайдеться підходящий резервний принтер, який можна буде використовувати до усунення аварії. У деяких випадках договором може бути передбачена тимчасова заміна несправного устаткування на період ремонту технічних засобів, що відмовили.

### 11.4. Особливості захисту статистичних БД

Зазвичай статистичні БД використовуються для генерації статистичної інформації. Наприклад, для визначення усереднених і зведених показників різних наборів даних. Інформація окремих записів у статистичній БД повинна зберігатися конфіденційно і не повинна бути доступна користувачам. Основна проблема при роботі з подібними типами БД полягає у можливості використання відповідей на припустимі запити для одержання відповідей на заборонені запити. Для розв’язання цієї проблеми можуть використовуватися різні стратегії.

1. Запобігання виконання запитів, що працюють із незначною кількістю записів БД.

2. Випадкове додавання додаткових рядків до основного набору даних, що оброблений запитом. У результаті відповідь буде містити певну помилку й дозволить лише оцінити правильне значення.

3. Використання для відповідей на запити тільки випадкових наборів вихідних даних.

4. Збереження історичних відомостей про результати виконання запитів і відхилення будь-яких запитів, що використовують значну кількість вихідних даних, оброблених попереднім запитом.

## 11.5. Модель управління доступом користувачів до БД

### 11.5.1. Базові моделі управління доступом

**11.5.1.1. Мандатна модель** базується на засадах секретного документообертy, що використовується у державних установах [23]. В моделі рівні допуску, які призначаються кожному об’єкту і користувачу, чітко визначені та упорядковані по зростанню секретності. Діють два основних правила:

1. Користувач може читати тільки ті об’єкти, що мають допуск на вище за його.

2. Користувач може змінювати тільки ті об’єкти, що мають однаковий з ним рівень допуску.

Перше правило зрозуміле. Друге правило доповнює перше. Воно запобігає навмисному або ненавмисному розкриттю тайн для користувачів з нижчими рівнями допуску.

Така організація допуску до об’єктів має деякі недоліки:

1. Модель не має механізму обмеження допуску до об’єктів одного рівня секретності для випадку коли два користувача з одноковими рівнями допуску виконують різні обов’язки, та не мають знати щодо діяльності один до одного.

2. Суттєве обмеження комунікативних можливостей між користувачами, які мають різні рівні допуску. З одного боку Користувач не має зворотного зв’язку зі своїми колегами, що мають більш високий рівень допуску. Він ніколи не дізнається, чи отримали його інформацію, яку він виклав для них на своєму рівні доступу. З іншого боку коментарі користувачів ніколи не дійдуть до тих, хто має нижчий рівень доступу.

Тому на практиці мандатну модель використовують сумісно з іншими моделями.

**11.5.1.2. Дискреційна модель** базується на управління доступом, що реалізується шляхом явного надання повноважень користувачам або групам користувачів на проведення дій з кожним об’єктом системи.

Для наглядної формалізації надання повноважень використовують матрицю доступу. Строки матриці відповідають користувачам або їх групам, а стовпці – об’єктам, або групам об’єктів, до яких надається доступ. Кожна чарунка матриці містить набір прав, які відповідний користувач має по відношенню до відповідного об’єкту. Користувач, який створив об’єкт, як правило має усю повноту повноважень на дії, що можливо виконати над цим об'єктом.

В порівнянні з мандатною моделлю дискреційна модель дозволяє створити більш гнучку систему безпеки. З іншого боку адміністрування дискреційної моделі набагато складніше ніж адміністрування мандатної моделі. Це найбільш гостро проявляється при великій кількості користувачів та об’єктів, до яких треба надавати привілеї доступу.

Рішення цієї проблеми лежить в площині зменшення розмірності матриці доступу за рахунок групування користувачів, типізації об’єктів, до яких треба надавати доступ та використання типових наборів прав, які називають схемами доступу.

При використанні групування користувачів права надаються безпосередньо групам, а не конкретним користувачам. Причому один і той користувач може бути членом декількох груп. За рахунок такого підходу скорочується кількість строк матриці доступу. Типізація об’єктів доступу здійснюється за спільними ознаками, які дають можливість створювати групи типових об’єктів. Як і з користувачами об’єкти можуть входити до декількох груп. Це дозволяє скоротити кількість стовпців матриці доступу.

На практиці навіть при скороченні розмірності матриці доступу продумати політику безпеки, тобто грамотно розділити повноваження між користувачами системи, дуже складно. Методи та засоби, які дозволяють це зробити описані в спеціальній літературі по захисту інформації [1, 3, 4].

**11.5.1.3. Ролева модель** базується на групуванні низки операцій або дій в набір, який називають роллю. Кожен користувач системи грає в ній одну або декілька ролей. Його роль в системі співпадає з роллю, яку він виконує в організації. Для недопущення перетинання між операціями в межах декількох різних ролей введена ієрархічна залежність між ролями. Таким чином роль, яка є вищою за ієрархією, може включати до себе інші ролі (читай операції), що знаходяться на нижчих сходинках, та одну або декілька окремих операцій, які притаманні тільки цієї ролі.

Особливість рольової моделі заключається в тому, що у об’єктів системи немає хазяїна. Користувачу не можна надати доступ к об’єкту або низки об'єктів. Уся інформація розглядається як така, що належить організації у цілому. Тому адміністрування ролевої моделі суттєво простіше ніж дискреційної.

З простоти організації рольової моделі випливає ряд недоліків:

1. Прийняття ролі користувача по відношенню до всіх об'єктів системи призводить до необхідності розподілення сфер керування або впливу користувачів на бізнес процеси шляхом утворення низки однотипних ролей, які стосуються адміністративно самостійних одиниць організації. Таким чином ролей „керівник підрозділу” буде утворено стільки, скільки однотипних підрозділів нараховує структура організації. А в межах підрозділу могуть бути і інші ролі. Ще одним шляхом виходу із даної ситуації є розподіл системи на домени та використання окремої рольової моделі для кожного домену. Деякі автори такий підхід вважають більш задовільним.

2. Звуження застосовування рольової моделі до рамок систем, в яких не передбачено використання авторських матеріалів в документах.

Цей недолік випливає з відсутності можливості визначення хазяїна об’єкту.

3. Дії, які можуть виконувати користувачі, стосуються без винятку усіх об'єктів системи. Таким чином, наприклад, „вилучати” з системи будь який об’єкт може будь хто, незалежно від того створював або використовував він цей об’єкт чи ні. Теж саме стосується „утворення” та „корегування” об’єкту, що може стати несподіваним для колег, сферу діяльності яких це торкає безпосередньо. Вихід з цієї ситуації в рамках рольової моделі є диференціація елементарних операцій, в назву яких додається необхідний ідентифікатор об'єкту. Наприклад: „вилучення договору”, „вилучення накладної” і таке інше.

В деяких випадках недоліки рольової моделі компенсуються зовнішніми засобами. Наприклад, надання об’єктам системи певних властивостей: „хазяїн об'єкту”, „рівень секретності” і тому подібне. В такому випадку процедура перевірки розміщується в необхідному місті прикладної програми для перевірки інформації, що надається як модулем рольової безпеки так і об'єктом, над яким запрошено виконання поточної дії. Деякі бібліотеки рольової безпеки надають можливості написання сценаріїв перевірки виконання дій над об’єктам. Ці сценарії відрізняються від згаданих вище процедур перевірки містом розташування (вони зберігаються в настройках бібліотеки) та тим, що написані на іншому язику програмування.

### 11.5.2. Реалізація дискреційної моделі доступу до об’єктів БД

Стандарт язика SQL використовує дискреційну модель доступу до об'єктів БД. Як об’єкти захисту так і механізм захисту вбудовані безпосередньо в БД. СУБД дає можливості налаштування цього механізму. Користувачі БД регіструються на рівні СУБД, але інформація про них віддзеркалена безпосередньо в БД, де їм надані ті чи інші привілеї. ***Привілей доступу*** *– це можливість виконувати визначений вид дій над конкретним об'єктами БД*, яка надана користувачу [4]. Такий підхід дуже ускладнює можливість викрадання інформації з БД без знання імен на паролів користувачів методом її переносу в систему, де встановлений інший екземпляр СУБД.

Привілеї доступу можуть установлюватися як системним адміністратором (наприклад, за замовченням в Interbase це користувач з іменем SYSDBA), так і користувачем, якому системний адміністратор надав таке право. Наприклад, для БД «БІБЛІОТЕКА» це користувач із іменем S. Кожний об'єкт, що створений у середовищі SQL, має свого власника. Власник об'єкту є єдиною персоною, яка знає про його існування й має право виконувати із ним будь-які операції. Надання й скасування користувачам привілеїв в SQL реалізується операторами *GRANT* і *REVOKE* відповідно [10, 14, 15]. Оператор *GRANT* має наступний формат:

**GRANT** <Привілеї> **ON** [TABLE] {Ім'я таблиці | Ім'я представлення даних}

**TO** { <об'єкт> | <список користувачів> **[WITH GRANT OPTION]**

| **GROUP** ім'я групи користувачів **UNIX**}
| **EXECUTE ON PROCEDURE** Ім'я **TO**
{ < об'єкт > | < список користувачів >}
| < рольові привілеї> **TO** {**PUBLIC** | < список рольових привілеїв >}
**[WITH ADMIN OPTION],** де
< Привілеї > = {**ALL [PRIVILEGES**] | < список привілеїв >};
< список привілеїв > **= SELECT | DELETE | INSERT**
| **UPDATE** [( атр. [, атр. …])] | **REFERENCES** [(атр. [, атр. …])]
[, < список привілеїв > …];
< об'єкт > **= PROCEDURE** Ім'я
| **TRIGGER** Ім'я
| **VIEW** Ім'я
| **PUBLIC**
[, < об'єкт > …];
< список користувачів > = [**USER**] Ім'я користувачів
| ім'я ролі
| ім'я групи користувачів **UNIX**}
[, < список користувачів > …];
< рольові привілеї > = ім'я ролі[, ім'я ролі …];
< список рольових привілеїв > = [**USER**] Ім'я користувача
**[, [USER]** Ім'я користувача …].

Після пропозиції *GRANT* визначаються привілеї доступу з набору, установленого в стандарті ISO: *SELECT* – право вибирати дані з таблиці; *INSERT* – право вставляти в таблицю нові рядки; *UPDATE* – право змінювати дані в таблиці; *DELETE* – право вилучати рядки з таблиці; *REFERENCES* – право посилатися на стовпці зазначеної таблиці в описах вимог підтримки цілісності даних; *USAGE* – право використовувати домени, перевірки, набори символів і трансляції. Якщо користувачеві дається весь <список привілеїв>, то замість їхнього перерахування вказується пропозиція *ALL* [*PRIVILEGES*].

Після пропозиції *TO* обов'язково вказується список об'єктів або список користувачів, яким надаються привілеї доступу до таблиці або представлення даних, зазначеному після *ON* [*TABLE*] або дозволу на виконання збереженої процедури (*EXECUTE ON PROCEDURE* Ім'я). Список користувачів формується з імен, які зареєстровані в СУБД. Список об'єктів може містити збережені процедури, тригера й представлення даних, що визначені усередині БД. Слово *PUBLIC* надає привілей не тільки всім існуючим користувачам, але також і всім тим користувачам, які будуть визначені в БД згодом.

Фраза *WITH GRANT OPTION* дозволяє всім зазначеним у списку користувачам передавати іншим користувачам усі надані їм відносно зазначеного об'єкта привілеї. Якщо ці користувачі також передадуть власні повноваження іншим користувачам із вказівкою фрази *WITH*

*GRANT OPTION*, то останні, у свою чергу, також одержать право передавати свої повноваження іншим користувачам. Якщо ця фраза не буде зазначена, одержувач привілею не зможе передати свої права іншим користувачам. Таким чином, власник об'єкта може чітко контролювати, хто одержав право доступу до об'єкта і які повноваження йому надані.

Конструкція *GROUP* ім'я групи користувачів *UNIX* дає можливість давати привілеї доступу до таблиць і виставам даних користувачам, що визначені на рівні операційної системи *UNIX*. Ця опція додана винятково прикладних програм-клієнтів, що працюють під управлінням цієї операційної системи. Вона не є стандартною для SQL. Ім'я групи повинне бути визначене в */etc/group*. З іншими особливостями застосування оператора *GRANT* можна ознайомитися в документації СУБД.

Оператор *REVOKE* має наступний формат:

**REVOKE <**Привілеї**> ON [TABLE]** {Ім'я таблиці | Ім'я представлення даних}
**FROM** {<об'єкт> | <список користувачів>
**| GROUP** ім'я групи користувачів **UNIX**}
| **EXECUTE ON PROCEDURE** Ім'я **TO**
{<об'єкт> | <список користувачів>}
| <рольові привілеї> ТЕ {**PUBLIC**| <список рольових привілеїв>}
**| GRANT OPTION FOR** {<список користувачів>}

Усі параметри ідентичні по змісту параметрам оператору *GRANT*, за винятком параметра *GRANT OPTION FOR*, який вилучає право видачі привілеїв у користувачів зі <списку користувачів>. Привілей може ліквідувати тільки той, хто його надав. Якщо користувач втрачає привілеї певного виду, то призначені йому привілеї всіх інших видів залишаються в його власності.

Наявність операторів *CREATE ROLE* та *DROP ROLE* в СУБД не вказує на присутність елементів рольової моделі доступу до об'єктів БД (дивись < рольові привілеї > синтаксису оператора *GRANT*). Ці оператори використовують для утворення груп користувачів, які мають однакові повноваження. Це цілковито укладається в рамки дискреційної моделі. Формат операторів для вибраної вами СУБД треба дивитись у відповідній документації. Наприклад в СУБД Interbase він такий:

**CREATE ROLE** <ім'я ролі>;
**DROP ROLE** <ім'я ролі>.

## 11.6. КОНТРОЛЬНІ ПИТАННЯ:

1. Чи можливо обмежитись мірами щодо захисту БД, які стосуються тільки даних, що зберігаються?

2. Від яких факторів залежить швидкість оновлення БД після негативного впливу?

3. У чому полягають відомі вам потенційні загрози щодо БД?

4. Які комп'ютерні засоби контролю застосовуються для захисту БД?

5. Які комп'ютерні засоби контролю стосуються надання прав та визначення особи користувача?

6. Який механізм комп'ютерного контролю дає можливість обмежити доступ користувача до даних?

7. Які механізми захисту стосуються безпосередньо БД?

8. Які особливості застосування відомих вам допоміжних процедур, які відповідають за захист БД?

9. Як класифікують некомп'ютерні заходи контролю?

10. У чому різниця між заходами забезпечення безпеки й планування захисту від непередбачених обставин?

11. Які некомп'ютерні заходи стосуються захисту приміщень і контролю над персоналом?

12. Які юридичні заходи вам відомі, що забезпечують захист даних?

13. Які особливості захисту інформації статистичних БД?

14. Дайте визначення відомих вам базових моделей управління доступом?

15. Які особливості має мандатна модель управління доступом?

16. У якій моделі управління доступу і яким чином використовується матриця доступу?

17. Які прийоми використовуються для скорочення розмірності матриці доступу?

18. Які переваги та недоліки рольової моделі доступу вам відомі?

19. Яким чином компенсуються недоліки рольової моделі?

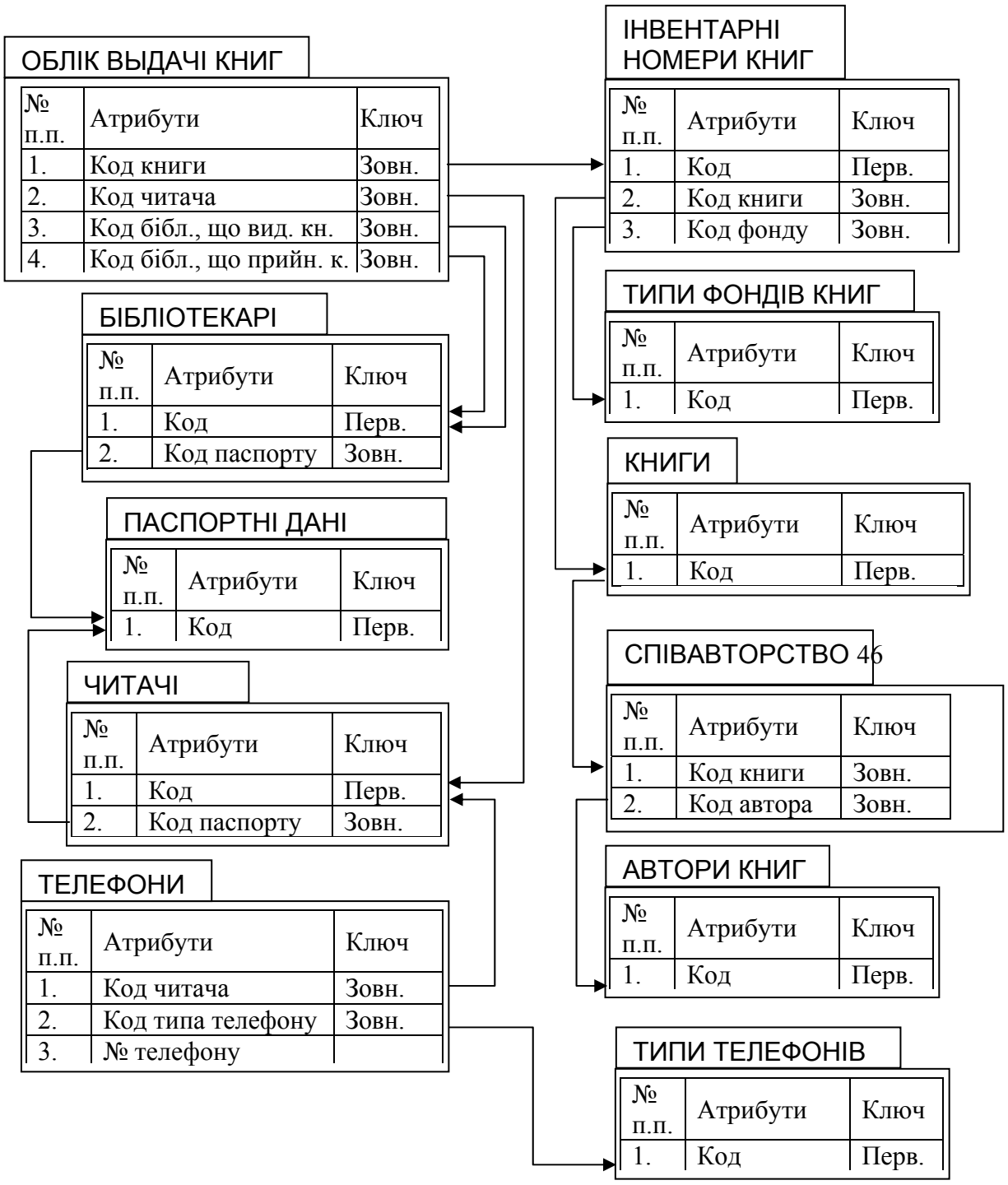
20. Які можливості установки та скасування привілеїв доступу на рівні СУБД вам відомі?

*Знайомство з цим матеріалом допоможе студентові на практиці організувати захист БД, що розробляється, від несанкціонованого доступу.*

# Література


1. Базы данных. Интеллектуальная обработка информации [Текст] : учеб. пособие / В.В.Корнеев, А.Ф.Гореев, С.В. Васютин, В.В.Райх. - 2-е изд. - М.: Издатель Молгачева С.В.: Нолидж, 2001. - 494 с.
2. Карпова Т.С. Базы данных: модели, разработка, реализация [Текст] / Т.С. Карпова - СПб: Питер, 2002. - 303с.
3. Чекалов А.П. Базы данных от проектирования до разработки приложений [Текст] / А.П. Чекалов - СПб.: БХВ-Петербург, 2003 - 380с.
4. Коннолли Т. Базы данных. Проектирование, реализация и сопровождение. Теория и практика. [Текст]: учебник/ Т. Коннолли , К. Бегг ;-2-е изд., испр. и доп. - М.: Изд. дом "Вильямс": 2001. - 1111с.
5. Гофман В.З Работа с базами данных в Delphi. [Текст]: учебник/ В.З. Гофман, А. Хомоненко : 2-е изд. - СПб.: БХВ Петербург, 2002. - 621с.
6. Дейт К. Дж. Введение в системы баз данных [Текст] / 6-е издание: Пер. с англ. - К.; М.; СПб.: Издательский дом "Вильямс", 1999 – 848 с.
7. Боуман, Д. Практическое руководство по SQL. [Текст]: учебное пособие/ Д. Боуман, С. Эмерсон, М. Дарновски. 3-е издание.: Пер. с англ. - К.: Диалектика, 1997. – 320 с.
8. Волфенгаген В.Э. Реляционные методы проектирования банков данных [Текст]: учебное пособие / В.Э. Волфенгаген, Л.Т. Кузин, В.И. Саркисян. Киев: Вища школа, 1979. - 192с.
9. Гайджорги В.І. Основи проектування та використання баз даних [Текст]: Навч. посібник / В.І. Гайджорги, О.А. Дацюк; Нац. техн. ун-т України "Київський політехн. ін-т". 2-ге вид, випр. і доп. - К.: Політехніка, 2004. - 254с.
10. Цикритзис, Дж. Модели данных. [Текст]: Учебное пособие / Цикритзис, Дж., Лоховски, Фр. / Пер. с англ. [и предисл.] О.М. Вейнерова. - М.: Финансы и статистика, 1985. - 343с.
11. Мейер, Д. Теория реляционных баз данных. [Текст]: учебное пособие / Д. Мейер, Пер. с англ. М.К. Валиева и др.; Под ред. М.Ш. Цаленко. - М.: Мир, 1987. - 608с.
12. Мещеряков Е.В. Публикация баз данных в Интернете. [Текст]: Учебное пособие / Е.В. Мещеряков, А.Д. Хоменко - М.: БХВ-Петербург, 2001. - VIII, 552 с.
13. Ковязин А.Н. Мир Interbase. Архитектура, администрирование и разработка приложений баз данных в Interbase/Firebird/Yaffil. [Текст]: Учебное пособие / А.Н. Ковязин , С.М. Востриков С.М. - 4-е издание. - КУДИЦ-ОБРАЗ, 2005 – 496 с.
14. Бондарь А.Г. Interbase и Firebird. Практическое руководство для умных пользователей и начинающих разработчиков (+CD). А.Г. Бондарь [Текст]: Учебное пособие / - СПб.: БХВ-Петербург, 2007 - 592с.
15. Заверач М.М. Бази даних: Лабораторний практикум: М.М. Заверач, В.В. Третько. [Текст]: Навч. посібник /Хмельницький: ТУП, 2003. - 210с.
16. Пирогов В.Ю. MS SQL Server 2000: Управление программирование. В.Ю. Пирогов [Текст]: Учебник / СПб: БХВ-Петербург, 2005 - 587с.
17. Велинг, Л. Разработка веб-приложений с помощью PHP и MySQL. Л. Велинг, Л. Томсон. [Текст]: Учебное пособие / 4-е изд.: Пер. с англ. - М.: ООО "И.Д.Вильямс", 2010. - 848 с.
18. Ален К. Oracle PL/SQL.101. К. Ален [Текст]: Учебник / М.: Лори. 2001. - XII, 350 с.
19. Лебедев А.Н. Visual FoxPro 9: Новейшая версия. [Текст]: Учебник / А.Н. Лебедев / М.: НТ Пресс, 2005 - 327с. - Самоучитель
20. Загальні положення щодо захисту інформації в комп'ютерних системах від несанкціонованого доступу // Департамент спеціальних телекомунікаційних систем та захисту інформації Служби безпеки України – Київ: 1999. – 19 с.
21. Термінологія в галузі захисту інформації в комп'ютерних системах від несанкціонованого доступу // Департамент спеціальних телекомунікаційних систем та захисту інформації Служби безпеки України – Київ: 1999. – 28 с.
22. Майоров А.В. Улучшенная ролевая модель управления доступом к объектам, http://blogs.byte-force.com/media/p/1147.aspx

# ДОДАТОК А. Діаграма зв'язків між відношеннями БД «БІБЛІОТЕКА».



# ДОДАТОК Б. Фізична модель БД «БІБЛІОТЕКА».

Таблиця Б.1

**Імена таблиць у БД LIBRARY.GBD**

| № | Найменування | Ім'я в LIBRARY.GBD |
|---|---|---|
| 1 | ЧИТАЧІ | Readers |
| 2 | БІБЛІОТЕКАРІ | Librarians |
| 3 | КНИГИ | Books |
| 4 | ПАСПОРТНІ ДАНІ | PasportData |
| 5 | ТЕЛЕФОНИ | Phones |
| 6 | АВТОРИ КНИГ | BookAuthors |
| 7 | СПІВАВТОРСТВО | CoAuthorship |
| 8 | ІНВЕНТАРНІ НОМЕРИ КНИГ | BookInventoryNumbers |
| 9 | ОБЛІК ВИДАЧІ КНИГ | BookGiveOutRecord |
| 10 | ТИПИ ФОНДІВ КНИГ | BookFunds |
| 11 | ТИПИ ТЕЛЕФОНІВ | PhoneTypes |

Таблиця Б.2

**Специфікація таблиці Readers (ЧИТАЧІ)**

| № | Заголовок | Ім'я поля | Тип | Довжина | Ключ |
|---|---|---|---|---|---|
| 1 | Код | Code | INTEGER | | P |
| 2 | Прізвище | FamilyName | CHAR | 30 | |
| 3 | Ім'я | Name | CHAR | 30 | |
| 4 | По батькові | Patronymic | CHAR | 30 | |
| 5 | № квитка читача | ReaderCardNumber | INTEGER | | |
| 6 | Код паспорта | PasportCode | INTEGER | | F |
| 7 | Місце осн. роботи | Job | CHAR | 60 | |
| 8 | Посада | Post | CHAR | 30 | |
| 9 | Примітки | Note | BLOB | | |

Таблиця Б.3

**Специфікація таблиці Librarians (БІБЛІОТЕКАРІ)**

| № | Заголовок | Ім'я поля | Тип | Довжина | Ключ |
|---|---|---|---|---|---|
| 1 | Код | Code | INTEGER | | P |
| 2 | Табельний номер | ClockNumber | INTEGER | | |
| 3 | Прізвище | FamilyName | CHAR | 30 | |
| 4 | Ім'я | Name | CHAR | 30 | |
| 5 | По батькові | Patronymic | CHAR | 30 | |
| 6 | Код паспорта | PasportCode | INTEGER | | F |
| 7 | Посада | Post | CHAR | 30 | |
| 8 | Домашній телефон | HomePhone | CHAR | 7 | |
| 9 | Примітки | Note | BLOB | | |

Таблиця Б.4

**Специфікація таблиці Books (КНИГИ)**

| № | Заголовок | Ім'я поля | Тип | Довжина | Ключ |
|---|---|---|---|---|---|
| 1 | Код | Code | INTEGER | | P |
| 2 | Назва | Name | CHAR | 200 | |
| 3 | Рік видання | IssueYear | DATA | | |
| 4 | Тираж | Drawing | INTEGER | | |
| 5 | УДК | UDK | CHAR | 20 | |
| 6 | Шифр | Cipher | CHAR | 10 | |
| 7 | Примітки | Note | BLOB | | |

Таблиця Б.5

**Специфікація таблиці PasportData (ПАСПОРТНІ ДАНІ)**

| № | Заголовок | Ім'я поля | Тип | Довжина | Ключ |
|---|---|---|---|---|---|
| 1 | Код | Code | INTEGER | | P |
| 2 | Серія паспорта | Series | CHAR | 2 | |
| 3 | № паспорта | Number | INTEGER | | |
| 4 | Дата народження | Birthday | DATE | | |
| 5 | Місце народження | Birthplace | CHAR | 30 | |
| 6 | Стать | Sex | CHAR | 1 | |
| 7 | Місце видачі паспорта | IssuePlace | CHAR | 100 | |
| 8 | Дата видачі паспорта | IssueDate | DATE | | |
| 9 | Прописка | Note | BLOB | | |

Таблиця Б.6

**Специфікація таблиці Phones (ТЕЛЕФОНИ)**

| № | Заголовок | Ім'я поля | Тип | Довжина | Ключ |
|---|---|---|---|---|---|
| 1 | Код читача | ReaderCode | INTEGER | | F |
| 2 | Код типу телефону | PhoneTypeCode | INTEGER | | F |
| 3 | № телефону | PhoneNumber | CHAR | 20 | |

Таблиця Б.7

**Таблиця BookInventoryNumbers (ІНВЕНТАРНІ НОМЕРИ КНИГ)**

| № | Заголовок | Ім'я поля | Тип | Довжина | Ключ |
|---|---|---|---|---|---|
| 1 | Код | Code | INTEGER | | P |
| 2 | Код книги | BookCode | INTEGER | | F |
| 3 | Код фонду | FundCode | INTEGER | | F |
| 4 | Інвентарний номер | InventoryNumber | INTEGER | | |
| 5 | Вартість | Cost | FLOAT | | |

Таблиця Б.8

**Специфікація таблиці BookAuthors (АВТОРИ КНИГ)**

| № | Заголовок | Ім'я поля | Тип | Довжина | Ключ |
|---|---|---|---|---|---|
| 1 | Код | Code | INTEGER | | P |
| 2 | Прізвище | FamilyName | CHAR | 30 | |
| 3 | Ім'я | Name | CHAR | 30 | |
| 4 | По батькові | Patronymic | CHAR | 30 | |
| 5 | Дата народження | Birthday | DATE | | |
| 6 | Дата смерті | DeatheDay | DATE | | |
| 7 | Коротка біографія | ShortBiography | BLOB | | |
| 8 | Примітки | Note | BLOB | | |

Таблиця Б.9

**Специфікація таблиці BookFunds (ТИПИ ФОНДІВ КНИГ)**

| № | Заголовок | Ім'я поля | Тип | Довжина | Ключ |
|---|---|---|---|---|---|
| 1 | Код | Code | INTEGER | | P |
| 2 | Найменування | Name | CHAR | 20 | |

Таблиця Б.10

**Специфікація таблиці PhoneTypes (ТИПИ ТЕЛЕФОНІВ)**

| № | Заголовок | Ім'я поля | Тип | Довжина | Ключ |
|---|---|---|---|---|---|
| 1 | Код | Code | INTEGER | | P |
| 2 | Найменування | Name | CHAR | 20 | |

Таблиця Б.11

**Специфікація таблиці CoAuthorship (СПІВАВТОРСТВО)**

| № | Заголовок | Ім'я поля | Тип | Довжина | Ключ |
|---|---|---|---|---|---|
| 1 | Код книги | BookCode | INTEGER | | F |
| 2 | Код автора | AuthorCode | INTEGER | | F |

Таблиця Б.12

**Специфікація таблиці BookGiveOutRecord (ОБЛІК ВИДАЧІ КНИГ)**

| № | Заголовок | Ім'я поля | Тип | Довж. | Ключ |
|---|---|---|---|---|---|
| 1 | Код | Code | INTEGER | | P |
| 2 | Код читача | ReaderCode | INTEGER | | F |
| 3 | Код бібл., вид. книгу | OutLibrarianCode | INTEGER | | F |
| 4 | Код інвентарного № книги | InventoryCode | INTEGER | | F |
| 5 | Дата видачі | IssueDate | DATE | | |
| 6 | Дата повернення | ReturnDate | DATE | | |
| 7 | Фактична дата повернення | FactReturnDate | DATE | | |
| 8 | Код бібліотекаря, що прийняв книгу | InLibrarianCode | INTEGER | | F |

# ДОДАТОК В. Інформація БД «БІБЛІОТЕКА».

Таблиця В.1

**Інформація в таблиці Readers**

| Code | FamilyName | Name | Patronymic | Readercard Number | PasportCode | Job | Post | Note |
|---|---|---|---|---|---|---|---|---|
| 1 | Іванов | Петро | Іванович | 317 | 4 | НГУ, каф. ЕОМ | Асистент | Blob |
| 2 | Федорец | Ірина | Олегівна | 28 | 1 | НГУ, АХЧ | Вахтер | Blob |
| 3 | Ільїн | Іван | Петрович | 1345 | 11 | НГУ, каф. фізики | Доцент | Blob |
| 4 | Суренко | Дмитро | Павлович | 543 | 6 | НГУ, каф. геофізики | Ст. викладач | Blob |
| 5 | Коршунова | Наталя | Юріївна | 128 | 8 | НГУ, каф. геоінформатики | Асистент | Blob |
| 6 | Носенко | Олег | Владими-Рович | 5672 | 5 | НГУ, ИКК | Інженер | Blob |
| 7 | Брусов | Владимир | Михайлович | 485 | 24 | НГУ, каф. геодезии | Лаборант | Blob |
| 8 | Козирєв | Олексій | Сергійович | 759 | 15 | НГУ, каф. кримінології | Професор | Blob |
| 9 | Левченко | Юлія | Павлівна | 146 | 18 | НГУ, каф. політичної теорії | Завідувач кафедри | Blob |
| 10 | Світла | Тетяна | Іванівна | 2021 | 22 | НГУ, каф. перекладу | Ст. препо-даватьль | Blob |
| 11 | Щиглів | Петро | Євгенович | 997 | 14 | НГУ, каф. електропостачання | Асистент | Blob |
| 12 | Кириленко | Віктор | Олександрович | 1010 | 17 | НГУ, каф. електропривода | Заступник декана | Blob |

Таблиця В.2

**Інформація в таблиці Librarians**

| Code | Clock Number | FamilyName | Name | Patronymic | Pasport Code | Post | HomePhone | Note |
|---|---|---|---|---|---|---|---|---|
| 1 | 28 | Іванова | Олена | Владимирівна | 2 | Бібліотекар | 52-07-75 | Blob |
| 2 | 12 | Ніколаєнко | Любов | Миколаївна | 10 | Бібліотекар | 46-32-19 | Blob |
| 3 | 187 | Иноземцева | Іванна | Модестівна | 9 | Ст. бібліотекар | 775-34-00 | Blob |
| 4 | 83 | Мальцева | Діана | Петрівна | 12 | Бібліотекар | 29-06-15 | Blob |
| 5 | 10 | Сызранцева | Тетяна | Ігорівна | 3 | Бібліотекар | 370-98-22 | Blob |
| 6 | 100 | Ставка | Лілія | Іванівна | 7 | Бібліотекар | 22-00-01 | Blob |
| 7 | 50 | Лещенко | Алла | Федорівна | 13 | Бібліотекар | 722-36-36 | Blob |
| 8 | 36 | Сіра | Лідія | Іванівна | 19 | Бібліотекар | 254-78-02 | Blob |
| 9 | 45 | Прохіна | Тамара | Львівна | 21 | Бібліотекар | 63-05-01 | Blob |
| 10 | 78 | Самійленко | Вікторія | Ігорівна | 20 | Бібліотекар | 125-45-80 | Blob |
| 11 | 69 | Степанова | Олександра | Миколаївна | 16 | Ст. бібліотекар | 445-45-65 | Blob |
| 12 | 17 | Петрова | Алина | Кузина | 23 | Бібліотекар | 999-12-05 | Blob |

Таблиця В.3

**Інформація в таблиці Books**

| Code | Name | Issue Year | Drawing | UDK | Cipher | Note |
|---|---|---|---|---|---|---|
| 1 | Автоматизація виробничих процесів на збагачувальній фабриці | 1985 | «Надра» | NULL | 622.7-52/Т | Blob |
| 2 | Розв'язок завдань по автоматизації процесів збагачення й металургії | 1969 | «Наука» | NULL | 622.7-52(075)/Т | Blob |
| 3 | Асимптотичні методи оптимального керування | 1987 | «Автомат» | NULL | 681.513.5:/А | Blob |
| 4 | Синтез оптимальних автоматичних систем | 1984 | «Автомат» | NULL | 681.513.5:/ ДО | Blob |
| 5 | Методи оптимізації стохастичних систем | 1987 | «Матстат» | NULL | 681.513.5:/ ДО | Blob |
| 6 | Автоматизовані системи керування технологічним процесом збагачення руди | 1987 | «Автомат» | NULL | 622.7-52/П | Blob |
| 7 | C/C++ Програмування мовою високого рівня | 2007 | «Пітер» | NULL | 681.3.06(075) | Blob |
| 8 | Комп'ютерні мережі. Принципи, технології, протоколи. | 2006 | «Пітер» | NULL | 004.72(075) | Blob |
| 9 | Довідник по диференціальних рівняннях із частками похідними першого порядку | 2003 | «ФИЗМАТ ЛИТ» | NULL | 517.9 | Blob |
| 10 | Теорія ймовірностей й математична статистика. | 2004 | «Пітер» | NULL | 519.2 | Blob |
| 11 | C#. Програмування мовою високого рівня | 2009 | «Пітер» | NULL | 004.43 | Blob |
| 12 | Теорія імовірностей й математична статистика | 2005 | «Вища школа» | NULL | 519.2 | Blob |
| 13 | Теорія імовірностей й математична статистика | 2002 | «ФИЗМАТ ЛИТ» | NULL | 519.2 | Blob |
| 14 | Дискретно-групові методи інтегрування звичайних диференціальних рівнянь | 1991 | «ЛИИАН» | NULL | 517.9-37 | Blob |

Таблиця В.4

**Інформація в таблиці BookAuthors**

| Code | FamilyName | Name | Patronymic | Birthday | Deathed ay | Short Biography | Note |
|---|---|---|---|---|---|---|---|
| 1 | Тихонов | Олег | Миколайович | NULL | NULL | Blob | Blob |
| 2 | Акуленко | Леонід | Дмитрович | NULL | NULL | Blob | Blob |
| 3 | Процуто | Віктор | Сергійович | NULL | NULL | Blob | Blob |
| 4 | Колосов | Геннадій | Євгенович | NULL | NULL | Blob | Blob |
| 5 | Козаков | Ігор | Єлисейович | NULL | NULL | Blob | Blob |
| 6 | Павловська | Тетяна | Олександрівна | NULL | NULL | Blob | Blob |
| 7 | Зайцев | Валентин | Федорович | NULL | NULL | Blob | Blob |
| 8 | Полянин | Андрій | Дмитрович | NULL | NULL | Blob | Blob |
| 9 | Андронов | Олександр | Михайлович | NULL | NULL | Blob | Blob |
| 10 | Копитов | Євгеній | Олександрович | NULL | NULL | Blob | Blob |
| 11 | Гринглаз | Леонід | Якович | NULL | NULL | Blob | Blob |
| 12 | Баврін | Іван | Іванович | NULL | NULL | Blob | Blob |
| 13 | Пугачов | Владимир | Семенович | NULL | NULL | Blob | Blob |
| 14 | Оліфер | Віктор | Григорович | NULL | NULL | Blob | Blob |
| 15 | Оліфер | Наталя | Олексіївна | NULL | NULL | Blob | Blob |
| 16 | Флегонтов | Олександр | Володимирович | NULL | NULL | Blob | Blob |

Таблиця В.5

**Інформація в таблиці Pasportdata**

| Code | Series | Number | Birthday | Birthplace | Sex | Issueplace | Issuedate | Note |
|---|---|---|---|---|---|---|---|---|
| 1 | АА | 45003 | 30.05.1930 | Росія, г. Опочки | Ж | Дніпропетровськ | 12.01.1995 | Blob |
| 2 | АА | 15700 | 23.02.1930 | Росія. м. Володимир | Ж | Житомир | 16.03.2000 | Blob |
| 3 | АБ | 87134 | 20.01.1963 | Дніпропетровська область, село Солоне | Ж | Дніпропетровська область, село Солоне | 10.01.1998 | Blob |
| 4 | АЕ | 12300 | 12.11.1960 | Україна, м. Донецьк | Ч | Донецьк | 15.12.1991 | Blob |
| 5 | АЕ | 01067 | 19.07.1981 | Україна, Дніпропетровськ | Ч | Дніпропетровськ | 25.08.1997 | Blob |
| 6 | АЖ | 01568 | 14.09.1956 | Казахстан, місто Павлодар | Ч | Київ | 24.05.1999 | Blob |
| 7 | АЗ | 43188 | 13.11.1970 | Дніпропетровська область, м. Дніпродзержинськ | Ж | Дніпропетровська область, м. Дніпродзержинськ | 15.05.1998 | Blob |
| 8 | АК | 23490 | 05.01.1961 | Росія, місто Самара | Ж | Дніпропетровськ | 13.09.2000 | Blob |
| 9 | АС | 90843 | 10.10.1949 | Молдова, місто Кишинів | Ж | Дніпропетровськ | 13.12.1998 | Blob |
| 10 | АЯ | 90764 | 14.11.1950 | Україна, місто Миколаїв | Ж | м. Миколаїв | 11.11.1998 | Blob |
| 11 | ІК | 10842 | 19.07.1949 | Україна, м. Кіровоград | Ч | Дніпропетровськ | 6.01.1998 | Blob |
| 12 | ІК | 45190 | 18.07.1983 | Дніпропетровська область, село Петропавлівка | Ж | Дніпропетровська область, село Петропавлівка | 20.09.1999 | Blob |
| 13 | АН | 61327 | 1.10.1960 | Росія, Санкт-Петербург | Ж | Санкт-Петербург | 12.10.1976 | Blob |
| 14 | АН | 64277 | 23.12.1972 | Україна, м. Львів | Ч | Левів | 6.01.1988 | Blob |
| 15 | АК | 89125 | 7.05.1980 | Україна, м. Київ | Ч | Київ | 10.01.1998 | Blob |
| 16 | АК | 55706 | 07.04.1965 | Донецьк | Ж | Донецьк | 20.04.1982 | Blob |
| 17 | АС | 73271 | 5.07.1950 | Крим | Ч | Сімферополь | 23.08.1970 | Blob |
| 18 | АЖ | 45879 | 4.02.1961 | Дніпропетровськ | Ж | Дніпродзержинськ | 14.03.1980 | Blob |
| 19 | АС | 12548 | 8.04.1974 | Трускавець | Ж | Прикарпаття | 28.05.1989 | Blob |
| 20 | АК | 12578 | 11.11.1987 | Донецьк, Краматорськ | Ж | Київ | 26.01.2000 | Blob |
| 21 | АС | 55489 | 25.09.1981 | Суми | Ж | Харків | 6.11.1999 | Blob |
| 22 | АЯ | 45789 | 7.8.1972 | Угорщина | Ж | Івано-Франківськ | 3.10.1988 | Blob |
| 23 | АЖ | 35126 | 18.03.1975 | Одеса | Ж | Одеса | 19.6.1993 | Blob |
| 24 | АН | 15625 | 19.06.1966 | Дніпропетровськ | Ч | Дніпропетровськ, Петриковка | 12.08.1982 | Blob |

Таблиця В.6

**Інформація в таблиці BookInventoryNumbers**

| Code | BookCode | FundCode | InventoryNumber | Cost |
|---|---|---|---|---|
| 1 | 1 | 1 | 4567890 | 15,56 |
| 2 | 2 | 1 | 4510000 | 22,33 |
| 3 | 3 | 1 | 4532477 | 34,01 |
| 4 | 4 | 1 | 4512890 | 12,99 |
| 5 | 5 | 2 | 4678532 | 56,78 |
| 6 | 6 | 2 | 4632112 | 10,10 |
| 7 | 7 | 2 | 7569832 | 73,50 |
| 8 | 7 | 2 | 5478956 | 45,10 |
| 9 | 8 | 2 | 2145876 | 59,25 |
| 10 | 9 | 1 | 5214786 | 36,05 |
| 11 | 10 | 1 | 5268933 | 74,20 |
| 12 | 11 | 2 | 7865890 | 21,32 |
| 13 | 12 | 1 | 6589321 | 36,69 |
| 14 | 13 | 1 | 7812639 | 48,13 |
| 15 | 14 | 1 | 7523690 | 27,99 |

Таблиця В.7

**Інформація в таблиці Phones**

| ReaderCode | PhoneTypeCode | PhoneNumber |
|---|---|---|
| 1 | 1 | 29-06-15 |
| 1 | 2 | 98-78-88 |
| 1 | 3 | 380531987987 |
| 2 | 2 | 47-77-10 |
| 3 | 1 | 68-03-09 |
| 4 | 1 | 370-10-20 |
| 4 | 3 | 380975678954 |
| 5 | 1 | 744-33-00 |
| 6 | 1 | 33-34-35 |
| 6 | 3 | 380962314783 |
| 8 | 1 | 68-03-58 |
| 8 | 2 | 47-71-45 |
| 8 | 3 | 380632577788 |
| 9 | 1 | 144-98-48 |
| 9 | 2 | 32-02-02 |
| 9 | 3 | 380975550022 |
| 10 | 1 | 56-66-01 |
| 10 | 2 | 89-08-98 |
| 10 | 3 | 380534566552 |
| 11 | 1 | 789-45-97 |
| 11 | 2 | 47-77-96 |
| 11 | 3 | 380540221584 |
| 12 | 1 | 777-45-45 |
| 12 | 2 | 41-41-39 |
| 12 | 3 | 380674545121 |

Таблиця В.8

**Інформація в таблиці BookGiveOutRecord**

| Code | ReaderCode | OutLlibrarianCode | Inventary Code | IssueDate | ReturnDate | Factreturn Date | InLibrarian Code |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 4 | 6 | 11.09.2004 | 25.09.2004 | 24.09.2004 | 3 |
| 2 | 3 | 4 | 4 | 02.09.2004 | 16.09.2004 | 11.12.2004 | 3 |
| 3 | 6 | 4 | 3 | 02.09.2004 | 16.09.2004 | 16.09.2004 | 1 |
| 4 | 4 | 3 | 6 | 30.10.2004 | 13.11.2004 | 10.01.2005 | 6 |
| 5 | 7 | 10 | 7 | 10.11.2009 | 24.11.2009 | 24.11.2009 | 12 |
| 6 | 9 | 7 | 12 | 15.12.2009 | 29.12.2009 | NULL | 9 |
| 7 | 11 | 8 | 10 | 06.02.2009 | 20.02.2009 | 19.02.2009 | 7 |
| 8 | 7 | 9 | 8 | 07.03.2009 | 21.03.2009 | 10.04.2009 | 10 |
| 9 | 9 | 8 | 12 | 05.02.2010 | 29.02.2010 | NULL | 11 |
| 10 | 12 | 10 | 15 | 21.09.2010 | 05.10.2010 | 03.10.2010 | 9 |

Таблиця В.9

**Інформація в таблиці BookFunds**

| Code | Name |
|---|---|
| 1 | НТБ |
| 2 | Студентський |

Таблиця В.10

**Інформація в таблиці Phonetypes**

| Code | Name |
|---|---|
| 1 | Домашній |
| 2 | Робочий |
| 3 | Мобільний |
| 4 | Супутниковий |

Таблиця В.11

**Інформація в таблиці Coauthorship**

| BookCode | AuthorCode |
|---|---|
| 1 | 1 |
| 2 | 1 |
| 3 | 2 |
| 4 | 4 |
| 5 | 5 |
| 6 | 3 |
| 7 | 6 |
| 8 | 14 |
| 8 | 15 |
| 9 | 7 |
| 9 | 8 |
| 10 | 9 |
| 10 | 10 |
| 10 | 11 |
| 11 | 6 |
| 12 | 12 |
| 13 | 13 |
| 14 | 16 |
| 14 | 7 |

# Предметний покажчик

Навчальне видання

**Куваєв** Ярослав Геннадійович
**Жукова** Олена Андріївна
**Сечкін** Ігор Арнольдович

# Організація реляційних баз даних

Навчальний посібник

Видання друге, доповнене та перероблене

Видано в редакції авторів

Підписано до друку 20.09. 2017. Формат 30х42/2.
Папір офсетний. Ризографія. Ум. друк. арк. 8,8.
Обл.-вид. арк. 8,8. Тираж 35 пр. Зам. №

Підготовлено до друку та видрукувано
у Державному вищому навчальному закладі
«Національний гірничий університет»
Свідоцтво про внесення до Державного реєстру ДК № 1842 від 11.06.2004..

49005, м. Дніпро, просп. Д. Яворницького, 19