

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ
ТЕХНІЧНИЙ УНІВЕРСИТЕТ

С. К. Корнієнко

ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ АВТОМАТИЗОВАНИХ СИСТЕМ

*Рекомендовано Міністерством освіти і науки,
молоді та спорту України як навчальний посібник
для студентів вищих навчальних закладів, які навчаються за
напрямом підготовки «Програмна інженерія»*

*Посібник розроблено відповідно до плану проекту
«TEMPUS-PROMENG»
«Practice oriented master programmes
in engineering in RU, UA,UZ»
Проект фінансується при підтримці Європейської комісії*

Запоріжжя • ЗНТУ • 2015

УДК 004.4.001.2(075.8)
ББК 32.97я73
К67

*Розповсюдження та тиражування без
офіційного дозволу видавництва заборонено*



*Гриф надано Міністерством освіти і науки, молоді та спорту України
(лист №1/11-5574 від 18 березня 2013 року)*

Рецензенти:

- С. І. Гоменюк* – д-р техн. наук, професор, декан Запорізького національного університету;
О. Д. Шамровський – д-р фіз.-мат. наук, професор Запорізького державної інженерної академії;
В. Є. Бахрушин – д-р фіз.-мат. наук, професор, зав. кафедри Класичного приватного університету, академік АН вищої школи України

Корнієнко С. К.

К 67 Проектування інформаційного забезпечення автоматизованих систем : навч. посіб. – Запоріжжя : ЗНТУ, 2015. – 210 с.; іл.

ISBN 978-617-529-096-2

У навчальному посібнику розглянуті питання проектування інформаційного забезпечення автоматизованих систем; архітектури та функціонування СКБД; використання технології клієнт-сервер; розподілені бази даних; інструментальні засоби та методика концептуального проектування, логічного проектування реляційних баз даних; адміністрування баз даних; використання мови SQL; захист інформації в системах баз даних.

Для студентів вищих навчальних закладів професійного спрямування «Програмна інженерія». Може бути корисним розробникам і користувачам баз даних.

Посібник розроблено відповідно до плану проекту «TEMPUS-PROMENG» «Practice oriented master programmes in engineering in RU, UA, UZ». Проект фінансується при підтримці Європейської комісії.

Зміст матеріалу відображає думку авторів та Європейська комісія не несе відповідальності за використання інформації що міститься в навчальному посібнику.

УДК 004.4.001.2(075.8)
ББК 32.97я73

ISBN 978-617-529-096-2

© Запорізький національний
технічний університет (ЗНТУ), 2015
© С. К. Корнієнко 2015

ЗМІСТ

Словник скорочень	6
Вступ.....	7
1 МІСЦЕ ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ В ЗАГАЛЬНІЙ СТРУКТУРІ АВТОМАТИЗОВАНОЇ СИСТЕМИ	9
1.1 Поняття автоматизованої системи	9
1.2 Принципи побудови автоматизованих систем	9
1.3 Класифікація автоматизованих систем	10
1.4 Види забезпечення автоматизованих систем	14
2 ОСНОВИ ОРГАНІЗАЦІЇ ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ...	20
2.1 Структура інформаційного забезпечення	20
2.2 Бази даних	21
2.3 Система керування базою даних	23
2.3.1 Призначення СКБД.....	23
2.3.2 Загальна архітектура СКБД.....	24
2.3.3 Лінгвістичні засоби СКБД.....	26
2.3.4 Інструментальні засоби СКБД	29
2.3.5 Словник даних	30
2.3.6 Функціонування СКБД	31
2.3.7 Архів.....	33
2.3.8 Адміністрування систем баз даних.....	33
2.4 Архітектура багатокористувацьких СКБД.....	36
2.4.1 Телеобробка.....	36
2.4.2 Файловий сервер	37
2.4.3 Технологія «клієнт-сервер»	39
2.4.4 Поняття технології відкритого доступу до даних	40
2.5 Розподілені бази даних	43
2.5.1 Поняття розподіленої бази даних	43
2.5.2 Стратегії розподілу даних.....	45
2.5.3 Фрагментація даних	47
2.5.4 Гомогенні та гетерогенні розподілені СКБД.....	51
2.5.5 Забезпечення прозорості в розподілених СКБД.....	53
2.5.6 Загальна модель розподіленої СКБД.....	56
2.5.7 Підтримка цілісності даних в розподілених базах даних.....	60
2.6 Життєвий цикл інформаційної системи	64
3 КОНЦЕПТУАЛЬНЕ ПРОЕКТУВАННЯ БАЗ ДАНИХ.....	68
3.1 Інструментальні засоби концептуального проектування.....	68
3.1.1 Сутності та атрибути.....	68
3.1.2 Зв'язки.....	69
3.2 Моделювання локальних подань	71
3.3 Об'єднання моделей локальних подань	79

4 ЛОГІЧНЕ ПРОЕКТУВАННЯ РЕЛЯЦІЙНИХ БАЗ ДАНИХ	87
4.1 Реляційна модель даних	87
4.1.1 Поняття моделі даних	87
4.1.2 Базисні засоби маніпулювання реляційними даними	88
4.1.3 Визначення відношення.....	89
4.1.4 Властивості та типи відношень	90
4.1.5 Операції над відношеннями	92
4.2 Одержання відношень з діаграм ER-типу	96
4.2.1 Попередні відношення для бінарних зв'язків ступеня 1:1	97
4.2.2 Попередні відношення для бінарних зв'язків ступеня 1:N	98
4.2.3 Попередні відношення для бінарних зв'язків ступеня M:N	99
4.3 Нормалізація баз даних.....	100
4.3.1 Аномалії ненормалізованого відношення	100
4.3.2 Типи функціональних залежностей	101
4.3.3 Поняття нормалізації.....	104
4.3.4 Перша та друга нормальні форми	105
4.3.5 Третя нормальна форма	107
4.3.6 Нормальна форма Бойса-Кодда (НФБК)	113
4.3.7 Четверта нормальна форма.....	115
4.4 Індексція	117
4.5 Критерії оцінки якості логічної моделі даних	118
4.6 Подання (курсори)	120
4.6.1 Поняття подання.....	120
4.6.2 Призначення подань.....	121
4.7 Збережені процедури.....	122
4.8 Проектування інтерфейсу користувача	123
5 ВИКОРИСТАННЯ МОВИ SQL.....	127
5.1 Загальні питання	127
5.2 Типи даних	129
5.2.1 Символьні рядки.....	129
5.2.2 Бітові рядки	129
5.2.3 Числові типи	130
5.2.4 Типи дата-час і інтервальні типи	130
5.3 Класифікація команд SQL	132
5.4 Створення баз даних і таблиць	133
5.4.1 Створення баз даних	133
5.4.2 Створення таблиць	133
5.4.3 Обмеження на множину припустимих значень.....	134
5.4.4 Підтримка посилальної цілісності	137
5.4.5 Створення індексів	138
5.5 Проектування запитів.....	140

5.5.1 Загальний формат команди SELECT	140
5.5.2 Вибір рядків: речення WHERE	141
5.5.3 Агрегатні функції	146
5.5.4 Групування даних	147
5.5.5 Сортування результатів запиту	149
5.5.6 З'єднання таблиць	151
5.6 Структуровані запити та підзапити	153
5.6.1 Загальне поняття	153
5.6.2 Некорельоване опрацювання	154
5.6.3 Корельовані запити	160
5.6.4 Використання EXISTS із зв'язаними підзапитами.....	163
5.6.5 Об'єднання множини запитів в один.....	164
5.7 Додавання, зміна та вилучання даних	165
5.7.1 Додавання нового рядку	165
5.7.2 Вилучання рядків із таблиці	167
5.7.3 Зміна значень полів.....	168
5.8 Подання	169
5.8.1 Створення та використання подань	169
5.8.2 Модифікація даних за допомогою подань	172
5.9 Питання адміністрування баз даних	173
6 ЗАХИСТ ДАНИХ	178
6.1 Відновлення	178
6.1.1 Транзакції.....	178
6.1.2 Транзакції та відновлення	180
6.2 Паралелізм	184
6.3 Безпека.....	186
6.3.1 Безпека та цілісність	186
6.3.2 Аспекти проблеми безпеки	186
6.3.3 Привілеї.....	187
6.3.4 Розподілення привілеїв.....	187
6.3.5 Вибірче керування доступом.....	188
6.3.6 Обов'язкове керування доступом	191
6.3.7 Контрольний слід виконання операцій	191
6.4 Цілісність	192
6.4.1 Обмеження цілісності	192
6.4.2 Типи обмежень цілісності	194
6.4.3 Обмеження стану та переходу.....	195
Література.....	197
Термінологічний словник	200
Предметний покажчик	206

СЛОВНИК СКОРОЧЕНЬ

4GL (Fourth-Generation Language) – мова четвертого покоління

API (Application Programming Interface) – прикладний програмний інтерфейс;

DDB (Distributed Database) – розподілена БД;

DDL (Data Definition Language) – мова опису даних;

DML (Data Manipulation Language) – мова маніпулювання даними;

ODBC (Open Database Connectivity) – відкритий доступ до даних;

SQL (Structured Query Language) – мова структурованих запитів;

АБД – адміністратор бази даних;

АД – адміністратор даних;

АІС – автоматизована інформаційна система;

АС – автоматизована система;

ІІПС – інформаційно-пошукова система;

ІЗ – інформаційне забезпечення;

ІС – інформаційна система;

ІСК (EIS – Executive Information System) – інформаційна система керівника;

КЗА – комплекс засобів автоматизації;

ММД – мова маніпулювання даними;

МОД – мова опису даних;

ПО – предметна область;

РБД – розподілена база даних;

СБД – система баз даних;

СКБД – система керування базою даних;

ВСТУП

Досягнення останніх років в галузі комп'ютерних і інформаційних технологій обумовили можливість створення автоматизованих систем (АС) обробки інформації в цілому ряді областей людської діяльності, ще недавно практично не охоплених процесами автоматизації.

Зокрема, з розвитком засобів і методів розподіленої обробки даних, створення автоматизованих систем організаційного керування стало одним з основних напрямків підвищення ефективності розв'язку задач, які характеризуються стислими строками виконання й нечіткістю формального опису.

Незважаючи на різноманіття типів АС і достатньо велику сферу їхнього застосування, їх об'єднує те, що обов'язковим компонентом будь-якої автоматизованої системи є інформаційне забезпечення (ІЗ), від якості якого багато в чому залежить працездатність і якість усієї системи в цілому.

Основне призначення даного навчального посібника – систематичний вступ в ідеї та методи, що використовуються в процесі проектування та експлуатації сучасного інформаційного забезпечення, зокрема, баз даних.

В курсі не розглядається яка-небудь одна популярна система керування базами даних (СКБД) – матеріал в рівному ступені відноситься до будь-якої сучасної системи. Як показує досвід, без знання основ баз даних важко на серйозному рівні працювати з конкретними системами, як би добре вони не були документовані.

Даний посібник складається з 6 розділів.

У *першому розділі* дається загальне поняття автоматизованої системи, розглядаються принципи побудови та класифікація АС, а також склад і призначення складових частин комплексу засобів автоматизації та місця в ньому інформаційного забезпечення.

Другий розділ присвячений розгляду основних питань організації інформаційного забезпечення. Описується структура ІЗ та призначення її основних компонентів. Розглядаються архітектура багатокористувацьких СКБД, організація розподілених баз даних, а також зміст основних етапів життєвого циклу інформаційної системи.

У *третьому розділі* розглянуто концептуальне проектування баз даних. Описуються інструментальні засоби концептуального

проектування. Розглядаються конкретні приклади моделювання локальних представлень, а також їх інтеграція до єдиної інформаційної моделі предметної області.

Четвертий розділ присвячений проектуванню реляційних баз даних. Розглянуто реляційну модель даних, типи та властивості відношень, а також основні операції над ними. Розглянуті питання нормалізації баз даних, індексації, використання курсорів, проектування інтерфейсу користувача.

У **п'ятому розділі** розглядається використання мови структурованих запитів SQL для проектування та обробки інформації в базах даних, бо саме ця мова стала фактичним стандартом, який використовується у сучасних СКБД.

Шостий розділ присвячений розгляду основних питань захисту інформації в базах даних: використання транзакцій для відновлення системи, паралелізм, безпеку та цілісності баз даних.

В основу навчального посібника покладений матеріал лекцій, які автор викладав на протязі останніх десяти років у Запорізькому національному технічному університеті для студентів спеціальності «Програмне забезпечення автоматизованих систем».

Матеріал посібника може бути корисним як для розуміння основних питань побудови та використання інформаційного забезпечення, так і при курсовому та дипломному проектуванні.

1 МІСЦЕ ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ В ЗАГАЛЬНІЙ СТРУКТУРІ АВТОМАТИЗОВАНОЇ СИСТЕМИ

1.1 Поняття автоматизованої системи

Автоматизована система (АС) являє собою організаційно-технічну систему, що забезпечує виробку рішень на основі автоматизації інформаційних процесів у різних сферах діяльності (керування, проектування, виробництво тощо) або їх комбінаціях.

Якщо процес, який автоматизується, пов'язаний в основному з обробкою інформації, то така система називається автоматизованою інформаційною системою.

Автоматизована інформаційна система – це людино-машинна система, що забезпечує автоматизовану підготовку, пошук і обробку інформації в рамках інтегрованих мережних, комп'ютерних і комунікаційних технологій для оптимізації економічної та іншої діяльності в різних сферах керування [1, 7, 22].

АС реалізують інформаційну технологію у вигляді певної послідовності інформаційно зв'язаних функцій, завдань або процедур, які виконуються в автоматизованому (інтерактивному) або автоматичному режимах.

Доцільність створення й впровадження АС визначається соціальним, науково-технічним та іншими корисними ефектами, що отримуються в результаті автоматизації.

Головною метою створення АС є не спрощення, але категоризація та стандартизація процесу, що автоматизується. Це дозволяє забезпечувати стабільність роботи системи, прозорість її контролю й аналізу слабких місць і підстави для її розвитку або згортання (списання, заміни).

У випадку правильної автоматизації діяльності організацій, вона спрощує прийняття рішень і зменшує необхідний час для вирішення проблем для керівників будь-якого рівня.

1.2 Принципи побудови автоматизованих систем

Автоматизовані системи створюють відповідно до технічного завдання, що є основним вихідним документом, на підставі якого виконується створення АС і приймання її замовником.

При створенні АС необхідно керуватися засадами [22, 24]:

- системності;
- розвитку (відкритості);
- сумісності;
- стандартизації (уніфікації);
- ефективності.

Принцип **системності** полягає в тому, що при декомпозиції повинні бути встановлені такі зв'язки між структурними елементами системи, які забезпечують цілісність АС і її взаємодію з іншими системами.

Принцип **розвитку** (відкритості) полягає в тому, що виходячи з перспектив розвитку об'єкта автоматизації, АС повинна створюватися з урахуванням можливості поповнення й оновлення функцій та складу АС без порушення її функціонування.

Принцип **сумісності** полягає в тому, що при створенні систем повинні бути реалізовані інформаційні інтерфейси, завдяки яким вона може взаємодіяти з іншими системами відповідно до встановлених правил.

Принцип **стандартизації** (уніфікації) полягає в тому, що при створенні систем повинні бути раціонально застосовані типові, уніфіковані та стандартизовані елементи, проектні рішення, пакети прикладних програм, комплекси, компоненти.

Принцип **ефективності** полягає в досягненні раціонального співвідношення між витратами на створення АС і цільовим ефектом, включаючи кінцеві результати, отримані в результаті автоматизації.

При створенні (модернізації) об'єктів автоматизації повинно бути передбачено проведення робіт зі створення (модернізації) автоматизованої системи.

1.3 Класифікація автоматизованих систем

Класифікація АС здійснюється за низкою ознак. Залежно від завдання, що вирішується, можна вибрати різні ознаки класифікації. При цьому АС може характеризуватися одним або декількома ознаками. У якості ознак класифікації АС використовуються [1, 22]:

- напрям діяльності;
- область і специфіка застосування;
- охоплювана територія;
- організація інформаційних процесів;
- призначення;
- структура тощо.

Класифікацію АС за напрямком діяльності (областю застосування) наведено на рис. 1.1. Як можна побачити з цього рисунку, у промисловій сфері превалює ієрархічна побудова АС.

За сферою (специфікою) застосування АС різняться в такий спосіб (рис. 1.2). Із цього класу АС історично першими стали застосовуватися АС на виробництві.

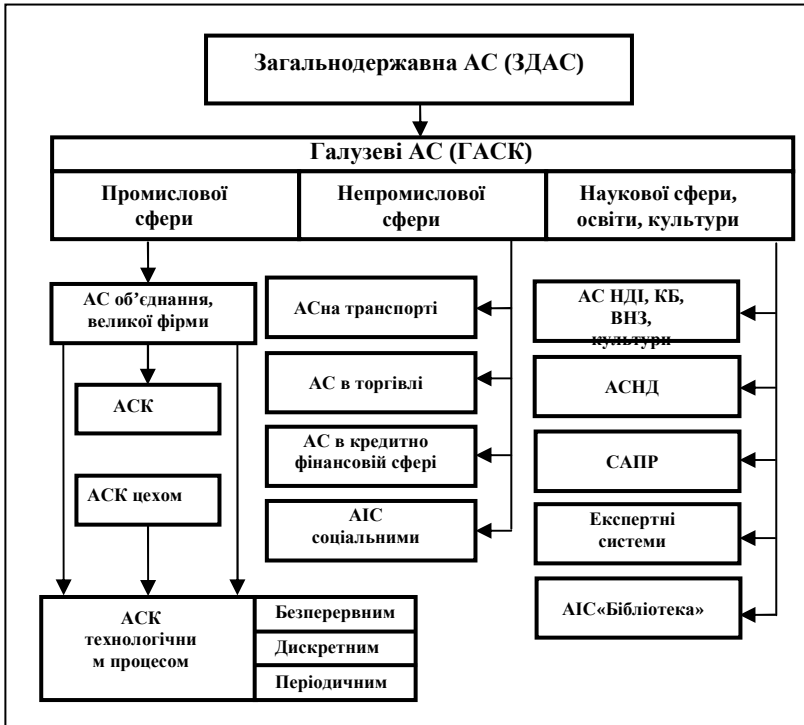


Рисунок 1.1 – Класифікація автоматизованих систем за напрямком діяльності (області застосування)

Класифікація АС за організацією інформаційних процесів передбачає розподіл автоматизованих систем на два великі класи: керуючі та інформаційні.

В інформаційних системах (ІС) керування відсутнє, наприклад: автоматизовані системи наукових досліджень – АСНД, «Бібліотека», системи автоматизованого проектування – САПР, експертні системи – ЕС тощо.

На відміну від чисто інформаційних систем у таких АІС, як автоматизовані системи керування технологічними процесами – АСКТП, АСК підприємствами – АСКП, керування займає важливе місце й буває або автоматичним, або автоматизованим.

Інформаційно-пошукові системи (ІПС) – у них об'єктом керування є процедура пошуку необхідної інформації в дуже великих обсягах цієї інформації. Типовий приклад – бібліотечні системи, різноманітні довідкові служби.



Рисунок 1.2 – Класифікація автоматизованих систем за специфікою застосування

Системи автоматизованого проектування (САПР) – у них об'єктом керування є процес проектування виробів будь-якої природи (верстата, літака, ЕОМ тощо).

Наступний клас систем – АС наукових досліджень і комплексних випробувань (АСНД). Тут об'єктом керування є процес дослідження об'єкта будь-якої роботи (дослідження процесу роботи двигуна, польоту літака, роботи реактора тощо).

Інтелектуальні системи (експертні системи) на відміну від попередніх систем оперують знаннями, що зберігаються в банку знань. Типові приклади – це медичні експертні системи, геологічні експертні системи тощо.

База знань (або банк знань) формується в результаті узагальнення знань провідних вчених, практиків, а також інформації, що зберігається в монографіях, статтях, книгах тощо.

Класифікація АС за призначенням наведена на рис. 1.3.

Класифікація АС за територіальною ознакою показана на рис. 1.4. За рівнем у системі державного керування виділяють територіальні, галузеві й міжгалузеві АІС, які одночасно є системами організаційного керування, але більш високого рівня.



Рисунок 1.3 – Класифікація автоматизованих систем за призначенням

Територіальні АІС призначені для керування адміністративно-територіальними районами. Територіальні системи автоматизують виконання управлінських функцій у регіоні, формування звітності, видачу оперативних відомостей місцевим державним і господарським органам.

Галузеві АІС створюються в сферах промислового та агропромислового комплексів, у будівництві, на транспорті. Ці системи вирішують задачу інформаційного забезпечення апарата управління відповідних відомств.

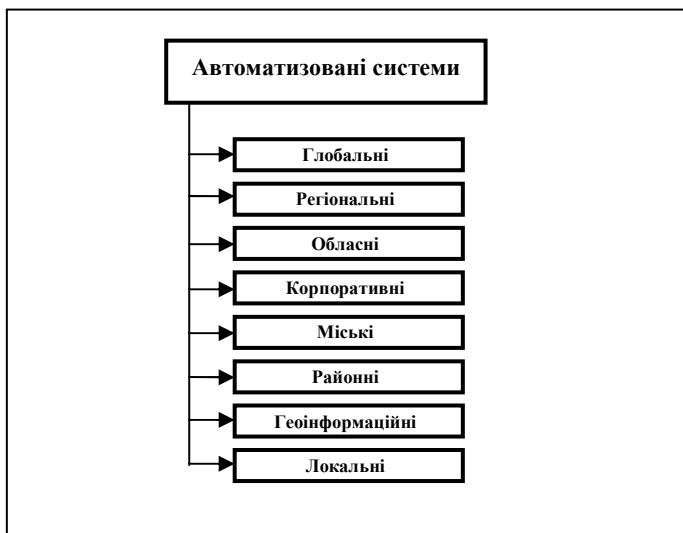


Рисунок 1.4 – Класифікація автоматизованих систем за територіальною ознакою

1.4 Види забезпечення автоматизованих систем

Загальну структуру складної системи можна розглядати як сукупність підсистем незалежно від сфери застосування. У цьому випадку кажуть про структурну ознаку класифікації, а підсистеми називають такими, що забезпечують.

Таким чином, структура будь-якої складної системи, зокрема автоматизованої системи, може бути представлена сукупністю забезпечуючих підсистем (комплексом засобів автоматизації – КЗА) (рис. 1.5), серед яких зазвичай виділяють [22, 24, 29]:

- технічне;
- математичне;
- програмне;
- інформаційне;
- лінгвістичне;
- ергономічне;
- організаційне;
- правове забезпечення.

Технічне забезпечення – комплекс технічних засобів, які забезпечують роботу автоматизованої системи, документація на ці засоби та технологічні процеси.

Комплекс технічних засобів становлять:

- комп'ютери (будь-яких платформ);
- обладнання збору, накопичення, обробки та виводу інформації;
- обладнання передачі/прийому даних і лінії зв'язку;
- оргтехніка та інше допоміжне обладнання тощо.

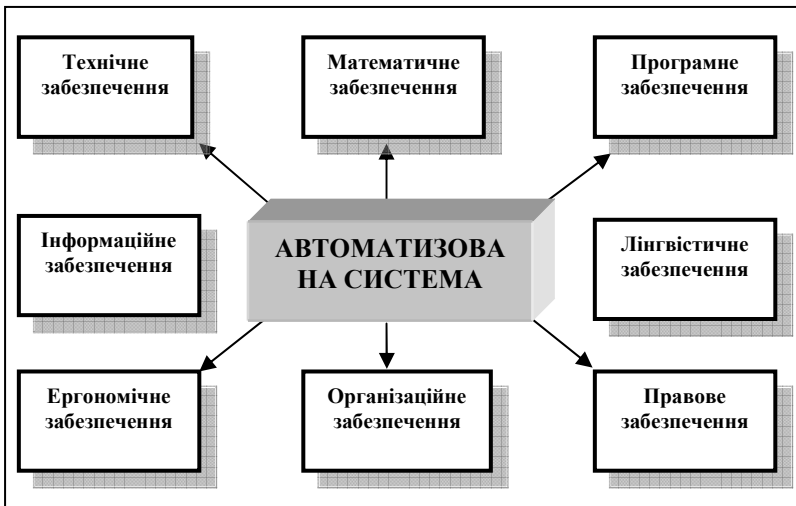


Рисунок 1.5 – Комплекс засобів автоматизації

До теперішнього часу склалися дві основні форми організації технічного забезпечення (форми використання технічних засобів) – централізована й частково або повністю децентралізована.

Централізоване технічне забезпечення базується на використанні в автоматизованій системі великих комп'ютерів (мейнфреймів, супер-ЕОМ) і обчислювальних центрів.

Децентралізація технічних засобів припускає реалізацію функціональних підсистем складної АС на робочих станціях, персональних комп'ютерах, промислових комп'ютерах безпосередньо на робочих місцях.

Перспективним підходом слід вважати, вочевидь, частково децентралізований підхід – організацію технічного забезпечення на базі розподілених мереж, які складаються із персональних і великих комп'ютерів для зберігання баз даних, загальних для будь-яких функціональних підсистем.

Для автоматизації складних об'єктів або процесів територіально розташованих на обмеженому просторі часто виявляється доцільною централізована структура. Для об'єктів, розподілених на великій території, наприклад у різних будинках, відокремлених філіях, або таких об'єктах як перекачувальні станції, електричні мережі єдиною доцільною структурою виявляється децентралізована структура.

Математичне й програмне забезпечення – сукупність математичних методів, моделей, алгоритмів і програм для реалізації цілей і завдань автоматизованої системи, а також нормального функціонування комплексу технічних засобів.

До засобів математичного забезпечення відносяться:

- засоби моделювання систем і процесів автоматизації;
- типові алгоритми процесів автоматизації;
- методи математичної теорії систем, системотехніки, математичної статистики, теорії масового обслуговування, математичного програмування тощо.

До складу програмного забезпечення входять загальносистемні й спеціальні програмні продукти, а також технічна документація.

До загальносистемного програмного забезпечення відносяться комплекси програм, орієнтованих на користувачів і призначених для розв'язання типових задач обробки інформації та керування. Вони служать для розширення функціональних можливостей комп'ютерів, контролю й керування процесом обробки даних.

Спеціальне програмне забезпечення являє собою сукупність програм, розроблених при створенні конкретної автоматизованої системи. До його складу входять пакети прикладних програм, що реалізують розроблені моделі різного ступеню адекватності, що відображають функціонування реального об'єкта.

Технічна документація на програмні засоби повинна містити опис задач, їх алгоритмізацію, економіко-математичну модель задач, контрольні приклади.

Лінгвістичне забезпечення – сукупність мовних засобів для формалізації природної мови, побудови й комбінації

інформаційних одиниць, які використовуються в автоматизованій системі при функціонуванні системи для спілкування з КЗА.

Ергономічне забезпечення – сукупність взаємозалежних вимог, спрямованих на узгодження психологічних, психофізіологічних, антропометричних, фізіологічних характеристик і можливостей людини-оператора, технічних характеристик КЗА, параметрів робочого середовища на робочім місці.

Організаційно-методичне забезпечення – сукупність методів і засобів, що регламентують взаємодію працівників з технічними засобами й між собою в процесі розробки та експлуатації АС. Даний вид забезпечення створюється за результатами передпроектного обстеження організації (підприємства, виробництва) і реалізує такі функції:

- аналіз існуючого об'єкта, де буде використовуватися АС, і виявлення задач, що підлягають автоматизації;
- підготовку задач до обробки на комп'ютері, включаючи технічне завдання на проектування АС і техніко-економічне обґрунтування ефективності;
- розробку управлінських рішень по складу та структурі організації, методології розв'язування задач, спрямованих на підвищення ефективності.

Правове забезпечення – сукупність правових норм, які визначають створення, юридичний статус і функціонування автоматизованих систем, а також порядок одержання, перетворення й використання інформації.

Головною метою правового забезпечення є зміцнення законності. Воно базується на законах, указах, постановах державних органів влади, наказах, інструкціях і інших нормативних документах міністерств, відомств, організацій, місцевих органів влади.

У правовім забезпеченні можна виділити як загальну частину, що регулює функціонування будь-якої АС, так і локальну частину, що регулює функціонування конкретної підсистеми.

Призначення підсистеми **інформаційного забезпечення** полягає в своєчасному формуванні та видачі достовірної інформації для прийняття управлінських рішень. Інформаційне забезпечення визначається як сукупність:

– системно-орієнтованих даних, що описують прийнятий у системі словник базових описів (єдина система класифікації й

кодування інформації, уніфіковані системи документації, класифікатори, типові моделі, елементи автоматизації, формати документації тощо);

– даних про стан інформаційної моделі об'єкту автоматизації, що актуалізуються на всіх етапах його життєвого циклу;

– схем інформаційних потоків, які циркулюють в організації;

– методології побудови систем баз даних [7, 8, 13, 24].

Уніфіковані системи документації створюються на державному, республіканському, галузевому й регіональному рівнях. Головна мета – це забезпечити порівнянність показників різних сфер суспільного виробництва. Розроблені стандарти, де встановлюються вимоги:

– до уніфікованих систем документації;

– до уніфікованих форм документів різних рівнів керування;

– до складу й структури реквізитів і показників;

– до порядку впровадження, ведення й реєстрації уніфікованих форм документів.

Однак, незважаючи на існування уніфікованої системи документації, при обстеженні більшості організацій постійно виявляється цілий комплекс типових недоліків:

– надзвичайно великий обсяг документів для ручної обробки;

– одні й ті ж показники часто дублюються в різних документах;

– робота з великою кількістю документів відволікає фахівців від вирішення безпосередніх задач;

– є показники, що створюються, але не використовуються тощо.

Тому усунення зазначених недоліків є одним із завдань при створенні інформаційного забезпечення.

Схеми інформаційних потоків відбивають маршрути руху інформації та її обсяги, місця виникнення первинної інформації й використання результатної інформації. За рахунок аналізу структури подібних схем можна виробити заходи щодо вдосконалювання всієї системи керування.

Побудова схем інформаційних потоків, що дозволяють виявити обсяги інформації й провести її детальний аналіз, забезпечує:

– виключення інформації, що дублюється та не використовується;

– класифікацію та раціональне подання інформації.

При цьому докладно повинні розглядатися питання взаємозв'язку руху інформації по рівням керування. Слід виявити, які показники

необхідні для прийняття управлінських рішень, а які ні. До кожного виконавця повинна надходити тільки та інформація, яка використовується.

Методологія побудови баз даних базується на теоретичних основах їх проектування.

Для створення інформаційного забезпечення необхідно:

- ясне розуміння цілей, завдань, функцій усієї системи керування організацією;

- виявлення руху інформації від моменту виникнення й до її використання на різних рівнях керування, поданої для аналізу у вигляді схем інформаційних потоків,

- удосконалення системи документообігу;

- наявність і використання системи класифікації й кодування;

- володіння методологією створення концептуальних та інформаційно-логічних моделей, що відбивають взаємозв'язок інформації;

- створення масивів інформації на машинних носіях, що вимагає наявності сучасного технічного забезпечення.

Організація інформаційного забезпечення визначається складом об'єктів предметної області, задач і даних, а також сукупністю інформаційних потреб користувачів автоматизованої системи.

Інформаційне забезпечення є системоутворючим фактором у структурі АС у цілому [24].

Докладно теоретичні та практичні питання проектування інформаційного забезпечення будуть розглянуті в наступних розділах.



Питання для самоперевірки

1. Що називається автоматизованою системою?
2. На яких принципах будуються автоматизовані системи?
3. Які типи автоматизованих систем використовуються?
4. З чого складається комплекс засобів автоматизації?

2 ОСНОВИ ОРГАНІЗАЦІЇ ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Структура інформаційного забезпечення

Інформаційне забезпечення включає повний набір показників, документів, класифікаторів і кодифікаторів інформації, файлів, баз даних, баз знань, методів їх використання в предметній діяльності, а також способи подання, накопичення, зберігання, перетворення, передачі інформації, прийняті у конкретній системі для задоволення інформаційних потреб користувачів у потрібній формі й у необхідний час [8, 16].

У рамках інформаційного забезпечення існують позамашинні та машинні масиви інформації, що сприймаються людиною без технічних засобів, наприклад, наряди, акти, накладні тощо.

Машинні масиви інформації зберігаються на носіях і складаються з файлів. Вони можуть бути створені як сукупність окремих файлів, кожний з яких відбиває множину однорідних управлінських документів (нарядів, накладних і т. п.), або як бази даних.

Організація інформаційного забезпечення визначається складом об'єктів предметної області, що відображається, задач, даних і сукупністю інформаційних потреб користувачів автоматизованої системи.

Укрупнене структура інформаційного забезпечення (ІЗ) подана на рис. 2.1.

Основними компонентами ІЗ є:

- система керування базою даних (СКБД);
- база даних (БД);
- архів системи;
- системний каталог (словник даних);

База даних і СКБД – основні складові компоненти інформаційного забезпечення..

Призначення компонентів системи докладніше буде розглянуто у наступних параграфах.

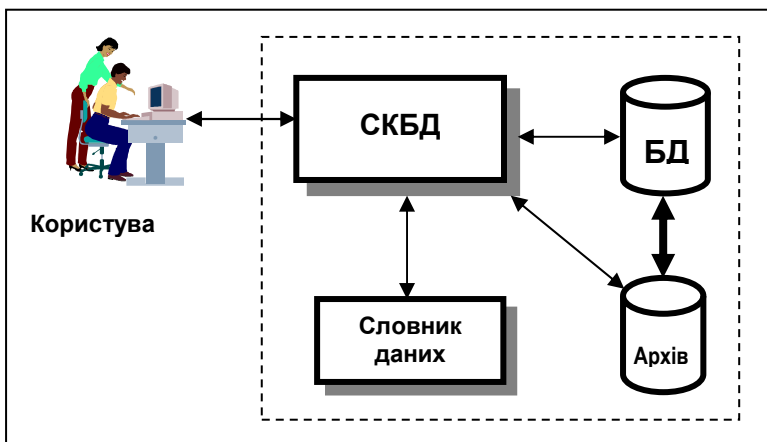


Рисунок 2.1 – Структура інформаційного забезпечення

2.2 Бази даних

 **База даних** – це поійменована структурована сукупність взаємозв’язаних даних, які описують певну предметну область і використовуються багатьма користувачами.

 **Предметною областю** (ПО) називається сукупність об’єктів реального світу, що розглядаються у контексті задачі, що вирішується. У якості об’єкту реального світу можуть бути як матеріальний об’єкт, так і процес, явище тощо

Існує багато інших визначень поняття «база даних», однак загальновизнане єдине формулювання відсутнє. Найбільше часто використовуються такі відмінні ознаки:

– *БД зберігається й обробляється в обчислювальній системі*
Таким чином, будь-які позакомп’ютерні сховища інформації (архіви, бібліотеки, картотеки тощо) не являються базами даних;

– *дані в БД логічно структуровані (систематизовані)* з метою забезпечення можливості їх ефективного пошуку та обробки в обчислювальній системі. Структурованість передбачає явне виділення складових частин (елементів), зв’язків між ними, а також типізацію елементів і зв’язків, при якій з типом елемента (зв’язку) співвідноситься певна семантика та припустимі операції;

– БД включає метадані, що описують логічну структуру БД у формальному виді (у відповідності з деякою метамоделлю).

Головним завданням БД є гарантоване збереження значних обсягів інформації та надання доступу до неї користувачам або ж прикладним програмам. З метою забезпечення ефективності доступу записи даних організовують як множину фактів (елемент даних).

Централізований підхід щодо збереження та керування даними має такі переваги:

- 1.Значно зменшується надлишок інформації.
- 2.Зменшення надлишку інформації дозволяє усунути протиріччя та забезпечити логічну цілісність даних.
- 3.Доступ до даних може здійснюватися одночасно багатьма прикладними програмами чи користувачами.
- 4.Завдяки повному контролю над базою даних з'являється можливість забезпечення заходів безпеки баз даних.

Бази даних поділяються на структуровані та неструктуровані.

Структуровані БД використовують структури даних, тобто структурований опис типу фактів за допомогою моделі даних.

До неструктурованих БД відносяться повнотекстові бази даних, які містять неструктуровані тексти статей чи книг у формі, що дозволяє здійснювати швидкий пошук.

Часто зустрічається характеристика БД на основі певних параметрів або необхідних вимог, наприклад:

- значна кількість даних
- незалежність даних
- відкритий доступ до даних
- підтримка транзакцій з гарантією відповідних властивостей
- гарантована відсутність збоїв
- одночасна робота з багатьма користувачами

З подальшим розвитком БД змінюються й ці вимоги та додаються нові, тому однастайності щодо повноти цієї характеристики немає.

2.3 Система керування базою даних

2.2.1 Призначення СКБД



СКБД – це комплекс програмних і лінгвістичних засобів, які забезпечують створення, завантаження, супроводження та експлуатацію бази даних

СКБД грає центральну роль у функціонуванні системи баз даних. Вона володіє засобами безпосереднього доступу до БД і надає кожній прикладній програмі інтерфейс для взаємодії з БД.

До основних функцій СКБД відносяться:

- Визначення схеми бази даних. У процесі роботи прикладних програм і термінальних користувачів база даних змінюється. Однак ці зміни не можуть бути довільними. Зазвичай у БД існують достатньо жорсткі обмеження на можливості маніпулювання даними. Ці обмеження дозволяють заздалегідь виконати опис незмінних властивостей даних у БД, який отримав назву схеми бази даних.

- Здійснення операцій над даними БД. СКБД повинна вміти обробляти запити користувачів на вибірку, оновлення чи вилучання даних, які містяться в БД, або додавання нових даних до БД.

- Надання доступу до БД, який контролюється за рахунок наступних засобів (будуть докладно розглянуті у главі 6):

- ✓ системи забезпечення безпеки, що запобігає несанкціонованому доступу до бази даних з боку користувачів;
- ✓ системи підтримки цілісності даних, яка забезпечує несуперечливий стан даних у БД;
- ✓ системи керування паралельною роботою додатків;
- ✓ системи відновлення, що дозволяє відновити стан БД після системного збою;
- ✓ каталогу, який доступний користувачам і містить опис інформації, що зберігається в БД.

2.2.2 Загальна архітектура СКБД

У складі СКБД можна виділити ядро (**Data Base Engine**), яке забезпечує організацію введення, обробки та зберігання даних, а також компоненти, що забезпечують налагодження системи, засоби тестування, утиліти, призначені для виконання допоміжних функцій.

Ядро СКБД має власний інтерфейс, який недоступний користувачам напряму й використовується у програмах, вироблених компілятором SQL (або в підсистемі підтримки виконання таких програм) і утилітах БД. Ядро СКБД є основною резидентною частиною СКБД.

На рис. 2.2 зображена узагальнена архітектура СКБД. Тут умовно показано, що СКБД повинна керувати зовнішньою пам'яттю, у якій розташовані файли з даними, фаїли журналів і файли системного каталогу.

З іншого боку, СКБД керує й оперативною пам'яттю, в якій розташовуються буфери з даними, буфери журналів, дані системного каталогу, необхідні для підтримки цілісності й перевірки привілеїв користувачів. Крім того, в оперативній пам'яті під час роботи СКБД розташовується інформація, що відповідає поточному стану обробки запитів, там зберігаються плани виконання скомпільованих запитів тощо.

Модуль керування зовнішньою пам'яттю забезпечує створення необхідних структур зовнішньої пам'яті як для зберігання даних, що безпосередньо входять до БД, так і для службових цілей, наприклад для прискорення доступу до даних у деяких випадках (зазвичай для цього використовуються індекси). Потрібно підкреслити, що в розвинених СКБД користувачі в кожному разі не зобов'язані знати, чи використовує СКБД файлову систему, і якщо використовує, то як організовані файли. Зокрема, СКБД підтримує власну систему іменування об'єктів БД.

Модуль керування буферами оперативної пам'яті призначений для рішення завдань ефективної буферизації, що використовується практично для виконання всіх інших функцій СКБД.

СКБД зазвичай працюють із БД значного розміру. Принаймні, цей розмір зазвичай істотно більше доступного обсягу оперативної пам'яті. Зрозуміло, що якщо при зверненні до будь-якого елемента даних буде виконуватися обмін із зовнішньою пам'яттю, то вся система буде працювати зі швидкістю пристрою зовнішньої пам'яті. Практично єдиним способом реального збільшення цієї швидкості є буферизація даних в оперативній пам'яті.

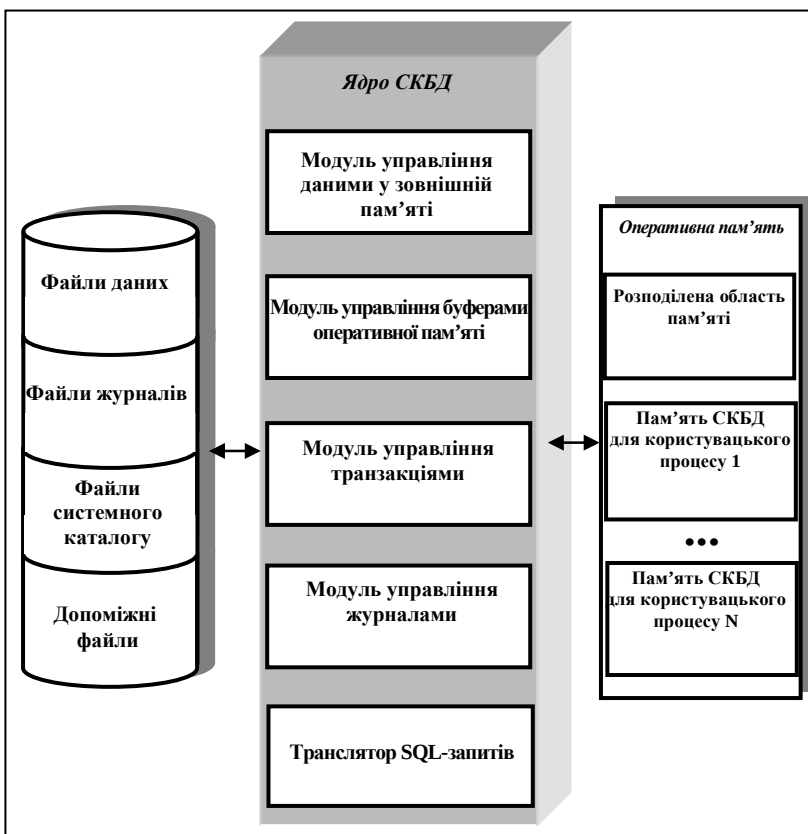


Рисунок 2.2 – Узагальнена структура СКБД

Модуль керування транзакціями являє собою механізм, який гарантує при виконанні операцій відновлення БД перехід її з одного стійкого стану в інший. **Транзакція** являє собою послідовність дій, виконуваних окремим користувачем або прикладною програмою з метою доступу або зміни вмісту БД. При цьому транзакція або виконується цілком, або всі зміни, виконані в процесі її виконання, відміняються (детально механізм транзакцій описано у розділі 6).

Модуль керування журналами призначений для фіксації всіх змін умісту БД у процесі роботи з нею з метою забезпечення можливості відновлення системи у випадку системного збою.

Журнал – це особлива частина БД, недоступна користувачам СКБД і підтримувана з особливою старанністю (іноді підтримуються дві копії журналу, розташовані на різних фізичних дисках), до якої надходять записи про всі зміни основної частини БД (див. розділ 6).

У різних СКБД зміни БД журналізуються на різних рівнях: іноді запис у журналі відповідає деякій логічній операції зміни БД (наприклад, операції вилучення рядку з таблиці реляційної БД), іноді – мінімальній внутрішній операції модифікації сторінки зовнішньої пам'яті; у деяких системах одночасно використовуються обидва підходи.

Транслятор SQL-запитів компілює (перекладає) оператори мови БД у деяку програму, що виконується. Основною проблемою реляційних СКБД є те, що мови цих систем (а це, як правило, SQL) є непроцедурними, тобто в операторі такої мови специфікується деяка дія над БД, але ця специфікація не є процедурою, а лише описує в деякій формі умови здійснення бажаної дії. Результатом компіляції є виконувана програма, що подається в деяких системах у машинних кодах, але більш часто у виконуваному внутрішньому машинно-незалежному коді.

2.2.3 Лінгвістичні засоби СКБД

Мова опису даних



Мова опису даних (Data Definition Language – DDL) – це мова декларативного (описового) типу, що призначена для створення та модифікації схеми бази даних.

За її допомогою описуються типи даних, що повинні зберігатися у БД або вибиратися з неї, їх структура та зв'язки між собою, а також засоби завдання обмежень для інформації, що зберігається в БД. Таким чином, МОД призначена для опису структури інформації на логічному рівні.

Схема БД складається з набору визначень, які зроблені за допомогою мови опису даних. МОД використовується як для визначення нової схеми, так і для модифікації вже існуючої. Цю мову неможливо використовувати для керування даними.

Зазвичай у системі розрізняють завдання всієї схеми БД та завдання її частин – підсхем. Підсхеми використовуються для опису частин БД, які відображають інформаційні потреби окремого користувача чи прикладної програми (зовнішні подання).

Опис даних, виконаний на МОД, після компіляції відображається до управляючих таблиць, які знаходяться в системному каталозі й містять [16]:

- адресні константи, що вказують на розміщення даних у пам'яті ЕОМ та на зв'язки між собою;
- константи, що характеризують розмірність даного та формат, у якому воно представлене;
- інша інформація, що необхідна для роботи СКБД.

Мова маніпулювання даними



Мова маніпулювання даними (Data Manipulation Language – DML), або мова запитів до БД, є засобом, що використовується прикладними програмами або користувачами для виконання операцій над БД.

Наявність централізованого сховища всіх даних і їхнього опису дозволяє використовувати мову DML як загальний інструмент організації запитів, який також називають мовою запитів (**Query Language**). Наявність мови запитів дозволяє усунути обмеження, що властиві файловим системам, коли користувачам доводиться мати справу тільки з фіксованими наборами запитів або з програмами, кількість яких постійно зростає.

Найбільш поширеною мовою запитів є мова структурованих запитів (**SQL – Structured Query Language**), яка стала фактичним стандартом сучасних СКБД (розгляду цієї мови присвячений розділ 5).

До операцій маніпулювання даними відносяться (у дужках вказані відповідні оператори SQL):

- **додавання** до БД нових відомостей (**insert**);
- **модифікація** відомостей, які зберігаються у БД (**update**);
- **вибірка** відомостей, що містяться у БД (**select**);
- **вилучення** відомостей з БД (**delete**).

За допомогою цієї мови можна виконати, наприклад, такі команди:

- виконати запис даного до конкретного поля;

- здійснити вибірку з бази конкретного даного, що задовольняє заданим умовам;
- знайти у базі позицію даного та помістити туди його нове значення або вилучити;
- здійснити сортування даних за заданими критеріями;
- визначити середнє значення даних, які відповідають заданим умовам тощо.

Мови 4GL

Абревіатура **4GL** являє собою скорочений англійський варіант написання терміна **мова четвертого покоління (Fourth-Generation Language)**. Не існує чіткого визначення цього поняття, хоча, по суті, мова йде про деякий стенографічний варіант мови програмування. Якщо для організації деякої операції з даними мовою третього покоління (3GL) типу COBOL буде потрібно написати сотні рядків коду, то для реалізації цієї ж операції мовою четвертого покоління буде досить 10–20 рядків.

У той час як мови третього покоління є процедурними, мови 4GL виступають як непроцедурні, оскільки користувач визначає, **що** повинне бути зроблене, але не говорить, **як** саме бажаний результат повинен бути досягнутий.

Реалізація мов четвертого покоління значною мірою заснована на використанні компонентів високого рівня, котрі часто називають «інструментами четвертого покоління». Користувачеві не буде потрібно визначати всі етапи виконання програми, необхідні для рішення поставленого завдання, а досить буде лише визначити потрібні параметри, на підставі яких згадані вище інструменти автоматично здійснять генерацію прикладного застосування.

Виділяють такі типи мов четвертого покоління:

- мови подання інформації, наприклад мови запитів або генератори звітів;
- спеціалізовані мови, наприклад мови електронних таблиць і баз даних;
- генератори додатків, які при створенні додатків забезпечують визначення, вставку, оновлення або витяг відомостей з бази даних;
- мови дуже високого рівня, призначені для генерації коду додатків.

Як приклад мови четвертого покоління можна вказати згадувану вище мову SQL.

2.2.4 Інструментальні засоби СКБД

Генератори форм

Генератор форм являє собою інтерактивний інструмент, призначений для швидкого створення шаблонів введення й відображення даних в екранних формах. Генератор форм дозволяє користувачеві визначити зовнішній вигляд екранної форми, її вміст і місце розташування на екрані. З його допомогою можна задавати кольори елементів екрану, а також інші характеристики, наприклад напівжирне, підкреслене, мерехтливе або реверсивне накреслення тощо. Більш вдосконалені генератори форм дозволяють створювати атрибути, що обчислюються з використанням арифметичних операторів або узагальнюючих функцій, а також задавати правила перевірки даних, які вводяться.

Генератори звітів

Генератор звітів є інструментом створення звітів на основі збереженої в базі даних інформації. Він подібний мові запитів у тому розумінні, що користувачеві надаються засоби створення запитів до бази даних і витягу з неї інформації, що використовується для подання у звіті. Однак генератори звітів, як правило, передбачають набагато більші можливості керування зовнішнім виглядом звіту. Генератор звіту дозволяє або автоматично визначати вид одержуваних результатів, або за допомогою спеціальних команд створювати свій власний варіант зовнішнього вигляду документа, що друкується.

Генератори графічного подання даних

Цей генератор являє собою інструмент, призначений для витягу інформації з бази даних і відображення її у вигляді діаграм із графічним поданням існуючих тенденцій і зв'язків. Зазвичай за допомогою подібного генератора створюються гістограми, кругові, лінійчаті, крапкові діаграми тощо.

Генератори застосувань

Генератор застосувань (*applications*) являє собою інструмент для створення програм, які взаємодіють з базою даних. Застосовуючи генератор застосувань, можна скоротити час, необхідний для проектування повного обсягу потрібного прикладного програмного забезпечення.

Генератори застосувань зазвичай складаються з попередньо створених модулів, що містять фундаментальні функції, що потрібні для роботи більшості програм. Ці модулі, що зазвичай створюються на мовах високого рівня, утворюють «бібліотеку» доступних функцій. Користувач указує, які завдання програма повинна виконати, а генератор застосувань визначає, як їх варто виконати.

2.2.5 Словник даних

	Словник даних (системний каталог) – це сховище даних, які описують інформацію, що зберігається в базі даних, тобто метадані.
---	---

Метадані, або «дані про дані», описують об'єкти БД, а також дозволяють спростити доступ до них і керування ними.

Словник даних (СД) є одним із основних компонентів системи. Перед доступом до реальних даних СКБД зазвичай звертається до системного каталогу, до якого крім СКБД мають доступ і користувачі.

Зазвичай в системному каталозі містяться такі відомості:

- імена, типи та розміри елементів даних;
- імена зв'язків;
- обмеження підтримки цілісності, що накладаються на дані;
- імена санкціонованих користувачів, яким надано право доступу до даних, і дозволені типи доступу до них – вставка, оновлення, вилучання чи читання (привілеї);
- статистичні дані, наприклад, частота транзакцій і лічильники звернень до об'єктів БД.

СД дуже важливий, особливо в умовах колективного використання даних, оскільки він забезпечує розв'язання проблеми достовірності, надлишковості та контролю за раціональним зберіганням і використанням даних. Словник даних повинен виконувати такі основні функції:

- встановлення зв'язків між користувачами БД;
- виконання простого та ефективного управління елементами даних при введенні в систему як нових елементів, так і при модифікації опису існуючих;
- зменшення надмірності даних;
- запобігання неузгодженості даних;
- централізоване управління даними з метою спрощення процесу проектування БД та її розширення.

Словник даних використовується кінцевими користувачами під час роботи із системою, прикладними програмами – при

написанні програм.

Словник даних може вміщувати відомості про джерело інформації, формати та взаємозв'язок між даними, відомості про частоту виникнення та характер використання даних, терміни коректування та осіб, відповідальних за це тощо.

До СД можна занести також інформацію про місце фізичного зберігання даних, а також відомості про обмеження секретного характеру, санкціонованості доступу та інші питання, пов'язані із захистом інформації.

Одне з основних призначень СД є документування даних. Йому відводиться роль засобу централізованого ведення та управління даними на всіх етапах проектування системи, а також забезпечення ефективної взаємодії між усіма користувачами при розподіленій БД.

Робота з СД спрощує процес підготовки документації програмістами, забезпечує користувачів інформацією про наявні в системі дані, дозволяє контролювати відповідність у назвах і форматах даних та їх кодах.

2.2.6 Функціонування СКБД

СКБД, основною функцією яких є виконання операцій по обробці даних для прикладних програм, використовують під час роботи схему та підсхему. Обробка запитів здійснюється у певній послідовності (рис. 2.3):

- 1 Прикладна програма *A* видає до СКБД запит на читання запису. При цьому програма також повідомляє ім'я користувача, що вимагає ці дані.
- 2 СКБД отримує у розпорядження підсхему, яку використовує прикладна програма *A*, та здійснює в ній пошук опису даних, на які видано запит.
- 3 СКБД проглядає опис фізичної організації бази даних і визначає, який фізичний запис (або записи) потрібно зчитати.
- 4 СКБД видає операційній системі команду читання потрібного запису (або записів).
- 5 Операційна система взаємодіє з фізичною пам'яттю, в якій зберігаються дані.
- 6 Запитані дані передаються із пам'яті до системних буферів.
- 7 СКБД передає дані з системних буферів до робочої області

прикладної програми А.

8 Прикладна програма обробляє дані, які знаходяться в її робочій області.

У випадку, коли прикладна програма оновлює запис, здійснюється аналогічна послідовність дій. Запис спочатку звичайним способом зчитується та модифікується в робочій області програми, а потім до СКБД передається команда записати модифіковані дані. При цьому СКБД має здійснювати необхідні перетворення даних, обернені тим перетворенням, які були виконані при зчитуванні даних. Потім СКБД видає операційній системі команду **ЗАПИСАТИ**.

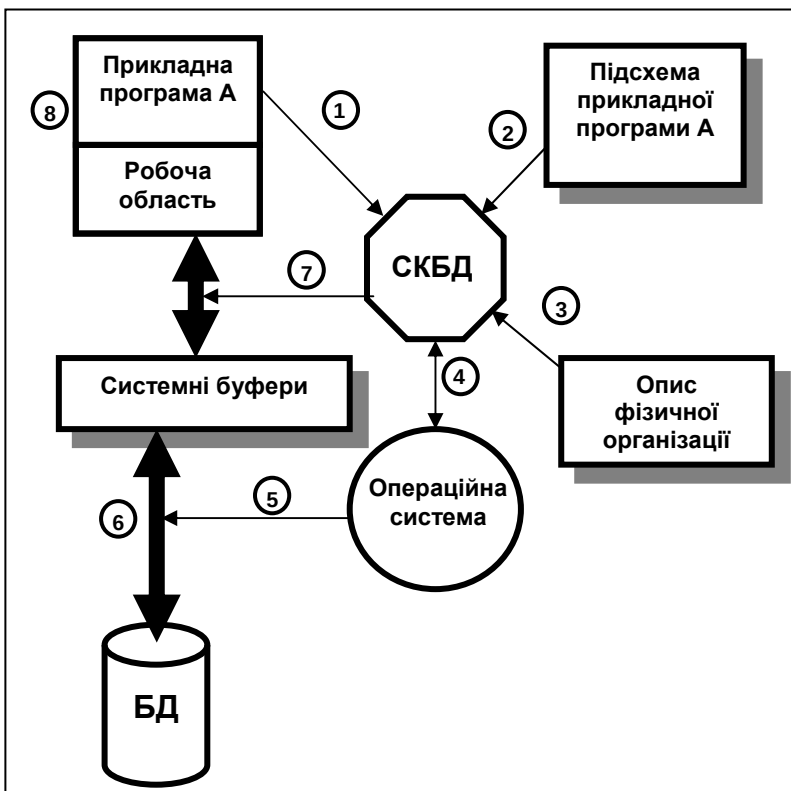


Рисунок 2.3 – Послідовність дій СКБД при обробці запитів

2.2.7 Архів

Усі дані, що зберігаються в БД, поділяються на фонд і архів даних, що пов'язано з відмінністю в технологічних режимах використання даних.

Фонд даних – це активні дані, з якими постійно працюють прикладні програми та користувачі, що зберігаються на вінчестері й перебувають безпосередньо під керуванням СКБД.

Архів – це копії файлів БД. В архівах зберігаються неактивні дані, що вже відпрацювали, але їх необхідно зберігати згідно з нормативними та законодавчими актами досить тривало. Прикладом може бути банківська та деякі види облікової інформації, яка згідно із законодавчими актами має зберігатися кілька років. В архівах також зберігаються страхові копії файлів БД, які використовуються для відновлення БД в разі її руйнування при різних збоях.

Крім того, перенесення «відпрацьованих» даних до архіву зменшує об'єм БД, що, в свою чергу, зменшує час доступу до даних, тобто підвищує продуктивність системи.

2.2.8 Адміністрування систем баз даних

Адміністрування базами даних передбачає виконання функцій, спрямованих на забезпечення надійного та ефективного функціонування системи баз даних, адекватності змісту бази даних інформаційним потребам користувачів, відображення в базі даних актуального стану предметної області [7].

Залежно від об'єму бази даних, специфіки та особливостей СКБД служба адміністрації СБД може відрізнитись як за чисельністю складу, так і за рівнем фахової підготовки. Чисельність групи адміністрації та функції, що вона виконує, залежать від масштабу системи баз даних, специфіки інформації, що в неї зберігається, типу СКБД і особливостей програмних засобів. Як правило, для великих розподілених баз даних створюється спеціальна адміністрація.

До складу групи адміністрування можуть входити системні аналітики, проектувальники структур даних і зовнішнього, щодо банку, інформаційного забезпечення, проектувальники технологічних процесів обробки даних, системні та прикладні програмісти, оператори, спеціалісти з технічного обслуговування.

В комерційних банках даних до адміністративної групи обов'язково повинні входити спеціалісти з маркетингу.

Отже, адміністратор БД (АБД) – це спеціаліст, який має цілкове уявлення про інформаційні потреби користувачів, співпрацює з ними в тісному контакті й відповідає за завантаження, ведення та підтримку БД в актуальному стані, а також за захист і ефективність функціонування системи. Він повинен враховувати як поточні, так і перспективні інформаційні вимоги предметної області.

Не можна розглядати адміністратора баз даних тільки як кваліфікованого технічного спеціаліста, так як це не відповідає цілям адміністрування. Рівень адміністратора баз даних в ієрархії організації досить високий: щоб визначати структуру даних і право доступу до них, адміністратор повинен знати, як працює підприємство і як використовуються відповідні дані.

До основних функцій адміністратора відносяться такі [16, 34]:

- сумісна робота з проєктувальниками для визначення умов використання БД;
- розробка опису БД;
- концептуальне проєктування БД;
- початкове завантаження БД;
- підтримка цілісності БД;
- організація захисту зберігання даних;
- відновлення БД в разі виникнення помилок програмного забезпечення чи збоїв пристроїв, які призводять до руйнування БД;
- накопичення статистики по роботі з БД;
- реорганізація та реструктуризація БД в разі необхідності.

Різноманітні задачі, які повинен розв'язувати адміністратор, можна поділити відповідно до етапів розробки СБД на чотири групи:

- планування;
- проєктування;
- експлуатація;
- використання.

При плануванні адміністратор бере участь у виборі програмного забезпечення та обладнання. Він співпрацює з кінцевими користувачами, щоб встановити реальні потреби, мету та вимоги прикладних програм до бази даних. Адміністратор бере участь у довгостроковому плануванні, в якому визначаються перспективи розвитку системи та розширення бази даних.

Під час проектування адміністратор надає спеціалістам необхідні дані для розробки логічної та фізичної моделей даних. Якщо виникають нові вимоги до даних, адміністратор визначає, як можна включити ці дані до баз даних і управляє процесом внесення змін.

На етапі експлуатації до обов'язків адміністратора входять розробка та контроль дій, які гарантують збереження цілісності бази даних, включаючи процедури її копіювання та відновлення, а також визначення засобів захисту інформації за допомогою механізму управління доступом до ресурсів БД.

Крім того, адміністратор увесь час спілкується з користувачами бази даних, тому він визначає стандарти на зміст і використання даних, супроводжує спеціальне програмне забезпечення роботи з базою даних (словники даних, статистику роботи з БД тощо), проводить консультації та ведення БД.

Адміністратор взаємодіє також із системними програмістами, вирішуючи питання технології та доведення її до відповідних експлуатаційних характеристик, а також у разі зміни версій системи чи операційного середовища.

При супроводженні баз даних, особливо розподілених, адміністраторові часто доводиться співпрацювати із співробітниками, котрі відповідають за експлуатацію технічних засобів СБД. Така співпраця пов'язана з необхідністю виявлення причин збоїв обладнання, що призводять до руйнування баз даних, і розробки заходів запобігання цього в ході подальшої експлуатації системи.

До технічних засобів СБД належать ЕОМ і периферійні засоби для введення даних у базу даних, передавання даних по мережі та відображення підсумкової інформації, що виводиться користувачеві. У кожному конкретному випадку залежно від особливостей СКБД, яка використовується, та особливостей об'єкта управління проектується й різна конфігурація технічних засобів.

У технічній документації на СКБД наводиться мінімальна конфігурація технічних засобів, яка необхідна для організації БД, а також обмеження на склад і кількість технічних засобів.

2.4 Архітектура багатокористувацьких СКБД

2.4.1 Телеобробка

Традиційною архітектурою багатокористувацьких систем раніше вважалася схема, що одержала назву «телеобробки», при якій один комп'ютер із єдиним процесором був з'єднаний з декількома терміналами так, як показано на рис. 2.4. При цьому вся обробка виконувалася в рамках єдиного комп'ютера, а приєднані до нього користувальницькі термінали були типовими «неінтелектуальними» пристроями, не здатними функціонувати самостійно.

Із центральним процесором термінали були пов'язані за допомогою кабелів, по яких вони посилали повідомлення користувальницьким додаткам (через підсистему керування обміном даними операційної системи).

У свою чергу, користувальницькі додатки зверталися до необхідних служб СКБД. У такий же спосіб повідомлення верталися назад на користувальницький термінал. На жаль, при такій архітектурі основне й надзвичайно велике навантаження покладалося на центральний комп'ютер, що повинен був виконувати не тільки дії прикладних програм і СКБД, але й значну роботу з обслуговування терміналів (форматування даних, виведених на екрани тощо).



Рисунок 2.4 – Топологія архітектури телеобробки

В останні роки був досягнутий істотний прогрес у розробці високопродуктивних персональних комп'ютерів і складених з них мереж. При цьому у всій індустрії спостерігається помітна тенденція

до **децентралізації** (downsizing), тобто заміні дорогих мейнфреймів більш ефективними, з погляду експлуатаційних витрат, мережами персональних комп'ютерів, які дозволяють отримувати такі ж результати, якщо не кращі. Ця тенденція призвела до появи наступних двох типів архітектури СКБД: технології файлового сервера й технології «клієнт / сервер» [16, 29, 41].

2.4.2 Файловий сервер

Комп'ютер, який керує тим чи іншим ресурсом, прийнято називати сервером цього ресурсу, а комп'ютер, який бажає їм скористатися, – клієнтом. Конкретний сервер визначається видом ресурсу, яким він володіє. Якщо ресурсом є бази даних, то мова йде про сервер баз даних, призначення якого – обслуговувати запити клієнтів, пов'язані з обробкою даних; якщо ресурс – це файлова система, то кажуть про файловий сервер або файл-сервер.

Цей принцип розповсюджується й на взаємодію програм. Якщо одна з них виконує деякі функції, надаючи іншим відповідний набір послуг, то така програма розглядається в якості сервера. Програми, що користуються цими послугами, прийнято називати клієнтами. Так, ядро реляційної SQL-орієнтованої СКБД часто називають сервером бази даних або SQL-сервером, а програму, що звертається до нього за послугами з обробки даних – SQL-клієнтом.

При роботі в комп'ютерній мережі БД може розмішуватися на файл-сервері або декількох файл-серверах. У якості файл-сервера може бути використаний або спеціально виділений комп'ютер, або одна з об'єднаних у мережу найбільш потужних машин.

Функції файл-сервера містяться в основному у зберіганні БД та забезпеченні доступу до неї користувачів, які працюють на різних комп'ютерах. При цьому прикладні програми користувачів і база даних розміщуються на різних комп'ютерах, як показано на рис. 2.5.

Звернення до файлового сервера здійснюються по мірі необхідності одержання потрібних файлів. Таким чином, файловий сервер функціонує як жорсткий диск, який використовується сумісно

Такий підхід характеризується значним мережним трафіком, що може призвести до суттєвого зниження продуктивності системи в цілому. Це обумовлено збільшенням об'єму даних, які передаються по мережі, тому що вся обробка виконується на комп'ютері користувача.

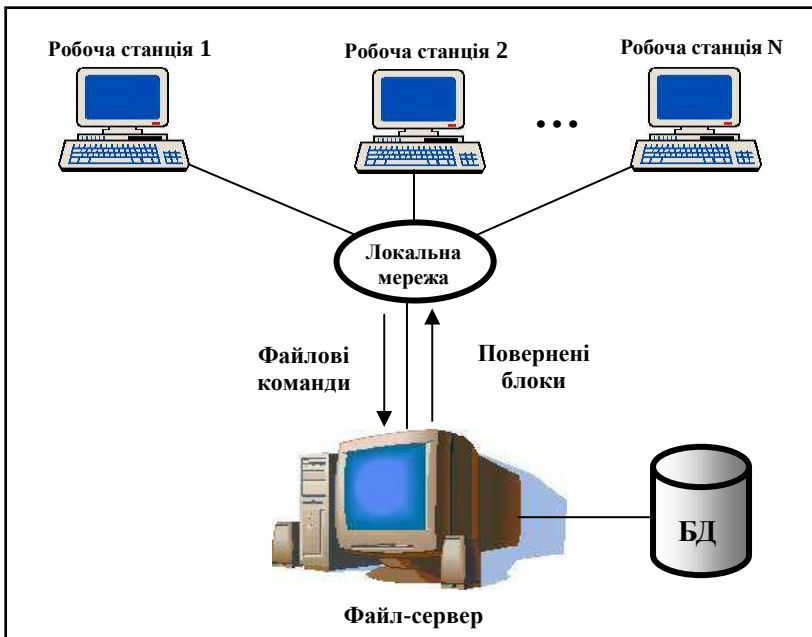


Рисунок 2.5 – Архітектура «файл-сервер»

Наприклад, якщо користувачу потрібні декілька рядків із таблиці об'ємом у сотні тисяч записів, то спочатку вся таблиця з файл-сервера передається на його комп'ютер, а вже потім СКБД відбирає необхідні записи.

Таким чином, архітектура з використанням файлового сервера має наступні основні недоліки:

- великий обсяг мережного трафіку;
- на кожній робочій станції повинна знаходитися повна копія СКБД;
- захист даних ускладнюється, оскільки доступ до одного й того ж файлу даних може здійснювати одночасно декілька примірників СКБД.

В цьому випадку продуктивність системи можливо збільшити за рахунок використання технології клієнт-сервер.

2.4.3 Технологія «клієнт-сервер»

При такому підході система баз даних складається з двох частин: набору клієнтів і сервера (рис. 2.6).

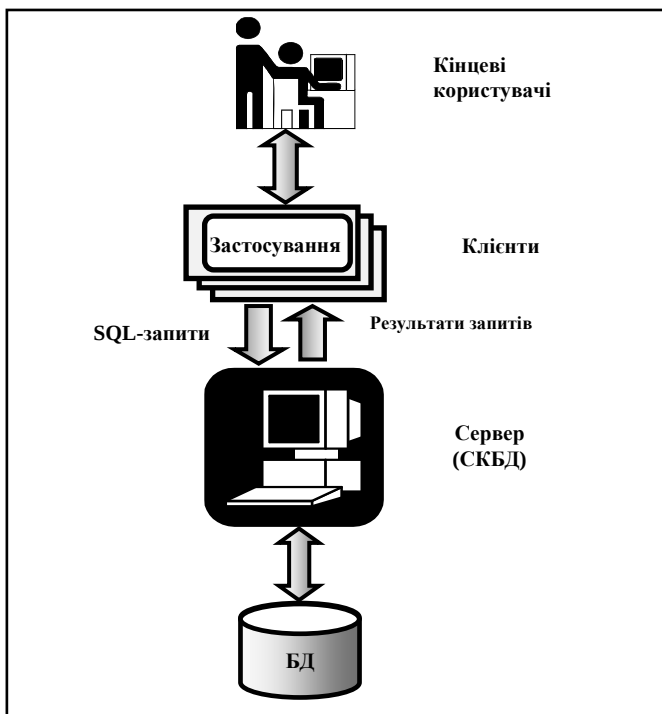


Рисунок 2.6 – Архітектура «клієнт-сервер»

Клієнт (зазвичай називається локальним вузлом – *local node*), керує інтерфейсом користувача та логікою застосувань, діючи як складна робоча станція, на якій виконуються застосування користувача. Клієнт одержує від користувача запит, перевіряє синтаксис, генерує запит до бази даних на мові SQL або іншій мові бази даних і передає його серверу. Отримавши відповідь, клієнт форматує отримані дані для надання користувачеві.

Сервер (зазвичай його називають віддаленим вузлом – *remote node*) – це взагалі СКБД. Він знаходиться на спеціально виділених комп'ютерах і підтримує всі основні функції СКБД: визначення даних, обробка даних, захист і цілісність даних, адміністрування тощо.

Сервер отримує та обробляє запити до бази даних, а потім передає отримані результати клієнту. Така обробка містить перевірку повноважень клієнт, забезпечення вимог цілісності, підтримку системного каталогу, а також виконання запиту та оновлення даних. Крім цього, підтримується керування паралельністю та оновленням (ці питання будуть детально розглянуті у розділі «Захист інформації»).

Операції, що виконуються клієнтом і сервером, наведені у табл. 2.1 [15].

Таблиця 2.1 – Функції клієнта й сервера

Клієнт	Сервер
Керує інтерфейсом користувача	Приймає та обробляє запити з боку клієнта до бази даних
Приймає й перевіряє синтаксис запиту користувача	Перевіряє повноваження користувачів
Виконує застосування	Гарантує дотримання обмежень цілісності
Генерує запит до бази даних і передає його серверу	Виконує запити/оновлення й повертає результати клієнту
Відображає отримані результати користувачу	Підтримує системний каталог Забезпечує паралельний доступ до бази даних Забезпечує управління відновленням

Усі проблеми, що виникають при взаємодії клієнта та сервера, повинен вирішувати спеціальний компонент СКБД, який називається комунікаційним сервером (**Communication Server, DBMS Server Net**). Для підтримки взаємодії клієнта та сервера він повинен функціонувати на віддаленому вузлі. В той же час на локальному вузлі повинна виконуватися програма зв'язку, що взаємодіє з комунікаційним сервером (**DBMS Client Net**).

2.4.4 Поняття технології відкритого доступу до даних

Важливим етапом у побудові додатку клієнт-сервер є встановлення його зв'язку із джерелом даних, що знаходяться на сервері БД. Загальновизнаним стандартом є технологія відкритого доступу до даних **ODBC (Open Database Connectivity)**, запропонована фірмою Microsoft.

Він представляє собою стандарт прикладного програмного інтерфейсу (**Application Programming Interface – API**) і дозволяє програмам, які працюють в середовищі Microsoft Windows, взаємодіяти (за допомогою операторів мови SQL) с різними СКБД, як персональними, так і багатокористувацькими, що функціонують в різних операційних системах.

Фактично, інтерфейс ODBC універсальним чином віддаляє суто прикладний, змістовний бік застосувань (обробка електронних таблиць, статистичний аналіз, ділова графіка) від безпосередньої обробки та обміну даними з СКБД. Основна ціль ODBC – зробити взаємодію застосування та СКБД прозорою, незалежною від класу та особливостей конкретної СКБД. Стандарт ODBC є невід’ємною частиною сімейства стандартів, які полегшують написання застосувань та забезпечують вертикальну відкритість застосувань (**WOSA – Windows Open Services Architecture** – відкрита архітектура сервісів системи Windows).

Інтерфейс ODBC забезпечує взаємну сумісність серверних і клієнтських компонентів доступу до даних. У якості базового стандартного засобу доступу до даних виступає мова структурованих запитів **SQL (Structure Query language)**.

Для реалізації уніфікованого доступу до різних СКБД було введено поняття драйвера ODBC (представляє собою бібліотеку, що динамічно завантажується).

ODBC-архітектура містить чотири компоненти (рис. 2.7):

- застосування;
- менеджер драйверів;
- драйвери;
- джерела даних.

Їхні ролі розподілені наступним чином:

- застосування викликає функції ODBC для виконання SQL-інструкцій, отримує та інтерпретує результати;
- менеджер драйверів завантажує ODBC-драйвери, коли цього вимагає застосування;
- ODBC-драйвери обробляють виклики функцій ODBC, направляють SQL-запити до конкретних джерел даних, а також повертають результат у застосування. При необхідності драйвери виконують модифікацію початкового запиту застосування з метою приведення його у відповідність до синтаксичних вимог цільової СКБД;
- джерела даних (data source) – компонент, який містить ті дані, доступ до яких потрібен користувачеві застосування.

Дії, що виконуються застосуванням, яке використовує інтерфейс ODBC, зводяться до наступного. Для початку сеансу роботи з базою даних застосування повинне підключитися до джерела даних, яке її приховує. Потім застосування звертається до бази даних, надсилаючи

SQL-інструкції, запитує результати, відслідковує та реагує на помилки тощо; тобто має місце стандартна схема взаємодії застосування та сервера БД. Завершуючи сеанс роботи, застосування повинне відключитися від джерела даних.

Технологія ODBC передбачає використання єдиного інтерфейсу доступу до баз даних. У якості базового стандартного засобу доступу до даних виступає мова структурованих запитів **SQL**.

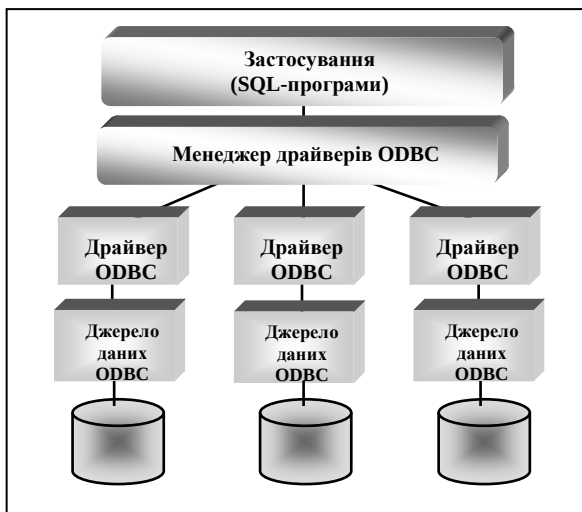


Рисунок 2.7 – Архітектура ODBC

Інтерфейс ODBC забезпечує високий ступінь універсальності, в результаті чого застосування може одержувати доступ до даних, які зберігаються в різних СКБД, без необхідності внесення змін до його програмного тексту. ODBC дозволяє застосуванню клієнта, що написане, наприклад, на **Access** або **Visual FoxPro**, працювати з командами та функціями, які підтримуються сервером.

ODBC знаходиться між застосуванням, яке використовує дані, та самими даними, що зберігаються у форматі, котрий не дозволяє обробити дані у застосуванні напряму.

Одна з головних цілей створення ODBC – приховати складність з'єднання із сервером і по можливості автоматизувати виконання багатьох процедур, пов'язаних із отриманням даних. ODBC вимагає від користувача тільки ім'я джерела даних. При цьому функції драйвера, адреси серверів, мережі та шлюзи приховані від користувача.

2.5 Розподілені бази даних

2.5.1 Поняття розподіленої бази даних

При роботі з базами даних можливі різні варіанти їх використання, що у загальному випадку можливо подати у вигляді схеми (рис. 2.8) [2,35].

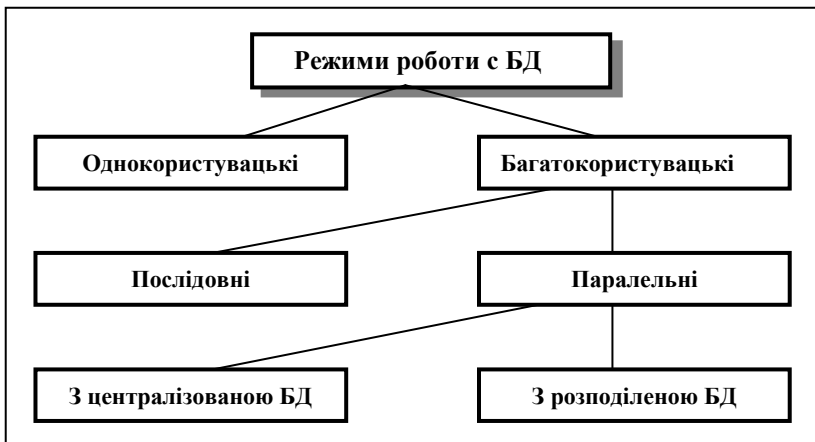


Рисунок 2.8 – Режими роботи с базою даних

У другому розділі даного посібника описувалися переваги централізованого підходу до збереження даних. Але в теперішній час у зв'язку з широким поширенням комп'ютерних мереж намітилася тенденція до децентралізації збереження та обробки даних.

Децентралізований підхід відображає організаційну структуру компанії, що логічно складається з окремих підрозділів, які фізично розподілені по різних офісам, відділенням або філіям, причому кожна окрема структурна одиниця має справу з особистим набором даних, які обробляються.

Розробка розподілених баз даних, які відображають організаційні структури підприємств, дозволяють зробити загальнодоступними дані, що підтримуються кожним підрозділом, забезпечивши при цьому їх зберігання саме в тих місцях, де вони частіше використовуються. Такий підхід розширює можливості спільного використання даних, одночасно підвищуючи ефективність доступу до них.

В централізованих базах даних термінальні користувачі та прикладні програми звертаються з локальних і віддалених терміналів до центральної бази даних, яка розташована на одному комп'ютері. Такий паралельний доступ називається *розподіленою обробкою даних*.

 **Розподілена база даних (РБД)** – це сукупність логічно взаємозв'язаних баз даних, фізично розподілених у комп'ютерній мережі.

У розподіленій БД (**Distributed Database – DDB**) не всі дані зберігаються централізовано; вони розподілені по мережі вузлів, які відділені географічно, але зв'язані комунікаційними лініями. Кожний вузол має свою власну базу даних; крім того, він може звертатися до даних, які зберігаються на інших вузлах.

 **Розподілена СКБД (РСКБД)** – це програмний комплекс, який призначений для керування розподіленими базами даних і який дозволяє зробити розподіленість системи прозорою для користувача.

Прозорість означає незалежність даних у розподілених системах, яка передбачає, що користувач у цій системі працює з розподіленою базою даних як з логічно цілісною сукупністю даних, тобто на його роботу не повинно впливати те, як дані розподілені між вузлами мережі. Отже, в розподіленій системі користувачеві надається логічно цілісне подання фізично розподіленої бази даних.

Серед причин розвитку й застосування розподілених систем баз даних можна виділити наступні.

1. Часто організації мають філії або відділення в різних місцях. Для конкретного вузла мережі може існувати набір даних, який найчастіше (або майже завжди) використовується саме на ньому. Крім того, користувачам цього вузлу іноді можуть вимагатися дані іншого вузла.
2. Оскільки кожний вузол підтримує власну базу даних, досягається швидкий та ефективний доступ до даних, які найбільш часто використовуються. Такі дані можуть використовуватися й іншими вузлами, але зазвичай рідше. Аналогічно, у міру необхідності можна звертатися до даних інших вузлів.

3. Розподілені бази даних можуть підвищити надійність. Якщо комп'ютер одного вузла виходить із ладу, то інша частина мережі зазвичай може продовжувати працювати. Більше того, якщо дані тиражовані (скопійовані) на двох або більше вузлах, то потрібні дані можна одержати з того вузла, що продовжує функціонувати.

4. Коли кожному вузлу дозволено контролювати дані, якими на ньому найчастіше користуються, користувачі більшою мірою задоволені роботою системи бази даних, тому що локальні бази даних здатні точніше відповідати адміністративній структурі.

2.5.2 Стратегії розподілу даних

Перш ніж перейти до розгляду стратегій розподілу даних, коротко охарактеризуємо ідею централізованого збереження. Оскільки дані зберігаються в одному місці, то значно простіше реалізувати проблему забезпечення цілісності та захисту інформації. При централізованій стратегії спрощується технологія створення та ведення файлів БД, оскільки можна скористатися єдиними стандартними процедурами та методами ведення й підтримування БД в актуальному стані. Проектування такої БД досить просте.

Але такий підхід має й певні недоліки. За такої стратегії можуть виникати черги, що призводить до різкого збільшення часу реакції системи. Крім того, витрачається певний час і на процедури, пов'язані з передаванням інформації. Обсяг бази даних обмежений пам'яттю однієї ЕОМ для зберігання даних.

Існує декілька стратегій розподілу даних у РБД [16]: розподілена без дублювання, розподілена з дублюванням і змішана (комбінована).

Розподілена стратегія без дублювання. За такої стратегії визначають дані, які потрібно зберігати в кожному вузлі мережі. При цьому розподілену базу даних проектують як неперетинні між собою підмножини даних, розподілені по вузлах мережі. Проектування даних за такої стратегії є складною задачею.

Ключовим фактором, який впливає на надійність і доступність бази даних, є так звана ***локалізація посилань***. Якщо БД розподілена так, що дані, розміщені в цьому вузлі, викликаються винятково його користувачем, то це свідчить про високий ступінь локалізації посилань.

Якщо подібне розчленування даних здійснити неможливо та для виконання запитів користувача потрібно звертатись за інформацією до інших вузлів, то це свідчить про невисокий ступінь локалізації посилань.

Переваги цієї стратегії полягають у тому, що зменшуються витрати на передавання інформації та вірогідність виникнення черг, коли кілька користувачів одночасно звертаються до одного файлу БД. Але водночас цю стратегію важко контролювати з точки зору дублювання даних, чим ускладнюється реалізація проблеми узгодженості та цілісності даних. Значно складнішими є проблеми адміністрування та підтримування БД даних в актуальному стані.

Розглянута стратегія підходить для тих предметних областей, в яких практично немає дублювання даних у різних вузлах мережі; користувач кожного вузла працює із своїми файлами та досить рідко використовує дані інших вузлів мережі.

Економічні задачі за своїми інформаційними властивостями характеризуються дуже тісними інформаційними взаємозв'язками, тому для даного класу задач реалізація цієї стратегії досить складна, неефективна й недоцільна.

Розподілена стратегія з дублюванням. Ця стратегія полягає в тому, що база даних проектується як при централізованому підході, але фізично дублюється в кожному вузлі мережі. Кожний вузол має свою копію, продубльовану стільки разів, скільки вузлів у мережі. Стратегія розподілу з дублюванням найбільш ефективно розв'язує проблеми доступу та вибірки даних із мінімальними витратами часу. Система досить проста при проектуванні.

Однак разом з перевагами цей підхід характеризується складністю адміністрування та розв'язання проблеми узгодженості файлів БД у різних вузлах мережі. Ця проблема узгодженості досить гостро може постати тоді, коли зв'язок у мережі порушується та в копіях в різних вузлах виникають розбіжності. У цьому разі потрібно розробити спеціальний механізм для узгодження деяких копій БД.

Змішана стратегія розподілу даних поєднує два підходи, пов'язані з розподілом без дублювання та з дублюванням даних, з метою використання їх переваг. Ця стратегія поділяє БД на багато логічних фрагментів, як це зроблено в стратегії розподілу без дублювання.

Крім того, вона повинна дозволяти мати довільну кількість фізичних копій кожною фрагмента. Такий підхід до створення

РБД дає змогу дублювати дані довільну кількість разів і в кожному вузлі; водночас у кожному вузлі може міститися потрібна частина бази даних. Система, побудована за цією стратегією, допускає досить просту реалізацію паралельної обробки даних, що скорочує час відгуку системи. Ця стратегія також забезпечує дуже велику надійність даних, за рахунок дублювання даних їх легко можна відновити при помилках чи збоях обладнання. Однак, як і при стратегії дублювання, виникає проблема узгодженості копій бази даних у всіх вузлах мережі.

2.5.3 Фрагментація даних

Фрагментація даних стосується реляційних систем баз даних. Вона пов'язана з тим, яким чином реляційні таблиці можуть бути розділені та розподілені між вузлами мережі. Це продовження стратегії розчленування даних, яка зазвичай стосується розподілу по вузлах таблиць (файлів) цілком.

Фрагментація реляційної таблиці означає, що вона розділена на підмножини (фрагменти) $R_1, R_2, \dots, R_n$. Об'єднання цих фрагментів складає початкову таблицю R . Існують три основних типи фрагментації: *горизонтальна*, при якій по різних фрагментах розподіляються кортежі (рядки), *вертикальна*, при якій по різних фрагментах розподіляються атрибути (стовпчики) (рис. 2.9) та змішана (рис. 2.10).

Якщо фрагментація можлива, то таблиця необов'язково зберігається цілком в одному вузлі. Її підмножини можуть бути розподілені між декількома вузлами з метою покращення роботи системи. Більш того, ці підмножини можуть тиражуватися.

Наприклад, в базі даних банку може бути доцільним зберігати підмножини БД у тих філіях, де знаходяться відповідні рахунки.

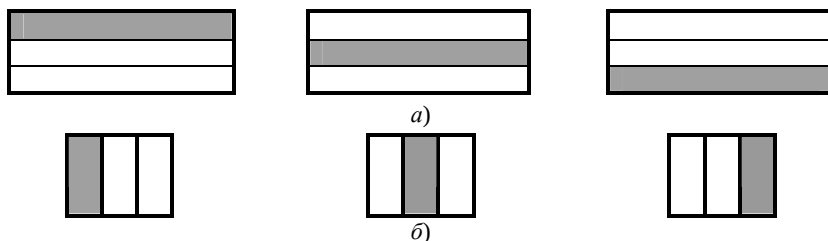


Рисунок 2.9 – Горизонтальна (а) та вертикальна (б) фрагментація

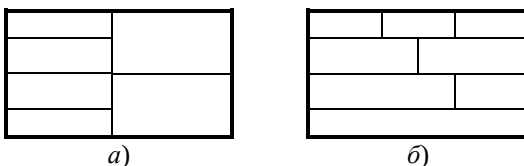


Рисунок 2.10 – Змішана фрагментація:

- a) горизонтально розділені вертикальні фрагменти;
- б) вертикально розділені горизонтальні фрагменти

Якщо фрагментація можлива, то таблиця необов'язково зберігається цілком в одному вузлі. Її підмножини можуть бути розподілені між декількома вузлами з метою покращення роботи системи. Більш того, ці підмножини можуть тиражуватися.

Наприклад, в базі даних банку може бути доцільним зберігати підмножини БД у тих філіях, де знаходяться відповідні рахунки.

Горизонтальна фрагментація



Горизонтальна фрагментація – це розподіл кортежів таблиці за фрагментами.

Зазвичай фрагменти не перетинаються й можуть бути тиражовані. В цьому випадку тиражування керується на рівні фрагментів, а не на рівні окремих кортежів.

Горизонтальний фрагмент створюється шляхом завдання предиката (умови), за допомогою якого виконується відбір кортежів початкового відношення. Цей тип фрагмента визначається за допомогою оператора вибірки **SELECT** над відношенням **R**:

$$R_i = \text{SELECT } (R: \text{<умова на один або декілька атрибутів>})$$

Наприклад, маємо таблицю **Потяги** (табл. 2.2), яку можна розділити на два фрагменти, кожен з яких буде містити відомості про потяги з однаковим пунктом відправлення (Запоріжжя та Київ) (табл. 2.3, табл. 2.4).

Таблиця 2.2 – Відношення Потяги

Станція_відпр	Станція_призн	Час_відпр	Час_прибуття
Запоріжжя	Київ	20.30	6.30
Запоріжжя	Львів	12.40	13.15
Запоріжжя	Москва	17.25	23.15
Київ	Запоріжжя	19.40	6.30
Київ	Москва	12.45	14.00
Київ	Львів	16.10	18.00
Київ	Софія	12.35	23.15

Фрагментація виконується за командами:

Потяги-З = SELECT (Потяги: Станція_відпр = «Запоріжжя»)

Потяги-К = SELECT (Потяги: Станція_відпр = «Київ»)

В результаті отримаємо горизонтальні фрагменти, подані у табл. 2.3 і 2.4.

Таблиця 2.3 – Горизонтальний фрагмент Потяги_З

Станція_відпр	Станція_призн	Час_відпр	Час_прибуття
Запоріжжя	Київ	20.30	6.30
Запоріжжя	Львів	12.40	13.15
Запоріжжя	Москва	17.25	23.15

Таблиця 2.4 – Горизонтальний фрагмент Потяги_К

Станція_відпр	Станція_призн	Час_відпр	Час_прибуття
Київ	Запоріжжя	19.40	6.30
Київ	Москва	12.45	14.00
Київ	Львів	16.10	18.00
Київ	Софія	12.35	23.15

Початкову таблицю **Потяги** можна відновити, використовуючи операцію об'єднання UNION над таблицями **Потяги_З** і **Потяги_К**.

Вертикальна фрагментація



Вертикальна фрагментація – це розділення множини атрибутів таблиці на підмножини, які, можливо, перетинаються.

Фрагменти одержуються проекцією початкової таблиці на кожну підмножини атрибутів. Для виключення можливості втрати даних кожний вертикальний фрагмент повинен включати ключ таблиці.

Вертикальна фрагментація задається наступним чином:

$$R_i = \Pi R(<\text{список імен атрибутів}>).$$

Відношення R можна реконструювати з фрагментів, використовуючи операцію з'єднання:

$$R = \text{JOIN} (R_1, R_2, \dots, R_n).$$

В табл. 2.4 доданий ключовий атрибут **Ном_потягу**. В табл. 2.5 і 2.6 показані вертикальні фрагменти **Потяги_відправлення** та **Потяги_прибуття**. Ці таблиці одержуються в результаті такого розбиття:

Потяги_відпр = Потяги (Ном_потягу, Станція_відпр, Час_відпр)

Потяги_приб = Потяги (Ном_потягу, Станція_призн, Час_прибуття)

Таблиця 2.4 – Розширене відношення Потяги

Ном_потягу	Станція_відпр	Станція_призн	Час_відпр	Час_прибуття
1	Запоріжжя	Київ	20.30	6.30
2	Запоріжжя	Львів	12.40	13.15
3	Запоріжжя	Москва	17.25	23.15
4	Київ	Запоріжжя	19.40	6.30
5	Київ	Москва	12.45	14.00
6	Київ	Львів	16.10	18.00

Таблиця 2.5 – Вертикальний фрагмент Потяги_відпр

Ном_потягу	Станція_відпр	Час_відпр
1	Запоріжжя	20.30
2	Запоріжжя	12.40
3	Запоріжжя	17.25
4	Київ	19.40
5	Київ	12.45
6	Київ	16.10

Таблиця 2.6 – Вертикальний фрагмент Потяги_приб

Ном_потягу	Станція_призн	Час_прибуття
1	Київ	6.30
2	Львів	13.15
3	Москва	23.15
4	Запоріжжя	6.30
5	Москва	14.00
6	Львів	18.00

Початкова таблиця Потяги відновлюється командою:

$$\text{Потяги} = \text{JOIN} (\text{Потяги_відпр}, \text{Потяги_приб})$$

2.5.4 Гомогенні та гетерогенні розподілені СКБД

Розподілені СКБД можна класифікувати як гомогенні та гетерогенні. В гомогенних системах всі вузли використовують один і той же тип СКБД. В гетерогенних системах у вузлах можуть функціонувати різні типи СКБД, які використовують різні моделі даних, тобто гетерогенна система може включати вузли з реляційними, мережними, ієрархічними або об'єктно-орієнтованими СКБД.

Гомогенні системи значно простіше проектувати та супроводжувати. Крім того, подібний підхід дозволяє поетапно нарощувати розміри системи, послідовно додаючи нові вузли до вже існуючої розподіленої системи. Додатково з'являється можливість підвищувати продуктивність системи за рахунок організації на різних вузлах паралельної обробки інформації.

Гетерогенні системи зазвичай виникають в тих випадках, коли незалежні вузли, що вже експлуатують свої власні системи з базами даних, інтегруються до розподіленої системи, що наново створюється. В гетерогенних системах для організації взаємодії між різними типами СКБД буде потрібно організувати трансляцію повідомлень, які передаються. Для забезпечення прозорості відносно типу СКБД, яка використовується, користувачі кожного з вузлів повинні мати можливість вводити запити на мові тієї СКБД, яка використовується на даному вузлі. Система повинна взяти на себе локалізацію необхідних даних і виконання трансляції повідомлень, які передаються. В загальному випадку дані можуть бути запитані з іншого вузла, який характеризується такими особливостями, як:

- інший тип обладнання, що використовується;
- інший тип СКБД, що використовується;
- інший тип обладнання та СКБД.

Якщо використовується інший тип обладнання, проте на вузлі встановлений такий же тип СКБД, методи виконання трансляції цілком очевидні та включають заміну кодів і зміну довжини слова.

Якщо типи СКБД, які використовуються на вузлах, різні, то процедура трансляції ускладнюється тим, що необхідно

відображати структури даних однієї моделі у відповідні структури даних іншої моделі. Наприклад, відносини в реляційній моделі даних повинні бути перетворені в записи та набори, типові для мережної моделі даних. Крім того, буде потрібно транслювати текст запитів з однієї мови, що використовується, на іншу (наприклад, запити з SQL-оператором SELECT буде потрібно перетворити в запити з операторами FIND і GET мови маніпулювання даними мережної СКБД).

Якщо відрізняються і тип обладнання, що використовується, й тип програмного забезпечення, то буде потрібно виконувати обидва види трансляції. Все це надзвичайно ускладнює обробку даних в гетерогенних СКБД.

Додаткові складнощі виникають, при спробах розробки єдиної концептуальної схеми, що створюється шляхом інтеграції незалежних локальних концептуальних схем.

Типове рішення, що використовується в деяких реляційних системах, полягає в тому, що окремі частини гетерогенних розподілених систем повинні використовувати *шлюзи*.

 **Шлюз** – сукупність програмних засобів, які використовуються у гетерогенних розподілених СКБД і призначені для перетворення мови та моделі даних кожного типу СКБД, яка використовується, в мову та модель даних реляційної системи.

Проте підходу з використанням шлюзів властиві деякі серйозні обмеження. По-перше, шлюзи не дозволяють організувати систему управління транзакціями навіть для окремих пар систем. Іншими словами, шлюз між двома системами є не більше ніж транслятором запитів. Наприклад, шлюзи не дозволяють системі координувати управління паралельністю та процедурами відновлення транзакцій, які включають оновлення даних в обох базах.

По-друге, використання шлюзів покликане лише вирішити задачу трансляції запитів з мови однієї СКБД на мову іншої СКБД. Тому вони не дозволяють вирішити проблеми, викликані неоднорідністю структур і представленням даних в різних схемах.

2.5.5 Забезпечення прозорості в розподілених СКБД

Під прозорістю розуміється приховування від користувачів деталей реалізації системи. Розподілені СКБД можуть забезпечувати різні рівні прозорості. Проте разом в якій-небудь одній РСКБД вони дуже рідко зустрічаються.

Можна виділити чотири основні типи прозорості [2]:

- прозорість розподіленості;
- прозорість транзакцій;
- прозорість виконання;
- прозорість використання СКБД.

2.5.5.1 Прозорість розподіленості

Прозорість розподіленості бази даних дозволяє кінцевим користувачам сприймати базу даних як єдине логічне ціле. При цьому користувачу не треба яких-небудь знань про фрагментацію даних (прозорість фрагментації) або їх розміщення (прозорість розташування).

Прозорість фрагментації

Прозорість фрагментації є найвищим рівнем прозорості розподіленості. Якщо забезпечується прозорість фрагментації, то користувачу не треба знати, як саме були фрагментовані дані. В цьому випадку доступ до даних здійснюється на основі глобальної схеми й користувачу немає необхідності вказувати імена фрагментів або розташування даних.

Наприклад, для вибірки відомостей про всіх керівників відділень (у них атрибут ***Посада*** має значення ***Менеджер***), за наявності в системі прозорості фрагментації, можна скористатися наступним SQL-оператором:

```
SELECT Прізвище  
FROM Співробітники  
WHERE Посада = «Менеджер»;
```

Це той же самий SQL-оператор, який було б потрібно ввести для отримання вказаних результатів в централізованій системі.

Прозорість розташування

Прозорість розташування є середнім рівнем прозорості розподіленості. В цьому випадку користувач повинен мати відомості про способи фрагментації даних в системі, але не потрібно знати про розташування даних. В цьому випадку в запиті необхідно вказувати імена фрагментів, що використовуються.

Основна перевага прозорості розташування полягає в тому, що база даних може бути піддана фізичній реорганізації, але це не повинно впливати на програми, що здійснюють до неї доступ.

Прозорість реплікації

З прозорістю розташування дуже тісно зв'язаний ще один тип прозорості – прозорість реплікації. Він означає, що користувачу не потрібно мати відомості про існуючу реплікацію (тиражування) фрагментів. Під прозорістю реплікації мається на увазі прозорість розташування реплік. Проте можуть існувати системи, що не забезпечують прозорості розташування, але підтримують прозорість реплікації.

Прозорість локального відображення

Якщо користувачу необхідно мати відомості про фрагментацію даних і розташування фрагментів, то цей тип прозорості будемо називати *прозорістю локального відображення*.

Це найнижчий рівень прозорості розподіленості. При наявності в системі прозорості локального відображення користувачу необхідно вказувати як імена фрагментів, що використовуються, так і розташування відповідних елементів даних.

Прозорість іменування

Прямим наслідком варіантів прозорості розподіленості, що обговорювалися вище, є вимога наявності прозорості іменування. Як і у разі централізованої бази даних, кожний елемент розподіленої бази даних повинен мати унікальне ім'я. Отже, РСКБД повинна гарантувати, що ніякі два вузли системи не зможуть створити деякий об'єкт бази даних, що має одне й те ж ім'я.

Одним з варіантів рішення цієї проблеми є створення центрального серверу імен, який нестиме відповідальність за повну унікальність всіх існуючих в систем імен. Проте подібному підходу властиві такі недоліки, як:

- втрата певної частини локальної автономії;
- поява проблем з продуктивністю (оскільки центральний вузол перетворюється на вузьке місце всієї системи);
- зниження доступності. Якщо центральний вузол з якоїсь причини стане недоступним, вся решта вузлів системи не зможе створювати ніяких нових об'єктів бази даних.

Альтернативне рішення полягає у використанні префіксів, які входять в імена об'єктів як ідентифікатор вузла, що створив цей

об'єкт. Наприклад, відношення **ТОВАРИ**, створене на вузлі **T1**, могло б отримати ім'я **T1.ТОВАРИ**. Аналогічним чином, необхідно мати нагоду ідентифікувати кожний фрагмент і кожен його копію. Тому другої копії третього фрагмента відношення **ТОВАРИ**, створеного на вузлі **T1**, можна було б привласнити ім'я **T1.ТОВАРИ.F3.C2**. Проте подібний підхід призводить до втрати прозорості розподіленості.

2.5.5.2 Прозорість транзакцій

Прозорість транзакцій в середовищі розподілених СКБД означає, що при виконанні будь-яких розподілених транзакцій гарантується збереження цілісності й узгодженості розподіленої бази даних. Розподілена транзакція здійснює доступ до даних, що зберігаються більш ніж в одному вузлі. Кожна з транзакцій розділяється на декілька субтранзакцій – по одній для кожного вузла, до даних якого здійснюється доступ. На віддалених вузлах субтранзакції представляються агентами.

Неподільність залишається фундаментальною концепцією поняття транзакції й у разі розподілених транзакцій, проте додатково РСКБД повинна гарантувати неподільність і кожної з її субтранзакцій. Отже, РСКБД повинна гарантувати не тільки синхронізацію субтранзакцій з іншими локальними транзакціями, що виконуються паралельно з ними на одному вузлі, але й забезпечити синхронізацію субтранзакцій з глобальними транзакціями, що виконуються одночасно з ними на цьому та інших вузлах системи.

Прозорість транзакцій в розподілених СКБД додатково ускладнюється за рахунок наявності фрагментації, розподілу даних і використання реплікації.

2.5.5.3 Прозорість виконання

Прозорість виконання вимагає, щоб робота в середовищі РСКБД виконувалася точно так, як і в середовищі централізованої СКБД. В розподіленому середовищі робота системи не повинна демонструвати ніякого зниження продуктивності, пов'язаного з її розподіленою архітектурою, наприклад, з присутністю повільних мережних з'єднань.

Прозорість виконання також вимагає, щоб РСКБД була здатна знаходити найефективніші стратегії виконання запитів. В централізованій СКБД обробник запитів повинен оцінювати

кожний запит на доступ до даних і знаходити оптимальну стратегію його виконання у вигляді впорядкованої послідовності операцій з базою даних.

В розподіленому середовищі обробник розподілених запитів відображає запит на доступ до даних у впорядковану послідовність операцій локальних баз даних. Додаткова складність виникає через необхідність враховувати наявність фрагментації, реплікації та певної схеми розміщення даних. Обробник розподілених запитів повинен з'ясувати:

- до якого фрагмента слід звернутися;
- яку копію фрагмента використовувати, якщо його дані реплікуються;
- яке з місцеположень повинне використовуватися.

Обробник розподілених запитів виробляє стратегію виконання, що є оптимальною з погляду деякої вартісної функції. Зазвичай розподілені запити оцінюються за такими показниками:

- час доступу, що включає фізичний доступ до даних на диску;
- час роботи центрального процесора, що затрачений на обробку даних в первинній пам'яті;
- час, необхідний для передачі даних по мережним з'єднанням.

Перші два чинники аналогічні тим, які враховуються в централізованих системах. Проте в розподіленому середовищі РСКБД необхідно враховувати й витрати на передачу даних, які у багатьох випадках серед інших витрат виявляються домінуючими.

2.5.5.4 Прозорість використання СКБД

Прозорість використання СКБД дозволяє приховати від користувача РСКБД той факт, що на різних вузлах можуть функціонувати різні локальні СКБД. Тому даний тип прозорості застосовується тільки у разі гетерогенних розподілених систем. Як правило, це один з найскладніших в реалізації типів прозорості.

2.5.6 Загальна модель розподіленої СКБД

Розподілена СКБД складається з декількох вузлів, кожний з яких працює з локальною базою даних у тих видах діяльності, де потрібні тільки **локальні дані**. Крім того, кожний вузол може обробляти транзакції, для яких необхідні дані, що зберігаються на інших вузлах (**глобальні дані**).

Для цього необхідно, щоб різні вузли бази даних могли передавати один одному дані. Комунікаційні лінії, що забезпечують необхідні можливості обміну даними, називаються **з'єднаннями**. Структура з'єднань утворює базову архітектуру розподіленої системи керування базою даних (РСКБД), тобто системного програмного забезпечення, що управляє розподіленою базою даних.



Локальні дані – це дані, що підтримуються у власній базі даних вузла мережі.



Глобальні дані – це дані, що підтримуються в базі даних, розташування якої відрізняється від розташування хоча б одного з її користувачів.



Розподілена СКБД – це система, що складається з декількох СКБД, що працюють на локальних вузлах, з'єднаних засобами обробки повідомлень.

Словник даних РСКБД містить звичайну інформацію, необхідну для керування даними, а також інформацію, що стосується місцезнаходження, тиражування й фрагментації даних різних реляційних таблиць. При обробці запитів, які задають вибірку або оновлення даних, словник даних РСКБД може надати потрібну інформацію про розташування та копії даних, стежачи при цьому, щоб зміни даних були передані на всі відповідні вузли. Словник даних РСКБД може підтримуватися в центральному вузлі, або ж його підмножини можуть бути розподілені між різними вузлами, якщо буде потреба. Повну копію словника даних можна одержати, об'єднавши всі розподілені підмножини.

Користувач взаємодіє із РСКБД, запускаючи програми, які називаються транзакціями. **Транзакція** в таких системах більше не зводиться до одного процесу, що контролюється одним програмним модулем; вона може викликати кілька процесів у різних вузлах, які контролюються незалежними програмними модулями.



Агент – це процес, який приймає участь у виконанні транзакції.



Локальна транзакція – це транзакція, що складається з одного агента.



Глобальна транзакція – це транзакція, що складається з декількох агентів.

Конкретний агент може звертатися тільки до даних, які контролює його локальне програмне забезпечення. Доступ до даних інших вузлів вимагає взаємодії з агентами цих вузлів. Агент, який починає транзакцію, називається *ініціюючим агентом*. Ініціюючий агент може вимагати активації агентів на інших вузлах з метою доступу до потрібних даних. Будучи активізованими, два або більше агентів можуть обмінюватися повідомленнями.

Транзакції звертаються до записів операціями **read** (читати) та **write** (записати). **Read(x)** повертає поточне значення *x*. **Write(x)** (*x* - нове значення) заміняє поточне значення *x* на нове значення. Транзакція передає команди читання й запису до РСКБД і виконує кінцеві введення та виведення даних.

На кожному вузлі РСКБД зазвичай запускається один або декілька з наступних програмних модулів: диспетчер транзакцій (ДТ), диспетчер даних (ДД) або планувальник. На рис. 2.11 представлені їхні взаємини. Транзакції спілкуються із ДТ; ДТ обмінюється інформацією з планувальником; планувальники спілкуються між собою та з ДД; ДД керує даними. Послідовність обробки операцій наведена на рис. 2.12.

Кожна транзакція повідомляє про всі свої операції читання й запису одному ДТ. Транзакція також на початку виконання повідомляє своєму ДТ операцію **begin** і по закінченні виконання – операцію **end**.

ДТ повідомляє про кожну операцію читання або запису планувальникові. Вибір планувальника визначається алгоритмом контролю паралельної обробки; однак, частіше за все вибирається планувальник, який перебуває на тій же ділянці, що й дані, які обробляються.

Планувальник керує послідовністю, у якій ДД виконує команди читання й запису, а також робить контроль паралельної обробки. Коли планувальник одержує команду читання або запису, він може виконати її негайно; відкласти виконання, утримуючи команду; відхилити команду у випадку помилки передачі, порушення доступу й подібних проблем.

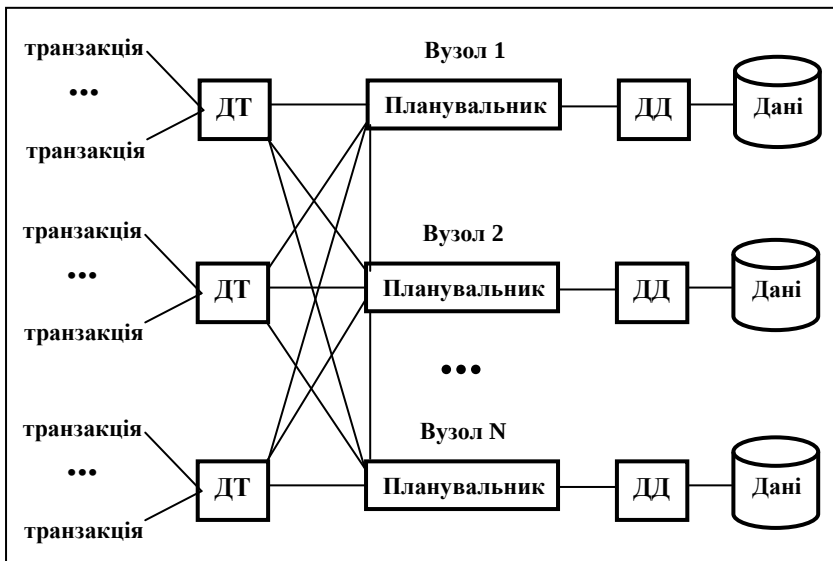


Рисунок 2.11 – Архітектура розподіленої СКБД

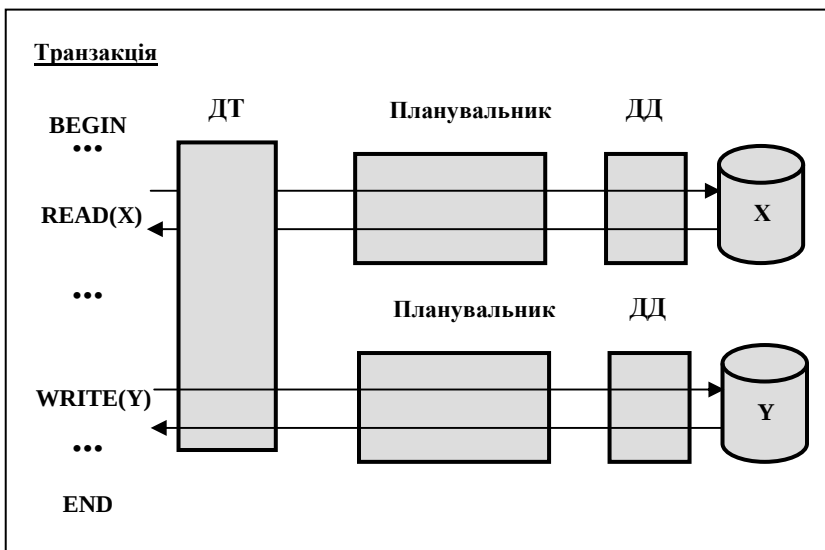


Рисунок 2.12 – Обробка операцій

ДД виконує кожну одержану команду читання або запису. По команді читання ДД переглядає локальну базу даних і повертає необхідне значення. По команді запису ДД змінює локальну базу даних і повертає підтвердження планувальникові. Планувальник передає його назад ДТ, який, у свою чергу, передає його назад транзакції.

Кожний вузол у розподіленій системі бази даних повинен бути здатний обробляти як транзакції, що звертаються до даних тільки цього вузла, так і транзакції, що вимагають даних інших вузлів.

2.5.7 Підтримка цілісності даних в розподілених базах даних

В главі 6 розглядалася проблема захисту даних стосовно централізованих баз даних. В РБД внаслідок розподілення даних по окремим вузлам проблема підтримки цілісності дещо ускладнюється.

У розподілених системах виділяють чотири типи збоїв:

- збій транзакції,
- збій вузла,
- збій носія (диска),
- збій комунікаційної лінії.

Збій транзакції може бути спричинений помилками у вхідних даних і виявленням тупику. У всіх перелічених ситуаціях збою транзакцію необхідно перервати та виконати відкат бази даних. Для попередження «зависання» системи необхідний контроль за часом виконання транзакції. Якщо ліміт часу, відведеного для виконання транзакції, перевищено, то її потрібно відмітити як таку, що підлягає аварійному завершенню незалежно від результату початкових кроків, і всі файли, задіяні при її виконанні, перевести в початковий стан.

Для підтримання цілісності бази даних при різного роду збоях використовують протоколи журналізації, що вміщують інформацію про всі зміни, які вносяться до бази даних під час виконання транзакції.

В одну транзакцію доцільно об'єднувати операції, що виконують логічно зв'язані між собою зміни файлів бази даних.

Підтримка транзакцій – це основа забезпечення цілісності БД.

У сучасних розподілених СКБД підтримку транзакцій можна виконувати за допомогою двох методів:

1. двофазною фіксацією транзакцій;
2. реплікацією (дублюванням) даних.

Ці два методи відрізняються тим, що в першому випадку багато транзакцій працюють одночасно з однією розподіленою базою даних. У другому випадку кожна транзакція має змогу копіювати необхідні для її виконання дані з розподіленої бази даних і працювати з нею як із своєю особистою базою даних, модифікувати ці дані, а потім після виконання транзакції повертати їх до розподіленої бази даних.

Двофазна фіксація транзакцій

При використанні цього методу транзакцію виконують у два етапи, спираючись на такі правила:

1. Якщо хоча б один вузол не може з будь-яких причин зафіксувати транзакцію, то розподілена транзакція припиняється на всіх інших вузлах, які беруть участь у її виконанні.

2. Якщо всі вузли погоджуються з фіксацією транзакції, то вона фіксується на всіх вузлах, які беруть участь в її реалізації.

Перший етап, або перша фаза, полягає в тому, що на вузлі, де виконуватимуть транзакцію, створюють **процес-координатор**, а на всіх інших вузлах – **процеси-учасники**. Спочатку процес-координатор розсилає повідомлення учасникам про початок транзакції, та кожний з них самостійно вирішує, чи може транзакція бути завершеною на даному вузлі. Учасники, що готові завершити транзакцію, надсилають повідомлення на вузол-координатор про свою готовність до завершення транзакції. Учасники, що не зможуть зафіксувати свою частину транзакції, надсилають повідомлення про те, що транзакцію буде перервано.

На *другому етапі* координатор, зібравши повідомлення всіх учасників, вирішує долю транзакції згідно з правилами. Якщо всі учасники надіслали згоду, координатор надсилає повідомлення про глобальну фіксацію транзакції, згідно з яким вносяться зміни на всіх вузлах. При відмові хоча б одного з учасників приймається рішення про припинення виконання транзакції та надсилається повідомлення про глобальне припинення транзакції.

Ця ситуація називається одностороннім припиненням виконання транзакції. В результаті на всіх вузлах транзакція завершується однаково (фіксується або переривається). Якщо всі вузли мають змогу зафіксувати транзакцію, то всі перетворення виконують на всіх вузлах; якщо хоча б один вузол не в змозі зафіксувати результати транзакції, то її припиняють на всіх вузлах і виконують відкат.

Протоколом обслуговування транзакцій передбачається обмін повідомленнями між координатором і учасниками двічі, тому цей механізм отримав назву двофазної фіксації транзакцій.

Механізм двофазного виконання транзакцій має ряд недоліків: одночасне захоплення всіх ресурсів може досить надовго заблокувати доступ до даних. Крім того, вихід з ладу під час виконання транзакції того вузла мережі, що заблокував ресурси, призведе до блокування деяких даних усієї мережі доти, доки не буде відновлено його роботу.

Основні проблеми, що можуть виникнути при двофазній фіксації транзакцій – незабезпечення коректної поведінки системи в разі виходу з ладу сервера чи пошкодження ліній зв'язку. Тому організація розподіленої системи в даному варіанті потребує досить надійних і швидких ліній зв'язку.

Іншою проблемою, що досить часто виникає при такій організації підтримування транзакцій, є вирішення конфліктів, коли одна транзакція вступає в конфлікт з іншою транзакцією з приводу доступу до певних даних. Тому системи, що базуються на двофазній фіксації транзакцій повинні мати якийсь механізм вирішення конфліктів, пов'язаних з одночасним доступом.

Реплікація даних

Суть цього механізму полягає в тому, що необхідні для виконання транзакції дані копіюють на той сервер, де їх оброблятимуть. Усі зміни, що внесені іншими користувачами протягом виконання запиту, не впливають на його виконання, оскільки вони фіксуються в основних файлах і не відображаються в їх копіях. Такий механізм дає змогу завершити транзакцію з ланцюжком пошукових запитів будь-якої довжини, не порушивши логічної цілісності даних, а також є засобом уникання конфліктів під час роботи з базою даних.

Наприклад, кілька користувачів можуть одночасно модифікувати одні й ті самі дані, не очікуючи один одного. Але коли змінені дані будуть повернені до основної бази даних, то вона може мати кілька версій, тому для таких систем потрібен механізм управління версіями.

Після успішного завершення транзакції зміни вносять на всі сервери у відповідні файли, які брали участь у виконанні транзакції. При цьому виникає потреба якимсь чином забезпечити синхронізацію процесу тиражування змін (реплікації) на різних серверах, щоб база даних не втратила своєї цілісності. Можливі

два режими тиражування змін: синхронний і асинхронний.

Синхронна реплікація – це процес, при якому зміни поширюються одразу по всіх вузлах після їх виконання. Синхронний варіант внесення змін слід використовувати в тих інформаційних системах, які повинні функціонувати в режимі реального часу. Прикладом таких систем може бути автоматизована банківська система, в якій зміни залишку рахунків повинні відображатись одразу, щоб не було можливості виконати з ними якісь несанкціоновані дії.

Цей варіант виконання транзакції порівняно з їх двофазною фіксацією має деякі переваги. По-перше, запит виконують на одному сервері, при цьому не потрібно передавати дані та розподіляти виконання запиту між серверами. По-друге, хоч і потрібно тиражування змін даних, які виконала транзакція, але ці зміни вносять асинхронно щодо самої транзакції; отже, користувач отримує результат виконання запиту швидше, оскільки йому не потрібно очікувати закінчення обміну даними між серверами по внесенню змін.

Ще одна важлива перевага синхронного тиражування – підтримування «дзеркальної» бази даних на резервному сервері, причому обидва сервери (основний і резервний) в цьому випадку можуть працювати паралельно.

При **асинхронній реплікації** зміни запам'ятовуються, а момент їх передачі вибирають за якимсь певним правилом. При такому способі підтримки логічної цілісності розподіленої БД спостерігається деяка розсинхронізація стану локальних баз даних, яка полягає в тому, що зміни, внесені до деяких локальних баз, відстають за часом їх внесення одна від одної. Інтервал часу, через який поновлюються дані в інших вузлах, має узгоджуватися з інтервалом розв'язання задачі в цих вузлах.

Асинхронне дублювання має недолік, оскільки не забезпечує виконання процедур поновлення даних у масштабі реального часу. Цей режим можна використовувати в таких інформаційних системах, у яких складно підтримувати постійний зв'язок між серверами, та термін тимчасової неузгодженості стану бази даних не конфліктує з періодичністю розв'язання задач у цій системі.

Порівняльна характеристика розглянутих методів підтримки транзакцій в розподіленій БД [15] наведена в табл. 2.7.

Таблиця 2.7 – Порівняння методів підтримки транзакцій в ЦБД

Методи	Двофазна фіксація транзакцій	Реплікація
Характеристики	Однорангові сервери Зміни вносяться у реальному масштабі часу Синхронні зміни Блокування нових транзакцій на час внесення змін	Сервери ведучий/підлеглий Зміни вносяться у близькому до реального режимі часу Асинхронні зміни Безперервна робота сервера
Результати	Інформаційна система має обмежені можливості управління внесенням змін	Інформаційна система має можливості управління синхронізацією та трафіком мережі

2.6 Життєвий цикл інформаційної системи

Інформаційна система (ІС) є набором ресурсів, які дозволяють збирати, підтримувати актуальність, контролювати й поширювати інформацію усередині організації.

База даних є фундаментальним компонентом інформаційної системи, а її розробку й використання варто розглядати з погляду можливого розширення організації. Отже, життєвий цикл інформаційної системи організації нерозривно пов'язаний з життєвим циклом бази даних.

Основні етапи життєвого циклу бази даних включають (рис. 2.13):

- планування розробки бази даних;
- визначення вимог до системи;
- збір і аналіз вимог користувачів;
- проектування бази даних;
- вибір СКБД (якщо буде потрібно);
- розробку застосунків (applications);
- створення прототипів (при необхідності);
- реалізацію;
- конвертування та завантаження даних;
- тестування;
- експлуатацію й супровід.

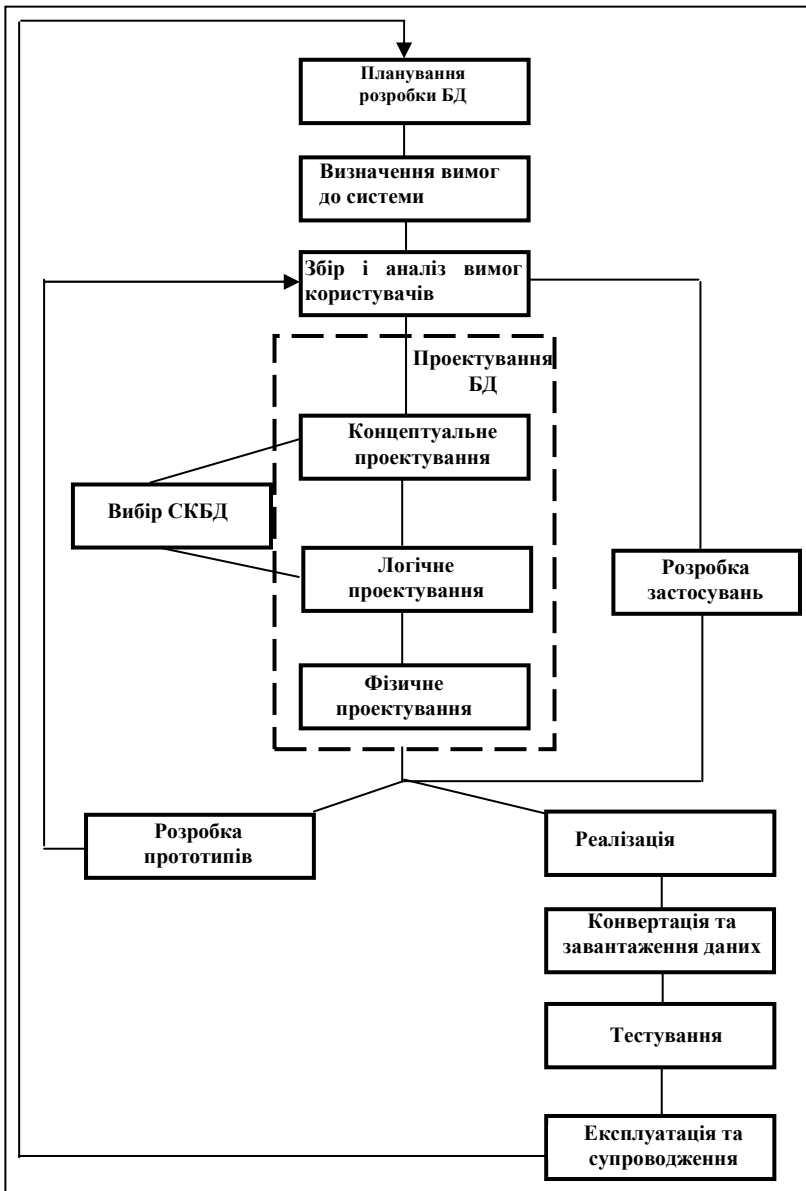


Рисунок 2.13 – Життєвий цикл інформаційної системи

Планування розробки бази даних – це сукупність організаційних дій, які дозволяють із максимальною ефективністю реалізувати етапи створення додатку бази даних.

Визначення вимог до системи включає визначення предметної області й границь застосування бази даних, у тому числі основних областей його використання та користувальницьких груп.

Збір і аналіз вимог користувачів являє собою процес збору й аналізу інформації про ту частину організації, що буде обслуговуватися створюваним застосуванням баз даних, а також використання цієї інформації для визначення вимог користувачів до нової системи.

Проектування бази даних включає створення проекту БД, призначеної для підтримки функціонування організації й досягнення її бізнес-цілей. Цей етап охоплює:

- *концептуальне проектування бази даних* (розділ 3), яке являє собою процес створення моделі використання інформації в організації, що не залежить від всіх фізичних подробиць її подання;

- *логічне проектування бази даних* – процес створення моделі використання інформації в організації, побудованої з урахуванням обраної моделі подання даних у базі, але незалежно від подробиць фізичної реалізації;

- *фізичне проектування бази даних* – процес створення опису реалізації бази даних у зовнішній пам'яті. Воно включає визначення обраних структур зберігання й методів доступу, що забезпечують ефективну обробку даних.

Вибір СКБД передбачає вибір найбільш прийнятної СКБД, яка буде використовуватися для підтримки застосувань БД.

Розробка застосувань включає проектування інтерфейсу користувача й складних програм, які обробляють інформацію в базі даних.

Створення прототипів означає побудову робочих моделей застосувань БД, які дозволяють проектувальникам і майбутнім користувачам візуально ознайомитися з системою та оцінити її можливості.

Реалізація включає фізичне втілення розробленої бази даних і її застосувань.

Конвертування й завантаження даних передбачає перенос будь-яких існуючих даних у нову базу даних і додання будь-яким існуючим застосуванням виду, придатного для роботи з новою базою даних.

Тестування являє собою процес виконання прикладних програм з метою пошуку помилок. Для оцінки повноти й коректності створених застосувачів баз даних можуть використовуватися різні стратегії тестування: спадне тестування, висхідне тестування, тестування потоків і інтенсивне тестування.

Експлуатація й супровід полягає в промисловому використанні створеної системи, що супроводжується постійною перевіркою її поточних показників функціонування, а також необхідною підтримкою.

Стадія експлуатації може включати вдосконалення системи та розробку нових застосувачів у випадку розвитку та зміни предметної області. При цьому інформаційна система повинна мати відкриту архітектуру, що дозволяє відсунути термін її морального старіння.



Питання для самоперевірки

1. Назвіть основні компоненти інформаційного забезпечення та їхнє призначення.
2. Які основні функції виконує СКБД?
3. Які лінгвістичні засоби входять до складу СКБД, їхнє призначення?
4. Для чого призначений словник даних?
5. У чому полягає функція адміністрування бази даних?
6. Чим відрізняються технології файл-сервера від клієнт-сервера?
7. Для чого призначена технологія відкритого доступу до даних?
8. Що називається розподіленою БД?
9. Що називається розподіленою СКБД?
10. Які існують стратегії розподілу даних?
11. Що таке фрагментація даних, які види фрагментації існують?
12. Чим відрізняються гомогенні та гетерогенні системи?
13. Що таке прозорість розподіленої СКБД, які типи прозорості існують?
14. Чим відрізняються локальні та глобальні дані?
15. Які існують методи підтримки цілісності даних в розподілених базах даних?
16. Які основні етапи життєвого циклу інформаційної системи?

3 КОНЦЕПТУАЛЬНЕ ПРОЕКТУВАННЯ БАЗ ДАНИХ

3.1 Інструментальні засоби концептуального проектування

При розробці БД можна виділити декілька рівнів моделювання:

- сама предметна область;
- модель предметної області (концептуальна модель);
- логічна модель даних;
- фізична модель даних;
- безпосередньо база даних і додатки.

На етапі концептуального проектування у якості неформальної моделі ПО використовується модель типу «*сутність – зв’язок*» (**ER – Entity Relation**), що була запропонована Ченом [10] у 1976 р. Конструктивними елементами моделі служать *сутності*, *атрибути* та *зв’язки*. Основним конструктивним елементом являється сутність.

Користувач описує ті об’єкти ПО, що його зацікавили, за допомогою сутностей, потім визначає властивості сутностей, використовуючи атрибути, та, нарешті, описує відповідності між сутностями за допомогою зв’язків.

3.1.1 Сутності та атрибути

Сутність – це збиральне поняття, деяка абстракція реально існуючого об’єкта навколишнього світу, процесу чи явища.

Згідно з Ченом, об’єкти, що описуються сутностями, можна підрозділити на *правильні об’єкти* та *слабкі об’єкти*. Слабким називається об’єкт, який знаходиться в залежності від деякого іншого об’єкта, тобто він не може існувати, якщо не існує цей інший об’єкт. Наприклад, сутність **ПОСТАЧАЛЬНИК** являється правильним об’єктом, а **ПОСТАЧАЄМИЙ_ТОВАР** – слабким об’єктом. Правильні об’єкти називають також *батьківськими* або *основними*, а слабкі – *підкореними*.

Із сутністю пов’язані два поняття: тип сутності та екземпляр сутності. *Тип сутності* визначає набір однорідних об’єктів, що виступають як єдине ціле, а *екземпляр сутності* відноситься до

конкретного об'єкта у наборі. Наприклад, типом сутності може бути **КОМПЛЕКТУЮЧИ**, а екземплярами – «процесор», «материнська плата» тощо. Кожний тип сутності повинен бути пойменованим.

Атрибут – це пойменована характеристика сутності. Атрибут являється засобом, за допомогою якого визначаються властивості сутності. Атрибути мають сенс тільки по відношенню до сутностей, яких вони визначають.

Для ідентифікації конкретних екземплярів сутностей використовуються спеціальні атрибути – **ідентифікатори (ключі)**. Це може бути один або декілька атрибутів, значення яких дозволяють беззаперечно відрізнати один екземпляр сутності від іншого. Наприклад, ідентифікуючий атрибут (ключ) **ТАБЕЛЬНИЙ_НОМЕР**.

Складовим називається ключ, що складається з двох і більше атрибутів, які сприймаються як єдине ціле.

Якщо для опису набору об'єктів обрана сукупність атрибутів, що не містить ключа, то до складу атрибутів вводиться додатковий, ключовий атрибут, наприклад, **НОМЕР_РАХУНКУ**.

3.1.2 Зв'язки

Зв'язки виступають у моделі в якості засобу, за допомогою якого представляються стосунки між двома й більш сутностями та атрибутами. Розрізняють три види зв'язків, що позначаються **1:1** («*один до одного*»), **1:M** («*один до багатьох*») та **M:N** («*багато до багатьох*»). Під час аналізу ПО можливе виявлення трьох типів зв'язку:

- сутність-сутність;
- сутність-атрибут;
- атрибут-атрибут.

Зв'язок «один до одного» (між двома типами сутностей)

При такому зв'язку (рис. 3.1) кожному екземпляру сутності **A** відповідає один і тільки один екземпляр сутності **B** і, навпаки, кожному екземпляру сутності **B** відповідає один і тільки один екземпляр сутності **A**. При цьому ідентифікація екземплярів сутності унікальна в обох напрямках.

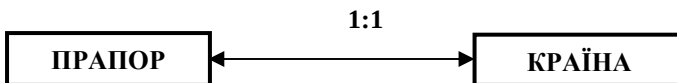


Рисунок 3.1 – Приклад зв'язку 1:1 між двома сутностями

Зв'язок «один до багатьох» (між двома типами сутностей)

При такому зв'язку кожному екземпляру сутності **A** може відповідати 0, 1 або декілька екземплярів сутності **B**, але кожному екземпляру сутності **B** відповідає тільки один екземпляр сутності **A**.

Це означає, що ідентифікація екземплярів сутності унікальна тільки в напрямку від **B** до **A**. Наприклад, кожний продавець може обслуговувати декілька замовлень, але кожне замовлення обслуговується лише одним продавцем (рис. 3.2).

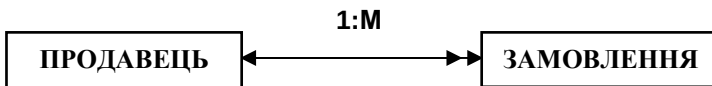


Рисунок 3.2 – Приклад зв'язку 1:M між двома сутностями

Зв'язок «багато до багатьох» (між двома типами сутностей)

Це відображення є найбільш складним видом зв'язку між сутностями. При цьому кожному екземпляру сутності **A** може відповідати 0, 1 або декілька екземплярів сутності **B** і навпаки, кожному екземпляру сутності **B** може відповідати 0, 1 чи декілька екземплярів сутності **A**, тобто ідентифікація екземплярів сутності неунікальна в обох напрямках. Наприклад, зв'язок між сутностями **ПРОДАВЕЦЬ** і **ПОКУПЕЦЬ** (рис. 3.3):

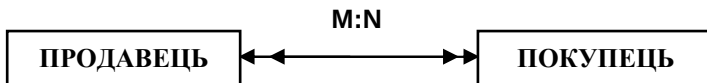


Рисунок 3.3 – Приклад зв'язку M:N між двома сутностями

Зв'язок «один до одного» (між двома атрибутами)

Якщо при описанні сутності використовуються два унікальних ідентифікатора, наприклад, **КОД_КЛІЄНТА** та **НОМЕР_ПАСПОРТУ**, то між ними існує зв'язок 1:1 (рис. 3.4).

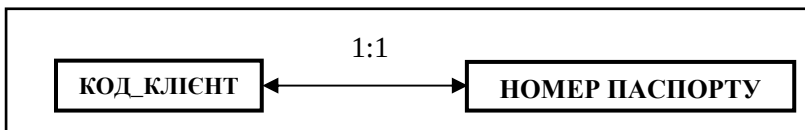


Рисунок 3.4 – Приклад зв'язку 1:1 між двома атрибутами

Зв'язок «один до багатьох» (між двома атрибутами)

Наприклад, між двома атрибутами НОМЕР_ГРУПИ й ПРІЗВИЩЕ_СТУДЕНТА існує зв'язок 1:М (рис. 3.5):

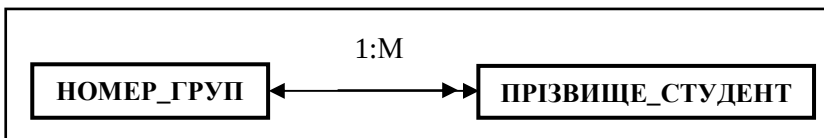


Рисунок 3.5 – Приклад зв'язку 1:М між двома атрибутами

Зв'язок «багато до багатьох» (між двома атрибутами)

Наприклад, атрибути ВИКЛАДАЧ і ДИСЦИПЛІНА сутності ВИКЛАДАННЯ_ДИСЦИПЛІНИ пов'язані зв'язком М:N (рис. 3.6)



Рисунок 3.6 – Приклад зв'язку М:N між двома атрибутами

3.2 Моделювання локальних подань

Враховуючи великий об'єм реальних баз даних (за кількістю сутностей), а також труднощі обліку всіх деталей предметної області, процес проектування концептуальної моделі складається з таких етапів:

- виділення у ПО окремих локальних областей (декомпозиція ПО);
- проектування локальних подань відповідних областей;
- об'єднання локальних подань у єдину глобальну концептуальну модель ПО.

При моделюванні використовуються такі основні правила:

- використовуються тільки три типи конструктивних елементів: сутності, атрибути та зв'язки;

–в окремому проектному поданні кожний компонент інформації моделюється лише одним конструктивним елементом, тобто необхідно уникати надмірності у використанні конструктивних елементів.

Вибір локального подання залежить від масштабів предметної області. Для зручності проектування в окремому локальному поданні бажано використовувати 6–7 типів сутностей, тому що згідно з правилом із теорії інформації, існує «магічне число 7 ± 2 », тобто кількість факторів (інформаційних кластерів), якими людина може одночасно оперувати, приблизно дорівнює 7.

Зазвичай локальне подання відповідає окремому зовнішньому додатку, наприклад, окремій функціональній задачі або окремому користувачу. Але воно може відповідати й цілій незалежній області даних, яка використовується декількома додатками.

При виборі області для локального подання проектувальник відшукує компромісне рішення, оскільки вузька область призводить до зниження рівня інтеграції даних і їх дробленню, а велика область – до нечіткості та складності проектування.

Формулювання сутностей

Для кожного локального подання необхідно сформулювати сутності, що потрібні для його опису. В окремих випадках це зробити складно, оскільки деяка порція інформації може бути представлена будь-яким із типів конструктивних елементів. Наприклад, факт знаходження в сімейних стосунках може бути виражений сутністю **СІМ'Я**, зв'язком **ОДРУЖЕНИЙ_З (ЗАМУЖЕМ_ЗА)** або атрибутом **ДРУЖИНА (ЧОЛОВІК)**.

У цих випадках рекомендується проробити декілька варіантів моделей локального подання та вибрати більш гнучкий з точки зору представлення інформації.

Кожній вибраній сутності повинно бути присвоєне чітке найменування, що відображає смисловий зміст поняття, що вводиться. Розпливчаті найменування, наявність *синонімів* (одне й теж поняття має різні найменування) або *омонімів* (різні поняття мають одне й теж саме найменування) призводять до помилок у проектуванні та неприпустимі.

Приклад 1

Нехай у деякому локальному представленні виконується опис поставок товару на склад. Передбачається, що в одній поставці може брати участь лише один постачальник, який поставляє лише один вид товару. При цьому постачальник може брати участь у декількох поставках.

Для опису використаємо лише дві основні конструкції, наприклад, сутність і атрибут. Введемо до розгляду сутність **ПОСТАВКА** та опишемо її за допомогою наступних атрибутів (рис. 3.7):

- ІНДЕКС_ПОСТАВКИ;
- ІНДЕКС_ПОСТАЧАЛЬНИКА;
- АДРЕСА_ПОСТАЧАЛЬНИКА;
- ІНДЕКС_ТОВАРУ;
- НАЗВА_ТОВАРУ;
- КІЛЬКІСТЬ_ПОСТАВЛЕНОГО_ТОВАРУ;
- ЦІНА_ОДИНИЦІ_ТОВАРУ;
- ШИФР_СКЛАДУ;
- ДАТА_ПОСТАВКИ.

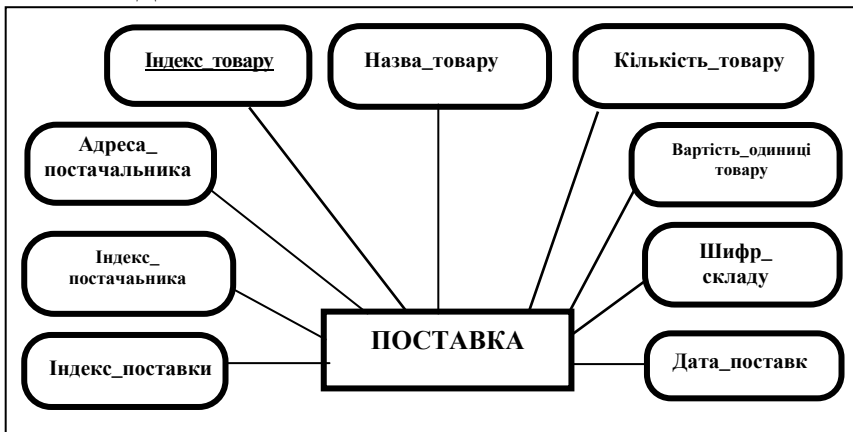


Рисунок 3.7 – Початкова графічна діаграма локального подання

Такій моделі властиві певні недоліки. За її допомогою неможливо представити інформацію про окремого постачальника, якщо він не виконує поставчань у теперішній час.

Для цього необхідно ввести до моделі сутність **ПОСТАЧАЛЬНИК**, призначити їй відповідні атрибути, зв'язати з сутністю **ПОСТАВКА** та вилучити надлишкові елементи (рис. 3.8).

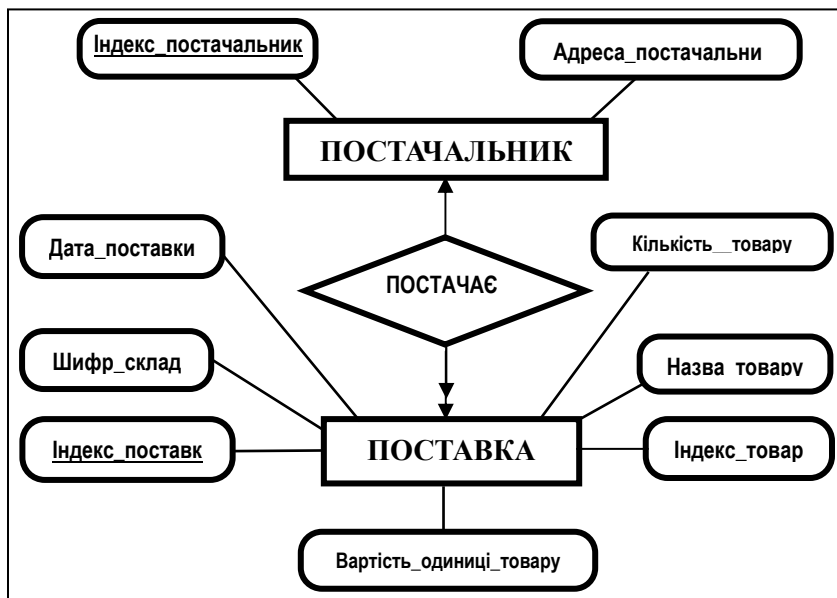


Рисунок 3.8 – Графічна діаграма з введенням до моделі сутності **ПОСТАЧАЛЬНИК**

При цьому можна визначити, який конкретно постачальник виконав постачання, використовуючи зв'язок **ПОСТАЧАЄ** між сутностями **ПОСТАЧАЛЬНИК** і **ПОСТАВКА**. В інформаційному плані дана модель зберігає всі можливості попередньої. Крім того, вона дає інформацію ще й про окремих постачальників, незалежно від того, виконували вони постачання товару чи ні.

Але отриманий варіант не дає інформацію про окремі товари, якщо вони відсутні у постачаннях. Необхідно ввести до моделі сутність **ТОВАР** та виконати відповідні процедури побудови (рис. 3.9).

Отриманий варіант, у свою чергу, не дозволяє представити інформацію «які товари може постачати окремий постачальник» та «які постачальники можуть постачати даний товар». Для реалізації в моделі подібної інформації необхідно організувати відповідні зв'язки між сутностями **ПОСТАЧАЛЬНИК** і **ТОВАР** (рис.3.10).

Для даного локального подання вибираємо останній варіант моделі, оскільки він найбільш гнучкий для представлення інформації.

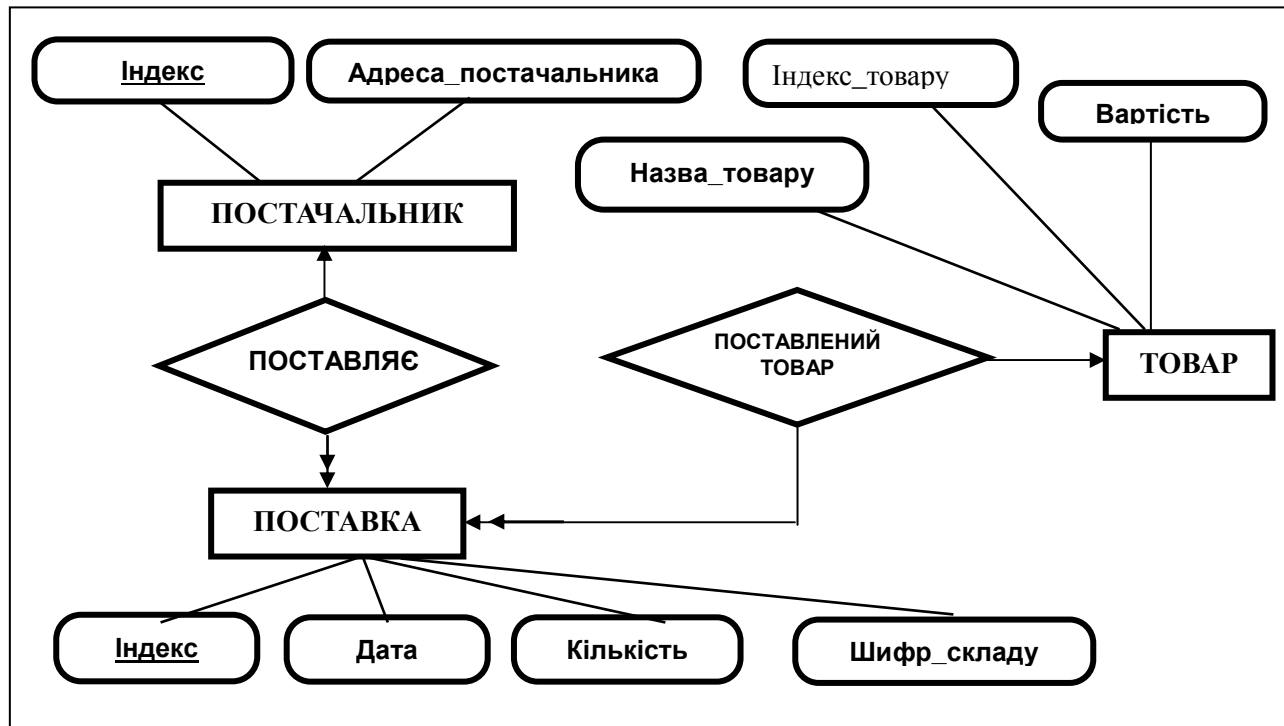


Рисунок 3.9 – Графічна діаграма з введенням до моделі сутності ТОВАР

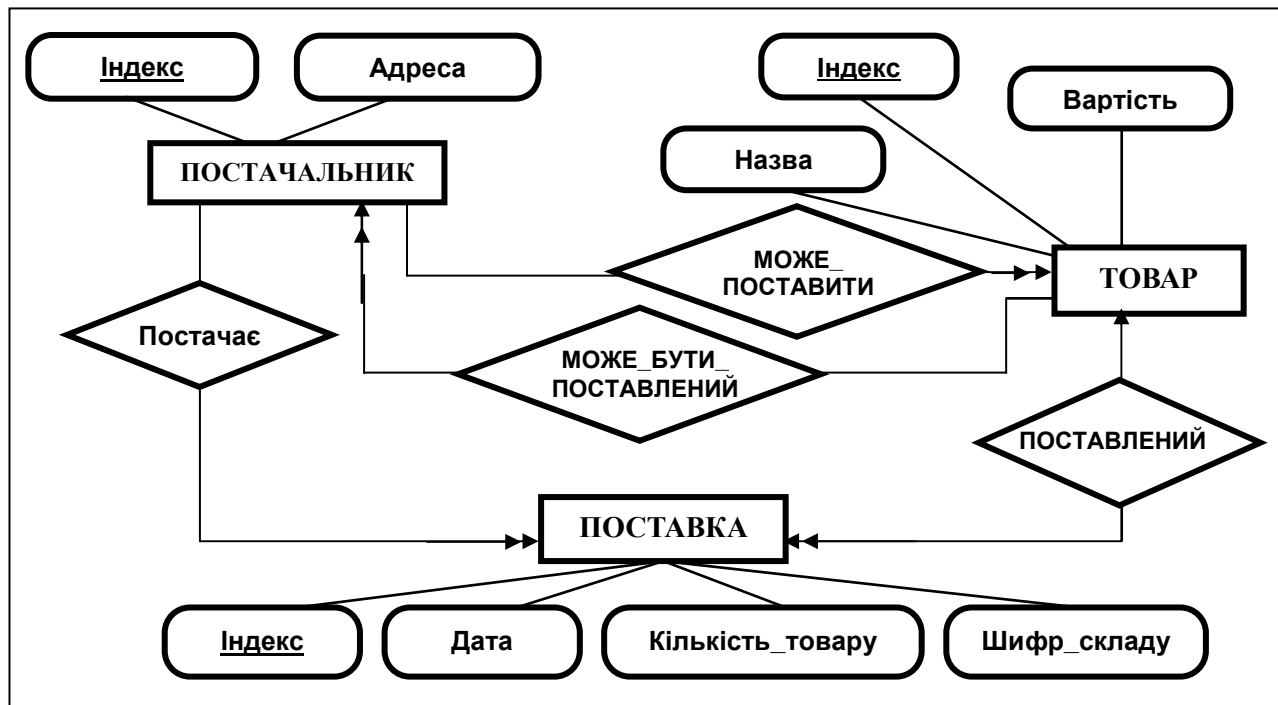


Рисунок 3.10 – Остаточний вигляд графічної діаграми

Призначення сутностям описових атрибутів

Виділеним сутностям окрім ключів назначаються описові атрибути. Наприклад, сутність **СЛУЖБОВЕЦЬ** може мати такі описові атрибути, як **ДАТА_НАРОДЖЕННЯ**, **ОСВІТА**, **ДОМАШНЯ_АДРЕСА**.

Для атрибутів вказується тип даних (символьний, числовий тощо) та діапазон можливих значень для числових величин або кількості символів для подання символьної інформації. Крім того, необхідно вказати всі обмеження на значення атрибутів, якщо вони мають місце.

Специфікація зв'язків

На цьому етапі виявляються залежності між двома та більше сутностями та атрибутами. Для виявлених зв'язків визначаються їх характеристики, кожен тип зв'язку «сутність-сутність» іменується.

Визначаються також зв'язки типу «атрибут-атрибут», які уявляють собою відношення між атрибутами однієї сутності чи одного зв'язку типу «сутність-сутність».

Прикладом зв'язку типу «сутність-сутність» є зв'язок **СКЛАД_ПОСТАВКИ** між сутностями **ПОСТАВКА** та **ТОВАР**, який може мати атрибути **Номер_поставки**, **Номер_товару**, **Назва_товару**, **Ціна_товару**, **Кількість_товару**, **Сума_за_даний_товар**.

Приклад 2

У локальному поданні для сутності **СЛУЖБОВЕЦЬ** були призначені атрибути, представлені на рис. 3.11.

Між атрибутами існують певні зв'язки. Оскільки всі атрибути описують ту саму сутність **СЛУЖБОВЕЦЬ**, яка має ідентифікуючий атрибут **Табельний_номер**, то всі атрибути мають залежність від цього атрибуту. При описі конкретних екземплярів сутності **СЛУЖБОВЕЦЬ** описові атрибути не можуть приймати довільні значення, їхні значення залежать від значень ідентифікуючого атрибуту. Крім того, серед атрибутів спостерігається й ряд інших залежностей: значення атрибуту **Адреса_ВНЗ** визначаються значеннями атрибуту **Назва_ВНЗ**, а значення атрибуту **Дата_народження_дитини** визначаються значенням атрибуту **Ім'я_дитини**. Слід також зазначити, що для кожного значення атрибуту **Таб_номер** може бути декілька значень атрибутів **Адреса_ВНЗ** і **Ім'я_дитини**.

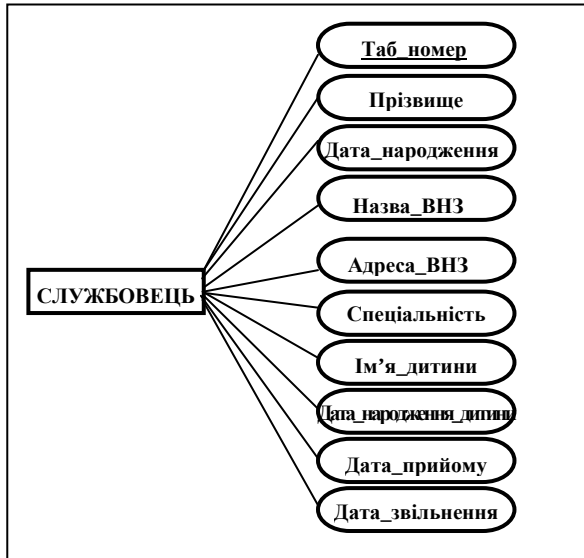


Рисунок 3.11 – Приклад локального подання

Цю інформацію доцільно представити у вигляді окремої діаграми бінарних зв'язків між атрибутами (зв'язки показані стрілками) (рис. 3.12).

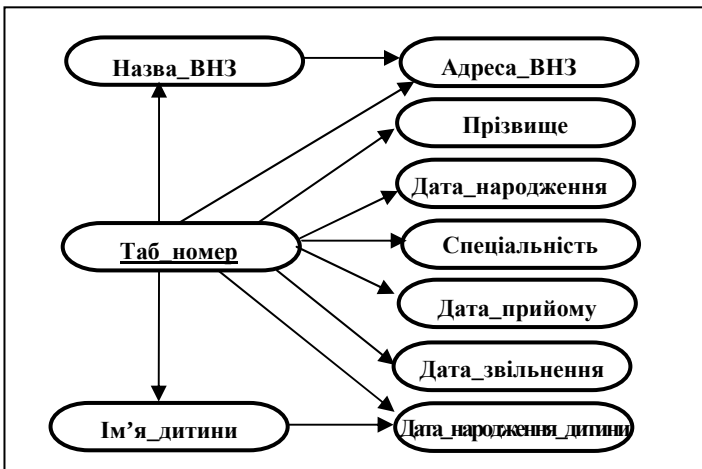


Рисунок 3.12 – Діаграма бінарних зв'язків між атрибутами

Аналіз діаграми зв'язків між атрибутами сутності дозволяє судити про те, наскільки вдалий був вибір основних конструкцій моделі (у даному випадок – сутність і атрибут) з погляду простоти подання змістовних зв'язків між конструкціями, що використовуються.

Доцільно прагнути до випадку, коли залежність «атрибут-атрибут» для сутності має вигляд, представлений на рис. 3.13, тобто спостерігаються тільки залежності описових атрибутів від ідентифікуючих, інших залежностей немає. З цього погляду в нашому прикладі доцільно сутність **СЛУЖБОВЕЦЬ** представити за допомогою наступної графічної діаграми (рис. 3.14). При цьому початкова діаграма бінарних зв'язків розкладається на три діаграми для сутностей **СЛУЖБОВЕЦЬ**, **ДИТИНА** та **ВНЗ**.

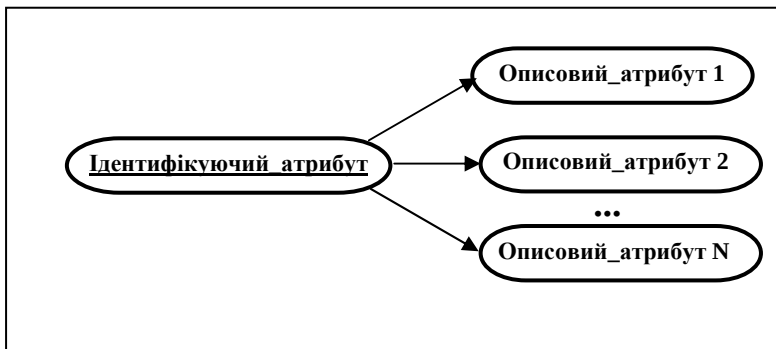


Рисунок 3.13 – Діаграма бінарних зв'язків атрибутів сутності

Моделювання локального подання закінчується графічним описом усіх виявлених сутностей, зв'язків, атрибутів, а також складанням усіх перелічених вище специфікацій

3.4 Об'єднання моделей локальних подань

Після етапу формування локальних подань здійснюється їх об'єднання в єдину глобальну інформаційну структуру. Ця структура не тільки взаємно відповідає кожному поданню, але й уявляє собою інтегровану БД у стиснутому вигляді.

При великій кількості подань (більше 5–6) зазвичай використовується бінарне об'єднання. При цьому результат

об'єднання $N1$ об'єктів одного подання з $N2$ об'єктами другого подання дає $(N1 + N2 - X)$ об'єктів, де X – кількість об'єктів, які співпадають в об'єднаних поданнях.

Якщо перед процесом об'єднання виконати відповідне групування локальних подань, то можна збільшити значення фактору X і тим самим зменшити кількість операцій порівняння при виконанні об'єднання.

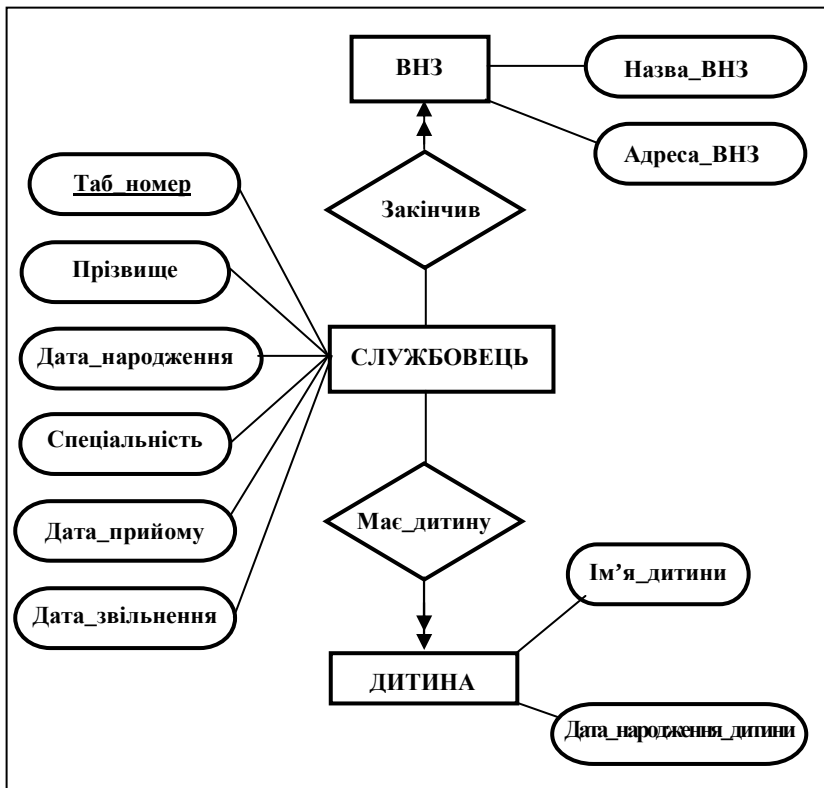


Рисунок 3.14 –Графічна діаграма перетвореного початкового подання

Існують три основні концепції об'єднання: *ідентичність*, *агрегація* та *узагальнення*.

Два та більше елементів у моделі *ідентичні*, якщо вони мають однакове семантичне (змістове) значення. Але для ідентичних елементів зовсім необов'язково мати однакове синтаксичне представлення.

Агрегація дозволяє розглядати зв'язок між елементами моделі як новий елемент більш високого рівня. Наприклад, елемент **СЛУЖБОВЕЦЬ** може бути сприйнятий як агрегація елементів **ПРИЗВИЩЕ**, **НОМЕР_ПАСПОРТУ** та **АДРЕСА** (рис. 3.15).



Рисунок 3.15 – Структура агрегатованого даного СЛУЖБОВЕЦЬ

Приклад 3 Об'єднанню підлягають два подання (рис.3.16 і 3.17). Використовуючи агрегацію, можна об'єднати ці подання (рис. 3.18)

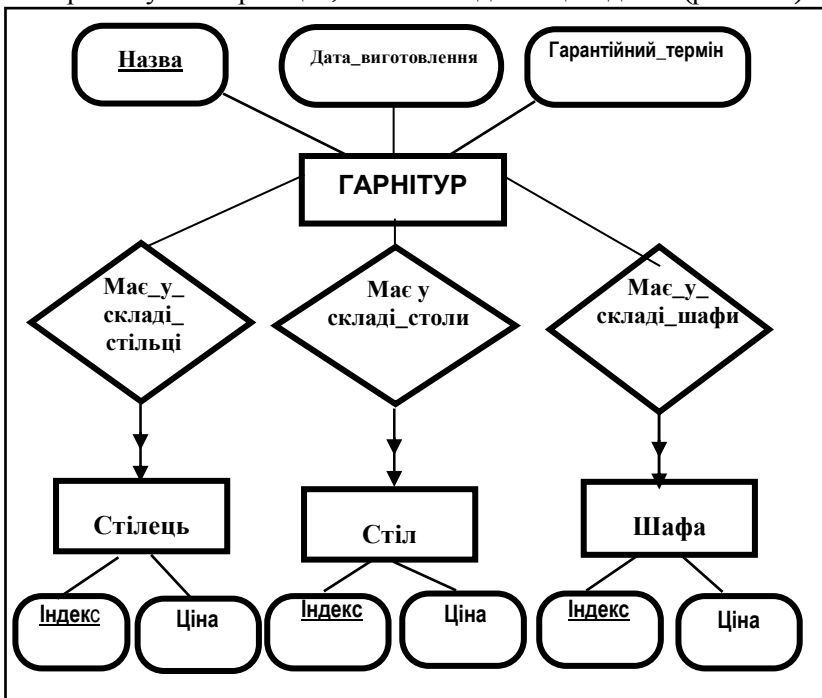


Рисунок 3.16 – Перше початкове подання

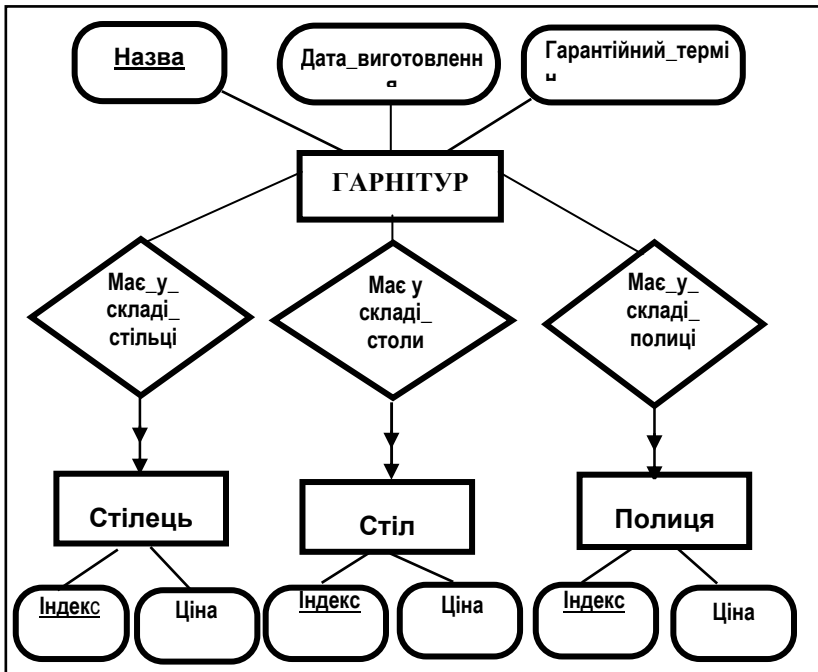


Рисунок 3.17 – Друге початкове подання

Узагальненням називається абстракція даних, яка дозволяє трактувати клас різних подібних об'єктів як один поійменований узагальнений тип об'єкту.

Наприклад, клас об'єктів {собака, кішка, слон} може бути узагальнений в об'єкт **ТВАРИНА**. В узагальненні підкреслюється загальна природа об'єктів, а їхні індивідуальні відмінності ігноруються.

В загальному випадку, якщо клас $\{o_1, \dots, o_n\}$ може бути узагальнений в O , то кажуть, що O_i є **категорією** O . Наприклад, **собака** є категорією об'єкту **ТВАРИНА**.

У випадку багаторівневої ієрархії узагальнень структура узагальнень утворює **родову ієрархію**, що призводить до появи поняття родової та видової сутності. За визначенням, вид є рід у сукупності з видовою ознакою. Це означає, що усі властивості родової сутності повинні наслідуватися її видом, але при цьому вид може мати свої додаткові властивості.

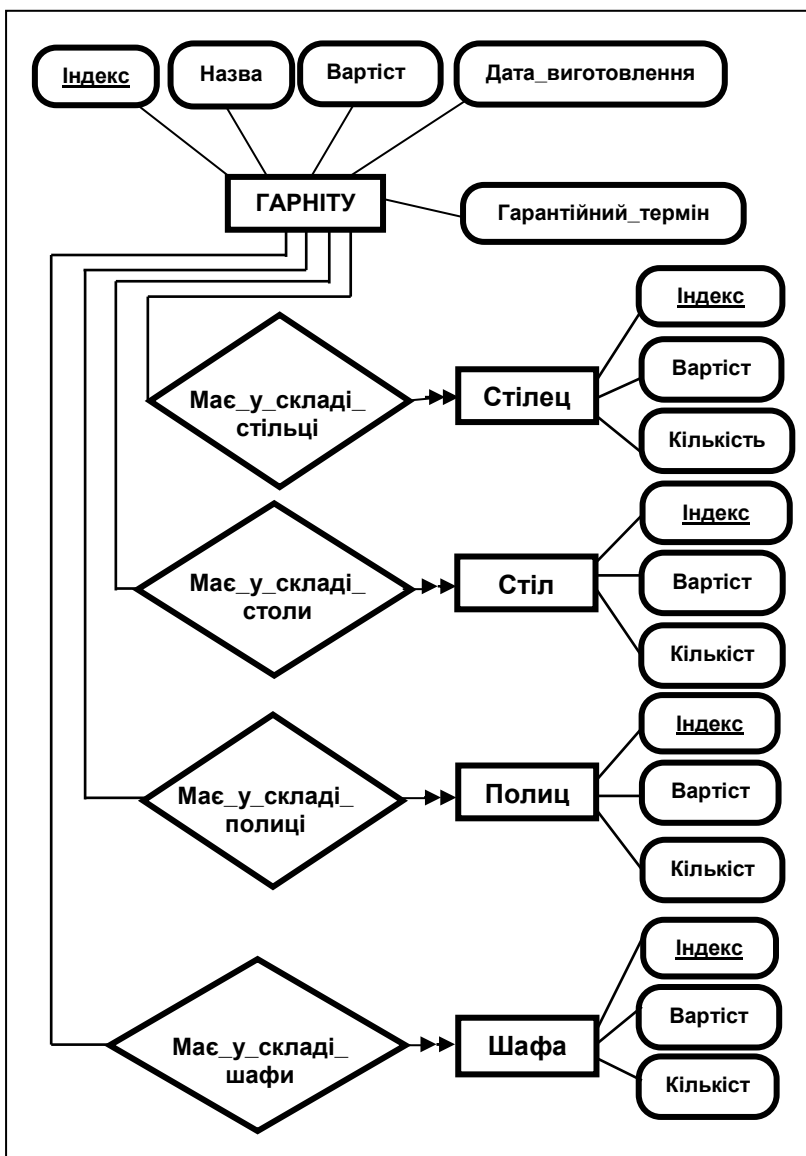


Рисунок 3.18 – Остаточний вигляд агрегатного представлення

Приклад 4 У попередньому прикладі (рис. 3.18) процес узагальнення обох подань слід продовжити, використовуючи узагальнення, оскільки сутності **СТІЛЕЦЬ**, **СТІЛ**, **ШАФА**, **ПОЛИЦЯ** уявляють собою різні категорії типів узагальненого поняття «*компонент гарнітуру*».

Надамо цьому узагальненому поняттю найменування **КОМПОНЕНТ** та введемо його у модель подання, використовуючи конструктивний елемент «сутність». Щоб надати в моделі інформацію про категорії, додамо до сутності **КОМПОНЕНТ** атрибут **найменування**, який буде приймати значення з множини: {**стіл**, **стілець**, **шафа**, **полиця**}. Значення цієї множини відповідають категоріям, на які поділяється введене узагальнення (рис. 3.19).

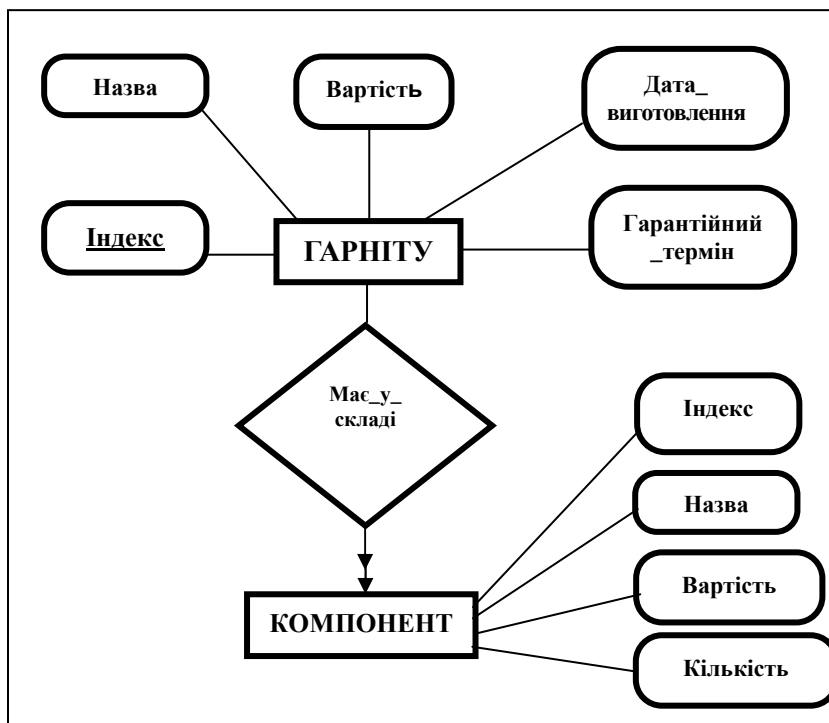


Рисунок 3.19 – Приклад використання узагальнення

Приклад 5 В поєднаних початкових поданнях присутні наступні сутності: ДЕТАЛІ_ВЛАСНОГО_ВИРОБНИЦТВА, ДЕТАЛІ_ПОКУПІ, ЗБІРНІ_ОДИНИЦІ_ПОКУПІ, ЗБІРНІ_ОДИНИЦІ_ВЛАСНОГО_ВИРОБНИЦТВА.

Для цього прикладу можуть бути два варіанти об'єднання, що наведені на рис. 3.20 і 3.21, які можуть бути поданнями, наприклад, начальника цеху та відділу постачання, відповідно.

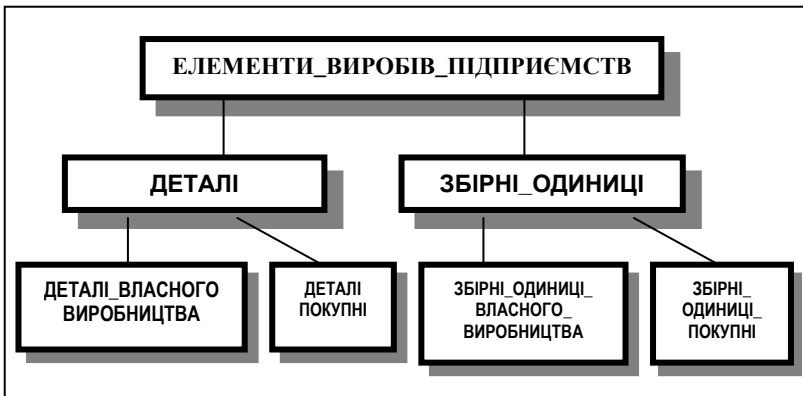


Рисунок 3.20 – Перший варіант об'єднання

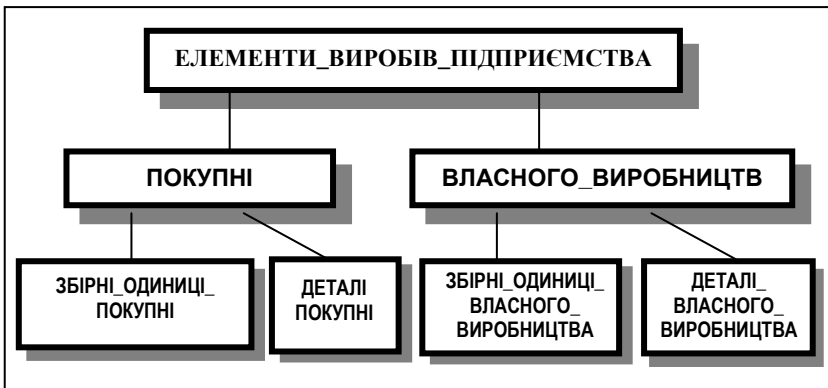


Рисунок 3.21 – Другий варіант об'єднання

Застосування узагальнень дозволяє організовувати для користувачів доступ до даних БД з використанням різних рівнів абстракції. Це дозволяє підвищити гнучкість системи для спільного використання даних. Крім того, при пошуку даних у відповідності зі смисловими категоріями суттєво скорочується простір пошуку.

Слід мати на увазі, що **агрегація** формує об'єкт як зв'язок між іншими об'єктами. **Узагальнення** формує об'єкт із класу інших об'єктів.

Агрегація та узагальнення не є взаємовиключними категоріями. Агрегатний об'єкт може бути узагальненням деякого класу об'єктів. У цьому випадку він називається **родовим об'єктом**. Родовий об'єкт, у свою чергу, може бути агрегацією деякого зв'язку між об'єктами. В цьому випадку він називається агрегатним об'єктом. У загальному випадку кожен об'єкт може бути як агрегатним, так і родовим.

Процес об'єднання продовжується до тих пір, доки не будуть інтегровані всі подання, узгодженні та усунуті всі суперечності, що можуть виникати між окремими локальними поданнями.



Питання для самоперевірки

1. З яких конструктивних елементів складається ER-модель?
2. У чому різниця між батьківськими та підкореними об'єктами?
3. Чим відрізняються тип і екземпляр сутності?
4. На які типи поділяються ключі сутностей?
5. Що таке атрибут сутності?
6. Які типи зв'язків існують?
7. Навіщо виконується декомпозиція предметної області?
8. З яких міркувань обирається обсяг локальної області?
9. Які основні методи використовуються для об'єднання локальних подань?
10. У яких формах зустрічається агрегація при об'єднанні локальних подань?
11. Чим відрізняється агрегація від об'єднання?

4 ЛОГІЧНЕ ПРОЕКТУВАННЯ РЕЛЯЦІЙНИХ БАЗ ДАНИХ

4.1 Реляційна модель даних

4.1.1 Поняття моделі даних

Система баз даних підтримує у пам'яті ЕОМ модель предметної області (ПО). Результат моделювання залежить від СКБД, яка використовується, оскільки кожна система надає свій інструментарій відображення ПО, який прийнято називати *моделлю даних* [19].

Модель даних, яку підтримує СКБД, визначається трьома компонентами:

- припустимою структурою (організацією) даних;
- множиною припустимих операцій над даними;
- обмеженнями цілісності (семантичної).

Припустима організація даних визначає різноманітність та кількість типів об'єктів моделі, обмеження на структуру даних. Наприклад, кількість типів записів і типів зв'язків.

СКБД забезпечує будівництво, що мають назву *схема даних*. ***Схемою даних*** називається опис структури даних деякого типу на формалізованій мові. Схема описує ті властивості структури, що не змінюються у процесі виконання прикладних програм і які можна об'явити наперед, до виконання програм.

Множина операцій визначає види обробок, яким можуть підлягати об'єкти моделі даних. Сюди, в першу чергу, входять операції вибірки даних і операції, що змінюють склад БД.

Обмеження цілісності покладаються на значення даних та зв'язки, що характеризують достовірні стани БД. Наприклад, якщо між записами типу **ПОЛІКЛІНІКА** та **ПАЦІЄНТ** встановлений ієрархічний зв'язок, то СКБД заборонить вилучення відомостей про поліклініку, якщо з цією поліклінікою зв'язаний хоча би один запис про пацієнта.

Обмеження цілісності можуть поширюватись і на значення окремих полів. Наприклад, при описі запису можна задати область припустимих значень. Система не допустить оновлення даного, якщо нове значення буде кваліфіковано як некоректне (дивиться розділ 6).



Модель даних – це сукупність правил формування структури даних в БД, операцій над ними, а також обмежень цілісності, що визначають припустимі зв'язки та значення даних.

У загальноприйнятій класифікації в цей час відомі наступні моделі даних:

- ієрархічна;
- мережна;
- реляційна;
- об'єктно-орієнтована;
- об'єктно-реляційна.

Об'єктно-орієнтовану та об'єктно-реляційну моделі часто називають постреляційними моделями даних. Відповідно, кажуть про мережні, ієрархічні, реляційні, об'єктно-орієнтовані, об'єктно-реляційні та постреляційні БД. Сучасні комерційні СКБД у своїй більшості підтримують саме реляційну модель даних.

4.1.2 Базисні засоби маніпулювання реляційними даними

В маніпуляційній складовій реляційної моделі визначаються два базових механізми маніпулювання даними: реляційна алгебра, заснована на теорії множин, і реляційне числення, що базується на математичній логіці (точніше, на численні предикатів першого порядку).

У свою чергу, зазвичай розглядаються два види реляційного числення – числення доменів і числення предикатів. Всі ці механізми володіють однією важливою властивістю: вони замкнені відносно поняття відношення. Це означає, що вирази реляційної алгебри та формули реляційного числення визначаються над відношеннями реляційної БД і результатом обчислення також є відношення. В результаті будь-якій вираз або формула можуть інтерпретуватися як відношення, що дозволяє використати їх в інших виразах або формулах.

Реляційні алгебра та числення володіють великою виразною потужністю: дуже складні запити до бази даних можуть бути подані за допомогою одного виразу реляційної алгебри або однієї формули реляційного числення. Саме з цієї причини ці механізми включені до реляційної моделі даних.

Відомо, що механізми реляційної алгебри та реляційного числення еквівалентні, тобто для будь-якого припустимого виразу реляційної алгебри можна побудувати еквівалентну формулу реляційного числення та навпаки [10].

4.1.3 Визначення відношення

Реляційна модель даних (РМД) введена *Едвардом Коддом (Kodd E.E.)* в 1970 році. В основі РМД лежить математичне поняття відношення (*relation*) [10,25].

 **Відношення** – це підмножина декартового добутку доменів.

Домен – це множина певних елементів, наприклад, множина цілих чисел, множина назв міст, прізвищ студентів тощо.

Кортеж – це упорядкований набір взаємозв'язаних величин (даних).

Декартовим добутком k доменів $D_1, D_2, D_3, \dots, D_k$

$$D = D_1 \times D_2 \times D_3 \times \dots \times D_k$$

називається множина всіх кортежів виду $d = \{d_1, d_2, d_3, \dots, d_k\}$ довжини k , де $d_1 \in D_1, d_2 \in D_2$, та $d_k \in D_k$.

Припустимо, $D_1 = \{1, 2, 3\}; D_2 = \{a, b, c\}$. Тоді

$D_1 \times D_2 = \{(1, a), (1, b), (1, c), (2, a), (2, b), (2, c), (3, a), (3, b), (3, c)\}$ або у матричному вигляді

$$D_1 \times D_2 = \begin{vmatrix} 1, a & 1, b & 1, c \\ 2, a & 2, b & 2, c \\ 3, a & 3, b & 3, c \end{vmatrix}$$

Декартовий добуток дозволяє отримати всі можливі комбінації елементів початкових множин (елементів доменів, які розглядаються).

Підмножина декартового добутку R_n називається ***n*-місним відношенням**. Число n (кількість атрибутів або стовпців) називається ***мірою*** відношення.

Число всіх різних кортежів $d = \langle d_1, d_2, d_3, \dots, d_n \rangle$, які утворюють відношення R_n , називається ***потужністю*** цього відношення.

Міра відношення зазвичай не змінюється після утворення відношення, але потужність буде змінюватися по мірі додавання нових та вилучання старих кортежів.

4.1.4 Властивості та типи відношень

Кожне відношення в БД надається у вигляді двомірної таблиці, в якій рядок є кортеж (запис). Стовпці відношень називаються атрибутами. Атрибути іменуються.

Схема відношення **R**, яке має атрибути **A1, A2, ..., Ak**, позначається **R (A1, A2, ..., Ak)**. **Реляційна база даних** (РБД) – це набір відношень. Схему РБД можна уявити у вигляді сукупності схем відношень і зв'язку між ними.

Відношення поділяють на **об'єктні** та **зв'язні**.

Об'єктні відношення

Об'єктне відношення зберігає дані про об'єкти (екземпляри сутності). В об'єктному відношенні один із атрибутів однозначно ідентифікує окремий об'єкт. Такий атрибут називається **первинним ключем** або просто **ключем**. Решта атрибутів залежить від цього ключа. Ключі бувають прості та складові.

Простим називається **ключ**, який складається з одного атрибута, причому атрибут повинен бути атомарним, а екземпляри даного атрибута – унікальні.

Атомарним називається атрибут, значення якого сприймається програмою чи СКБД як неподільний елемент даних, навіть якщо він утворений з декількох об'єктів, наприклад, **ДАТА** чи **АДРЕСА**.

Складовим називається **ключ**, який містить декілька атрибутів. При цьому в ключі повинна бути відсутня надмірність, тобто при вилученні з ключа будь-якого атрибута він переставє однозначно ідентифікувати кортежі даного відношення. Один і той же набір атрибутів може мати декілька ключів, але тільки один із них призначається первинним. Усі інші ключі при цьому називаються **можливими (альтернативними)**.

Вторинний ключ (ключ пошуку) – це атрибут або сукупність атрибутів, яка ідентифікує не унікальний об'єкт у наборі, а всі об'єкти, що мають визначені значення цих атрибутів. Наприклад, усі товари, ціна яких перевищує 100 грн.

В об'єктному відношенні не повинно бути рядків з однаковими ключами, тобто не повинно бути дублювання об'єктів. Це **основне обмеження реляційної моделі** для забезпечення цілісності даних.

Зв'язні відношення

Зв'язне відношення зберігає ключі двох або більше об'єктних відношень, тобто по ключам встановлюються зв'язки між об'єктами відношень.

Роздивимось зв'язне відношення **ВИКЛАДАЄ**. Нехай у БД маютьс'я об'єктні відношення **ВИКЛАДАЧ** і **ДИСЦИПЛІНА** (табл. 4.1, 4.2).

Таблиця 4.1 – ВИКЛАДАЧ

Прізвище	Посада	Вчений ступень	Кафедра
Рисіков	доцент	к.т.н.	ПЗ
Сердюк	доцент	к.т.н.	ПЗ
Скачко	ст. викл.		ПЗ

Таблиця 4.2 –
ДИСЦИПЛІНА

Назва	Семестр
СОД	3
СПЗ	5
СКБД	4

Зв'язне відношення **ВИКЛАДАЄ** повинно містити в якості атрибутів первинні ключі об'єктних відношень, які зв'язуються, а також може мати й інші атрибути, що функціонально залежать від цього зв'язку (табл.4.3, 4.4):

Таблиця 4.3 – ВИКЛАДАЄ_1

Викладач	Дисципліна
Рисіков	СОД
Сердюк	СПЗ
Скачко	БД та БЗ

Таблиця 4.4 – ВИКЛАДАЄ_2

Викладач	Дисципліна	Семестр
Рисіков	СОД	3
Сердюк	СПЗ	5
Скачко	БД та БЗ	4

Ключі у зв'язних відношеннях мають назву *зовнішні ключі*, оскільки вони посилаються на первинні ключі інших відношень.

На зовнішні ключі накладається обмеження для забезпечення цілісності даних, яке має назву *посилальна цілісність*. Це означає, що кожному значенню зовнішнього ключа повинен відповідати певний рядок батьківського відношення. Інакше зовнішній ключ може посиляється на неіснуючий об'єкт. Наприклад, на непрацюючого викладача. Спроба вводу такого даного буде блокована.

Контроль за цілісністю даних покладається на СКБД (питання підтримки цілісності даних будуть розглянуті у розділі 6).

4.1.5 Операції над відношеннями

✎ **Реляційною алгеброю** називають систему операцій маніпулювання відношеннями, кожний оператор якої в якості операндів має одно чи більше відношень.

До таких операцій належать: об'єднання, перетин, різниця, декартовий добуток, ділення, проекція, з'єднання та селекція.

При виконанні операцій об'єднання, перехрещення та різниці відношення, що приймають участь у них, повинні задовольняти властивості сумісності за об'єднанням.

Два відношення **сумісні за об'єднанням**, якщо вони мають одну й ту ж кількість атрибутів і однакові типи відповідних атрибутів.

Для подальшого розглядання цих операцій введемо два відношення СТУД_1 і СТУД_2 (табл. 4.5, 4.6).

Таблиця 4.5 – Відношення СТУД_1

Прізвище	Група
Попов	РП-411
Бочков	РП-411
Шахова	РП-421

Таблиця 4.6 – Відношення СТУД_2

Прізвище	Група
Бочков	РП-411
Сокол	РП-411
Супрун	РП-421

Об'єднання $R = R1 \cup R2$. Операція виконується над двома сумісними відношеннями. Результат об'єднання включає кортежі першого відношення та кортежі другого відношення, що відсутні в першому відношенні (табл. 4.7).

Таблиця 4.7 – Відношення СТУДЕНТ

Прізвище	Група
Попов	РП-411
Бочков	РП-411
Шахова	РП-421
Сокол	РП-411
Супрун	РП-421

Перетин $R = R1 \cap R2$ Результат операції містить тільки ті кортежі, котрі є загальними для обох відношень (табл. 4.8).

Таблиця 4.8 – Відношення СТУДЕНТ

Прізвище	Група
Бочков	РП-411

Різниця $R = R1 \setminus R2$ Результат різниці включає тільки ті кортежі першого відношення, яких немає в другому (табл. 4.9)

Таблиця 4.9 – Відношення СТУДЕНТ

Прізвище	Група
Попов	РП-411
Шахова	РП-421

Декартовий добуток $R \times S$ Відношення-операнди (табл. 4.10, 4.11) можуть мати різну міру ($k1$ и $k2$).

Декартовим добутком відношень R і S називається численність кортежів міри $k1 + k2$, перші $k1$ компонентів якої належать R , а останні $k2$ – кортежі, що належать S .

Міра результуючого відношення дорівнює сумі мір відношень-операндів, а потужність – добутку їх потужностей (табл. 4.12).

Таблиця 4.10 – Відношення СТУДЕНТИ

Прізвище
Іванов
Попов
Бочков

Таблиця 4.11 – Відношення ІСПИТИ

Предмет	Дата
БД та БЗ	5.06.98
МП	20.01.98

Таблиця 4.12 – Відношення ЕКЗАМЕНАЦІЙНА_ВІДОМІСТЬ

Прізвище	Предмет	Дата
Іванов	БД та БЗ	5.06.98
Іванов	МП	20.01.98
Попов	БД та БЗ	5.06.98
Попов	МП	20.01.98
Бочков	БД та БЗ	5.06.98
Бочков	МП	20.01.98

Ділення. Відношення-дільник (табл. 4.14) повинно містити підчисленність атрибутів відношення-діленого (табл. 4.13). Результуюче відношення (табл. 4.15) містить тільки ті атрибути діленого, яких немає в дільнику. В нього включають тільки ті кортежі, декартові добутки яких з дільником містяться в діленому:

Таблиця 4.13 – Відношення
ЕКЗ_ВІДОМІСТЬ

Прізвище	Дисципліна	Оцінка
Іванов	БД та БЗ	4
Іванов	МП	4
Попов	БД та БЗ	5
Попов	МП	5
Бочков	БД та БЗ	5
Бочков	МП	4

Таблиця 4.14 – РЕЗУЛЬТАТИ

Дисципліна	Оцінка
БД та БЗ	5
МП	4

Таблиця 4.15 – СТУДЕНТИ

Прізвище
Бочков

Проекція. Операція складається з того, що береться відношення **R**, вилучаються деякі з його атрибутів, і (або) переупорядковуються компоненти, що залишились.

Ця операція є унарною й дозволяє з заданого відношення **R** (табл. 4.16) отримати інше відношення **R'** (табл. 4.17), в якому численність атрибутів є підчисленністю атрибутів відношення **R**.

$$R' = \Pi R(A)$$

де **Π** – позначення операції проєкції;

A – підчисленність атрибутів початкового відношення **R**.

Таблиця 4.16 – Відношення
СПІВРОБІННИКИ

Прізвище	Відділ	Посада
Іваненко	1	Інженер
Петренко	1	Інженер
Сидоров	2	Технік
Швець	2	Інженер
Яценко	3	Технік

Таблиця 4.17 – ПОСАДИ

Відділ	Посада
1	Інженер
2	Технік
2	Інженер
3	Технік

Порядок слідування атрибутів в результуючому відношенні може відрізнитись від порядку слідування цих же атрибутів в початковому відношенні.

З одного початкового відношення за допомогою проєкції можна отримати декілька різних відношень.

З'єднання (JOIN) $R = R1 \bowtie R2$. Операція дозволяє створити нове відношення (табл. 4.20) з двох і більше початкових відношень (табл. 4.18, 4.19), які мають хоч би один спільний атрибут. При цьому для кожного початкового відношення можна вказати лише ті атрибути, які повинні увійти до результуючого відношення.

Таблиця 4.18 – Відношення
ПОСТАВКИ

Код_поставки	Код_товару
1	1
2	4
3	5
4	3

Таблиця 4.19 – Відношення ТОВАРИ

Код_товару	Назва	Ціна
1	Монітор	1500
2	Принтер	400
3	Клавіатура	25
4	Процесор	600
5	Пам'ять	250
6	Сканер	400

В якості атрибута для з'єднання оберемо «Код_товару»:

Таблиця 4.20 – Відношення НАКЛАДНА

Код_поставки	Код_товару	Назва	Ціна
1	1	Монітор	1500
2	4	Процесор	600
3	5	Пам'ять	250
4	3	Клавіатура	25

Вибір (селекція). Операція є унарною, тобто виконується над одним відношенням (табл. 4.21). Результуюче відношення (табл. 4.22) містить підчисленність кортежів, обраних за деякою умовою, наприклад, «Вік» > 55:

Таблиця 4.21 – Відношення
СПІВРОБІТНИКИ

Прізвище	Вік
Іванов	60
Петров	57
Бочков	21
Шахова	17

Таблиця 4.22 – Відношення
ПЕНСІОНЕРИ

Прізвище	Вік
Іванов	60
Петров	57

→

При виконанні операції оновлення над відношеннями бази даних можуть виникнути побічні ефекти, якщо між атрибутами відношень є небажані функціональні залежності.

Усунення побічних ефектів досягається нормалізацією відношень бази даних, тобто раціональним групуванням атрибутів у відношеннях (цей матеріал буде розглянутий у розділі 4).

4.2 Одержання відношень з діаграм ER-типу

Загальний підхід до проектування БД із використанням ER-методу складається з таких кроків [15]:

- побудова діаграми ER-типу, що включає до себе всі сутності та зв'язки, важливі з погляду інтересів організації; тобто з урахуванням бізнес-правил організації;
- побудова набору попередніх відношень із указівкою можливого первинного ключа для кожного відношення та з урахуванням бізнес-правил організації;
- підготування списку всіх атрибутів, що подають інтерес (крім зазначених ключів), і призначення кожного з цих атрибутів одному з попередніх відношень із тією умовою, щоб ці відношення задовольняли умовам нормалізації.

Бізнес-правила – це спеціалізований вид логіки, що описує обмеження на образ дій, які система або люди повинні враховувати у своїй поведінці. Ці правила визначаються цілим рядом факторів, включаючи директиви розпорядчих органів, промислові стандарти, ділову хватку й простий здоровий глузд.

Нерідко вони змінюються від країни до країни, від галузі до галузі, і навіть від бізнесу до бізнесу. Як приклад бізнес-правила в банківській справі можна навести закон, за яким про будь-яку угоду, що перевищує суму 10 000 доларів готівкою, необхідно повідомляти державу. Безумовно, дане бізнес-правило необхідно брати до уваги при створенні банківської системи вкладення/зняття готівки.

На останньому кроці проектування для кожного відношення повинні бути визначені міжатрибутні функціональні залежності, за допомогою яких перевіряється відповідність відношень нормальним формам.

Якщо отримані в результаті відношення виявляються ненормалізованими або якщо деяким атрибутам не знаходиться логічно обґрунтованих місць у попередніх відношеннях, то в цих випадках необхідно переглянути ER-діаграми на предмет усунення виниклих утруднень.

4.2.1 Попередні відношення для бінарних зв'язків ступеня 1:1

Як вже відмічалось, при побудові відношень БД необхідно враховувати бізнес-правила конкретної організації.

Наприклад, між сутностями **БРИГАДА** та **МАЙСТЕР** (табл. 4.23, 4.24) існує зв'язок 1:1. З точки зору спрощення структури БД доцільно об'єднати їх у одне відношення (табл. 4.25).

Таблиця 4.23 – Екземпляри сутності БРИГАДА

Ном	Спеціалізація	Дільниця
1	теслі	6
2	слюсарі	6
3	електрики	5
4	слюсарі	6

Таблиця 4.24 – Екземпляри сутності БРИГАДИР

Прізвище	Тел
Іваненко П.А.	218-34-45
Петренко П.Р.	289-34-76
Сидоренко А.О.	741-13-13
Махно О.Р.	289-45-67

Таблиця 4.25 – Відношення БРИГАДИ

Ном	Спеціалізація	Дільниця	Бригадир	Тел
1	теслі	6	Іваненко П.А.	218-34-45
2	слюсарі	6	Петренко П.Р.	289-34-76
3	електрики	5	Сидоренко А.О.	741-13-13
4	слюсарі	6	Махно О.Р.	289-45-67

З іншого боку, якщо взяти, приклад, БД «Відділ кадрів», то в одному відношенні **ПРАЦІВНИКИ** зберігаються дані двох типів: особиста інформація про працівників (дата народження, адреса, сімейний стан тощо) та службова інформація (підрозділ, посада, дата прийому на роботу, підвищення кваліфікації тощо).

Таке сумісне зберігання в одному відношенні різномірної інформації може призвести до певних негативних наслідків. Оскільки доступ до інформації в базах даних послідовний, то на його швидкодію напряму впливає розмір таблиць. З іншого боку, при

вибірці даних про працівників практично відсутня необхідність одночасного звернення до інформації обох типів.

Тому в цьому конкретному випадку доцільно розбити відношення **ПРАЦІВНИКИ** на два відношення, наприклад, *Дані_особисті* та *Дані_службові*, які зв'язані між собою залежністю 1:1 та мають менший розмір, ніж початкове відношення. Таке розбиття буде сприяти підвищенню швидкодії виконання запитів до цих відношень та спрощенню роботи з ними.

Слід лише зауважити, що подібне розбиття таблиці не має нічого спільного з нормалізацією, про яку піде мова в наступному розділі.

4.2.2 Попередні відношення для бінарних зв'язків ступеня 1:N

Якщо між двома сутностями існує зв'язок 1:N, то вони відображається за допомогою двох окремих відношень. Крім того, до складу атрибутів підлеглого відношення (з боку N) додається атрибут, який називається *зовнішнім ключем (foreign key)*. Він посилається на первинний або потенційний (альтернативний) ключ *батьківського відношення (parent key)*.

В цьому випадку кажуть, що ці відношення знаходяться у стані *посилальної цілісності*, ідея якої полягає у тому, що всі значення зовнішнього ключа відсилають до певного значення батьківського ключа. При цьому кожне значення зовнішнього ключа повинно бути подане в батьківському ключі тільки один раз.

Як тільки значення розміщується в зовнішньому ключі, батьківський ключ перевіряється на наявність в ньому такого значення. Якщо воно відсутнє, команда блокується.

Наприклад, відношення **ПОСТАЧАЛЬНИК** зв'язане 1: N із відношенням **ПОСТАЧАННЯ** (табл. 4.4, 4.5). Цей зв'язок реалізується за допомогою додавання зовнішнього ключа *Постачальник* до відношення **ПОСТАЧАННЯ**.

Таблиця 4.4 – Відношення ПОСТАЧАЛЬНИК

Код	Найменування	Місто	Адреса	Телефон
1	ФОТОКОМ	Запоріжжя	вул.Лермонтова, 5	(061)33-14-94
2	ДКТ	Донецьк	вул.Правди, 20	(062)24-09-68
3	КВАЗАР	Запоріжжя	вул.Верхня, 28	(061)32-45-15
4	dataLux	Одеса	вул.Приморська, 13	(048)23-39-95
5	ЮСТР	Севастополь	вул.Довженка 16	(044)241-75-00
6	РОМА	Донецьк	вул.Шахтарська, 39	(062)34-27-04
7	ЮСТР	Запоріжжя	вул.Мира,79	(061)234-45-45
8	КВАЗАРo	Харків	вул.Сумська, 23	(057)443-83-96
9	AT-company	Київ	пр.Перемоги, 15	(044) 232-78-89

Таблиця 4.5 – Відношення ПОСТАЧАННЯ

Код	Дата	Вартість, грн	Постачальник
1	12.01.12	15876	4
2	25.01.12	56432	2
3	01.02.12	23456	4
4	13.02.12	71635	1
5	15.02.12	52371	3
6	01.03.12	45234	3
7	25.03.12	61324	7
8	12.04.12	15876	4
9	25.04.12	56432	9

4.2.3 Попередні відношення для бінарних зв'язків ступеня $M:N$

Якщо між двома сутностями існує зв'язок $M:N$, то для їх відображення необхідні три відношення – по одному для кожної сутності та одне для зв'язку. Зв'язне відношення при цьому повинно містити зовнішні ключі для зв'язку з відповідними об'єктами відношеннями.

Наприклад, між відношенням **ПОСТАЧАННЯ** та відношенням **ТОВАРИ** (табл. 4.5, 4.6). існує зв'язок $M:N$. Цей зв'язок реалізується за допомогою використання додаткового відношення **СКЛАД_ПОСТАЧАННЯ** (табл. 4.6).

Таблиця 4.6 – Відношення ТОВАРИ

Код	Модель	Ціна
6	Cyber Drive 48x	240,00 грн
7	DeskJet 7420	538,00 грн
14	HP 920C	609,00 грн
16	Stylus Color S42UX	375,00 грн
17	HP LaserJet 1000w	1448,00 грн
18	ASUS 50x	360,00 грн
26	CD-RW GCE-8480BB	492,00 грн
29	GM56PCI-I	123,00 грн
30	DI5732	105,00 грн
32	DIMM 512Mb PC133	644,00 грн
35	CT-4810 Sound Blaster	96,00 грн
36	100 Gb WD100EB	366,00 грн
37	200 Gb ST320014A	468,00 грн
40	ML-1210	1088,00 грн
41	Duron 2800MHz	169,00 грн
44	Perfection 1650SU	840,00 грн

Таблиця 4.6 – Відношення СКЛАД_ПОСТАЧАННЯ

Код	Товар	Ціна	Кількість	Сума
1	29	123,00 грн	50	6 150,00 грн
1	18	360,00 грн	100	36 000,00 грн
1	29	123,00 грн	200	24 600,00 грн
1	16	375,00 грн	75	28 125,00 грн
2	32	644,00 грн	125	80 500,00 грн
3	18	360,00 грн	200	72 000,00 грн
4	32	644,00 грн	100	64 400,00 грн
4	16	375,00 грн	150	56 250,00 грн

4.3 Нормалізація баз даних

4.3.1 Аномалії ненормалізованого відношення

При проектуванні схеми реляційної БД атрибути групуються у відношення на підґрунті або здорового глузду розробника, або шляхом виводу реляційної схеми з розробленої ER-діаграми.

При експлуатації БД у відношенні можуть виникнути аномалії, пов'язані з виконанням операції підтримки його в актуальному стані. Виявити фактичний прояв тієї чи іншої аномалії можна, лише врахувавши конкретну семантику даних.

Для ілюстрації можливих аномалій розглянемо відношення **КОМПАНІЯ** (код_працівника, номер_відділу, керівник, тип_контракту) (табл. 4.23).

Таблиця 4.23 – Відношення КОМПАНІЯ

код_працівника	номер_відділу	керівник	тип_контракту
12	23	Притула	короткостроковий
56	4	Василенко	
94	12	Кличко	
15	8	Рисіков	короткостроковий
68	12	Кличко	

В цьому відношенні під час виконання операцій актуалізації стану БД можуть виникати наступні аномалії.

1 Аномалія оновлення (UPDATE). Заміна керівника відділу призведе до необхідності внесення змін і модифікації по кожному працівнику даного відділу. Отже, для підтримки узгодженості

даних необхідно виконати зміну не лише в кортежі бази, а цілий ряд змін, що може розглядатися як аномалія, оскільки характер самої зміни повинен стосуватися лише одного певного запису БД.

2 Аномалія поповнення (INSERT). Приймаючи на роботу нового співробітника, необхідно вносити до БД відомості не лише про нього, а й про керівника та тип контракту. Це також можна розглядати як аномалію, тому що не завжди можливо внести до БД відомості про відділ і контракт, оскільки часто співробітників беруть на роботу з випробувальним терміном і лише після його проходження підписують певний вид контракту та визначають відділ, де він працюватиме.

3 Аномалія вилучення (DELETE). Припустимо, що звільнились усі працівники певного відділу. Тоді разом з інформацією про них у БД буде втрачено інформацію про цей відділ. Якщо цю інформацію необхідно зберігати, то це також може розглядатись як аномалія.

4 Надмірність. Відомості про керівника відділу повторюються в ряді кортежів БД. Основні проблеми із зберіганням надлишку інформації пов'язані не лише з неефективним використанням пам'яті, а й із забезпеченням підтримки узгодженості цих даних.

Усунути можливість аномалій або, принаймні, зменшити їхню кількість можна, використовуючи формальний метод, запропонований Е. Коддом і який дозволяє проектувальнику знаходити оптимальне угруповання атрибутів для кожного відношення схеми БД. Перш ніж ознайомитися з цим методом, який називається **нормалізацією** схеми БД, треба розглянути поняття *функціональної залежності*.

4.3.2 Типи функціональних залежностей

Відношення реляційної БД містять як структурну, так і семантичну (сенсову) інформацію. Структурна інформація задається схемою відношень, а семантична виражається функціональними зв'язками між атрибутами, що враховуються в схемі.

Склад атрибутів відношень БД повинний задовольняти двом основним вимогам:

- між атрибутами не повинно бути небажаних функціональних залежностей;
- угруповання атрибутів повинні забезпечувати мінімальне дублювання даних, забезпечувати їхнє опрацювання та оновлення без труднощів.

Термін *функціональна залежність* (*ФЗ*) означає: атрибут *B* відношення *R* функціонально залежить від атрибута *A* того ж відношення, якщо в кожному момент часу кожному значенню атрибута *A* відповідає тільки одне значення атрибута *B*, зв'язаного з *A* у відношенні *R*:

$$f: A \rightarrow B.$$

Цій залежності відповідає співвідношення 1:1 між атрибутами (наприклад, між атрибутами **код_деталі** → **назва_деталі**).

Ліву і праву частини символічного запису ФЗ називають також *детермінантом* і *залежною частиною*, відповідно.

Роздивимося як приклад відношення **ВИКЛАДАЧ_ДИСЦИПЛІНА** із складовим ключем, що складається з двох атрибутів: **таб_ном** і **назва_дисц** (табл. 4.24).

У цьому прикладі маємо функціональні залежності **посада** → **оклад**; **таб_ном** → **прізвище** та інші.

Таблиця 4.24 – Відношення ВИКЛАДАЧ-ДИСЦИПЛІНА

таб_ном	назва_дисц	кільк_годин	прізвище	посада	оклад	каф	тел
201	ОФ ЕОМ	36	Рижов	Доцент	700	ЕОМ	2–52
201	СОД	36	Рижов	Доцент	700	ЕОМ	2–52
301	СКБД	48	Костенко	Доцент	700	ЕОМ	2–52
504	СКБД	32	Кравець	Професор	950	АСУ	3–14
410	Фізика	52	Головко	Асистент	500	фізики	4–53
324	ВМ	36	Бойко	Асистент	500	ВМ	4–64

Якщо неключовий атрибут залежить тільки від частини ключа, то він знаходиться в *частковій функціональній залежності*. Наприклад, неключовий атрибут **прізвище** залежить від частини ключа, тобто тільки від атрибуту **таб_ном**.

Якщо для атрибутів *A*, *B*, *C* виконуються умови $A \rightarrow B$ і $B \rightarrow C$, але обернена залежність відсутня, то *C* знаходиться в *транзитивній залежності* від *A*. Приклад транзитивної залежності: **прізвище** → **посада** → **оклад**. Транзитивна залежність призводить до дублювання даних.

У відношеннях між атрибутами може існувати й *багатозначна залежність*, яка є узагальненням функціональної залежності.

У відношенні **R** атрибут **B** багатозначно залежить від атрибута **A** ($A \rightarrow > B$), якщо кожному значенню **A** відповідає численність значень **B**, ніяк не зв'язаних з іншими атрибутами з **R**.

Наприклад, задано відношення **СПІВРОБІТНИК** (табельний_номер, державна_нагорода, прізвище_ім'я_та_по_батькові_члена_сім'ї, родинні_відносини) (табл. 4.25).

У цьому відношенні мають місце незалежні багатозначні залежності:

табельний_номер $\rightarrow >$ **державна_нагорода**;

табельний_номер $\rightarrow >$ **ПІБ_члена_сім'ї**,

тому що значення багатозначних атрибутів **державна_нагорода** та **ПІБ_члена_сім'ї** ніяк не зв'язані між собою й можлива зміна їхніх значень у будь-якому рядку відношення.

Таблиця 4.25 – Відношення СПІВРОБІТНИК

таб_ном	держ_нагорода	ПІБ_члена_сім'ї	родин_віднос
123	орден «За заслуги»	Шайко Іван Миколайович	син
123	орден «За заслуги»	Шайко Ганна Петрівна	дружина
123	орден «За мужність»	Шайко Іван Миколайович	син
123	орден «За мужність»	Шайко Ганна Петрівна	дружина

Присутність багатозначних залежностей у схемі відношення та незалежність їх правих частин призводить до комбінаторики правих частин залежностей, тобто до надлишкового дублювання даних.

Поняття багатозначної залежності складніше, ніж поняття функціональної залежності. Для виявлення багатозначної залежності потрібний значно глибший семантичний аналіз атрибутів.

Наявність розглянутих залежностей у відношеннях призводить до аномалій оновлення, що усуваються нормалізацією відношень.

Поняття функціональної залежності атрибутів не можна рахувати цілком еквівалентним математичному поняттю функціональної залежності, оскільки значення цієї залежності різні для різних станів відношення, та, головне, ці значення можуть змінюватися непередбачено.

Функціональна залежність стверджує лише те, що для кожного конкретного стану бази даних по значенню одного атрибута (детермінанта) можна однозначно визначити значення іншого атрибута (залежної частини). Але конкретні значення залежної частини можуть бути різними для різних станів бази даних.

4.3.3 Поняття нормалізації

Нормалізація – це процес, у результаті якого можна позбавитися дефектів проектування бази даних. У процесі нормалізації ми одержуємо ряд нормальних форм, використовуючи набір правил, що описують те, що слід і що не слід робити із структурою таблиць.



Нормалізація відношень – це покроковий зворотний процес декомпозиції (розкладання) вихідних відношень БД на інші, більш дрібні та прості відношення, внаслідок чого формується краща структура.

Під зворотністю процесу розуміється те, що операція з'єднання відношень, отриманих в результаті декомпозиції, має дати початкове відношення, тобто при виконанні декомпозиції повинна виконуватися умова з'єднання без втрат інформації.

Нормалізація таблиць бази даних – перший крок на шляху проектування структури реляційної бази даних. Теорія реляційних баз даних була розроблена в кінці 70-х років 20 століття. Відповідно до неї, виділяються шість нормальних форм, п'ять з яких так і називаються: перша нормальна форма, друга нормальна форма, третя нормальна форма, четверта нормальна форма, п'ята нормальна форма, а також нормальна форма Бойса-Кодда, що лежить між третьою і четвертою.



Нормальна форма – властивість відношення в реляційної моделі даних, що характеризує його з точки зору надмірності, яка потенційно може призвести до логічно помилкових результатів вибірки або зміни даних. Нормальна форма визначається як сукупність вимог, яким має задовольняти відношення.

База даних вважається нормалізованою, якщо її таблиці представлені як мінімум в третій нормальній формі. Часто багато таблиць нормалізуються до четвертої нормальної форми. Очевидно, що в нормалізованій базі даних зменшується ймовірність виникнення помилки, і вона займає менше місця на жорсткому диску.

Приведення моделі до необхідного рівня нормальної форми є основою побудови реляційної БД. У процесі нормалізації

елементи даних групуються в таблиці, що визначають об'єкти та їхні взаємозв'язки. Теорія нормалізації базується на тому, що певний набір таблиць має кращі властивості при додаванні, модифікації та вилучанні даних, ніж всі інші набори таблиць, за допомогою яких можуть бути представлені ті ж дані.

Введення нормалізації відношень при розробці інформаційної моделі забезпечує мінімальний обсяг фізичної БД і її максимальну швидкодію, що напряму відбивається на якості функціонування інформаційної системи. Нормалізація інформаційної моделі виконується в декілька етапів, під час яких вона послідовно приводиться до нормальних форм більш високого рівня.

4.3.4 Перша та друга нормальні форми



Відношення знаходиться в першій нормальній формі (1НФ), якщо всі його неключові атрибути прості, тобто містять атомарні, неподільні значення.

Приклад 1 Розглянемо відношення зі складовим ключем **ВИКЛАДАЧ_ДИСЦИПЛІНА** (табл. 4.24). Воно знаходиться в **1НФ**, оскільки крім ключа не містить складових атрибутів. У цьому відношенні є часткова функціональна залежність атрибутів **прізвища, посада, оклад, каф, тел** від частини **таб_ном** складового ключа. Така часткова залежність призводить до таких аномалій:

- має місце дублювання даних про викладача, оскільки викладач може читати декілька дисциплін;
- існує проблема контролю надмірності даних, тому що зміна, наприклад, окладу викликає необхідність пошуку та зміни значень окладів у всіх кортежах з цим викладачем;
- виникає проблема з викладачами, які зараз не ведуть дисципліни (підвищують кваліфікацію), оскільки викладача без дисципліни неможливо включити до відношення.

Аналогічно неприпустимо включення нової дисципліни, якщо за нею не закріплений жоден викладач. З іншого боку, якщо викладач звільняється та вилучається з відношення, то буде вилучена й дисципліна, що неприпустимо.

Таким чином, відношення в 1НФ потребує подальших перетворень.

Друга нормальна форма



Відношення знаходиться в 2НФ, якщо воно знаходиться в 1НФ і в ньому відсутні часткові функціональні залежності описових атрибутів від атрибутів первинного ключа.

Щоб усунути часткову залежність і привести відношення до 2НФ, необхідно розкласти його на два відношення в такий спосіб:

- побудувати проекцію без атрибутів, що знаходяться в частковій функціональній залежності від складового ключа;
- побудувати проекцію на частину складового ключа й атрибути, що залежать від цієї частини.

У результаті одержимо два відношення, **НАВАНТАЖЕННЯ** та **ВИКЛАДАЧ**, що знаходяться в 2НФ (табл. 4.26, 4.27):

Таблиця 4. 26 – Відношення НАВАНТАЖЕННЯ

таб_ном	назва_дисц	кільк_годин
201	ОФ ЕОМ	36
201	СОД	36
301	СКБД	48
504	СКБД	48
410	фізика	52
324	матаналіз	36

Таблиця 4. 27 – Відношення ВИКЛАДАЧ

таб_ном	прізвище	посада	оклад	каф	тел
201	Рижов	доцент	500	ЕОМ	2-52
301	Костенко	доцент	500	ЕОМ	2-52
504	Кравець	професор	750	АСУ	3-14
410	Головко	асистент	400	фізики	4-53
324	Бойко	асистент	400	ВМ	4-64

У отриманому відношенні **ВИКЛАДАЧ** є транзитивні функціональні залежності, наприклад:

таб_ном → каф → тел; таб_номер → посада → оклад.

Наявність транзитивних залежностей породжує незручності й аномалії такого характеру (на прикладі атрибута **тел**):

- має місце дублювання інформації про телефон для викладачів однієї кафедри;
- існує проблема надмірності, оскільки зміна номера телефону кафедри викликає необхідність пошуку та зміни номерів у всіх викладачів цієї кафедри;
- не можна включити дані про нову кафедру (назва й номер телефону), якщо на даний момент ще відсутні викладачі. Та навпаки, при звільненні усіх викладачів із кафедри дані про неї не можна зберегти.

Таким чином, відношення в 2НФ також може вимагати подальших перетворень.

4.3.5 Третя нормальна форма



Відношення знаходиться в 3НФ, якщо воно знаходиться в 2НФ і не містить транзитивних залежностей.

Транзитивна залежність обумовлює дублювання даних в одному відношенні. Приведення відношення до третьої нормальної форми полягає в усуненні транзитивних залежностей шляхом розщеплення початкового відношення (рис. 4.1, а) на два (рис. 4.1, б).

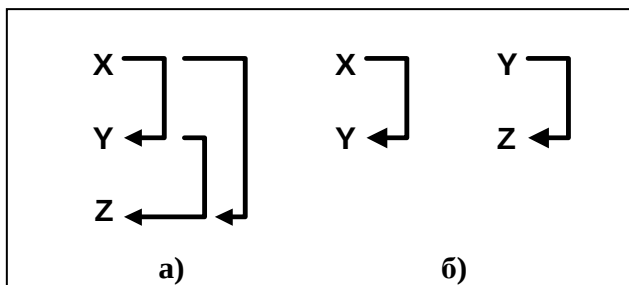


Рисунок 4.1 – Усунення транзитивних залежностей

Слід мати на увазі, що при наявності у відношенні ланцюжку типу $A \rightarrow B \rightarrow C \rightarrow D$ виділення залежностей (проекцію відношення) треба робити справа наліво, тобто спочатку виділити залежність $C \rightarrow D$, а потім поступово решту, що залишилися.

Приведення відношення **ВИКЛАДАЧ** (табл. 4.27) до 3НФ дає три відношення (таблиці 4.28–4.30).

Таблиця 4.28 – Відношення ВИКЛАДАЧ

таб_ном	прізвище	посада	каф
201	Рижов	доцент	ЕОМ
301	Костенко	доцент	ЕОМ
504	Кравець	професор	АСУ
410	Головко	асистент	фізики
324	Бойко	асистент	ВМ

Таблиця 4.29 – ПОСАДА

посада	оклад
доцент	700
професор	950
асистент	500

Таблиця 4.30 – КАФЕДРА

каф	тел
ЕОМ	2-52
АСУ	3-14
фізики	4-53
ВМ	4-64

Приклад 2 Розглянемо початкове відношення **СТУДЕНТ** (табл. 4.31), в якому міститься така інформація:

- с_ном – унікальний номер студента;
- с_прізвище – прізвище студента (можуть бути однаковими у декількох студентів);
- к_ном – номер кімнати у гуртожитку студентського містечка. В одній кімнаті можуть мешкати кілька студентів;
- т_ном – номер телефону студента (один у кімнаті);
- дисц – дисципліна, що вивчається;
- сем – семестр, у якому вивчається дисципліна (можливе вивчення дисципліни у кількох семестрах);
- оцінка – оцінка, одержана студентом за певну дисципліну в даному семестрі.

У цьому відношенні існують функціональні залежності, діаграми яких наведені на рис. 4.2:

с_ном → с_прізвище;

с_ном → к_ном → т_ном;

с_ном → т_ном;

т_ном → к_ном;

к_ном → т_ном;

(с_ном, дисц, сем) → оцінка

Таблиця 4.31 – Відношення СТУДЕНТ

с_ном	с_прізви	к_ном	т_ном	дисц	сем	оцінка
321	Шрамко	51	4-82	МПС	3	5
321	Шрамко	51	4-82	БД	4	4
321	Шрамко	51	4-82	Фізика	2	3
327	Кравчук	52	2-15	МПС	3	3
327	Кравчук	52	2-15	БД	4	4
327	Кравчук	52	2-15	Фізика	2	5
330	Чумак	51	4-82	МПС	3	3
330	Чумак	51	4-82	БД	4	4
340	Попович	59	3-36	Фізика	1	3

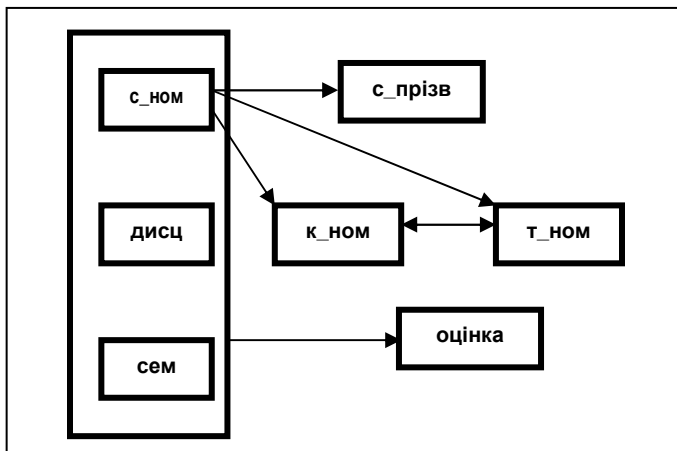


Рисунок 4.2 – Діаграма функціональних залежностей відношення СТУДЕНТ

У цьому відношенні маємо залежність атрибутів **к_ном**, **с_прізви**, **т_ном** від частки **с_ном** складового ключа (**с_ном**, **курс**, **сем**), яка свідчить, що відношення не знаходиться в 2НФ. Тому треба переробити цю діаграму шляхом проєкції вказаних атрибутів до нового відношення (рис. 4.3). Відповідні відношення наведені у табл. 4.32 та 4.33.

У відношенні **СТУДЕНТ** маємо транзитивну залежність **с_ном** → **к_ном** → **т_ном**. В якості детермінанта цього ланцюжка виступає атрибут **с_ном**, а не атрибут **с_прізви**, тому що кілька студентів можуть мати однакові прізвища.

Наявність цієї залежності призводить до надмірного дублювання даних про телефон, що потребує в разі зміни номеру пошук всіх кортежів, які містять старий номер. Отже, потрібне подальше перетворення одержаного відношення.

Треба провести декомпозицію відношення **СТУДЕНТ** (рис. 4.2, б та, відповідно, табл.4.33) на два відношення **СТУДЕНТ** і **КІМНАТА**, що наведені в табл. 4.34 і 4.35, а їхня діаграма функціональних залежностей – на рис. 4.4.

Таблиця 4.34 – Відношення **СТУДЕНТ**

с_ном	с_прізви	к_ном
321	Шрамко	51
327	Кравчук	52
330	Чумак	51
340	Попович	59

Таблиця 4.35 – Відношення **КІМНАТА**

к_ном	т_ном
51	4–82
52	2–15
59	3–36

ЗНФ звільняє від надмірності та аномалій виконання операцій додавання, вилучання та оновлення (зміни), якщо відношення має один ключ, а інші залежності, у тому числі багатозначні, у ньому відсутні. Але якщо при цьому є інші залежності, то ЗНФ не забезпечує відсутності аномалій операцій оновлення. У цьому випадку застосовують посилену ЗНФ, або НФБК.

Згідно з [10], форми 1НФ, 2НФ і 3НФ мають значення не самі по собі, а лише як проміжні етапи побудови НФБК (та форм більш високого рівня).

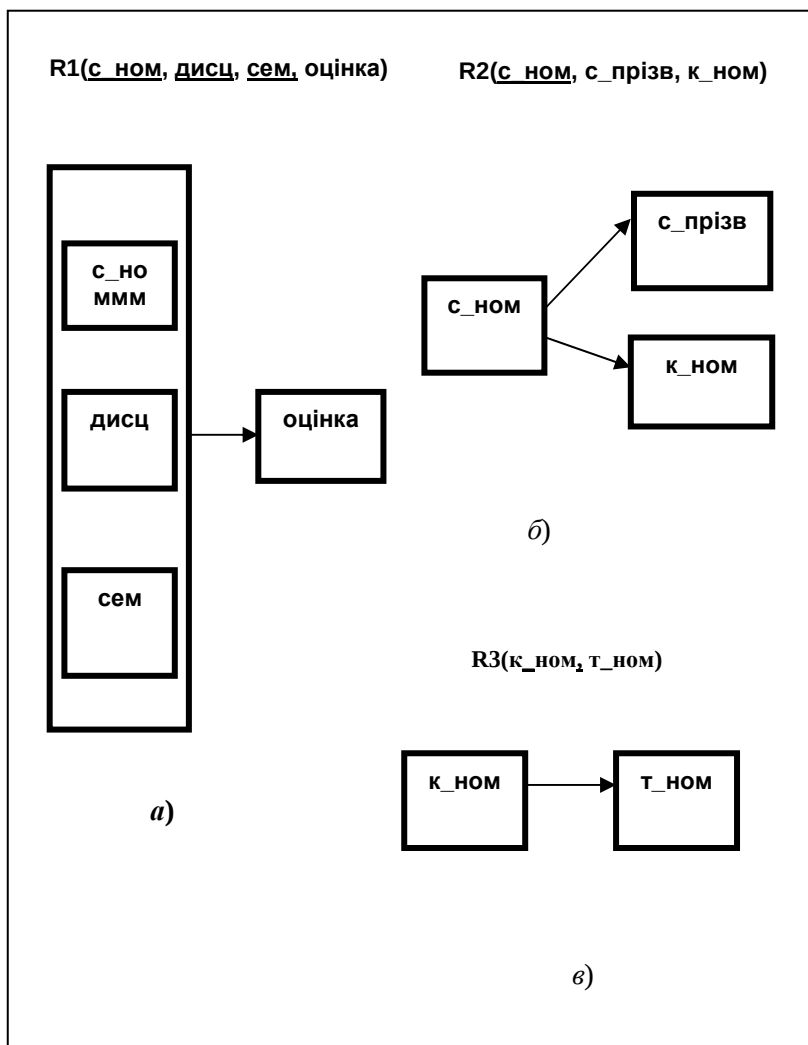


Рисунок 4.4 – Остаточна діаграма ФЗ попереднього відношення СТУДЕНТ (табл. 4.31)

4.3.6 Нормальна форма Бойса-Кодда (НФБК)



Відношення знаходиться в НФБК тоді й тільки тоді, коли детермінанти є потенційними ключами.

Іншими словами, на діаграмі функціональної залежності стрілки будуть починатися тільки з **потенційних** ключів.

Відношення може знаходитися в ЗНФ, але не бути при цьому в НФБК. Відношення ж, яке задане в НФБК, завжди знаходиться в ЗНФ.

Приклад 3 Нехай є відношення **ВИВЧЕННЯ_ДИСЦИПЛІНИ** з атрибутами **студент, дисципліна, викладач (СДВ)** (табл. 4.36). При цьому накладаються такі обмеження:

- кожний студент, що вивчає дану дисципліну, навчається тільки у одного викладача.
- кожний викладач веде тільки одну дисципліну.

Таблиця 4.36 – Відношення **ВИВЧЕННЯ_ДИСЦИПЛІНИ**

студент	дисципліна	викладач
Івахненко	МПС	Скачко
Петров	СОД	Рисіков
Чумак	МПС	Скачко
Кравець	БД	Степаненко
Сидоров	БД	Степаненко

Роздивимось функціональні залежності, що існують у даному відношенні (рис. 4.5). З першого обмеження випливає залежність атрибута **викладач (В)** від комбінації атрибутів **студент (С)** і **дисципліна (Д)**, а з другого – залежність атрибута **дисципліна** від атрибута **викладач**.

Дане відношення знаходиться в ЗНФ і характеризується деякими аномаліями оновлення. Наприклад, якщо потрібно вилучити інформацію про те, що Петров вивчає СОД, це не можна буде зробити, не втрачаючи інформацію про те, що викладач Рисіков веде дану дисципліну. Ці труднощі викликані тим, що атрибут **викладач** є **детермінантом**, але **не є потенційним** ключем.

Для вирішення цієї проблеми вихідне відношення варто розбити на дві проекції (табл. 4.37 і 4.38), що знаходяться в НФБК.

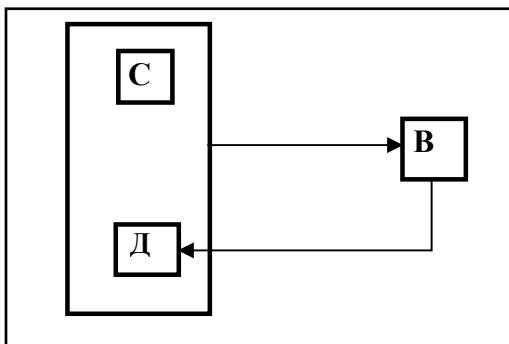


Рисунок 4.5 – Діаграма функціональних залежностей відношення ВИВЧЕННЯ_ДИСЦИПЛІНИ

Таблиця 4.37 – ВИКЛАДАЄ

викладач	дисципліна
Скачко	МПС
Рисіков	СОД
Степаненко	БД

Таблиця 4.38 – ВИВЧАЄ

студент	дисципліна
Івахненко	МПС
Петров	СОД
Чумак	МПС
Кравець	БД
Сидоров	БД

З'єднання отриманих відношень за атрибутом *дисципліна* дає початкове відношення **ВИВЧЕННЯ_ДИСЦИПЛІНИ**.

Слід мати на увазі, що відношення у ЗНФ не завжди можна привести до НФБК, не втративши залежності між його атрибутами.

Приклад 4 Нехай задано відношення R (місто (M), адреса (A), індекс (I)). В ньому мають місце такі нетривіальні залежності (рис.4.6):

(місто, адреса) $\rightarrow$ індекс; індекс $\rightarrow$ місто.

Це відношення перебуває в ЗНФ, тому що в ньому відсутні часткові та транзитивні залежності. Однак воно не знаходиться в НФБК, тому що в ньому присутня залежність **індекс $\rightarrow$ місто**, в якій детермінант **індекс** не є потенційним ключем.



Рисунок 4.6 – Діаграма ФЗ відношення R

При зведенні цього відношення до НФБК одержимо два відношення **R1(місто, адреса)** і **R2(індекс, місто)**, в яких будуть відображені не всі залежності початкового відношення. Так, не буде відображена залежність **індекс → адреса**, а саме індекс визначає відділення зв'язку, що обслуговує адресатів якоїсь вулиці певного міста.

Отже, зведення до НФБК може призвести до втрати важливих залежностей, а з'єднання отриманих в результаті декомпозиції відношень не дасть початкового відношення. Тому при зведенні до НФБК необхідно ретельно вивчити всі залежності та проводити його лише тоді, коли виконується вимога «**з'єднання без втрат**».

4.3.7 Четверта нормальна форма



Відношення знаходиться в 4НФ, якщо воно знаходиться в НФБК і в ньому відсутні незалежні багатозначні залежності, тобто всі вони виділені (рознесені) в окремі відношення з тим самим ключем.

Якщо у відношенні присутні багатозначні функціональні залежності, то схема відношення повинна знаходитися в 4НФ.

Нехай є відношення **СПВРОБІТНИК** (табл. 4.25) із незалежними багатозначними залежностями **таб_ном →> держ_нагорода** та **таб_ном →> ПІБ_члена_сім'ї**.

Аномалії виконання операцій тут можуть бути такими:

- при появі нової нагороди у співробітника до відношення треба додавати не один кортеж, а стільки, скільки членів сім'ї він має;
- аналогічна ситуація виникає з включенням до відношення нового члена сім'ї.

Усунення аномалії досягається розкладанням початкового відношення на декілька (тут на два) відношення з багатозначними залежностями від одного й того ж ключа (тут **таб_ном**) (табл. 4.39 і 4.40).

Таблиця 4.39 – НАГОРОДИ

таб_ном	держ_нагорода
123	орден «За заслуги»
123	орден «За мужність»

Таблиця 4.40 – ЧЛЕНИ_СІМ'Ї

таб_ном	член_сім'ї	родин_віднос
123	Шайко Іван Миколайович	син
123	Шайко Ганна Петрівна	дружина

Підсумки

Процес нормалізації відношень послідовно усуває такі типи функціональної залежності:

- часткові залежності неключових атрибутів від ключа;
- транзитивні залежності неключових атрибутів від ключа;
- залежності ключів від неключових атрибутів або залежності, в яких детермінанти не є потенційними ключами;
- незалежні багатозначні залежності, тощо.

На практиці достатньо довести нормалізацію до 4НФ, оскільки відношення, які потребують подальшої нормалізації, є екзотичними й зустрічаються досить рідко [11].

Слід зазначити, що нормалізація збільшує число відношень БД, проте завдяки коректності та усуненню дублювання даних прискорюється виконання операцій доступу до даних, а також підвищується ступінь достовірності даних.

Слід зауважити, що інколи доцільно розбивати велику таблицю на декілька, навіть якщо вона відповідає вимогам нормальних форм. Це виконується з метою зменшення простору пошуку та, відповідно, підвищення швидкодії системи, тому що час виконання запиту напряму залежить від розміру таблиці.

Наприклад, таблиця **Співробітники** БД «Відділ кадрів» містить інформацію, що складається з двох частин: особистих і виробничих відомостей про співробітників. При виконанні запиту до цієї таблиці, скоріше за все, нам знадобиться тільки один якийсь вид інформації. Тому з метою прискорення виконання запитів доцільно розділити цю таблицю на дві: **Особисті відомості** та **Виробничі відомості**.

4.4 Індексція

Таблиці реляційної БД зберігаються у пам'яті у вигляді файлів з послідовним доступом. Основним недоліком послідовних файлів є те, що для вибірки інформації з них необхідно здійснити послідовний пошук, який вимагає більших витрат часу.

Індексція є механізмом підвищення продуктивності. Подібно предметному покажчику, що допомагає знаходити необхідні сторінки в книзі, індексування прискорює вибірку інформації з БД. При пошуку теми в книзі не треба переглядати підряд усі сторінки. Аналогічно, при пошуку потрібних даних індекси служать логічними покажчиками на їхнє фізичне місце розташування.

Користувач СКРБД тільки вирішує, створювати йому індекс чи ні. Після свого створення індекси працюють і використовуються незалежно від користувачів БД. При цьому підтримку індексів бере на себе система.



Індекс (index) – це упорядкований певним чином список умісту стовпців або групи стовпців у таблиці з вказівкою їхнього розташування у пам'яті.

Таким чином, індекс забезпечує прямий доступ до окремих записів файлу, тобто певний запис може бути зчитаний без перегляду інших записів файлу.

Таблиці можуть мати велику кількість рядків і, оскільки рядки задаються в будь-якому довільному порядку, їхній пошук за значенням якогось з полів може зайняти досить багато часу. Індекси саме й призначені для рішення цієї проблеми.

Індекси створюються, наприклад, за допомогою команди SQL (див. розд.5):

**CREATE [UNIQUE] INDEX ім'я індексу
ON ім'я_таблиці (ім'я_стпчика [, ім'я_стпчика]..);**

Наприклад,
create index товар_ind
on Товари (назва);

Індекси можна створювати по безлічі полів. Якщо зазначено більш одного поля для створення єдиного індексу, дані упорядковуються за значеннями першого поля, за яким здійснюється індексування. Усередині отриманих груп здійснюється упорядкування за значеннями другого поля; для груп, отриманих в результаті, здійснюється упорядкування за значеннями третього поля тощо.

Коли організується індекс за значеннями якого-небудь поля, створюється упорядкований список значень для цього поля. Припустимо, таблиця **ПОСТАЧАЛЬНИКИ** має тисячі рядків і потрібно знайти постачальника з номером 2999. Оскільки рядки не упорядковані, програма повинна переглянути всю таблицю, рядок за рядком, і вибрати ту, у якій значення поля *код_постач* дорівнює 2999. Якби по полю *код_постач* був організований індекс, програма могла б відразу знайти в ньому значення 2999 і одержати інформацію про те, як знайти потрібний рядок таблиці.

Використання індексів може значно поліпшити виконання запитів на вибірку, але керування індексами істотно сповільнює час виконання операцій оновлення, таких як додавання та вилучання (дивиться розділ 5); крім того, сам індекс займає місце в пам'яті. Отже, перед створенням індексів варто ретельно проаналізувати ситуацію.

4.5 Критерії оцінки якості логічної моделі даних

Рішення, прийняті на попередньому етапі, при розробці моделі предметної області, визначають деякі границі, у межах яких можна розвивати логічну модель даних. У межах цих границь можна приймати різні рішення. Наприклад, модель предметної області складського обліку містить поняття «склад», «накладна», «товар». При розробці відповідної реляційної моделі ці терміни обов'язково повинні бути використані, але різних способів реалізації тут багато – можна створити одне відношення, у якому будуть присутні у якості атрибутів поняття «склад», «накладна», «товар», а можна створити три окремих відношення, по одному на кожне поняття.

При розробці логічної моделі даних виникають питання: чи добре спроектовані відносини? Чи правильно вони відображають модель предметної області, а отже й саму предметну область?

Для оцінки якості прийнятих рішень на рівні логічної моделі даних необхідно сформулювати деякі критерії оцінки таких рішень і проаналізувати, як різні рішення, прийняті в процесі логічного моделювання, впливають на функціональні характеристики бази даних.

Звичайно, таких критеріїв може бути дуже багато, а їхній вибір у достатньому ступені довільним. Розглянемо деякі з таких критеріїв, які є безумовно важливими з погляду одержання якісної БД [20]:

- адекватність відображення предметної області;
- швидкість виконання операцій модифікації даних (вставка, оновлення, вилучання кортежів);
- швидкість виконання операцій вибірки даних.

Розглянемо більш детально ці критерії.

Адекватність відображення предметної області

База даних повинна адекватно відображати предметну область. Це означає, що повинні виконуватися наступні умови:

- стан бази даних у кожний момент часу повинний відповідати стану предметної області;
- зміна стану предметної області повинна приводити до відповідного зміні стану бази даних;
- обмеження предметної області, закладені в модель ПО, повинні певним чином відбиватися й враховуватися у базі даних.

Швидкість операцій модифікації даних

Основними операціями, що змінюють стан бази даних, є операції вставки, оновлення та вилучання записів. У базах даних, що вимагають постійних змін (складський облік, системи продажів квитків тощо) продуктивність визначається швидкістю виконання великої кількості таких операцій.

Згідно дослідженням [28], швидкість виконання операції вставки зменшується при збільшенні кількості індексів у таблиці й мало залежить від числа рядків у таблиці.

Швидкість виконання операцій оновлення та вилучання також зменшується при збільшенні кількості індексів у таблиці та мало залежить від числа рядків у таблиці

Крім того, чим більше атрибутів мають відносини, розроблені в ході логічного моделювання, тим повільніше будуть виконуватися операції модифікації даних за рахунок витрати часу на перебудову більшої кількості індексів.

Швидкість операцій вибірки даних

Одне з призначень бази даних – надання інформації користувачам. Інформація витягається з реляційної бази даних за допомогою оператора SQL – SELECT. Однією з найбільш «важких» операцій при виконанні оператора SELECT є операція з'єднання таблиць. Чим більше взаємозалежних відносин було створено в ході логічного моделювання, тим більше імовірність того, що при виконанні запитів ці відносини будуть з'єднуватися, і тим повільніше будуть виконуватися запити. Таким чином, збільшення кількості відносин приводить до уповільнення виконання операцій вибірки даних.

4.6 Подання (курсори)

4.6.1 Поняття подання

У главі 1 при розгляді трирівневої архітектури зовнішнє подання описувалося як структура БД із погляду окремого користувача. У реляційній моделі слово «подання» (**view**) має декілька інше значення. Воно характеризує не всю зовнішню модель користувача, а лише якість **віртуальне відношення (virtual relation)**, що їм використовується, тобто відношення, яке насправді само по собі не існує, але динамічно відтворюється на підставі одного або декількох **базових відносин** (відносин, що реально існують у базі даних).

Подання може бути побудоване на базі виконання таких операцій реляційної алгебри, як вибірка, проєкція, з'єднання, або на основі інших операцій з даними з реально існуючих базових відносин.

 **Базове відношення** – це поймаєменоване відношення, що відповідає сутності в концептуальній схемі, кортежі якого фізично зберігаються в базі даних.

Поняття подання визначається на основі базових відносин.

 **Подання** – це динамічний результат однієї або декількох реляційних операцій над базовими відносинами з метою створення деякого іншого відношення. Подання є **віртуальним відношенням**, якого реально в базі даних не існує, але яке створюється на вимогу окремого користувача в момент надходження цієї вимоги.

З погляду користувача подання є відношенням, яке постійно існує та з яким можна працювати так само, як із базовим відношенням.

Однак подання не зберігається в базі даних так, як базові відносини. Вміст подання визначається як результат виконання запиту до одного або декількох базових відносин. Наприклад, на мові SQL можна створити подання, в якому буде міститися інформація про книги видавництв, розташованих у місті Києві.

```
create view видавництво_книги  
as  
select видавництво.назва, книги.назва  
from видавництво, книги  
where (видавництво.код = книги.код_видавництва) and  
(видавництво.місто = «Київ»);
```

Будь-які операції над поданням автоматично транслюються в операції над відносинами, на підставі яких це подання було створено. Подання мають *динамічну* природу, тобто зміни в базових відносинах, які можуть вплинути на вміст подання, негайно відбиваються на вмісті цього подання.

Якщо користувачі вносять у подання деякі припустимі зміни, ці зміни негайно заносяться в базові відносини подання. Більш докладно способи визначення й робота з поданнями описуються в розділі 5.8.

4.6.2 Призначення подань

Механізм подань може використовуватися з кількох причин:

- воно надає потужний і гнучкий механізм захисту за рахунок приховування частин бази даних від певних користувачів. Користувач не буде мати ніяких відомостей про існування будь-яких атрибутів або кортежів, відсутніх у доступних йому поданнях (більш докладно про це буде сказано в розділі 6.3);
- воно дозволяє організувати доступ користувачів до даних найбільш зручним для них чином. Тому одні й ті ж дані можуть розглядатися різними користувачами зовсім відмінними способами;
- воно дозволяє спрощувати складні операції з базовими відносинами. Наприклад, якщо подання буде визначено на основі з'єднання двох відносин, то користувач зможе виконувати над ним прості унарні операції вибірки й проєкції, що будуть автоматично перетворені засобами СКБД в еквівалентні операції з виконанням з'єднання базових відносин.

Подання варто проєктувати, вибираючи такий спосіб підтримки зовнішньої моделі, що був би найбільш зручним користувачеві.

4.7 Збережені процедури

Збережена процедура – об’єкт бази даних, який уявляє собою набір SQL-інструкцій, який компілюється один раз і зберігається на сервері. Збережені процедури дуже схожі на звичайні процедури мов високого рівня. В них можуть бути вхідні та вихідні параметри й локальні змінні, у них можуть проводитися числові обчислення та операції над символьними даними, результати яких можуть привласнюватися змінним і параметрам.

У збережених процедурах можуть виконуватися стандартні операції з базами даних (як DDL, так і DML). Крім того, у збережених процедурах можливі цикли й розгалуження, тобто в них можуть використовуватися інструкції керування процесом виконання.

Збережені процедури схожі на функції, обумовлені користувачем (UDF). Основна відмінність полягає в тому, що користувацькі функції можна використовувати як і будь-який інший вираз в SQL-запиті, в той час як збережені процедури повинні бути викликані за допомогою функції CALL:

CALL процедура(...)

або

EXECUTE процедура(...)

Збережені процедури дозволяють підвищити продуктивність, розширюють можливості програмування й підтримують функції безпеки даних.

Тригер (англ. *trigger*) – це збережена процедура особливого типу, яку користувач не викликає безпосередньо, та виконання якої обумовлено настанням певної події (дією) — по суті додаванням INSERT або видаленням DELETE рядка в заданій таблиці, або модифікації UPDATE даних у певному стовпці заданої таблиці реляційної бази даних.

Тригери застосовуються для забезпечення цілісності даних і реалізації складної бізнес-логіки. Тригер запускається сервером автоматично при спробі зміни даних у таблиці, з якою він зв’язаний.

Усі виконані їм модифікації даних розглядаються як такі, що виконуються в транзакції, в якій виконана дія, що викликала спрацьовування тригера. Відповідно, у випадку виявлення помилки або порушення цілісності даних може відбутися

скасування цієї транзакції. Питання використання транзакцій буде детально розглянуто у 6-му розділі даного посібника.

Момент запуску тригера визначається за допомогою ключових слів **BEFORE** (тригер запускається до виконання пов'язаного з ним події; наприклад, до додавання запису) або **AFTER** (після події). У випадку, якщо тригер викликається до події, він може внести зміни до запису, що модифікується подією (звичайно, за умови, що подія – не видалення запису). Деякі СКБД накладають обмеження на оператори, які можуть бути використані в тригері (наприклад, може бути заборонено вносити зміни в таблицю, на якій «висить» тригер тощо.)

Крім того, тригери можуть бути прив'язані не до таблиці, а до подання (**VIEW**). У цьому випадку з їхньою допомогою реалізується механізм «подання, що оновлюється». У цьому випадку ключові слова **BEFORE** і **AFTER** впливають лише на послідовність виклику тригерів, тому що власне подія (видалення, вставка або оновлення) не відбувається.

У деяких серверах тригери можуть викликатися не для кожного запису, що модифікується, а один раз на зміну таблиці. Такі тригери називаються табличними.

4.8 Проектування інтерфейсу користувача

База даних є невід'ємною складовою частиною будь-якої автоматизованої інформаційної системи. Користувачами таких систем зазвичай є люди, далекі від розуміння тонкощів обчислювальної техніки. Тому виникає задача розробки зручного інтерфейсу користувача для роботи з системою.

Під інтерфейсом користувача розуміється комплекс засобів, які надаються прикладною програмою для керування нею, перегляду результатів її роботи, а також організації діалогу з користувачем.

Для баз даних інтерфейс складається з форм вводу й редагування даних, а також звітів, які після перегляду на екрані можуть бути надруковані.

Перш ніж приступати до реалізації форми (або звіту), важливо ретельно спроектувати її (або його) макет. Можна дати деякі корисні рекомендації із створення макетів будь-яких форм і звітів. Зокрема, подібний макет повинний включати [15]:

- змістовну назву;
- ясні та зрозумілі інструкції;
- логічно обгрунтовані угруповання та послідовності полів;
- візуально привабливий вигляд вікна форми або поля звіту;
- погоджену термінологію та скорочення;
- погоджене використання кольорів;
- зручні засоби переміщення курсору;
- засоби виправлення окремих помилкових символів і цілих полів;
- засоби виводу повідомлень про помилки при введенні неприпустимих значень;
- особливе виділення необов’язкових для введення полів;
- засоби виводу пояснювальних повідомлень із описом полів;
- засоби виводу повідомлення про закінчення заповнення форми.

Розглянемо детальніше ці рекомендації.

Змістова назва

Інформація в назві повинна ясно та недвозначно ідентифікувати призначення звіту або форми.

Ясні та зрозумілі інструкції

В інструкціях повинна застосовуватися звична для користувачів термінологія. Інструкції повинні бути короткими, а на випадок необхідності надання додаткової інформації варто передбачити спеціальні довідкові екрани. Інструкції варто записувати в стандартному форматі, дотримуючи єдиного граматичного стилю.

Логічно обгрунтовані угруповання та послідовності полів

Логічно зв’язані поля в звіті або формі варто розташовувати разом, причому їхня послідовність повинна бути логічно обгрунтованою та погодженою.

Погоджена термінологія та скорочення

Скрізь повинні використовуватися тільки знайомі та зрозумілі терміни або скорочення, обрані зі заздалегідь погодженого списку.

Візуально привабливий вигляд вікна форми або полів звіту

Форма чи звіт повинні мати привабливий зовнішній вигляд і являти собою гармонічне сполучення полів або груп полів, рівномірно розподілених на поверхні форми/звіту. При цьому у формі/звіті не повинно бути ділянок з дуже малою або занадто великою концентрацією полів. Крім того, поля потрібно

розміщати через регулярні інтервали та вирівнювати їх по вертикалі та горизонталі. Якщо екранна форма має якесь еквівалентне представлення на папері, то їхній зовнішній вигляд повинний бути узгоджений.

Погоджене використання кольорів

Для поліпшення зовнішнього вигляду форми чи звіту можна використовувати кольорове оформлення. Крім того, виділення кольором може застосовуватися для найважливіших полів або повідомлень. Для досягнення оптимального результату колір варто використовувати узгоджено та продумано. Наприклад, у формах поля із білим тлом можуть позначати поля введення, а поля із синім тлом – поля з даними, призначеними тільки для відображення на екрані.

Зручні засоби переміщення курсору

Користувач повинний легко визначати, які операції доступні йому для переміщення курсору у формі чи звіті. Зазвичай для подібних цілей використовуються клавіші табуляції, клавіші із стрілками або покажчик миші.

Засоби виправлення окремих помилкових символів і цілих полів

Користувач повинний легко визначати, які саме операції доступні йому для виправлення помилки, допущеної при введенні даних. Для цієї мети зазвичай використовуються найпростіші механізми, подібні до натискання клавіші <Backspace> або повторному введенню поверх помилкових символів.

Реакція на помилки при введенні неприпустимих значень

При введенні в поле неправильних даних програма повинна виводити повідомлення про помилку. Це повідомлення повинне інформувати користувача про допущену помилку та вказати діапазон припустимих значень.

Особливе виділення необов'язкових для введення полів

Необов'язкові для введення поля повинні бути явно відзначені за допомогою відповідного напису або виділення особливим кольором. Подібні поля варто розташовувати після полів обов'язкового введення.

Засоби виводу пояснювальних повідомлень з описом полів

Коли користувач розміщує курсор миші в чергове поле, то в деякому стандартному місці (наприклад, у рядку стану даного вікна) варто вивести інформацію про це поле.

Засоби виводу повідомлень про закінчення заповнення форми

Користувач повинний ясно уявляти собі, коли процес заповнення форми буде закінчений. Однак завершення цього процесу не повинне бути автоматичним – доцільно виводити попереджуваче повідомлення, щоб при необхідності користувач зміг ще раз переглянути введені ним дані.



Питання для самоперевірки

1. З яких компонент складається модель даних на логічному рівні?
2. Що називається відношенням?
3. На які типи поділяються відношення реляційної БД?
4. Що таке посиальна цілісність?
5. З яких операцій складається реляційна алгебра?
6. Які операції реляційної алгебри висувають вимогу сумісності за об'єднанням?
7. Чим відрізняються операції об'єднання та з'єднання?
8. З якою метою використовується індексація?
9. Що таке подання, яке в нього призначення?
10. Яким чином зв'язуються дві або більше таблиці?
11. Які аномалії притаманні ненормалізованому відношенню?
12. Які існують типи функціональних залежностей?
13. Що таке нормалізація баз даних?
14. Які нормальні форми існують?
15. Чим відрізняються ЗНФ від НФБК?
16. Яким вимогам повинен відповідати «дружній інтерфейс»?

5 ВИКОРИСТАННЯ МОВИ SQL

5.1 Загальні питання

У якості серверу баз даних використовуються реляційні СКБД, для яких стандартним засобом доступу до даного є високорівнева мова структурованих запитів **SQL (Structured Query Language)** [4,5,9,11, 14,18].

У 1992 році Міжнародним комітетом із стандартизації (*ISO – International Standard Organization*) був затверджений стандарт SQL, який часто називають **SQL/92** або просто **SQL2**, вимоги якого враховуються більшістю виробників СКБД.

В 1999 році з'явився новий стандарт – SQL3, який якісно відрізняється від попередніх стандартів своєю об'єктною орієнтацією, а також деякими змінами у використанні транзакцій (поняття транзакції буде докладно розглядатися в главі 6).

SQL може використовуватися для маніпуляції з даними (вибірка і модифікація), опрацювання структури бази даних (створення та вилучення таблиць та індексів) і адміністрування бази даних (визначення списку користувачів, призначення прав доступу).

Використання SQL має багато переваг, найбільше важливими з яких є:

- SQL підтримується багатьма постачальниками СКБД. Програми, написані з використанням мови SQL, будуть працювати майже на будь-якому сервері бази даних;
- SQL простий у вивченні, оскільки в ньому для опису дій використовуються прості англійські слова, такі, як **SELECT, INSERT, UPDATE, DELETE, COMMIT, ROLLBACK**.

Мова SQL може використовуватися двома способами. Перший передбачає *інтерактивну роботу*, що міститься у вводі користувачем окремих SQL-операторів і негайного одержання результату.

Другий міститься у *впровадженні* SQL-операторів у програми на інших мовах програмування (наприклад, *Ci*). Таке впровадження може зробити програму більш потужною й ефективною. Проте несумісність цих мов програмування зі структурою SQL і властивим йому стилем керування даними потребує внесення ряду розширень в інтерактивний SQL.

Стандарт SQL визначений **ANSI (American National Standards Institute)**. Проте, з огляду на визначену обмеженість стандарту,

творці *SQL-продуктів* намагаються розробляти їх таким чином, щоб вони відповідали стандарту ANSI, але не були б занадто жорстко обмежені його вимогами.

Функції, котрі додаються до стандарту мови розробниками комерційних реалізацій, прийнято називати *розширеннями*. Кожна з реалізацій мови називається *діалектом*.

Синтаксис SQL

При описі синтаксису SQL використовуються наступні угоди. SQL являється мовою з *незумовленими формами (free-form language)*, тобто він не обмежує кількість слів в одному рядку та місця розривів рядків. Команди SQL у більшості версій мови можна вводити в будь-якому регістрі, проте варто пам'ятати, що в загальному випадку при завданні імен таблиць або стовпчиків реєстр враховується, тобто ідентифікатори **TOVAR** і **tovar** не те саме.

Фігурні дужки ({}), навколо слів або фраз означають, що потрібно вибрати принаймні одну з укладених у них опцій. Якщо опції розділені вертикальною рисою (|), можна вибрати тільки одну з них. Якщо ж опції розділені комою – одну або декілька.

Квадратні дужки ([]) означають, що укладені в них опції необов'язкові. Якщо, крім того, опції розділені вертикальною рисою, то можна або взагалі їх не використовувати, або використовувати тільки одну з них. Якщо ж опції розділені комою, то можна не використовувати жодну з них, використовувати одну або декілька.

Фігурні та квадратні дужки на відміну від круглих не являють частиною оператора, тому їх не потрібно набирати при введенні команди SQL.

Оператори SQL зазвичай потребують *термінаторів (terminator)*. **Термінатор** – це символ, слово або команда меню, що використовуються для позначення кінця оператора. Найбільше поширеними є крапка з комою (;), \ – «сleih обернений») або слово *go*.

Зауваження

У SQL замість реляційних термінів *відношення, кортеж і атрибут* використовуються терміни *таблиця, рядок і стовпчик*

5.2 Типи даних

В SQL/92 визначаються різні типи даних, які позначаються ключовими словами: CHARACTER, CHARACTERVARYING, BIT, BITVARYING, NUMERIC, DECIMAL, INTEGER, SMALLINT, FLOAT, REAL, DOUBLEPRECISION, DATE, TIME, TIMESTAMP і INTERVAL.

Типи даних CHARACTER і CHARACTERVARYING спільно називаються типами даних символьних рядків; типи даних BIT і BITVARYING – типами даних бітових рядків. Типи даних символьних рядків і типи даних бітових рядків спільно називаються рядковими типами даних, а значення рядкових типів називаються рядками.

Типи даних NUMERIC, DECIMAL, INTEGER і SMALLINT спільно називаються типами даних точних чисел. Типи даних FLOAT, REAL і DOUBLEPRECISION спільно називаються типами даних приблизних чисел. Типи даних точних чисел і типи даних приблизних чисел спільно називаються числовими типами. Значення числових типів називаються числами.

Типи даних DATE, TIME і TIMESTAMP спільно називаються типами дати-часу. Значення типів дати-часу називаються « дата-час». Тип даних INTERVAL називається інтервальним типом.

5.2.1 Символьні рядки

Тип даних символьних рядків може містити рядка фіксованої (CHARACTER) або змінної (CHARACTERVARYING) довжини. Для конкретного типу CHARACTER (CHAR) вказується довжина рядків цього типу; у випадку CHARACTERVARYING (VARCHAR) – максимальна довжина.

5.2.2 Бітові рядки

Бітовий рядок – це послідовність біт, кожний з яких має значення 0 або 1. Довжиною бітового рядка називається число бітів у цьому рядку (0 або позитивне ціле число). При визначенні конкретного типу бітових рядків вказується довжина (у випадку BIT) або максимальна довжина рядка (у випадку BITVARYING).

5.2.3 Числові типи

Представниками типів даних точних чисел є NUMERIC, DECIMAL (або DEC), INTEGER (або INT) і SMALLINT. Специфікатор типу NUMERIC має вигляд NUMERIC [(precision [, scale])]. Специфікуються точні числа точністю precision і масштабом scale. Якщо опущений масштаб, то він покладається рівним 0, а якщо опущена точність, то її значення за замовчуванням визначається в реалізації.

Специфікатор типу DECIMAL (або DEC) має вигляд DECIMAL [(precision [, scale])]. Специфікуються точні числа, представлені з масштабом scale і точністю, рівної або більшої значення precision.

INTEGER специфікує тип даних точних чисел з масштабом 0 і обумовленою в реалізації точністю. SMALLINT специфікує тип даних точних чисел з масштабом 0 і обумовленою в реалізації точністю, не більшою, ніж точність чисел типу INTEGER.

Літеральні значення точних чисел у загальному випадку представляються у формі

[+ -] <ціле-без-знаку> [.<ціле-без-знаку>].

Клас типів даних приблизних чисел складають типи FLOAT, REAL і DOUBLE PRECISION. Специфікатор типу FLOAT має вигляд FLOAT [(precision)]. Специфікуються приблизні числа із двійковою точністю, рівною або більшою значення precision.

REAL специфікує тип даних приблизних чисел з точністю, визначеною в реалізації. DOUBLE PRECISION специфікує тип даних приблизних чисел з точністю, визначеною в реалізації, більшою, ніж точність типу REAL.

Літеральні значення приблизних чисел у загальному випадку представляються у вигляді

<значення-точного-числа>E<ціле-зі-знаком>.

5.2.4 Типи дата-час і інтервальні типи

Ці типи даних називаються також темпоральними типами, тобто типами, що зв'язані з часом).

Елемент типу дата-час складається із суміжного піднабору полів:

YEAR	Рік від Різдва Христового
MONTH	Місяць у межах року
DAY	День у межах місяця
HOURL	Година в межах дня
MINUTE	Хвилина в межах години
SECOND	Секунда й, можливо, частки секунди в межах хвилини

Елемент типу INTERVAL складається із суміжного піднабору полів:

YEAR	Рік
MONTH	Місяць

або

DAY	Дні
HOURL	Годинник
MINUTE	Хвилини
SECOND	Секунди й, можливо, частки секунд

Реальний суміжний піднабір полів, що становлять елемент типу інтервал, визначається кваліфікатором дата-часу, що називається точністю елементу.

Усередині елементу типу інтервал перше поле не обмежується. Наступні поля обмежуються в такий спосіб:

MONTH	Місяці в межах року (0–11)
DAY	Не обмежується
HOURL	Годинник у межах дня (0–23)
MINUTE	Хвилини в межах години (0–59)
SECOND	Секунди в межах хвилини (0–59.999...)

Окремі версії мови SQL мають певні відмінності. У табл. 5.2 наведені типи даних ANSI SQL, еквівалентні ним типи даних мови SQL ядра бази даних **Microsoft Jet** і припустимі синоніми. Крім того, представлені еквівалентні типи даних **Microsoft® SQL Server™**.

5.3 Класифікація команд SQL

Команди SQL використовуються для виконання різноманітних дій над реляційними БД. Для зручності роботи вони розділяються на наступні групи [6,12,14] (табл.5.1):

- команди визначення даних (**Data Definition Commands**);
- команди маніпуляції даними (**Data Manipulation Commands**);
- команди вибірки даних (**Data Query Commands**);
- команди керування транзакціями (**Transaction Control Commands**);
- команди керування даними (**Data Control Commands**).

Таблиця 5.1 – Команди мови SQL

Команда	Призначення
1	2
Команди визначення даних	
<i>alter table</i>	Змінює структуру таблиці
<i>create index</i>	Створює індекс
<i>create table</i>	Створює таблицю
<i>create view</i>	Створює подання
<i>Drop</i>	Вилучає таблицю, індекс, подання
Команди маніпуляції даними	
<i>Delete</i>	Вилучає запису таблиці
<i>Insert</i>	Додаває запису в таблицю
<i>Update</i>	Змінює дані таблиці
Команда вибірки даних	
<i>Select</i>	Вибирає дані
Команди керування транзакціями	
<i>commit</i>	Робить зміни, проведені з початку транзакції, постійними
<i>rollback</i>	Відкочує всі проведені зміни до крапки зберігання або до початку транзакції
<i>savepoint</i>	Встановлює контрольну крапку, до якого згодом можна буде виконати відкат
Команди керування даними	
<i>check database</i>	Перевіряє цілісність бази даних
<i>grant</i>	Надає привілеї
<i>revoke</i>	Скасовує надані раніше привілеї

5.4 Створення баз даних і таблиць

5.4.1 Створення баз даних

До ANSI-стандарт SQL-92 не входить оператор **CREATE DATABASE**. Замість цього він передбачає команду **CREATE SCHEMA** для визначення частини бази даних, якою буде володіти конкретний користувач. Проте в комерційні версії SQL зазвичай включена команда **CREATE DATABASE**, спрощена форма котрої (тобто без обов'язкових пропозицій і розширень) має вигляд **CREATE DATABASE ім'я_бази_даних**.

Створення бази даних не означає автоматичну можливість її використання. У залежності від версії SQL для обертання до конкретної бази даних необхідно ввести

USE ім'я_бази_даних

або

DATABASE ім'я_бази_даних

5.4.2 Створення таблиць

За допомогою команд визначення даних **SQL** можна створювати, вилучати та маніпулювати таблицями (добавляти, вилучати, переставляти стовпчики та змінювати їхні параметри).

Щоб створити таблицю треба зробити, щонайменше, наступне (наведені приклади зроблено за допомогою СКБД Access).

- задати ім'я таблиці;
- задати імена її стовпчиків;
- визначити тип даних для кожного стовпчика;
- визначити (або використовувати по умовчання) нульовий статус для кожного стовпчика – припускається або забороняється використання в стовпчику нульових значень.

CREATE TABLE ім'я_таблиці
(ім'я_стовпчика тип_даних [NULL | NOT NULL]
[, ім'я_стовпчика тип_даних [NULL | NOT NULL]] ...);

Статус стовпчика **NOT NULL** означає обов'язкове заповнення відповідного стовпчика. Статус **NULL** – означає, що значення стовпчика можуть бути не визначені. Наприклад, відсутність

телефону у визначеного клієнта. Деякі системи по умовчання вставляють деяке значення (пробіл або звичайний нуль), якщо в стовпчик не були введені жодні дані.

Наприклад,

```
create table постачальник  
(назва varchar(25) not null,  
адреса varchar(35) not null,  
факс char(8) null);
```

Як вже згадувалося у §5.2, для даних типу **varchar** дисковий простір виділяється по мірі необхідності в рамках зазначеного розміру, що дозволяє заощаджувати дисковий простір.

5.4.3 Обмеження на множини припустимих значень

Більшість комерційних систем підтримують пропозиції (обмеження) **PRIMARY KEY**, **UNIQUE**, **DEFAULT**, **CHECK**, **REFERENCES** і **FOREIGN KEY** у команді **CREATE TABLE** у відповідності зі стандартом **SQL/92**. Ці елементи забезпечують захист цілісності даних.

- **PRIMARY KEY** помічає стовпчик (у якому не можуть бути присутніми нульові значення) у якості первинного ключа таблиці. Кожне значення, що вводиться в цей стовпчик, повинно бути унікальним. Якщо первинний ключ включає декілька стовпчиків (складовий ключ), **PRIMARY KEY** визначається на рівні таблиці.

```
create table клієнти  
(код int not null primery key,  
назва char(10) not null, місто char(10));
```

У наступному прикладі обмеження на первинний ключ є обмеженням на всю таблицю, оскільки первинний ключ є складовим.

```
create table ісnumи  
(дисципліна varchar(35) not null,  
група char(8) not null,  
дата date,  
primery key (дисципліна, група));
```

- **UNIQUE** – альтернативний ключ. Також гарантує унікальність елементів стовпчика. Це обмеження застосовується до тих полів, що були заявлені ***NOT NULL***.

Стовпчики, що повинні містити унікальні значення, але котрі не є первинними ключами, називаються ***можливими ключами (candidate keys)***.

За функціональними можливостями обмеження ***PRIMARY KEY*** й ***UNIQUE*** подібні, за винятком того, що тільки один первинний ключ (що складається з будь-якої кількості стовпчиків) може бути визначений для даної таблиці. Отже, існує розходження між первинним ключем і унікальними стовпчиками в засобах їхнього використання з зовнішніми ключами.

Унікальною можна також зробити й групу полів, вказавши ***UNIQUE*** в якості обмежень таблиці. При цьому комбінації значень зазначених стовпчиків повинні бути унікальними, хоча в самих стовпчиках значення можуть бути й неунікальними. При об'єднанні в групу важливий порядок. Наприклад, комбінації значень стовпчиків ***'a', 'b'*** і ***'b', 'a'*** відрізняються.

Наприклад, потрібно відстежити всі покупки, зроблені за день у кожного продавця.

create table покупки

(ном_продавця int not null,

дата date not null,

сума money,

unique (ном_продавця, дата);

- **DEFAULT** визначає значення, що автоматично вставляється системою, якщо користувач не введе необхідні дані. Наприклад, слово ***невизначений***, якщо значення ще не визначене.
- **CHECK** визначає, які дані можуть вводитися у визначений стовпчик, тобто дозволяє визначити область припустимих значень стовпчика. При цьому можна задати список, діапазон або шаблон припустимих значень. Наприклад, задати щоб у стовпчик вводилися значення, що складаються з двох букв і чотирьох цифр. Якщо користувач спробує порушити це обмеження, то команда виконуватися не буде. Ці обмеження іноді називають ***правилами (rules)***.

Обмеження на таблицю рекомендується розташовувати після списку всіх її елементів (хоча це й необов'язково). Столпчики можуть мати більш одного обмеження, котрі не розділяються комами:

```
CREATE TABLE ім'я_таблиці  
(ім'я_стовпчика тип [NULL|NOT NULL] [ DEFAULT  
значення_по_умовчанняю]  
[обмеження_на_стовпчик]...  
[, ім'я_стовпчика тип [NULL|NOT NULL] [ DEFAULT  
значення_по_умовчанняю]  
[обмеження_на_стовпчик]...]...  
[обмеження_на_таблицю]...);
```

Наприклад,

create table товар

(код char(6) not null primary key
check (код like '[A-Я] [A-Я] [0-9] [0-9][0-9] [0-9]'),
назва varchar(80) not null,
тип char(12)
default «невизначений» null
check (тип in («економіка», «психологія»,
«програмування», «невизначений»)),
ціна money null,
кількість int null)

Ключове слово **LIKE** означає завдання шаблону значень, а **IN** – список можливих значень.

Обмеження з перевіркою також можуть застосовуватися до всієї таблиці, якщо вони використовують більш одного стовпчика. Наприклад, у таблиці **Продажі** є стовпчики для опису кількості замовлених і відправлених товарів. У цьому випадку можна використовувати табличне обмеження, що перевіряє, щоб кількість відправлених книг не перевищувала кількості замовлених.

create table продажі

(назва_id char(6) not null primery key,
відправлено int not null,
замовлено int null,
дата_замовлення date null,
check (відправлено <= замовлено));

5.4.4 Підтримка посилальної цілісності

Таблиці бази даних можуть бути певним чином зв'язані між собою. Такі зв'язки між таблицями встановлюються шляхом завдання посилань (*references*) між окремими полями таблиць.

Коли поле однієї таблиці посилається на поле в іншій таблиці, воно називається *зовнішнім ключем (foreign key)*; поле, на яке воно посилається, називається його *батьківським ключем*. Імена зовнішнього та батьківського ключів можуть бути неоднаковими. Наприклад, таблиці *Покупці* та *Продавці* зв'язуються між собою полем *ном_прод*.

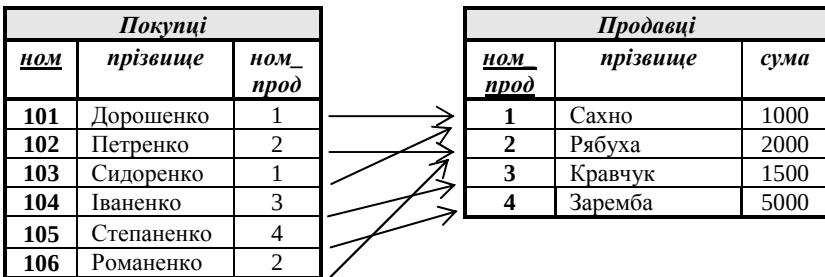


Рисунок 5.1 – Приклад використання зовнішнього ключа

Як видно з прикладу, кожне значення зовнішнього ключа повинно бути подане в батьківському ключі один і тільки один раз. У той же час на одне значення батьківського ключа можуть посилатися декілька полів зовнішнього ключа.

create table покупці

(ном integer not null primary key,

прізвище char(15),

ном_прод integer,

foreign key (ном_прод) references продавці(ном_прод));

Обмеження **REFERENCES** і **FOREIGN KEY** зв'язують разом первинні та зовнішні ключі. Якщо вводиться значення в стовпчик зовнішнього ключа, визначеного за допомогою речення **REFERENCES**, цей стовпчик і стовпчик, на який він посилається, повинні існувати в таблицях, інакше команда модифікації даних буде відхилена.

Зовнішній ключ, як і первинний, може бути визначений на будь-якій кількості полів, які разом розглядаються як єдине ціле (складовий ключ). Зовнішній і батьківський ключі повинні бути визначені на однаковій множині полів (за кількістю, типам і порядку слідування):

FOREIGN KEY (список стовпчиків) REFERENCES ім'я таблиці (список стовпчиків)

Версія обмеження ***FOREIGN KEY***, як обмеження на стовпчик, називається обмеженням ***REFERENCES***, оскільки реально не містить слів ***FOREIGN KEY***, а містить слово ***REFERENCES***, після якого впливають імена для батьківського ключа. Наприклад:

```
create table покупці  
(ном integer not null primary key,  
прізвище char(15),  
ном_прод integer references продавці(ном_прод));
```

Якщо список стовпчиків батьківського ключа має обмеження ***PRIMARY KEY***, то при завданні посилання на батьківський ключ список його стовпчиків можна опустити. Наприклад,

```
create table покупці  
(ном integer not null primary key,  
прізвище char(15),  
ном_прод integer references продавці);
```

Таким чином, основна ідея ***посилальної цілісності*** полягає в тому, що всі значення зовнішніх ключів відсилають до визначеного рядка батьківського ключа. Як тільки значення розміщається в зовнішньому ключі, батьківський ключ перевіряється для визначення того, що таке значення в ньому присутнє; у протилежному випадку команда відхиляється. Батьківський ключ повинний мати обмеження ***PRIMARY KEY*** або ***UNIQUE*** для гарантії того, що значення не подане більш ніж один раз.

5.4.5 Створення індексів

У системах реляційних баз даних механізм індексації є засобом підвищення продуктивності, прискорюючи вибірку інформації з БД.



Індекс (index) – це упорядкований (в алфавітному або чисельному порядку) список вмісту стовпчиків або групи стовпчиків у таблиці.

При пошуку потрібних даних індекси служать у якості логічних покажчиків на їхнє фізичне положення.

Після створення індекси працюють і використовуються незалежно від користувачів бази даних. У будь-який момент дані в таблиці можуть змінитися. При цьому відповідним чином повинний змінитися один або декілька індексів. У цьому випадку турботи про підтримку індексів бере на себе система.

Синтаксис команди зазвичай має вид:

```
CREATE [UNIQUE] INDEX ім'я_індексу  
ON ім'я_таблиці (ім'я_стовпчика [,ім'я_стовпчика]..);
```

Наприклад,

```
create index назва_ind  
on Товари (назва);
```

Складові індекси використовуються, коли одночасно краще здійснювати пошук по двох або декількох стовпчиках, що пояснюється їхнім логічним взаємозв'язком.

Наприклад,

```
create index товар_ind  
on Товари (група, назва);
```

У більшості діалектів **SQL** порядок стовпчиків в операторі створення індексу не обов'язково повинний повторювати їхній порядок в операторі **CREATE TABLE**. Проте, із погляду продуктивності, краще першим указувати стовпчик, по котрому частіше виконується пошук.

Унікальний індекс зазвичай створюється для ключових стовпчиків для підсилення їхньої можливості як унікальних ідентифікаторів рядків. Створювати унікальні індекси має сенс тільки тоді, коли це диктується самими даними. У загальному випадку має сенс індексувати тільки стовпчики, що часто використовуються в запитах на вибірку, у першу чергу ключові стовпчики та стовпчики, що використовуються для об'єднання та сортування.

5.5 Проектування запитів

5.5.1 Загальний формат команди **SELECT**

Для створення запитів використовується конструкція **SELECT**. Результатом виконання запиту є таблиця, що зберігається в тимчасовому буфері серверу бази даних. Обрані дані можна використовувати для перегляду, друку звітів, формування графіків або, наприклад, додати їх до постійних (базових) таблиць бази даних.

Найбільш проста конструкція **SELECT** наступна:

SELECT список полів, що вибираються FROM список таблиць;

Результатом запиту є інша таблиця, що утворюється з заданих у запиті таблиць. Наприклад:

***select код, назва, ціна
from товари;***

Якщо необхідно вивести всі стовпчики таблиці, то використовується символ «*»:

***Select *
from товари;***

Для виконання більш складних вибірок використовується загальна форма конструкції **select**:

SELECT [ALL DISTINCT] список полів, що вибираються FROM список таблиць [WHERE умова вибірки] [GROUP BY умова угруповання [HAVING умова вибірки групи]] [ORDER BY умова впорядкування];

DISTINCT – аргумент, який усуває дублювання значень із результату виконання речення **SELECT**. Наприклад:

***select distinct прізвище
from продавці;***

Якщо оператор **SELECT** витягає декілька полів, то **DISTINCT** виключає рядки, в яких всі вибрані поля ідентичні, тобто він діє на всю строку, а не на окремі поля.

Альтернативою **DISTINCT** є ключове слово **ALL**, що має протилежне значення, тобто дозволяє виведення повторюваних рядків у запиті. По умовчанням використовується **ALL**.

При роботі з декількома таблицями, що містять однойменні поля, перед іменами стовпчиків треба задавати імена відповідних таблиць. У більшості діалектів **SQL** для спрощення набору таблиць дозволяється задавати **псевдоніми (aliases)**. Псевдонім указується після імені таблиці в списку таблиць:

```
select T.код, T.назва, T.ціна  
from Товари [as] T;
```

Буква **T** перед іменами стовпчиків заміняє повне ім'я таблиці (**товари**). Цей запит еквівалентний наступному:

```
select Товари.код, Товари.назва, Товари.ціна  
from Товари;
```

5.5.2 Вибір рядків: речення **WHERE**

Речення **WHERE** задає предикат, умову, що може бути вірною або помилковою для кожного рядку таблиці. В результаті вибираються тільки ті рядки з таблиці, для яких предикат має значення «істина».

У **SQL** є ряд операторів і ключових слів для завдання умов.

Таблиця 5.4 – Оператори завдання умов

Оператор, ключове слово	Приклад використання
оператори порівняння (=, <, >, >=, <=, <>, !=);	where ціна > 1000;
комбінації умовних і логічних операцій – (AND,OR,NOT)	where ціна < 5000 and ціна >2000
діапазони (BETWEEN і NOT BETWEEN)	where ціна between 450 and 500
списки (IN, NOT IN)	where товар in («монітор», «принтер»)
невідомі значення (IS NULL і IS NOT NULL)	where телефон is null
відповідності символів (LIKE і NOT LIKE)	where телефон not like «415%»; where назва like «*БД*»

Оператори порівняння

Ці оператори використовуються в такий спосіб:

WHERE вираз оператор_порівняння вираз

У якості виразів можуть використовуватися константи, імена стовпчиків, функції, підзапити або їхні комбінації, що пов'язані арифметичними операторами.

Зазвичай оператори порівняння застосовуються до чисельних значень. У **SQL** вони також можуть застосовуватися до даного з типами **char** і **varchar** (< означає *раніш за абеткою*, > означає *пізніше*) і до дат (< означає *раніш у хронологічному порядку*, > означає *пізніше*). При цьому символічні значення та дати містяться у лапки.

```
select код, назва, місто  
from Постачальники  
where місто = «Київ»;  
select прізвище, посада  
from Посадовці  
where прізвище > «Іваненко»;  
select *  
from Продавці  
where рейтинг > 100;  
select *  
from titles  
where advance*2 > price + sales;
```

Комбінації умовних і логічних операцій

Логічні оператори виконуються в суворо визначеній послідовності (рис. 5.2). Якщо у виразі присутні обидва типи операторів, першими здійснюються арифметичні оператори.

Роздивимося пошук моніторів в таблиці **Товари**, незалежно від ціни, а також принтерів із ціною більше 700 грн.

```
Select модель, тип, ціна
```

```
From Товари
```

```
Where тип=«монітор» or тип=«принтер» and ціна >700;
```

Обмеження на ціну застосовується тільки до принтерів, тому що оператор **AND** виконується перед оператором **OR**.

Щоб першим виконувався оператор **OR**, у запиті треба використовувати дужки. Наприклад, якщо потрібно вибрати всі принтери та сканери з ціною вище 1000 грн:

```
Select модель, тип, ціна
```

```
From Товари
```

```
Where (тип=«монітор» or тип=«принтер») and ціна >1000;
```

Роздивимося запит, що здійснює пошук книг, витрати на який не окупилися, тобто прибуток від продажу котрих менше подвоєної суми, що сплачена авторам. Крім того, введемо контрольну дату.

select назва, тип, ціна, гонорар, продано, дата
from Книги
where ціна*продано < 2* гонорар and дата < '15/09/04';



Рисунок 5.2 – Послідовність виконання операцій

Діапазони (BETWEEN i NOT BETWEEN)

Діапазон можна визначити двома способами:

- за допомогою операторів порівняння <, >;
- за допомогою ключового слова (оператора) **BETWEEN**.

Ключове слово **BETWEEN** можна використовувати для створення включаючого діапазону, якщо в результатуючі значення повинні включатися й межі діапазону. Наприклад:

select модель, кількість
from Товари
where кількість between 100 and 150;

При використанні конструкції **not between** знаходяться рядки, значення яких лежать поза вказаним діапазоном. Наприклад:

select модель, кількість
from Товари
where кількість not between 100 and 150;

Аналогічний результат дає використання операторів порівняння:

select модель, кількість
from Товари
where кількість < 100 and кількість > 150;

Треба бути уважними при завданні діапазону символічних значень. Наприклад, необхідно провести вибірку покупців, прізвища яких потрапляють у заданий алфавітний діапазон:

*select **
from Автори
where прізвище between 'А' and 'Л'

При цьому прізвища авторів, які починаються на **Л**, виявляться пропущеними, хоча оператор **BETWEEN** є включаючим. У даному випадку порівнюються рядки різної довжини: рядок 'Л' і, наприклад, рядок 'Лупенко'. Тому **BETWEEN** доповнює рядок 'Л' пробілами. У більшості кодів знак пробілу передує символам кирилиці. Тому Лупенко не був вибраним. Для правильної вибірки потрібно вказати наступну букву алфавіту або приписати літеру 'я' (декілька літер 'я', якщо це необхідно).

Списки (IN, NOT IN)

Ключове слово **IN** дозволяє вибрати значення, що збігаються зі значеннями з заданого списку. Наприклад:

select name, royalty
from authors
where name in ('Довлатов', 'Бродський', 'Шмельов');

що еквівалентно запиту:

select name, royalty
from authors
where name='Довлатов' or name='Бродський'
or name='Шмельов';

Вибірка нульових (невідомих) значень

Значення **NULL** використовується для подання невідомої інформації. При цьому **NULL**-значення не є звичайним нулем або порожньою позицією. Його не можна порівнювати з будь-яким іншим значенням. Проте, якщо за логікою запиту необхідно вибрати саме таку інформацію, то потрібно використовувати наступну конструкцію:

```
select код, картина, художник  
from Живопис  
where художник is null;  
Пошук по підрядках (LIKE і NOT LIKE)
```

При необхідності вибору інформації за неповними даними використовується наступна конструкція:

<i>WHERE ім'я_стовпчика [NOT] LIKE 'зразок' [ESCAPE ключовий символ]</i>

Ключове слово **LIKE** може бути застосовано до символічних полів і в деяких системах до полів дати, проте для чисельних полів воно не може бути застосовано.

Зразок (в лапках) може містити один або декілька шаблонів – символів, які заміщають у зразку пропущені букви або рядки. Ключове слово **ESCAPE** використовується, якщо в зразку міститься один із шаблонів, але його потрібно розглядати як просту літеру.

У **ANSI SQL** використовуються два символи шаблону разом із ключовим словом **LIKE**: знак відсотка (%) і підкреслення (_).

% – будь-який рядок з будь-якою кількістю символів;

_ – будь-який одиночний символ.

Наприклад:

```
select автор, назва  
from Книги  
where назва like «%Fox Pro%»;  
select *  
from Покупці  
where прізвище like 'A%';  
select *  
from Автори  
where прізвище like 'Mc%' or прізвище like 'Mac%;
```

*select **

from Терміни

where термін like «»&[Введіть назву терміна]& «*»;*

Щоб знайти в рядку символ підкреслення або відсотка, у предикаті **LIKE** будь-який символ можна визначити як **ESCAPE-символ** (керуючий символ). Він використовується в предикаті безпосередньо перед символом шаблону й означає, що наступний за ним символ інтерпретується саме як звичайний символ, а не символ шаблону.

select notes

from table

where notes like «%/%% escape /»;

У результаті будуть обрані всі значення, що містять символ «%».

Приклади:

like «27%» – рядок, що починається з 27;

like «27@%» – 27%;

like «_n» – an, in, on тощо;

like «@_n» – _n.

5.5.3 Агрегатні функції

Агрегатні функції (табл. 5.5) використовуються для одержання узагальнюючих значень. Вони дають єдине значення для цілої групи рядків таблиці. Агрегатні функції завжди мають аргументи, що є виразами та містяться у дужках. У якості аргументів зазвичай використовуються назви стовпчиків, але також припускаються константи, функції та будь-які їхні комбінації з арифметичними операторами.

Таблиця 5.5 – Агрегатні функції

Функція	Результат
COUNT	Визначає кількість рядків або значень поля, обраних за допомогою запиту, що <i>не є NULL-значеннями</i>
SUM	Обчислює арифметичну суму всіх обраних значень даного поля
AVG	Обчислює середнє значення для всіх обраних значень даного поля
MAX	Обчислює найбільше зі всіх обраних значень даного поля
MIN	Обчислює найменше зі всіх обраних значень даного поля

Для **SUM** і **AVG** можуть використовуватися тільки цифрові поля. Для **COUNT**, **MAX** і **MIN** – цифрові та символічні поля. Стосовно до символічних полів **MAX** і **MIN** визначають значення відповідно коду **ASCII** згідно алфавіту.

Наприклад, середня зарплата у випадку індексації

```
select avg(оклад*індекс)  
from відомість;
```

Функції **COUNT(DISTINCT)** і **COUNT(*)** відрізняються тим, що перша рахує кількість непорожніх і неповторюваних рядків, а друга – загальну кількість рядків. Багато реалізацій підтримують аргумент **ALL** (не підтримуваний **ANSI**), що при використанні у функції **COUNT** дозволяє підраховувати і повторювані, але не порожні значення. Наприклад:

```
select count (*)  
from Покупці;  
select count (all рейтинг)  
from Покупці;  
select count (distinct назва)  
from Поставки;
```

В агрегатних функціях для обмеження кількості рядків, що беруть участь в обчисленнях, можна використовувати конструкцію **WHERE**. Наприклад:

```
select назва, sum(сума)  
from Продажі  
where назва = «Монітор»
```

5.5.4 Групування даних

Речення **GROUP BY** розділяє таблицю на набори, а агрегатна функція обчислює для кожного з них підсумкове значення. Ці значення називаються **агрегатним вектором**.

<pre>SELECT список вибору FROM список таблиць [WHERE умови] [GROUP BY список_угруповання];</pre>
--

Наприклад:

```
select художник, count(картина)  
from Музеї  
group by художник;
```

До списку вибору включаються стовпчик, за яким проводиться угруповання, та агрегатна функція.

Шляхом сортування одночасно за декількома елементами можна створювати групи всередині груп.

*select музей, художник, count(картина)
from Музеї*

group by музей, художник;

При відсутності агрегатної функції конструкція **GROUP BY** в запиті еквівалентна реченню **DISTINCT**:

select художник

from Музеї

group by художник;

select distinct художник

from Музеї;

Спільна робота речень **WHERE** і **GROUP BY** відбувається в такий спосіб. Спочатку знаходяться всі рядки, що відповідають умові **WHERE**. Потім речення **GROUP BY** поділяє відібрані рядки на групи. Рядки, що не задовольняють умовам речення **WHERE**, не включаються до жодної групи.

select музей, count(картина)

from Музеї

where painter = «Рубенс»

group by музей;

Речення HAVING

У самому загальному сенсі речення **HAVING** працює аналогічно реченню **WHERE**, але застосовується до груп. **WHERE** накладає обмеження на рядки, а **HAVING** – на групи. Як правило, речення **HAVING** використовується разом із реченням **GROUP BY**.

Якщо в списку вибору є агрегатні функції, речення **WHERE** виконується перед ними, тоді як речення **HAVING** застосовується до всього запиту в цілому, після розбивки на групи та обчислення значень функцій.

З погляду синтаксису умовного виразу речення **WHERE** і **HAVING** ідентичні. Відмінність полягає у тому, що в умові речення **WHERE** не можуть знаходитися агрегатні функції. Крім того, у більшості систем елементи речення **HAVING** повинні включатися до списку вибору. На речення **WHERE** це обмеження не поширюється. У реченні **HAVING** може міститися будь-яка кількість умов.

Пропозиція **HAVING** працює наступним чином: спочатку **GROUP BY** розділяє рядки на набори (за типом), потім на отримані групи накладаються умови речення **HAVING**

<i>select mun, count(*)</i>	<i>select автотр</i>
<i>from Товари</i>	<i>from Книги</i>
<i>group by mun</i>	<i>group by автотр</i>
<i>having count(*) > 1;</i>	<i>having автотр like «П%»;</i>

Якщо в реченні **HAVING** є декілька умов, вони об'єднуються за допомогою операторів **AND**, **OR** або **NOT**.

```
select pub_id, sum(advance), avg(price)  
from titles  
group by pub_id  
having sum(advance) > $15000  
and avg(price) < $20  
and pub_id > '0800';
```

Результати виконання речень **HAVING** і **WHERE** можуть відрізнитися. Це пов'язано з тим, що **WHERE** відсіває рядки перед угрупованням даних, а **HAVING** – після.

5.5.5 Сортування результатів запиту

Для поліпшення сприйняття результатів виконання запиту можна використовувати конструкцію **ORDER BY**, що дозволяє упорядкувати вихідні дані. Дані можуть сортуватися як по убутуванню (**DESC**), так і по зростанню (**ASC**). По умовчанню прийнята зростаюча послідовність сортування.

```
SELECT список вибору  
FROM список_таблиць  
[WHERE умови]  
[ORDER BY {вираз [ASC | DESC] | позиція [ASC | DESC]}]  
[, {вираз [ASC | DESC] | позиція [ASC | DESC]}]...];
```

При цьому задати сортування можна у такі способи:

- зазначити стовпчик або стовпчики, за якими буде виконуватися сортування;
- зазначити номер (позицію) виразу в списку вибору;
- зазначити ім'я виразу.

Наприклад,

```
select *  
from Посадовці  
order by прізвище asc;
```

При використанні більш одного стовпчика в пропозиції **ORDER BY** виконується *вкладене сортування*, тобто спочатку виконується сортування по першому полю, а потім по наступним. Кількість рівнів сортування не обмежується. Стандарт **ANSI**, а також більшість комерційних систем, потребують, щоб стовпчики, за якими проводиться сортування, були присутні в списку вибору.

Порядок указівки стовпчиків у реченні **ORDER BY** не зобов'язаний збігатися з порядком перерахування стовпчиків і виразів в операторі **SELECT**.

```
select товар, ціна  
from Поставлено  
order by ціна dsc, товар asc;
```

У реченні **ORDER BY** замість імені стовпчика можна зазначити його позицію в списку вибору.

```
select телефон, прізвище  
from Абоненти  
order by 1 asc;
```

Стовпчики, отримані за допомогою агрегатних функцій, а також вирази зі списку вибору **SELECT** можна використовувати також у реченні **ORDER BY**, якщо на них посилаються за номером. Речення **ORDER BY** завжди виконується останнім.

Наприклад, треба знайти середню вартість книг за кожним типом, витрати на які більше 5000 грн, і впорядкувати результати за ціною.

```
select min, avg(ціна)  
from Книги  
where витрати > 5000  
group by min  
order by 2 desc;
```

Якщо в списку вибору вираз визначений із заголовком, по ньому можна виконувати сортування.

```
select видавництво, ціна*уцінка as m_ціна, назва  
from Книги  
order by видавництво, m_ціна desc;
```

Деякі системи припускають завдання в явному вигляді виразу зі списку вибору в якості параметра речення **ORDER BY**, проте стандартом це не підтримується.

Порожні рядки розташовуються наприкінці або на початку списку, що не нормується стандартом **ANSI** і для кожної системи обмовляється окремо.

5.5.6 З'єднання таблиць

За допомогою операції з'єднання в одному операторі **SELECT** можна маніпулювати даними з декількох таблиць.

Кожне з'єднання задається для двох таблиць, хоча в одному операторі **SELECT** може виконуватися декілька з'єднань. При цьому в якості стовпчиків, за якими проводиться з'єднання, використовуються первинний (або потенційний) та зовнішній ключі (§ 4.1.3).

Існують два варіанти вказівки стовпчиків з'єднання: в реченні **FROM** з використанням операції **INNER JOIN** або в реченні **WHERE**:

```
SELECT список_вибору
FROM таблиця_1 INNER JOIN таблиця_2 ON
[таблиця_1.]стовпчик з'єднання = [таблиця_2.]
стовпчик з'єднання]
```

або

```
SELECT список_вибору
FROM таблиця_1, таблиця_2 [, таблиця_3]...
WHERE [ таблиця_1.]стовпчик з'єднання =
[ таблиця_2.] стовпчик з'єднання
```

Нехай в БД є таблиці *Товари(код, тип, назва, ціна)* і *Замовлення(ном_замовлення, дата, код_товару, ціна_оддиниці, кількість, вартість)*, між яким існує зв'язок 1:N. Необхідно визначити, коли та в якій кількості замовлявся конкретний товар.

```
select Товари.назва, Товари.тип, Замовлення.кількість,
Замовлення.дата
from Товари Inner Join Замовлення
On Товари. код= Замовлення. код_товару
where Товари.назва = [ Введіть назву товару];
```

або

```
select Товари.назва, Товари.тип, Замовлення.кількість,
Замовлення.дата
```

from Товари, Замовлення

*On Товари. код= Замовлення. код_товару
where (Товари.назва = [Введіть назву товару]) AND
(Товари. код= Замовлення. код_товару);*

В результаті виконання запиту з'єднуються ті рядки обох таблиць, в яких значення стовпчиків з'єднання співпадають.

Якщо в стовпчиках, за якими проводиться з'єднання, є нульові значення, вони пропускаються.

Запит на з'єднання не вносить ніяких змін у таблиці, а просто дозволяє системі маніпулювати даними з різних таблиць так, як якби вони належали одній таблиці.

Щоб прискорити ввід запитів і зробити їх більш зрозумілими, у списку таблиць можна визначити псевдоніми таблиць (аліаси). В якості аліаса можна використовувати будь-яке ім'я, навіть одну літеру.

*select T.назва, T.тип, Z.кількість, Z.дата
from товари as T, замовлено as Z
where T.код = Z.код_товару*

З'єднання двох копій однієї таблиці

З'єднання двох копій однієї таблиці означає наступне: будь-який рядок однієї таблиці (один в кожен момент часу) можна комбінувати з його ж копією або з будь-яким іншим рядком тієї ж таблиці. Кожна така комбінація оцінюється в термінах предиката, як і у випадку з'єднання декількох різноманітних таблиць.

СКБД виконує цю операцію так, як і над двома таблицями, що мають різні назви. Тому для таблиць треба задати псевдоніми. При з'єднанні двох копій однієї таблиці виконується самоз'єднання, якщо поля, що з'єднуються, містять однакові значення.

Наприклад, із списку публікацій **Публікації (назва, прізвище)** необхідно вибрати тільки роботи, написані а) у співавторстві і б) одним автором.

*а) select distinct а.назва, а.прізвище
from Публікації А, Публікації Б
where А.назва=Б.назва and А.прізвище!= Б.прізвище
order by А. назва;
б) select distinct а.назва, а.прізвище
from Публікації А, Публікації Б
where А.назва=Б.назва and А.прізвище = Б.прізвище
order by А. назва;*

5.6 Структуровані запити та підзапити

5.6.1 Загальне поняття

Підзапит (subquery) – це додатковий метод маніпуляцій із декількома таблицями. Це – оператор **SELECT**, вкладений:

- у речення **WHERE**, **HAVING** або **SELECT** іншого оператора **SELECT**;
- в оператор **INSERT**, **UPDATE** або **DELETE**;
- в інший підзапит.

Підзапити повертають результати внутрішнього запиту в зовнішнє речення та мають дві основні форми: некорельовану й корельовану (зв'язані підзапити). Перша реалізується (концептуально) «зсередини назовні», тобто зовнішній запит виконує ту або іншу дію, базуючись на результатах виконання внутрішнього запиту. Спрощена форма синтаксису підзапиту показана на рис. 5.3.

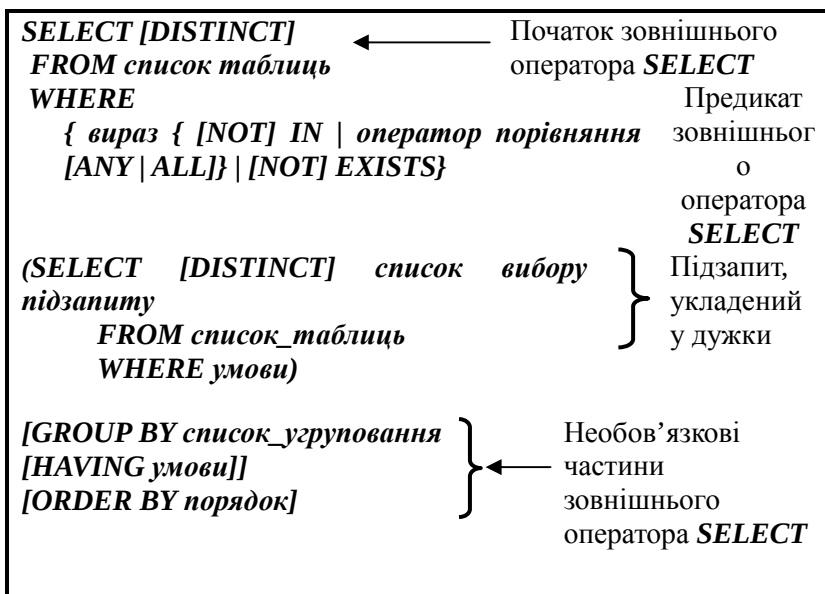


Рисунок 5.3 – Спрощена форма синтаксису підзапиту

У корельованому запиті зовнішній оператор **SQL** надає для підзапиту значення, котрі будуть використовуватися при його виконанні. Потім результати виконання підзапиту повертаються на зовнішній запит. Як корельовані, так і некорельовані підзапити бувають трьох типів, у залежності від елементів у реченні **WHERE** зовнішнього запиту:

- підзапити, що не повертають жодного або повертають декілька елементів (починаються з **IN** або з оператора порівняння, містять ключові слова **ANY** або **ALL**);
- підзапити, що повертають єдине значення (починаються з простого оператора порівняння);
- підзапити, що являють собою тест на існування (починаються з **EXISTS**).



Оператори **BETWEEN**, **LIKE**, **IS NULL** у підзапитах використовувати не можна

5.6.2 Некорельоване опрацювання

Підзапити, що не повертають значень або повертають декілька значень

Ця група включає підзапити, що починаються з **IN**, **NOT IN** або оператора порівняння з ключовими словами **ANY** або **ALL**.

Підзапити, що починаються з ключового слова **IN**, мають наступну загальну форму:

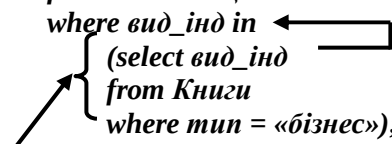
Початок операторів **SELECT**, **INSERT**, **UPDATE**, **DELETE** або підзапиту
WHERE вираз **[NOT] IN** (підзапити)
[Кінець операторів **SELECT**, **INSERT**, **UPDATE**, **DELETE** або підзапиту]

Список набору внутрішнього підзапиту, що починається з оператора порівняння або **IN**, може включати *тільки один вираз або ім'я стовпчика*. Тому варіант **SELECT*** не можна використовувати в підзапитах. Винятком є підзапити з оператором **EXISTS**. Стовпчик, ім'я якого вказується в реченні **WHERE** зовнішнього оператора, повинний бути сумісним для об'єднання зі стовпчиком, ім'я якого вказується в списку вибору підзапиту.

Результатом внутрішнього підзапиту є список, що включає від нуля до декількох значень. Після того, як підзапит поверне результати, зовнішній запит приступить до їхнього опрацювання.

Наприклад, необхідно визначити назви видавництв, що випускають книги по бізнесу, із таблиці **Книги**, що містить назви й тип випущених книг і індекс видавництва, і таблиці **Видавництва**, що містить індекси та назви видавництв:

```
select вид_назва
from Видавництва
where вид_інд in (select вид_інд
from Книги
where тип = «бізнес»);
```



Внутрішній запит виконується незалежно, передаючи результати до зовнішнього запиту

Цей запит можна також сформулювати як запит на з'єднання:

```
select distinct вид_назва
from Видавництва B, Книги K
where B.вид_інд = K.вид_інд and тип=«бізнес»;
```

Приклад 2: Знайти прізвища всіх других авторів, що мешкають у Запоріжжі й одержують менше 30% авторського гонорару за книги, співавторами яких вони є (номер автора у списку співавторів – 2).

```
select прізвище
from Автори
where місто = 'Запоріжжя' and код_автора in
(select код_автора
from Автори-Книги
where гонорар <.30 and ном_автора=2);
```

При цьому речення **WHERE** як внутрішнього, так і зовнішнього запитів, припускає вмикання декількох умов. Крім того, підзапит може включати й з'єднання, оскільки підзапити та з'єднання не є взаємовиключними елементами.

Використовуючи з'єднання, цей же запит можна уявити, як:
select Автори.прізвище

from Автори, Автори-Книги
where місто = 'Запоріжжя'
and Автори.код_автора = Автори-Книги.код_автора
and Автори-Книги.гонорар <.30
and Автори-Книги.ном_автора = 2;

Підзапити, що починаються з оператора порівняння, модифікованого ключовими словами **ANY** або **ALL**, мають таку форму:

Початок операторів **SELECT, INSERT, UPDATE, DELETE** або підзапиту
WHERE вираз оператор порівняння [**ANY** | **ALL**] (підзапити)
 [Кінець операторів **SELECT, INSERT, UPDATE, DELETE** або підзапиту]

Якщо, наприклад, скористатися оператором порівняння «>», то «> **ALL**» означає «більше, ніж кожне значення» (іншими словами, «більше, ніж найбільше значення»). Таким чином, «> **ALL (1, 2, 3)**» означає «більше, ніж 3».

«> **ANY**» означає «більше, ніж, принаймні, одне значення» (іншими словами, «більше, ніж найменше значення»). Таким чином, «> **ANY (1, 2, 3)**» означає «більше, ніж 1».

Таблиця 5.6 – Приклади результатів використання **ALL** і **ANY**

ALL	Результат	ANY	Результат
> ALL (1, 2, 3)	>3	> ANY (1, 2, 3)	>1
< ALL (1, 2, 3)	<1	< ANY (1, 2, 3)	<3
= ALL (1, 2, 3)	=1 або =2 або =3	= ANY (1, 2, 3)	=1 або =2 або =3

Наприклад, вибрати назви книг, гонорари за які перевищують найбільший гонорар, сплачений видавництвом «Освіта»:

select назва
from Книги
where гонорар > all
(select гонорар
from Видавництва, Книги
where Книги.код_вид = Видавництва.код_вид
and Видавництва.назва = 'Освіта');

Тут для кожної назви внутрішній запит знаходить список значень гонорарів, сплачених видавництвом «Освіта». Зовнішній

запит знаходить найбільше значення в списку та вибирає гонорари, що перевищують визначений.

Аналогічно працює підзапити з ключовим словом **ANY**.

Якщо підзапит, що починається з **ALL** або **ANY** та оператора порівняння, повертає в якості одного зі своїх значень **NULL**, вважається, що запит у цілому завершився невдало.

Порівняння ключових слів IN, ANY і ALL

Оператор «=**ANY**» еквівалентний оператору **IN**. Якщо, наприклад, потрібно знайти авторів, що мешкають у тому ж місті, де розташоване видавництво, можна скористатися або **IN**, або «=**ANY**»:

```
select прізвище, місто
  from Автори
 where місто in
 (select місто
  from Видавництва)
 order by місто;
```

```
select прізвище, місто
  from Автори
 where місто = any
 (select місто
  from Видавництва)
 order by місто;
```

Проте оператор «<>**ANY**» (або «!**ANY**») відрізняється від оператора **NOT IN**.

«<> **ANY**» означає «**не** =**a** або **не** =**b** або **не** =**c**». **NOT IN** означає «**не** =**a** і **не** =**b** і **не** =**c**».

Підзапити, що повертають єдине значення

Підзапит, що починається з немодифікованого оператора порівняння (оператор порівняння, не супроводжуваний ключовими словами **ANY** або **ALL**), повинний повертати єдине значення.

Початок операторів **SELECT, INSERT, UPDATE, DELETE** або підзапиту
WHERE вираз оператор порівняння (підзапити)
[Кінець операторів **SELECT, INSERT, UPDATE, DELETE** або підзапиту]

Агрегатні функції повертають лише єдине значення. *При цьому агрегатні функції використовуються без речень **GROUP BY**.*

Приклад 1 Потрібно дізнатися всі замовлення, вартість яких перевищує середню вартість замовлень за 10 квітня 2005 року:

```
select *
  from Замовлення
 where вартість >
```

*(select avg(вартість)
from Замовлення
where дата = 10/04/05);*

Приклад 2 Знайти назви всіх книг із цінами вище поточної мінімальної ціни:

*select title
from titles
where price >
(select min(price)
from titles)*

Підзапити, що виконують перевірку на існування

Коли запит починається з ключового слова **EXISTS**, він функціонує як «тест на існування». Ключове слово **EXISTS** у реченні **WHERE** виконує перевірку на існування (або неіснування) даних, що задовольняють критеріям відповідного підзапиту.

Початок операторів **SELECT, INSERT, UPDATE, DELETE** або підзапиту
WHERE [NOT] EXISTS (підзапит)
[Кінець операторів **SELECT, INSERT, UPDATE, DELETE** або підзапиту]

EXISTS виконує перевірку на наявність або відсутність «порожнього набору» рядків. Якщо підзапит повертає хоча б один рядок, цей результат оцінюється як «істина». Це означає, що результат виконання **EXISTS** буде успішним, а результат виконання **NOT EXISTS** буде невдалим. Якщо підзапит повертає порожній набір (рядків немає), цей результат оцінюється як «неправда». Це означає, що результат виконання фрази **NOT EXISTS** буде успішним, а результат виконання фрази **EXISTS** буде невдалим.

Наприклад, потрібно знайти назви усіх видавництв, що публікують книги по бізнесу:

*select distinct назва
from Видавництва
where exists
(select *
from Книги
where код_вид = Видавництва.код_вид
and тип = «бізнес»);*

Синтаксис підзапитів, що починаються з **EXISTS**, відрізняється від синтаксису інших підзапитів наступним чином:

- ключовому слову **EXISTS** не передує ім'я стовпчика, константа або який інший вираз;
- список вибору підзапиту, що починається з **EXISTS**, майже завжди містить «зірочку» (*). Немає ніякого сенсу вводити імена стовпчиків, оскільки виконується лише тест на існування рядків, що задовольняють умовам підзапиту, а вони вказуються в реченні **WHERE** цього підзапиту (а не в реченні **SELECT** цього підзапиту).

NOT EXISTS виконує дію, обернену **EXISTS**. Успішний результат виконання запитів із **NOT EXISTS** відповідає випадку, коли підзапит узагалі не повертає рядків.

Щоб знайти, наприклад, назви видавництв, що не публікують книги по бізнесу, запит повинний мати такий вид:

```
select distinct назва  
from Видавництва  
where not exists  
(select *  
from Книги  
where код_вид = Видавництва.код_вид  
and мун = «бізнес»);
```

Підзапити, що починаються з **EXISTS** і **NOT EXISTS**, можна використовувати для виконання двох операцій із теорії множин: **перетин** і **різниця**. **Перетин** двох множин містить елементи, що належать обом вихідним множинам. **Різниця** двох множин містить елементи, що належать тільки першій з двох множин.

Перетин таблиць **Автори** та **Видавництва** по стовпчику **місто** є множина міст, у яких мешкає якийсь автор і розташоване якесь видавництво:

```
select distinct місто  
from Автори  
where exists  
(select *  
from Видавництво  
where Автори.місто=Видавництво.місто)
```

Різницею таблиць **Автори** та **Видавництва** по стовпчику **місто** є множина міст, у яких мешкає якийсь автор, але немає жодного видавництва:

```
select distinct місто
from Автори
where not exists
(select *
from Видавництва
where Автори.місто=Видавництва.місто)
```

5.6.3 Корельовані запити

У корельованому (зв'язаному) запиті внутрішній запит не може бути реалізований негайно: він посилається на зовнішній запит і виконується по черзі для кожного рядка в зовнішньому запиті.

```
select вид_назва
from Видавництва as B
where «бізнес» in
    (select титл
     From Книги
     Where вид_код=B.вид_код);
```

Внутрішній запит для свого виконання повинен отримати дані з зовнішнього запиту

Цей приклад по кроках виконується наступним чином:

- 1.Зовнішній запит відшукує перше ім'я в таблиці **Видавництва**.
- 2.Внутрішній запит з'єднує таблиці **Видавництва** і **Книги** по стовпчику **вид_код** і шукає рядки, в яких стовпчик **титл** таблиці **Книги** має значення «бізнес», отримане з зовнішнього запиту.
- 3.Внутрішній запит повертає цей результат у речення **in** зовнішнього запиту, де значення стовпчика **титл** таблиці **Книги** порівнюються з рядком «бізнес».
- 4.В зовнішній таблиці вибирається наступний рядок, і повторюються пункти 2 і 3.

Приклад 2 Вибрати всіх авторів, що одержують 100% авторського гонорару

```
select distinct прізвище
from Автори
```

*where 1.0 in
 (select гонорар
 from Автори-Книги
 where код_автора = Автори. код_автора);*

Корельований запит до однієї таблиці, що починається з **IN**, можна використовувати, наприклад, для визначення, які типи книг є загальними для декількох видавництв:

*select distinct K1. type
 from книги K1
 where K1. mun in
 (select K2. mun
 from titles K2
 where K1.вид_код!= K2.вид_код);*

Корельовані IN-підзапити в реченні HAVING

Такий варіант запиту можна, наприклад, використовувати для перебування типів книг, максимальний аванс за які, щонайменше, у два рази перевищує розмір середнього авансу для цього типу книг.

*select K1.mun
 from Книги K1
 group by K1.mun
 having max(K1.аванс) in
 (select 2*avg(t2. аванс)
 from Книги K2
 where K1.mun = K2.mun);*

Корельовані запити з операторами порівняння

Наприклад, щоб знайти книги, ціна котрих вище середньої для книг цього типу

*select K1.mun, K1.mun
 from книги K1
 where K1.ціна >
 (select avg(K2.ціна)
 from Книги K2 where K1.mun=K2.mun);*

Зовнішній запит вибирає рядки з таблиці **Книги (K1)** по черзі. Тип книги передається в підзапит, і для нього обчислюється середнє значення ціни. Це значення потім передається в зовнішній запит, де речення **where** відбирає потрібні рядки.

У цьому запиті корельований підзапит імітує дію оператора **GROUP BY**. Немає необхідності виконувати групування за типом у явному вигляді, оскільки за допомогою з'єднання двох копій таблиці *Книги* у реченні **where** даного підзапиту визначаються середні ціни за кожним типом.

Визначити номери та імена всіх продавців, що мають більш одного покупця, можна наступним чином:

```
select прод_ном, прод_прізвище  
from Продавці Пр  
where 1 <  
      (select count (*)  
      from Покупці  
      where прод_ном=Пр.прод_ном);
```

Пошук помилок

Корельовані підзапити можна також використовувати для пошуку помилок. Наприклад, такий запит перевіряє таблицю *Замовлення* із метою встановити, чи відповідає співвідношення полів *прод_ном* і *покуп_ном* кожного її рядка співвідношенню цих же полів у таблиці *Покупці*, та здійснює вивід будь-якого рядку, для якого така відповідність не виявлена.

Іншими словами, перевіряє коректність зв'язків продавців із покупцями (передбачається, що *покуп_ном*, як первинний ключ таблиці *Покупці*, не повторюється).

```
select *  
from Замовлення З  
where not прод_ном=  
      (select прод_ном  
      from Покупці  
      where покуп_ном = З.покуп_ном);
```

Підзапити та з'єднання, як правило, дають однаковий результат. Проте доцільно використовувати

- підзапити — коли необхідно порівнювати значення агрегатних функцій з іншими значеннями
- з'єднання — коли інформація відображається з декількох таблиць, оскільки підзапит може відображати інформацію тільки з зовнішньої таблиці.

5.6.4 Використання EXISTS із зв'язаними підзапитами

При застосуванні зв'язаних підзапитів речення **EXISTS**, як і інші предикатні оператори, оцінюється окремо для кожного рядка таблиці, на якій є посилання в зовнішньому запиті. Це дозволяє використовувати його як правильний предикат, що генерує різноманітні відповіді для кожного рядку цієї таблиці.

Наприклад, необхідно знайти продавців, що мають декількох покупців:

```
select distinct про_ном  
from Покупці П1  
where exists  
(select*  
from Покупці П2  
where П1.прод_ном=П2.прод_ном  
and П1.покуп_ном<>П2.покуп_ном);
```

Для кожного рядку-кандидата зовнішнього запиту внутрішній запит знаходить рядки, що відповідають даному значенню **прод_ном** (той же продавець) і не значенню **покуп_ном** (інший покупець). Якщо рядок, який задовольняє подібному критерію, знайдений у внутрішньому запиті, це означає, що різні покупки обслуговуються даним продавцем. Отже, предикат **exists** вірний для поточного рядка, й номер поля продавця (**прод_ном**) із таблиці зовнішнього запиту включається до складу вихідних даних.

Комбінування EXISTS і з'єднань

Для одержання додаткової інформації про кожного продавця можна зв'язати таблиці **Продавці** та **Покупці**:

```
select distinct Пр1.прод_ном, Пр1.прод_прізви, Пк1.місто  
from Продавці Пр1, Покупці Пк1  
where exists  
(select*  
from Покупці Пк2  
where Пк1.прод_ном=Пк2.прод_ном  
and Пк1.покуп_ном<>Пк2.покуп_ном)  
and Пр1.прод_ном=Пк1.прод_ном;
```

Зовнішній запит є з'єднанням таблиць **Продавці** та **Покупці**. Оператор **and** зв'язує в зовнішньому запиті **exists** і додаткову умову.

Підзапит для предиката **EXISTS** теж може мати у своєму складі один або декілька підзапитів будь-якого типу. Наприклад, необхідно знайти всіх продавців, які мають покупців, зробивших більш одного замовлення:

```
select *  
from Продавці Пр  
where exists  
(select*  
from Покупці Пк  
where Пр.прод_ном=Пк.покуп_ном  
and 1 <  
(select count(*)  
from Замовлення З  
where  
З.покуп_ном=Пк.покуп_ном));
```

Зовнішній запит бере кожний рядок таблиці **Продавці** у якості рядку-кандидата. Для нього береться кожний рядок із таблиці **Покупці** (середній запит). Якщо поточний рядок покупця не відповідає поточному рядку продавця, то предикат середнього запиту хибний. Після знаходження в середньому запиті покупця, що відповідає продавцю в зовнішньому запиті, виконується перехід до третього запиту, щоб визначити істинність предиката середнього запиту.

Внутрішній запит виконує підрахунок замовлень для поточного покупця (із середнього запиту). Якщо це значення перевищує 1, то предикат середнього запиту приймає значення «істина» й рядок вибирається.

Це робить предикат **exists** зовнішнього запиту істинним для поточного рядку продавця. Це означає, що, принаймні, один із покупців даного продавця має більш одного замовлення.

5.6.5 Об'єднання множини запитів в один

Крім вкладення запитів їх можна також об'єднувати. **Об'єднання (unions)** відрізняються від підзапитів тим, що будь-який із двох (або більшого числа) запитів не може управляти іншим запитом.

Можна задати множину запитів одночасно та комбінувати їх вихідні дані з використанням речення **UNION**, що об'єднує вихідні дані декількох SQL-запитів у єдину множину рядків і стовпчиків.

Наприклад, для одержання відомостей про всіх продавців і покупців Запоріжжя у вигляді вихідних даних одного запиту:

```
select под_ном, покуп_ном  
from Продавці  
where місто = «Запоріжжя»  
  
union  
select под_ном, покуп_ном  
from Покупці  
where місто = «Запоріжжя»;
```

Стовпчики, обрані за допомогою двох команд, подані у вихідних даних так, якби вони вибиралися за допомогою одного запиту. При цьому заголовки стовпчиків опускаються, оскільки в результат об'єднання входять стовпчики з декількох різних таблиць.

Для об'єднання двох і більш запитів необхідно, щоб їхні стовпчики, що входять у набір вихідних даних, були **сумісні по об'єднанню**. Крім того, якщо NULL-значення заборонені для будь-якого стовпчика в об'єднанні, то вони повинні бути заборонені для усіх відповідних стовпчиків в інших запитах об'єднання.

Не можна використовувати **UNION** у підзапитах також, як і функції агрегування в реченнях **SELECT** запитів в об'єднанні (багато програмних продуктів пом'якшують ці обмеження).

5.7 Додавання, зміна та вилучання даних

У **SQL** використовуються три команди модифікації даних:

- **INSERT** додає нові рядки до таблиці;
- **UPDATE** змінює існуючі в таблиці рядки;
- **DELETE** вилучає рядки з таблиці.

5.7.1 Додавання нового рядку

Оператор **INSERT** дозволяє додавати рядки за допомогою ключового слова **VALUES** або за допомогою оператора **SELECT**. Для кожного рядку, що додається, використовується свій оператор **INSERT**.

<pre>INSERT INTO <i>ім'я_таблиці</i> [(<i>стовпчик1</i> [, <i>стовпчик2</i>]...)] VALUES(<i>константа1</i> [, <i>константа2</i>]...)</pre>
--

Наприклад, треба додати новий товар до таблиці **Товари(код, назва, ціна, кількість)**.

*insert into Товари (код, назв, ціна, кількість)
values('1756', «монітор», 2000, 6);*

Порядок слідування стовпчиків в операторі **INSERT** може бути будь-яким, проте він повинний відповідати порядку слідування значень.

Якщо рядок, який додається, містить значення окремих стовпчиків таблиці, то стовпчики, що залишилися, (що не модифікуються) повинні припускати нульовий статус. У протилежному випадку команда буде заперечена.

Для одержання даних з однієї або декількох таблиць у команді **INSERT** можна використовувати оператор **SELECT**:

```
INSERT INTO ім'я_таблиці [(список стовпчиків,  
які вставляються)]  
SELECT список_стовпчиків  
FROM список_таблиць  
WHERE умови
```

Оператор **SELECT** у команді **INSERT** дозволяє взяти дані з декількох або з усіх стовпчиків однієї таблиці та вставити їх в іншу таблицю. Якщо вставлені значення тільки для частини стовпчиків, визначити значення для інших стовпчиків можна буде пізніше за допомогою оператора **UPDATE**.

При вставці рядків з однієї таблиці в іншу ці таблиці повинні мати сумісну структуру. Якщо стовпчики в обох таблицях сумісні за типами та визначені в однаковому порядку у відповідних операторах **CREATE TABLE**, перераховувати їх у команді **INSERT** необов'язково.

Наприклад, до таблиці *Прайс-лист(тип, назва, ціна)* треба додати інформацію з таблиці *Товари(тип, назва, ціна_закуп, ціна_відпускна, кількість)*

```
insert into Прайс-лист(тип, назва, ціна)  
select тип, назва, ціна_відпускна  
from Товари
```

Оператор **SELECT** у команді **INSERT** може містити агрегатні функції. Наприклад, є таблиці *Підсумки(дата, сума)* та *Продаж(ном, дата, вартість)*. Необхідно підрахувати щоденну виручку за минулий місяць.

```

insert into Підсумки (дата, сума)
select дата, sum(вармість)
from Продаж
group by дата;

```



При використанні підзапитів у всіх командах оновлення в реченні **FROM** будь-якого підзапиту не можна посилатися на таблицю, що змінюється в основній команді.

5.7.2 Вилучання рядків із таблиці

За допомогою команди **DELETE** із таблиці вилучається не окреме поле, а рядок цілком.

DELETE *

FROM ім'я_таблиці

У результаті виконання цієї команди таблиця стає порожньою та її можна вилучити командою **DROP TABLE**. Для вилучання окремих рядків використовується предикат:

```

delete from Продавці      delete from Замовлення
where прод_ном = 103;     where місто = «Донецьк»;

```

У предикаті команди **DELETE** можна використовувати підзапити. Наприклад, для вилучання всіх покупців, що обслуговуються продавцями з **Запоріжжя**:

```

delete from Склад_Замовлення
where ном_заказа = any
(select ном_заказа
 from Замовлення
 where місто = «Запоріжжя»);

```

Хоча в реченні **FROM** підзапиту не можна посилатися на таблицю, з якої здійснюється вилучання, у предикаті можна посилатися на поточний рядок-кандидат, тобто на той рядок, що у даний момент перевіряється в основному предикаті. Таким чином, можна також використовувати зв'язані підзапити. Наприклад, вилучити всіх продавців, що мають покупців із рейтингом 100:

```

delete from Продавці Пр
where exists
(select *
 from Покупці Пк
 where рейтинг = 100
 and Пр.ном_прод=Пк.ном_прод);

```

Аналогічно:

```
delete from Продавці Пр  
where 100 in  
(select рейтинг  
from Покупці Пк  
where  
Пр.ном_прод=Пк.ном_прод);
```

Запит знаходить усі рейтинги покупців кожного продавця і вилучає продавця, що має покупців із рейтингом 100.

5.7.3 Зміна значень полів

```
UPDATE ім'я таблиці  
SET ім'я стовпчика = вираз  
[WHERE умова
```

При відсутності речення **WHERE** зміни вносяться у всього рядка стовпчиків, зазначених у **SET**.

```
update Видавництва           update Продавці  
set місто=«Запоріжжя», країна set ціна = ціна*2;  
= «Україна»;
```

```
update Замовлення  
set ціна = ціна*2  
where місто = «Київ»;
```

У предикаті команди **UPDATE** можна використовувати підзапити. Наприклад, потрібно підвищити комісійні для всіх продавців, які обслуговують не менше двох покупців:

```
update Продавці  
set ком = ком + 0.1  
where 2 <=  
(select count(покуп_ном)  
from Покупці  
where
```

```
Покупці.прод_ном=Продавці.прод_ном);
```

Зменшити комісійні для продавців, які одержали мінімальні замовлення:

```
update Продавці  
set ком = ком - 0.1  
where прод_ном in
```

```
(select прод_ном
from Замовлення А
where сума =
(select min(сума)
from Замовлення Б
where А.дата=Б.дата));
```

5.8 Подання

5.8.1 Створення та використання подань

Подання, або курсор (**view**), – це об’єкт, що не містить власних даних. Це свого роду віртуальна таблиця, що не існує як незалежний об’єкт у базі даних і вміст якої береться з інших таблиць за допомогою виконання запиту.

Оскільки значення в таблицях змінюються, це автоматично призводить до відповідних змін у поданнях. Подання можна уявити як рухливий кадр або вікно, через яке користувач бачить дані. Запити до подань виконуються в основному так само, як і до звичайних таблиць.

У термінах **ANSI** реальні таблиці, про які мова йшла дотепер, називаються **базовими таблицями (base table)**, а терміну «**подання**» відповідає термін «**таблиця, що переглядається (viewed table)**».

Подання створюється за допомогою команди:

```
CREATE VIEW ім'я_подання [(ім'я_стовпчика
[, ім'я_стовпчика]...)]
AS
SELECT_оператор
```

Наприклад:

```
create view Замовлення-1
as select *
from Замовлення
where квартал = 1;
```

У результаті створюється віртуальна таблиця **Замовлення-1**, до якої можна формулювати запити, виконувати оновлення, вставку, вилучання та з’єднання з іншими таблицями й поданнями. Вміст подання не фіксується, а повторно обчислюється при посиланні на нього.

Подання дозволяють налагодити загальну базу даних для потреб конкретного користувача, кожний із який може мати свої власні інтереси й рівні кваліфікації.

Крім того, подання забезпечують певний захист бази даних від несанкціонованого доступу шляхом обмеження доступу до всієї інформації, приховуючи секретні або не стосовні до конкретної задачі частини бази даних. Наприклад, кожний продавець може переглянути загальні відомості про інших продавців, але не може бачити значення їхніх комісійних. Детальніше розподіл доступу користувачів до БД описується у розділі 6.3.

Групові подання

Групові подання (*grouped views*) – це подання, що містять речення **GROUP BY** або базуються на інших поданнях.

Припустимо, що нам щодня потрібно відслідковувати кількість покупців, які мають замовлення, кількість продавців, які одержали замовлення, кількість замовлень, середню суму замовлень і загальну суму замовлень, які надійшли. Замість того, щоб багаторазово конструювати складний запит, можна просто створити таке подання.

create view Підсумки_за_день

***as select дата, count(distinct пок_ном), count(distinct
прод_ном), count(замовл_ном), avg(сума), sum(сума)
from Замовлення
group by дата;***

Тепер можна одержати всю необхідну інформацію за допомогою єдиного запиту:

select*

from Підсумки_за_день;

Як працюють подання

Створення подання означає його визначення в термінах базових таблиць. Визначення подання зберігається в словнику даних без зв'язаних із ним даних. Подання дозволяють одержати доступ до зв'язаних з ними таблиць, тобто при створенні та використанні подання не створюється ніякої нової копії даних.

При зміні даних у поданні змінюються дані в зв'язаних із ним базових таблицях. І навпаки, усі зміни в базовій таблиці автоматично відбиваються в створеному на її основі поданні.

Наприклад, для визначення книг, вартість котрих більше 15 грн і витрати на які перевищили 5000 грн:

```
select *  
from Книги  
where ціна > 15 and витрати > 5000;
```

Припустимо, що необхідно виконати стосовно цієї сукупності даних ряд операцій вибірки та оновлення. У цьому випадку для зручності можна створити подання, в якому будуть визначені тільки потрібні записи:

```
create view Ціни-мах  
as  
select *  
from Книги  
where ціна > 15 and витрати > 5000;
```

Коли **SQL** одержує цю команду, він насправді не виконує оператор **SELECT**, який стоїть за ключовим словом **AS**. Замість цього він запам'ятовує цей оператор **SELECT** у словнику даних. Фактично даний оператор **SELECT** є визначенням подання **Ціни-мах**.

Після цього, коли виконуються які-небудь операції стосовно подання **Ціни-мах**, **SQL** комбінує ці операції з визначенням подання, що запам'ятоване. Наприклад, процедура зміни всіх цін у **Ціни-мах** нічим не відрізнялася б від внесення змін до будь-якої іншої таблиці:

```
update Ціни-мах  
set ціна = ціна*2;
```

У дійсності **SQL** знаходить визначення подання в словнику даних і перетворює цю команду оновлення в такий оператор:

```
update Книги  
set ціна = ціна*2  
where ціна > 15 and витрати > 5000;
```

Іншими словами, із визначення подання **SQL** знає, що дані, котрі підлягають оновленню, знаходяться в таблиці **Книги**. Після виконання оператора оновлення подання відповідний результат можна побачити або за допомогою даного подання, або в таблиці **Книги**. Та навпаки, якщо подання створене та зроблено безпосереднє оновлення базової таблиці, то результат можна побачити також і за допомогою подання.

При створенні подань можна також використовувати з'єднання та підзапити. Далі у прикладі визначається подання, що містить три з'єднання та підзапит. У ньому знаходиться прізвище автора, назва книги, видавництво та ціни на книги, вартість яких перевищує середнє значення цін усіх книг.

```
create view Ціни_вище_середніх  
as  
select Автори.прізвище, Книги.код_книги, Книги.видавництво,  
Книги.ціна  
from Автори, Книги-Автори, Книги, Видавництва  
where Автори.код_авт = Книги-Автори.код_авт  
and Книги.код-кн = Книги-Автори.код-кн  
and Книги.код_вид = Видавництва.код_вид  
and ціна >  
(select avg(ціна)  
from Книги);
```

Подання можна також використовувати для визначення іншого подання. Наприклад, використовувати попереднє подання для створення нового подання, у якому відображаються книги, видані у видавництві «Освіта»:

```
create view Ціни-Освіта  
as select *  
from Ціни_вище_середніх  
where видавн = ' Освіта ';
```

Переглянути дані можна за допомогою запиту

```
select *  
from Ціни-Освіта;
```

5.8.2 Модифікація даних за допомогою подань

Переліки операторів модифікації даних, якими дозволено користуватися в поданнях, відрізняються в різних реалізаціях **SQL**.

Правила використання подань відповідно до ANSI

Стандарт **ANSI** встановлює, що є тільки подання, що читаються (тобто не модифікуються), якщо оператор **CREATE VIEW** містить один із таких елементів:

- **DISTINCT** у списку вибору;
- вираз (обчислюванні стовпчики, функції, тощо) у списку вибору;

- посилання на декілька таблиць або в реченні **FROM**, або в підзапиті, або в реченні **UNION**;
- посилання на подання, що не оновлюється, в реченні **FROM** або в підзапиті;
- речення **GROUP BY** або **HAVING**.

Речення With Check Option

```
CREATE VIEW ім'я_подання [(ім'я_стовпчика
[, ім'я_стовпчика]...)]
AS
SELECT_оператор
[WITH CHECK OPTION]
```

Дане речення змушує **SQL** відхиляти будь-які спроби модифікації подання, котрі призводять до того, що один або декілька рядків перестають задовольняти умовам цього подання. Іншими словами, якщо оператор модифікації даних (**UPDATE**, **INSERT** або **DELETE**) призводить до зникнення якихось рядків із подання, такий оператор вважається неприпустимим.

Наприклад, якщо у визначенні подання зазначено, що в стовпчику *ціна* містяться ціни, що перевищують 15 грн, то спроба введення рядка зі значенням *ціна* = 15 закінчиться невдачею, якщо у визначенні подання використовувалося речення **With Check Option**.

5.9 Питання адміністрування баз даних

У багатокористувацькій системі питання безпеки даних вирішуються на рівні **SQL** і СКБД. Для цього є два механізми. Перший заснований на виділенні повноважень. При цьому за допомогою команд **GRANT** і **REVOKE** указується, яким користувачам дається право виконувати визначені команди відносно визначених таблиць, представлень або стовпчиків. Другий механізм заснований на використанні представлень у сполученні з командами **GRANT** і **REVOKE** для надання виборчого доступу до різноманітних підмножин даних.

Обидва механізми припускають, що в системі передбачений засіб ідентифікації (розпізнавання) кожного конкретного користувача.

Ідентифікація користувача й особливих користувачів

Більшість багатокористувацьких систем керування базами даних розпізнають, принаймні, два види користувачів, наділених особливими привілеями:

– суперкористувача, якого найчастіше називають **DBA** (*database administrator – адміністратор бази даних*), або *системним адміністратором*;

– власників об'єктів бази даних.

Деякі системи баз даних розпізнають додаткові типи користувачів і встановлюють ієрархію, у якій привілеї призначаються користувачу в залежності від місця в цій ієрархії.

Адміністратор бази даних призначається в момент інсталяції системи. **DBA** – це особлива посада, що може бути (тимчасово) зайнята будь-яким користувачем, який знає відповідне ім'я та пароль.

У багатьох інсталяціях БД роль адміністратора спільно виконується кількома людьми. Кожний із цих користувачів може увійти до системи як **DBA** для виконання адміністративних задач, але для виконання інших функцій у цій базі даних він повинний користуватися своїм власним ім'ям.

DBA, як правило, наділяється рядом особливих привілеїв; крім того, на нього покладається відповідальність за нормальне функціонування прикладного програмного забезпечення БД.

Іншою особливою категорією користувачів є власник кожної конкретної таблиці або подання. Власник таблиці або подання може автоматично наділитися правом виконувати будь-які операції з цією таблицею або поданням, у тому числі (як правило) видавати дозвіл на це іншим користувачам.

Користувачі, що не є власниками таблиці або подання (а також **DBA**), повинні одержати в явному виді дозвіл на виконання певних операцій над цією таблицею або поданням. Такі дозволи реалізуються за допомогою команд **GRANT** і **REVOKE**.

Команди GRANT і REVOKE

Ці команди вказують, яким користувачам можна виконувати визначені операції у відношенні до певних таблиць, представлень і стовпчиків. Контрольованими операціями є **SELECT**, **UPDATE**, **INSERT**, **DELETE** і **REFERENCES**.

Команда **GRANT** призначає привілею та має два варіанти синтаксису, що відрізняються за місцем розташування списку стовпчиків:

1) **GRANT {ALL | список_повноважень}**
ON {ім'я таблиці [(список_стовпчиків)] |

```

ім'я_подання [(список_стовпчиків)]}
TO { PUBLIC | список_користувачів}
[WITH GRANT OPTION]
2) GRANT {ALL | список_повноважень [(список_стовпчиків)]}
ON {ім'я_таблиці | ім'я_подання}
TO { PUBLIC | список_користувачів}
[WITH GRANT OPTION]

```

Команда **REVOKE** анулює видані повноваження:

```

REVOKE {ALL | список_повноважень}
ON {ім'я_таблиці [(список_стовпчиків)] |
ім'я_подання [(список_стовпчиків)]}
FROM { PUBLIC | список_користувачів}
[WITH GRANT OPTION]
REVOKE {ALL | список_повноважень [(список_стовпчиків)]}
ON {ім'я_таблиці | ім'я_подання}
FROM { PUBLIC | список_користувачів}
[WITH GRANT OPTION]

```

Наприклад, **Іваненко** одержує дозвіл додавати дані в таблицю **Товари** та обновляти її:

```

grant insert, update
on Товари
to Іваненко;

```

Для заборони **Іваненко** обновляти стовпчики **ціна** та **кількість** у таблиці **Замовлення**:

revoke update (ціна, кількість)	revoke update
on Замовлення	on Замовлення (ціна,
from Іваненко;	кількість) from Іваненко;

У списку повноважень вказується або ключове слово **ALL**, або список повноважень, якими наділяється користувач (або які знімаються з нього). Повноваження в списку розділяються комами.

У операторі **ON** указується таблиця або подання, на які видаються або анулюються повноваження (по одній таблиці або одному представленню на оператор). Коли повноваження видаються на окремі стовпчики, імена стовпчиків перераховуються в круглих дужках. При цьому стовпчики повинні належати одній таблиці або представленню. Якщо список стовпчиків не зазначений, повноваження видаються або анулюються на всю таблицю або подання.

Ключове слово **PUBLIC** («загальнодоступний») означає всіх користувачів у системі. Альтернативою йому є список користувачів.

Необов'язковий оператор **WITH GRANT OPTION** визначає, чи може користувач, що наділяється повноваженнями, у свою чергу, передавати їх іншим користувачам. Нижче наведений оператор, який наділяє користувача *Петренко* правом виконувати операцію **SELECT** у таблиці *Товари* та дозволяє йому передавати це право іншим користувачам:

**grant select
on Товари
to Петренко
with grant option;**

Подання як механізм забезпечення безпеки

Дозвіл на доступ до підмножини даних у представленні повинний бути наданий або анульований явно, незалежно від чинної сукупності повноважень, які відносяться до таблиці (або таблиць), на якій засноване дане подання.

За допомогою подання користувачі можуть запитувати або модифікувати тільки ті дані, які вони можуть бачити. Інша частина бази даних є для них і невидимою, і недоступною.

Наприклад, може бути небажано, щоб якісь користувачі мали можливість доступу до стовпчиків у таблиці *Замовлення*, у якій зберігаються відомості про гроші та продажі. У цьому випадку можна створити подання (*Замовлення_перегляд*) на основі таблиці *Замовлення*, а потім надати всім користувачам права на це подання:

**revoke all
on Замовлення
from public;
grant all
on Замовлення_перегляд
to public;**

Визначаючи різноманітні подання та вибірково надаючи права на них, можна обмежити доступ користувача або будь-якої групи користувачів до різноманітних підмножин даних.



Питання для самоперевірки

1. Які типи даних підтримує SQL?
2. Які групи команд входять до складу SQL?
3. Формат команди **CREATE TABLE**?
4. Які обмеження на множину припустимих значень використовуються в SQL?
5. Яким чином підтримується посилальна цілісність?
6. Навіщо використовуються індекси? Як вони створюються та працюють?
7. Загальний формат команди SELECT?
8. Яке призначення речення WHERE?
9. Для чого використовуються агрегатні функції?
10. Яким чином відбувається групування даних у запитах?
11. Як можна відсортувати результат виконання запиту?
12. Як виконується з'єднання декількох таблиць у запитах?
13. Які типи підзапитів використовуються в SQL?
14. Що таке об'єднання запитів?
15. Які команди входять до складу команд модифікації даних?
16. Чим віртуальні таблиці відрізняються від базових?
17. Навіщо використовуються подання?
18. Що таке привілеї, як вони використовуються?

6 ЗАХИСТ ДАНИХ

При експлуатації баз даних актуальною є проблема захисту інформації, пов'язана з можливістю виникнення наступних ситуацій:

- система може бути зруйнована під час виконання деяких програм, залишивши при цьому базу даних у непередбаченому стані;
- при виконанні конкуруючих програм між ними може виникнути конфлікт, який призведе до одержання неправильних результатів;
- дані можуть бути зіпсовані або змінені анонімним користувачем;
- оновлення можуть змінювати базу даних неприпустимим чином та інші.

Отже, система повинна забезпечувати функції захисту бази даних від подібних проблем, зокрема, відновлення, паралелізм, захист і цілісність.

6.1 Відновлення

Відновлення в системі баз даних означає, в першу чергу, відновлення самої бази даних, тобто повернення її в правильний стан, якщо якийсь збій зробив поточний стан неправильним або підозрілим.

Основний принцип, на якому будується таке відновлення, – це **надмірність**. Але ця надмірність будується на фізичному рівні та схована від користувача. Це означає, що, якщо будь-яка частина інформації, що міститься в базі даних, може бути реконструйована з іншої збереженої в системі надлишкової інформації, то база даних є такою, що відновлюється [10].

6.1.1 Транзакції



Транзакція – це логічна одиниця роботи, яка з точки зору відновлення даних розглядається та обробляється системою як єдина неподільна дія.

При цьому під транзакцією у загальному випадку розуміється не одиночна операція, а сукупність узгоджених операцій, що змінюють стан бази даних

Загалом, це перетворення одного узгодженого стану бази даних в інший, причому в проміжних точках база знаходиться в

неузгодженому стані. З цього випливає, що не припустимо, щоб одне з оновлень було виконано, а інше ні, тому що база даних залишиться в неузгодженому стані.

Системний компонент, який забезпечує атомарність, *називається адміністратором* транзакцій (або диспетчером транзакцій).

В стандарті **ISO SQL/92** вказується, що транзакція автоматично запускається будь-яким оператором, який ініціює транзакцію та який виконується користувачем або програмою (наприклад, **SELECT**, **INSERT** або **UPDATE**). Зміни, внесені до бази даних в ході виконання даної транзакції, не будуть побачені будь-якими іншими транзакціями, що виконуються паралельно, доти, доки ця транзакція не буде явним чином завершена. Завершення транзакції може бути виконано одним із наступних способів.

- Введення оператора **COMMIT** означає успішне завершення транзакції. Після його виконання зміни, що були внесені до БД, набувають постійного характеру. Після обробки оператора COMMIT введення будь-якого оператора, що ініціює транзакцію, автоматично викличе запуск нової транзакції. Даний оператор установлює так звану *точку фіксації* (у комерційних продуктах її також називається *точкою синхронізації* – *syncpoint*).

- Введення оператора **ROLLBACK** означає відмову від завершення транзакції, в результаті чого виконується скасування (відкат) усіх змін у БД, які були внесені при виконанні цієї транзакції, тобто система знову повертається до попередньої точки фіксації. Після обробки оператора ROLLBACK введення будь-якого оператора, що ініціює транзакцію, автоматично викличе запуск нової транзакції.

- При використанні SQL-операторів у тексті програми успішне закінчення її роботи автоматично викличе завершення останньої транзакції, що була запущена програмою, навіть якщо оператор COMMIT для неї не був уведений явно.

- При використанні SQL-операторів у тексті програми аварійне закінчення її роботи автоматично викличе відкат останньої транзакції, що була запущена програмою.

По умовчанню транзакції (запити) задаються СКБД неявно. Проте користувач може сам визначати склад транзакції, задаючи СКБД

команду на виконання будь-якої кількості операторів як єдиного блока. Класичним прикладом транзакції, визначеної користувачем, є переказ грошей з одного рахунку на інший. З погляду БД ця операція складається з двох операцій: зняття грошей для відбитка дебету та оновлення другого рахунку на основі кредиту. Об'єднання цих двох операцій у єдину транзакцію, визначену користувачем, гарантує, що переказ грошей буде або завершений цілком, або не відбудеться взагалі.

Крім надання користувачу контролю над керуванням транзакціями, транзакції, визначені користувачем, підвищують загальну продуктивність системи, оскільки надлишкові (але неминучі) дії в системі мають місце не для кожної окремої команди, а лише однократно – для транзакції в цілому.

Транзакцію можна відмінити (або повернути до початкового стану) в будь-який момент до виконання команди **COMMIT**. Однак скасувати транзакцію після її виконання не можна.

Для захисту від збоїв у системі використовується **системний журнал транзакцій (Transaction log)**, у якому записуються деталі всіх операцій оновлення, зокрема, нове та старе значення модифікованого об'єкта. Таким чином, при необхідності скасування деякого оновлення система може використовувати відповідний журнал для повернення об'єкта в початковий стан.

Транзакції необхідні для **контролю збігів** (усунення конфліктів користувачів або додатків, що намагаються одночасно звернутися до бази даних) і **відновлення** (надання можливості системі повернути базу даних у нормальний стан після збою програмного або апаратного забезпечень).

6.1.2 Транзакції та відновлення

Транзакція є не тільки одиницею роботи, але й одиницею відновлення при невдалому виконанні операції. Вони забезпечують можливість відновлення системи після системного збою, тобто приведення системи в самий останній за часом поточний стан, який можна з упевненістю вважати працездатним і достовірним.

Системними називаються збої, що впливають на всі транзакції, що виконуються в даний момент, але не завдають фізичної шкоди самій базі даних.

У випадку системного збою механізм відновлення системи баз даних повинний з'ясувати два питання:

- Які транзакції не встигли завершитися до моменту збою та, отже, повинні бути скасовані.
- Які транзакції встигли завершитися до моменту збою, але відповідна інформація ще не переписалася із внутрішніх буферів системи до самої бази даних (фізичної) та, отже, повинні бути **виконані повторно**.

Ідентифікація транзакцій під час перезавантаження системи провадиться в такий спосіб. Через певний інтервал часу (або коли в журналі накопичується визначене число записів) система автоматично встановлює **контрольну точку**. При цьому відбувається фізичний запис вмісту робочих буферів бази даних безпосередньо до бази даних і фіксація списку виконуваних у даний момент транзакцій.

На рис. 6.1 показані п'ять можливих варіантів виконання транзакцій до аварійного збою системи.

Нижче наведені пояснення до рисунку.

- Відмова системи відбулася в момент часу **tf**.
- Найближча до моменту часу **tf** контрольна точка була прийнята в момент часу **tc**.
- Транзакція **T1** успішно завершена до моменту часу **tc**.
- Транзакція **T2** почата до моменту часу **tc** і успішно завершена після моменту часу **tc**, але до моменту часу **tf**.
- Транзакція **T3** також почата до моменту часу **tc**, але не завершена до моменту часу **tf**.
- Транзакція **T4** почата після моменту часу **tc** і успішно завершена до моменту часу **tf**.
- Транзакція **T5** також почата після моменту часу **tc**, але не завершена до моменту часу **tf**.

Вочевидь, що під час перезавантаження системи транзакції типу **T3** і **T5** повинні бути скасовані, а транзакції типу **T2** і **T4** – виконані повторно. Транзакції типу **T1** не включаються в процес перезавантаження, тому що оновлення зафіксовані до моменту часу **tc**. Транзакції, що завершилися невдало перед моментом часу **tf**, а також перервані в результаті збою, до процесу перезавантаження не включаються.

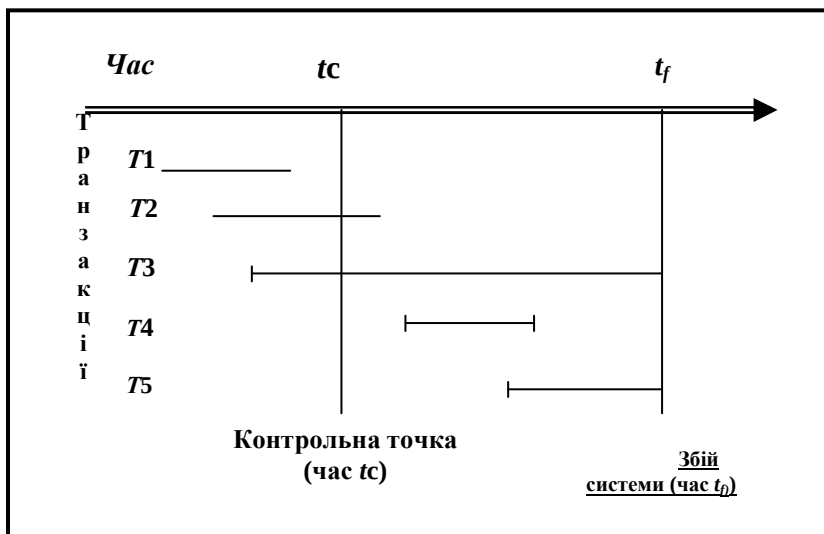


Рисунок 6.1 – П'ять варіантів виконання транзакцій

Отже, під час перезавантаження система спочатку проходить через процедуру ідентифікації всіх транзакцій типу **T1-T5**. При цьому виконуються наступні кроки.

1. Створюються два списки транзакцій: назвемо їх **Undo** (скасувати) і **ReDO** (виконати повторно). До списку **Undo** заносяться *всі* транзакції із запису контрольної точки (тобто всі транзакції, що виконувалися в момент часу t_c), а список **ReDO** залишається *порожнім*.

2. Здійснюється пошук у журналі транзакцій, починаючи з запису контрольної точки.

3. Якщо в журналі виявлений запис **BEGIN TRANSACTION** про початок транзакції T_i , то ця транзакція також добавляється до списку **Undo**.

4. Якщо в журналі виявлений запис **COMMIT** про закінчення транзакції T_i , то ця транзакція добавляється до списку **ReDO**.

5. Коли досягається кінець журналу реєстрації, списки **Undo** і **ReDO** аналізуються для ідентифікації транзакцій типу **T2** і **T4**, що з'явилися в списку **ReDO**, і транзакцій типу **T3** і **T5**, що залишилися в списку **Undo**.

Після цього система проглядає назад журнал реєстрації, вилучаючи транзакції зі списку **UnDO**, а потім переглядає знову вперед, повторно виконуючи транзакції зі списку **ReDO**.

Відновлення БД у правильний стан шляхом скасування виконаних операцій називають також **оберненим відновленням**. Аналогічно, відновлення її в правильний стан повторним виконанням називається **прямим відновленням**.

Механізм відновлення в основному схований від користувача, проте в деяких системах у розпорядженні користувача є така команда, як **CHECKPOINT**, що змушує систему переписати на диск зі своїх буферів результати всіх транзакцій, що завершилися.

При успішному завершенні транзакції система гарантує, що оновлення зафіксовані в БД, навіть якщо система потерпить крах у наступний момент.

Наприклад, у системі відбувся збій після успішного виконання операції **COMMIT**, але перед тим, як оновлення будуть **фізично** записані до БД (знаходячись у буфері оперативної пам'яті, вони можуть бути загублені в момент збою системи). Навіть якщо подібне трапилося, процедура перезавантаження системи повинна відновити БД відповідно до записів у журналі транзакцій.

З цього випливає, що оновлення фіксується в журналі транзакцій до того, як воно відбудеться в самій БД. Цей тип журналу, який має назву **журналу з випереджаючим записом**, гарантує можливість повного відновлення БД у випадку збоїв. При цьому буде потрібно повторне виконання транзакцій з наступною фізичною фіксацією в БД.

Таким чином, транзакціям притаманні чотири важливі властивості:

1. **Атомарність**. Транзакції атомарні (виконується все або нічого).
2. **Узгодженість**. Транзакції захищають базу даних узгоджено. Це означає, що транзакції переводять один узгоджений стан бази даних в інший без обов'язкової підтримки узгодженості у всіх проміжних точках.

3. **Ізоляція**. Транзакції відділені друг від друга. Це означає, що, якщо навіть буде запущена множина конкуруючих транзакцій, будь-яке оновлення визначеної транзакції буде приховано від інших доти, поки ця транзакція виконується. Іншими словами, для будь-яких двох транзакцій **T1** і **T2** справедливо таке

твердження: **T1** зможе побачити оновлення **T2** тільки після виконання **T2**, а **T2** зможе побачити оновлення **T1** тільки після виконання **T1**.

4. *Довговічність*. Коли транзакція виконана, її оновлення зберігаються, навіть якщо в такий момент відбудеться збій системи.

6.2 Паралелізм

Термін *паралелізм* означає можливість одночасного опрацювання в СКБД багатьох транзакцій, які звертаються до одних і тих же даних. Запобігання створення взаємних перешкод при одночасних транзакціях (*контроль збігу*) гарантує, що доти, доки процес оновлення даних не буде завершений, ніякий інший користувач не зможе працювати з цими даними або навіть переглядати їх [15].

У більшості систем керування реляційними базами даних для вирішення проблем паралелізму застосовується механізм, який має назву *блокування (locking)*. Принцип його роботи полягає в тому, що коли користувач вибирає якісь дані з БД, СКБД автоматично зачиняє їх «на замок» із тим, щоб ніякий інший користувач не міг оновлювати ці дані доти, доки блокування не буде зняте.

Існує два основні види блокування: *блокування без взаємного доступу (монопольне блокування)*, що має також назву *X-блокування (X lock – eXclusive lock)* і *блокування зі взаємним доступом* (спільне блокування), що має назву *S-блокування (S-lock – Shared lock)*.

X- і *S-блокування* називають також *блокуваннями запису та читання*, відповідно. Між цими видами блокування існують такі залежності:

1. Якщо транзакція **A** блокує кортеж **p** без можливості взаємного доступу (*X-блокування*), то запит іншої транзакції **B** із блокуванням цього кортежу **p** буде скасований.

2. Якщо транзакція **A** блокує кортеж **p** із можливістю взаємного доступу (*S-блокування*), то

– запит із боку деякої транзакції **B** на *X-блокування* кортежу **p** буде заборонений;

– запит із боку деякої транзакції **B** на *S-блокування* кортежу **p** буде прийнятий (тобто транзакція **B** також буде блокувати кортеж **p** за допомогою *S-блокування*).

Монопольне блокування застосовується в системах баз даних у ході операцій модифікації даних типу **UPDATE**, **INSERT**, **DELETE**.

Коли **X**-блокування застосовується до деякої сукупності даних, ніяка інша транзакція не може встановити стосовно цих даних ніякий вид блокування доти, доки наприкінці транзакції з модифікації даних не буде зняте початкове блокування.

Спільні блокування застосовуються при виконанні операцій, які не пов'язані з оновленням даних (вони також називаються операціями читання), таких як **SELECT**. Застосування **S**-блокування до деякої сукупності даних не дозволяє операції запису виконати виняткове блокування цих даних, але дозволяє іншим операціям читання виконувати свої власні спільні блокування навіть у тому випадку, якщо перша транзакція ще не завершилася.

Протокол доступу до даних на основі **X**- і **S**-блокування, що дозволяє уникнути проблем паралелізму, наступний:

1. Транзакція, що призначена для витягу кортежу, насамперед повинна накласти **S**-блокування на цей кортеж.

2. Транзакція, що призначена для оновлення кортежу, насамперед повинна накласти **X**-блокування на цей кортеж. Інакше кажучи, якщо, наприклад, для послідовності дій типу витяг/оновлення для кортежу вже задане **S**-блокування, то його необхідно замінити **X**-блокуванням.

3. Якщо блокування, що запитане з боку транзакції **B**, відкидається через конфлікт із деяким іншим блокуванням із боку транзакції **A**, то транзакція **B** переходить до стану чекання. Причому транзакція **B** буде знаходитися в стані чекання доти, доки не буде зняте блокування, задане транзакцією **A**.

4. **X**-блокування зберігається до кінця виконання транзакції (до операції «завершення виконання» або «скасування виконання»). **S**-блокування зазвичай також зберігаються до цього моменту, проте в окремих випадках воно може бути замінено **X**-блокуванням.

Блокування в транзакціях зазвичай задаються неявним чином. Наприклад, запит «*на витяг кортежу*» є неявним запитом із **S**-блокуванням, а запит «*на оновлення кортежу*» – неявним запитом із **X**-блокуванням відповідного кортежу. Зокрема, у стандарті **SQL** не передбачене явне завдання блокування.

У окремих системах такий механізм передбачений і реалізується за допомогою різного роду тверджень **LOCK**.

6.3 Безпека

6.3.1 Безпека та цілісність

Терміни **безпека** та **цілісність** у контексті обговорення баз даних часто використовуються спільно, хоча насправді, це цілком різні поняття. Термін **безпека** відноситься до захисту даних від несанкціонованого доступу, зміни або руйнації даних, а **цілісність** – до точності або істинності даних. Іншими словами:

- під безпекою припускається, що користувачам *дозволяється* виконувати деякі дії.

- під цілісністю припускається, що ці дії виконуються коректно.

Між ними є, звичайно, деяка подібність, оскільки як при забезпеченні безпеки, так і при забезпеченні цілісності система змушена перевірити, чи не порушують дії користувача деяких правил.

Ці правила повинні бути задані (зазвичай адміністратором БД) на деякій зручній для цього мові й збережені в системному каталозі. Причому в обох випадках СКБД повинна якимсь чином відстежувати всі дії користувача та перевіряти їхню відповідність заданим правилам.

6.3.2 Аспекти проблеми безпеки

Серед багатьох аспектів проблеми безпеки можна виділити такі:

- правові, суспільні й етичні аспекти (чи має право деяка особа одержувати певну інформацію, наприклад, про кредит клієнта);

- фізичні умови (наприклад, чи закритий даний комп'ютер або захищений будь-яким іншим засобом);

- організаційні питання (наприклад, як у рамках конкретного підприємства організований доступ до даних);

- питання реалізації керування (наприклад, якщо використовується метод доступу за паролем, то як організована реалізація керування та як часто змінюються паролі);

- апаратне забезпечення (чи забезпечуються заходи безпеки на апаратному рівні, наприклад, за допомогою захисних ключів або привілейованого режиму керування);

- безпека операційної системи (наприклад, чи затирає ОС зміст структури збереження та файлів із даними при припиненні роботи з ними);

- питання, що стосуються самої СКБД (наприклад, чи існує для БД деяка концепція надання прав володіння даними).

6.3.3 Привілеї



Привілеї (privileges) – це сукупність дій, які дозволено виконувати користувачеві по відношенню до конкретної таблиці або подання.

В стандарті ISO визначаються наступний набір привілеїв:

- **SELECT** – право вибирати дані з таблиці;
- **INSERT** – право додавати нові рядки до таблиці;
- **UPDATE** – право змінювати дані в таблиці;
- **DELETE** – право вилучати рядки з таблиці;
- **REFERENCES** – право посилатися на стовпці вказаної таблиці в описах вимог підтримки цілісності даних;
- **USAGE** – право використання доменів, перевірки наборів символів і трансляції.

Привілеї (**права** або **повноваження**) пов'язані як с користувачами, так і з таблицями. Коли користувач створює таблицю або подання, він автоматично стає її володарем і отримує по відношенню до неї повний набір привілеїв. Для надання доступу до цих об'єктів іншим користувачам необхідно, щоб володар явним чином надав їм ці права.

6.3.4 Розподілення привілеїв

У сучасних СКБД підтримується один із двох широко поширених підходів до питання забезпечення безпеки даних – **виборчий** та **обов'язковий** підходи. У обох підходах одиницею даних або «об'єктом даних», для яких повинна бути створена система безпеки, може бути як уся база цілком або який-небудь набір відношень, так і деяке значення даних для заданого атрибута у визначеному відношенні.

У випадку **виборчого** керування деякий користувач має різноманітні права при роботі з різноманітними об'єктами. Більш того, різні користувачі зазвичай володіють і різними правами доступу до одного й того ж об'єкта. Тому виборчі схеми характеризуються значною гнучкістю.

У випадку **обов'язкового** керування, навпаки, кожному об'єкту даних присвоюється деякий **класифікаційний** рівень, а кожний користувач має деякий рівень **допуску**. Отже, при такому підході доступом до визначеного об'єкта даних володіють тільки користувачі з відповідним рівнем допуску. Тому обов'язкові схеми достатньо жорсткі та статичні.

Незалежно від того, які схеми використовуються – виборчі або обов’язкові, усі рішення щодо допуску користувачів до виконання тих або інших операцій приймаються на стратегічному, а не технічному рівні. Тому вони перебувають за межами досяжності самої СКБД, і усе, що може в такій ситуації зробити СКБД, – це тільки пустити в хід уже прийняті раніше рішення. Виходячи з цього, можна відзначити наступне:

1. Результати стратегічних рішень повинні бути відомі системі (тобто виконані на основі тверджень, заданих за допомогою деякої мови) та зберігатися в ній (шляхом зберігання їх у каталозі у вигляді правил безпеки, що також називаються *повноваженнями*).

2. Отже, повинні бути деякі засоби регулювання запитів доступу стосовно відповідних правил безпеки. (Тут під «запитом доступу» припускається комбінація *операції* запиту, *об’єкта*, який запитується, *та користувача*, що запитує). Така перевірка виконується *підсистемою безпеки* СКБД, яка також називається *підсистемою повноважень*.

3. Для того, щоб розібратися, які правила безпеки до яких запитів доступу застосовуються, у системі повинні бути передбачені засоби упізнання джерела цього запиту, тобто упізнання користувача, що запитує. Тому в момент входу до системи користувач зазвичай повинен ввести не тільки свій ідентифікатор (наприклад, ім’я або посада), але також і пароль (щоб підтвердити свої права на заявлені раніше ідентифікаційні дані). Зазвичай передбачається, що пароль відомий тільки системі та деяким особам з особливими правами.

Слід зазначити, що різні користувачі можуть володіти одним ідентифікатором деякої групи. Таким чином, у системі можуть підтримуватися *групи користувачів* і забезпечуватися однакові права доступу для користувачів однієї групи, наприклад, для всіх осіб із розрахункового відділу. Крім того, операції додавання окремих користувачів до групи або їхнє вилучання з неї можуть виконуватися незалежно від операції завдання привілеїв для цієї групи.

6.3.5 Виборче керування доступом

Роздивися варіант завдання правил безпеки на деякій гіпотетичній мові.

CREATE SECURITY RULE правило
GRANT список_привілеїв_через_кому
ON вираз
TO список_користувачів_через_кому
[**ON ATTEMPTED VIOLATION** дія];

У загальному випадку правила безпеки задаються п'ятьма компонентами:

1. *Ім'я правила*, під яким воно буде зареєстровано в системному каталозі.

2. *Одна або декілька привілеїв*, які задані за допомогою директиви GRANT. У якості привілеїв можна використовувати одну з наведених нижче:

- **SELECT** [(список_атрибутів_через_кому)] – витяг;
- **INSERT** – вставка;
- **UPDATE** [(список_атрибутів_через_кому)] – оновлення;
- **DELETE** – вилучення;
- **ALL**.

Специфікація **ALL** є скороченим записом використання *всіх* привілеїв – **SELECT** (із *всіма* атрибутами), **INSERT**, **UPDATE** (із *всіма* атрибутами) і **DELETE**.

3. *Діапазон, до якого це правило застосовується* та який є деякою підмножиною кортежів єдиного відношення. Він задається за допомогою директиви ON.

4. *Один або декілька ідентифікаторів користувачів*, які мають задані привілеї в заданому діапазоні, що вказується за допомогою директиви TO. Якщо задана директива ALL, то вона означає всіх відомих системі користувачів.

5. *Реакція на порушення правил*, яка диктує системі визначені дії, якщо користувач порушив правила безпеки. Вона задається директивою ON ATTEMPTED. По умовчання приймається відмова у виконанні дії, що запитувалася. Проте в більш складних ситуаціях може виявитися корисним застосувати іншу дію; наприклад, у деяких випадках може знадобитися завершити виконання програми або заблокувати клавіатуру користувача.

Приклад 1

CREATE SECURITY RULE SR3

/* Створити правило безпеки SR3 */

GRANT SELECT (S#, SNAME, CITY), DELETE

/* Дозволити вибірку атрибутів (S#, SNAME, CITY) і вилучання */

ON S WHERE S. CITY ≠ 'London'

/* Для постачальників не з Лондона */

TO Jim, Fred, Mary

/* Для користувачів: Джим, Фред, Мері */

ON ATTEMPTED VIOLATION REJECT;

/* Не виконувати при порушенні цього правила */

Приклад 2

CREATE SECURITY RULE EX2

GRANT SELECT (S#, SNAME, CITY)

ON S

TO Іваненко, Петренко, Сидоренко;

В останньому прикладі вказані користувачі можуть переглядати «вертикальну підмножину» атрибутів відношення **S**, яка не залежить від якогось значення. Це приклад правила безпеки, що характеризується *незалежністю від значення*.

Приклад 3

CREATE SECURITY RULE EX3

**GRANT SELECT, INSERT, UPDATE (код_товару, назва),
DELETE**

ON Товар WHERE Товар. ціна < 100\$

TO ALL;

Всі користувачі можуть бачити «горизонтальну підмножину», точніше, підмножину кортежів, що залежить від деякого значення (у даному випадку тільки ті товари, у яких ціна менше 100\$). Це приклад правила безпеки, що *характеризується залежністю від значення*.

У системі також необхідно передбачити засіб усунення існуючих правил безпеки:

DESTROY SECURITY RULE правило.

6.3.6 *Обов'язкове керування доступом*

Методи обов'язкового керування доступом застосовуються до баз даних, у яких дані мають достатньо статичну або жорстку структуру, властиву, наприклад, урядовим або військовим організаціям.

Основна ідея такого керування полягає в тому, що кожний об'єкт даних має деякий *рівень класифікації*, наприклад: секретно, абсолютно секретно, для службового користування та інші, а кожний користувач має рівень допуску з *такими ж* градаціями, що й рівні класифікації.

Передбачається, що ці рівні створюють суворий ієрархічний порядок, наприклад: абсолютно секретно, секретно, для службового користування тощо. Тоді на підставі цих відомостей можна сформулювати два прості правила безпеки:

1. Користувач *A* має *доступ* до об'єкта *B*, тільки якщо його рівень допуску *більше або дорівнює* рівню класифікації об'єкта *B*.
2. Користувач *A* може модифікувати об'єкт *B*, тільки якщо його рівень допуску *дорівнює* рівню класифікації об'єкта *B*.

Перше правило достатньо очевидно, а друге потребує додаткових роз'яснень. Друге правило можна сформулювати інакше: будь-яка інформація, що записана *користувачем A*, автоматично набуває рівень *класифікації A*.

Таке правило необхідно, наприклад, для того, щоб запобігти запису секретних даних, який виконується користувачем із рівнем допуску «секретно», до файлу з меншим рівнем класифікації, що порушує всю систему таємності.

6.3.7 *Контрольний слід виконання операцій*

Важливо розуміти, що не буває невразливих систем безпеки. Тому при роботі з дуже важливими даними або при виконанні критичних операцій виникає необхідність реєстрації *контрольного сліду* операцій, які виконуються.

Якщо, наприклад, суперечливість даних призводить до підозри, що зроблено несанкціоноване втручання до бази даних, то контрольний слід повинний бути використаний для прояснення ситуації й підтвердження того, що всі процеси знаходяться під контролем. Якщо це не так, то контрольний слід допоможе, принаймні, виявити порушника.

Для зберігання контрольного сліду зазвичай використовується особливий файл, у якому система автоматично записує всі операції, що виконані користувачем при роботі з БД. У деяких системах контрольний слід може фізично зберігатися у файлі відновлення даних, а в інших – в окремому файлі. У будь-якому випадку користувачі повинні мати можливість звертатися до контрольного сліду за допомогою традиційної мови реляційних запитів (звичайно, за умови, що такі запити санкціоновані!).

Типовий запис у файлі контрольного сліду може містити таку інформацію:

- запит (вихідний текст запиту);
- термінал, з якого була викликана операція;
- користувач, який задав операцію;
- дата й час запуску операції;
- базові відношення, кортежі й атрибути, до яких зверталися в процесі виконання операції;
- старі значення;
- нові значення.

Як показує практика, навіть констатація факту, що в даній системі підтримується контрольне спостереження, у деяких випадках дуже суттєва для запобігання несанкціонованого проникнення до системи.

6.4 Цілісність

Термін *цілісність* використовується для опису точності та коректності даних у БД. Як при забезпеченні безпеки, так і при підтримці цілісності система повинна містити відомості про ті правила, які користувачу не можна порушувати. При цьому СКБД повинна стежити за дотриманням заданих правил при виконанні користувачем деяких операцій. Відмінністю обмежень цілісності від правил безпеки є те, що перші не залежать від користувача.

6.4.1 Обмеження цілісності

Роздивимося завдання обмежень цілісності на деякій гіпотетичній мові.

CREATE INTEGRITY RULE правило_4

FORALL Товар (товар.вага > 0)

ON ATTEMPTED VIOLATION REJECT;

У загальному випадку обмеження цілісності містять три компоненти:

1. Обмеження цілісності реєструються в системному каталозі під деяким ім'ям (у даному прикладі *правило_4*). Дане ім'я також використовується в будь-яких діагностичних повідомленнях системи у відповідь на спроби порушення цього правила.
2. Обмеження цілісності, що обов'язково повинні задовольнятися, задаються за допомогою істинного виразу, тобто обмеження цілісності задовольняється тоді та тільки тоді, коли цей вираз істинний, а порушується тоді та тільки тоді, коли цей вираз хибний. У даному прикладі в обмеженнях цілісності за допомогою квантора спільності «**FORALL** (For All) **Товар**» указано, що у відношенні **Товар** усі товари повинні мати вагу більше нуля.
3. Реакція на порушення, яка задана за допомогою речення **ON ATTEMPTED VIOLATION**, указує системі ті дії, які необхідно виконати при спробі користувача порушити обмеження цілісності. У даному прикладі це відмова від виконання оновлення, яке порушує цілісність. Реакція такого типу найбільш часто застосовується на практиці, тому її варто задавати по умовчання. Проте в більш загальному випадку реакція на порушення може бути процедурою довільної складності.

При виконанні речення **CREATE INTEGRITY RULE** система повинна спочатку перевірити, чи задовольняє поточний стан бази даних заданому обмеженню. Якщо це не так, то нове обмеження цілісності буде заборонено, у протилежному випадку воно буде прийняте (тобто збережено в каталозі) і з цього моменту буде застосовуватися далі.

На практиці обмеження цілісності майже завжди починається з квантора спільності, а у випадку, коли він відсутній, його наявність передбачається по умовчання. Відповідно квантор спільності може бути опущений. Приймаючи це допущення, а також прийняту по умовчання реакцію на порушення обмеження, вихідний запис аналізованого прикладу можна значно спростити:

CREATE INTEGRITY RULE правило_4 Товар.вага > 0

Усувається таке обмеження за допомогою правила:

DESTROY INTEGRITY RULE правило_4.

6.4.2 Типи обмежень цілісності

Обмеження цілісності зручно класифікувати за чотирма категоріями:

1. **Обмеження цілісності домену** визначають множину значень, що утворюють цей домен, тобто просто перераховуються всі припустимі значення домену. Тому в такому випадку немає необхідності в створенні окремого твердження типу **CREATE DOMAIN INTEGRITY RULE**, замість цього його можна зазначити у відповідному завданні домену **CREATE DOMAIN**, наприклад:

CREATE DOMAIN Колір **CHAR** (15)

FORALL Колір

(Колір = «блакитний» **OR**

Колір = «рожевий» **OR**

Колір = «чорний» **OR** Колір = «зелений»).

Наступний запис еквівалентний попередньому:

CREATE DOMAIN Колір **CHAR** (15)

VALUES («блакитний», «рожевий», «чорний», «зелений»).

Усунути обмеження цілісності домену можна тільки за рахунок усунення самого домену за допомогою речення **DESTROY DOMAIN**.

2. **Обмеження цілісності атрибута** задаються у вигляді окремої частини визначення цього атрибута при створенні відношення. Ці обмеження завжди перевіряються негайно, тобто будь-яка спроба (за допомогою операцій вставки **INSERT** або оновлення **UPDATE**) ввести до БД некоректне значення атрибута буде заборонена.
3. **Обмеження цілісності відношення** є правилом, що задається тільки для одного якогось відношення. При цьому вони також перевіряються негайно. Наприклад, можна задати, що поставальники зі Львову мають статус 20:

CREATE INTEGRITY RULE правило_5

FORALL Поставальник (**IF** Поставальник.місто= «Львів»

THEN Поставальник. статус =20)

ON ATTEMPTED VIOLATION REJECT;

Обмеження цілісності бази даних є правилом, що задається для двох і більш пов'язаних між собою відношень. Наприклад,

обмеження, що означає, що постачальники зі статусом менше 20 не можуть поставляти товари в кількості більш 500:

CREATE INTEGRITY RULE правило_6
FORALL Постачальник (**FORALL** Постачання
(**IF** Постачальник.статус <20 **AND** Постачальник. Код_
постачальника = Постачання. Код_постачальника
THEN Постачання.Кількість ≤ 500))
[ON ATTEMPTED VIOLATION REJECT];

або без кванторів спільності:

CREATE INTEGRITY RULE правило_6
IF Постачальник.статус <20 **AND** Постачальник.Код_
постачальника = Постачання. Код_постачальника
THEN Постачання.Кількість ≤ 500
[ON ATTEMPTED VIOLATION REJECT];

Приведення в дію обмежень бази даних, на відміну від інших обмежень, може бути відкладене до кінця транзакції.

Крім того, варто мати на увазі, що обмеження цілісності, що пов'язані з зовнішніми ключами, хоча і задаються, як правило, при створенні відношення, стосуються всієї бази даних.

6.4.3 Обмеження стану та переходу

У приведених раніше прикладах були надані обмеження стану, які співвіднесені з тими або іншими коректними станами бази даних. Проте іноді буває необхідно роздивитися *переходи* від одного стану до іншого. Наприклад, у БД персоналу для опрацювання припустимих змін повинні бути задані обмеження переходу в даних про родинний стан. До таких *припустимих* змін можна віднести наступні:

- ніколи не був одруженим – є одруженим;
- є одруженим – овдовів;
- є одруженим – розведений;
- розведений – є одруженим.

А серед неприпустимих змін можна назвати такі:

- ніколи не був одруженим – овдовів;
- ніколи не був одруженим – розведений;
- вдовий – розведений;

– розведений – овдовів.

У базі даних постачальників і товарів можна навести приклад обмеження, яке означає, що «*статус постачальника не повинний змєниуватися*» (при цьому **Постачальник** і **Постачальник1** означають одне й те ж відношення до та після операції оновлення, відповідно):

CREATE INTEGRITY RULE правило_7

FORALL Постачальник **FORALL** Постачальник 1

(**IF** Постачальник.Код_постачальника=

Постачальник1. Код_постачальника

THEN Постачальник.Статус ≤ Постачальник 1.Статус)

[ON ATTEMPTED VIOLATION REJECT];

Або, використовуючи неявне завдання кванторів спільності:

CREATE INTEGRITY RULE правило_8

IF Постачальник.Код_постачальника = Постачальник1.

Код_постачальника

THEN Постачальник.Статус ≤ Постачальник 1.Статус

[ON ATTEMPTED VIOLATION REJECT];

Оскільки це обмеження переходу задано для відношення, воно перевіряється *негайно*.



Питання для самоперевірки

1. Що таке системний збій?
2. Що таке транзакція, які властивості притаманні транзакціям?
3. Як виконується відновлення системи після збою?
4. Що означає термін паралелізм?
5. Що таке блокування та які види блокувань використовуються?
6. Яким чином підтримується безпека баз даних?
7. Що називається привілеями?
8. Що таке виборчий та обов'язковий підходи забезпечення безпеки?
9. З якою ціллю використовується контрольний слід виконання операцій?
10. Які види обмежень цілісності існують?

ЛІТЕРАТУРА

1. Автоматизированные информационные технологии в экономике : Учебник / ВЗФЭИ; Под ред. Г.А.Титоренко. – М. : ЮНИТИ. – 2005, 345 с.
2. Арсеньев Б. П. Интеграция распределенных баз данных / Б. П. Арсеньев, С. А. Яковлев. – СПб. : Изд. «Лань», 2001. – 286 с.
3. Архипенко С. Я. Хранилища данных. От концепции до внедрения / С. Я. Архипенко, Д. С. Голубев, О. Р. Максименко. – М. : ДИАЛОГ-МИФИ, 2006. – 528 с.
4. Бейли Д. Изучаем SQL / Д. Бейли. – СПб. : Питер, 2011. – 592 с.
5. Боуман Д. Практическое руководство по SQL / Д. Боуман, С. Эмерсон, М. Дарновски. – М., СПб. : Вильямс, 2002. – 352 с.
6. Васвани Д. MySQL: Использование и администрирование / Д. Васвани. – СПб. : Питер, 2011. – 368 с.
7. Гайдамакин Н. А. Автоматизированные информационные системы, базы и банки данных / Н. А. Гайдамакин. – М. : Гелиос, 2006. – 368 с.
8. Голенищев С. П. Информационное обеспечение систем управления. учеб. пособ. / С. П. Голенищев. – Ростов / Дон: Феникс, 2010. – 315 с.
9. Грабер М. Введение в SQL / М. Грабер. – М. : Лори, 2008. – 303 с.
10. Дейт К. Введение в системы баз данных / К. Дейт. – М., СПб. : Вильямс, 2008. – 781 с.
11. Джеймс Р. SQL. Полное руководство, 2-изд / Р. Джеймс, П. Вайнберг. – М., СПб. : BHV, 2001. – 632 с.
12. Информационные технологии в проектировании радио-электронных средств: учеб. пособ. для студ. высш. учебн. заведений / Ю. Л. Муромцев., Д. Ю. Муромцев, И. В. Тюрин и др – М. : Издательский центр «Академия», 2010. – 384 с.
13. Карпова Т. С. Базы данных: модели, разработка, реализация. – СПб. : Питер, 2001. – 304 с.
14. Клайн К. SQL: справочник / К. Клайн, Д. Клайн, Бр. Хант. – М.: КУДИЦ-Образ, 2006. – 560 с.
15. Конноли Т., Бегг К., Страчан А. Базы данных: проектирование, реализация и сопровождение. Теория и практика: Пер. с англ. – М. : Издательский дом «Вильямс», 2003. – 1120 с.
16. Корнієнко С. К. Системи баз даних: організація та проектування:

- навч. посіб. / С.К Корнієнко. – Запоріжжя: ЗНТУ, 2006. – 252 с.
17. Кренке Д. Теория и практика построения баз данных : [пер.с англ] / Д. Кренке. – 9-е изд. – СПб. : Питер, 2005. – 858 с.
 18. Кригель П. SQL. Библия пользователя / П. Кригель. – К. : Диалектика / Вильямс, 2010. – 752 с.
 19. Кузнецов С. Д. Базы данных : Языки и модели / М. : Бином, 2008. – 720 с.
 20. Малюх В. Н. Введение в современные САПР: Курс лекций / В.Н. Малюх – М. : ДМК Пресс, 2010. – 192 с.
 21. Нестеров А. Л. Проектирование АСУТП / А. Л. Нестеров. – М.: ДЕАН, 2010. – 349 с.
 22. Норенков И. П. Основы автоматизированного проектирования: учеб. для вузов – 4-е изд., перераб. и доп. / И. П. Норенков – М. : Изд-во МГТУ им. Н. Э. Баумана, 2009. – 430 с.
 23. Пасічник, В. В. Організація баз даних та знань / В. В. Пасічник, В. А. Резніченко. – К. : BNV, 2006. – 384 с.
 24. Петров В. Н. Информационные системы / В. Н. Петров. – СПб. : Питер, 2002, 322 с.
 25. Райордан Р. М. Основы реляционных баз данных / Р. М. Райордан. – М. : Русская Редакция, 2001. – 384 с.
 26. Рудикова А.П. Базы данных. Разработка приложений / А.П. Рудикова. – СПб. : BNV, 2006. – 496 с.
 27. Сиротинська А. Р. Інформаційні системи підприємств малого бізнесу. Навч. посіб. / А. Р. Сиротинська. – К. : ЦУЛ, 2008, 264 с.
 28. Советов Б. Я. Информационные технологии. / Б. Я. Советов, В. В. Цехановский. – М. : Высшая школа, 2007, 302 с.
 29. Советов Б. Я. Базы данных : теория и практика / Б. Я. Советов, В. В. Цехановский, В. Д. Чертовской. – М. : Высшая школа, 2005. – 352 с.
 30. Сховища даних / С. М. Капустін, О. О. Зайцев, В. П. Юрченко, Р. Л. Максименко та ін.; під ред. С. М. Капустіна. – К. : BNV, 2007. – 410 с.
 31. Федечкин С. А. Хранилище данных: вопросы и ответы / А. С. Федечкин // PCWeek. – 2003. – № 31. – С. 26–32.
 32. Харрингтон Д. Л. Проектирование реляционных баз данных / Д. Л. Харрингтон. – М. : Лори, 2006. – 313 с.
 33. Хомоненко А. Д., Базы данных : Учебник для высших учебных заведений / А. Д. Хомоненко, В. М. Цыганков, М. Г. Мальцев. – СПб. : КОРОНА, 2004. – 416с.

34. Чекалов А. Б. Базы данных от проектирования до разработки приложений / А. Б. Чекалов. – К. : ВHV, 2003. – 384 с.
35. Швецов А. Н., Распределенные интеллектуальные информационные системы / А. Н. Швецов, С. А. Яковлев. – СПб. : Изд-во СПбГЭТУ «ЛЭТИ», 2003, 267 с.
36. Akmal M. Relational Database Design for Starters / M. Akmal // Lulu.com, January 2008, p. 234–242.
37. Cindy J. M. Database Design and Development / J. M. Cindy/ – Amsterdam: John Wiley & Sons Inc., 2009. – 532 p.
38. Gray J. Transaction Processing: Concepts and Techniques / J. Gray, A. Reuter. – New York: Morgan Kaufmann Publishers, 2009. – 512 p.
39. Kroenke D. Database Concepts. 3rd ed. / D. Kroenke, J. David. – New York: Prentice, 2007. – 456 p.
40. Lightstone S. Physical Database Design : the database professional's guide to exploiting indexes, views, storage, and more / S. Lightstone, T. Teorey, T. Nadeau. – London–New York: Morgan Kaufmann Press, 2007. – 512 p.
41. Peter R. Database Systems-Design, Implementation, and Management / R Peter // Course Technology PTR, January 2012. – p. 341–349.
42. Teorey T. Database Modeling & Design: Logical Design. 4th edition / T. Teorey, S. Lightstone, T. Nadeau. – London–New York: Morgan Kaufmann Press, 2005. – 492 p.



ТЕРМІНОЛОГІЧНИЙ СЛОВНИК

Автоматизована система – система, що складається з персоналу, комплексу засобів автоматизації його діяльності й регламентів роботи, та реалізує інформаційну технологію виконання встановлених функцій.

Агрегація – метод об'єднання локальних подань, який дозволяє розглядати зв'язок між елементами моделі як новий елемент більш високого рівня.

Адміністратор бази даних – особа, що відповідає за виконання функцій адміністрування бази даних, тобто координацію дій по збору відомостей, проектуванню та експлуатації бази даних, а також по забезпеченню захисту даних.

Адміністратор (диспетчер) транзакцій – системний компонент, який забезпечує атомарність транзакцій.

Адміністрування БД – це виконання функцій визначення, організації, керування та захисту даних у базі даних.

Альтернативний ключ – атрибут (або група атрибутів), які не співпадають із первинним ключем і унікально ідентифікують кожний рядок у таблиці.

Атрибут – інформаційне відображення властивостей об'єкту.

База даних – поійменована структурована сукупність взаємозв'язаних даних, які описують певну предметну область і використовуються багатьма користувачами.

Безпека бази даних – захист від несанкціонованого доступу, зміни або руйнування даних.

Бізнес-правила – спеціалізований вид логіки, що описує обмеження на образ дій, які система або люди повинні враховувати у своїй поведінці. Ці правила визначаються цілим рядом факторів, включаючи директиви розпорядчих органів, промислові стандарти, ділову хватку й простий здоровий глузд.

Блокування (locking) – механізм захисту даних, який після звернення до даних забороняє їх оновлення іншим транзакціям.

Відновлення – надання можливості системі повернути базу даних у нормальний стан після збою програмного або апаратного забезпечення.

Відношення – підчисленність декартового добутку доменів.

Вітрини даних (data mart – DM) – невеличкі сховища зі спрощеною архітектурою, призначені для зберігання невеликої підмножини даних і зняття навантаження з основного інформаційного сховища підприємства.

Вторинний ключ (ключ пошуку) – це атрибут або сукупність атрибутів, яка ідентифікує не унікальний об'єкт у наборі, а всі об'єкти, що мають визначені значення цих атрибутів.

Домен – набір значень елементів даних одного типу, який відповідає поставленим умовам.

Екземпляр сутності – опис конкретного об'єкту в наборі.

Ергономічне забезпечення – сукупність взаємозалежних вимог, спрямованих на узгодження психологічних, психофізіологічних, антропометричних, фізіологічних характеристик і можливостей людини-оператора, технічних характеристик КЗА, параметрів робочого середовища на робочім місці.

Збережені процедури – процедури й функції, що зберігаються безпосередньо в базі даних у відкомпільованому вигляді та можуть запускатися користувачами або додатками, що працюють з базою даних.

Зв'язок – функціональна залежність між сутностями.

Зв'язне відношення – відношення, що зберігає ключі двох або більше об'єктних відношень, за якими встановлюється зв'язок між цими відношеннями.

Зовнішнє подання – це вміст БД, яким бачить його окремий користувач (тобто для користувача зовнішнє подання саме й є база даних).

Ідентифікатор (ключ) – один або декілька атрибутів, значення яких дозволяють беззаперечно відрізнити один екземпляр сутності від іншого.

Інформаційне забезпечення – сукупність єдиної системи класифікації й кодування інформації, уніфікованих систем документації, схем інформаційних потоків, що циркулюють в організації, методологія побудови баз даних.

Ключовий елемент даних – елемент, по якому можна визначити значення інших елементів даних.

Комплекс засобів автоматизації – сукупність усіх компонентів автоматизованої системи, за винятком користувачів і експлуатаційного персоналу.

Контрольна точка – певним чином визначений момент часу, коли система здійснює фізичний запис вмісту робочих буферів бази даних безпосередньо до фізичної бази даних і фіксацію списку виконуваних у даний момент транзакцій.

Концептуальне подання даних – сукупність усіх вимог щодо даних, отриманих із зовнішніх представлень користувача, являється загальною інформаційною моделлю ПО.

Концептуальне проектування – отримання семантичних (змістових) моделей, які відображають інформаційний зміст конкретної предметної області.

Кортеж – упорядкований набір взаємозв'язаних величин (даних).

Лінгвістичне забезпечення – сукупність мовних засобів для формалізації природної мови, побудови й комбінації інформаційних одиниць, які використовуються в автоматизованій системі при функціонуванні системи для спілкування з комплексом засобів автоматизації.

Математичне забезпечення – сукупність математичних методів, моделей та алгоритмів для реалізації цілей і завдань автоматизованої системи, а також нормального функціонування комплексу технічних засобів.

Метадані – логічна структура БД у формальному вигляді.

Міра відношення – кількість його атрибутів або стовпців.

Мова маніпулювання даними (Data Manipulation Language – DML), або мова запитів до БД, є засобом, що використовується прикладними програмами або користувачами для виконання операцій над БД.

Мова опису даних (Data Definition Language – DDL) – це мова високого рівня декларативного (описового) типу, що призначена для створення та модифікації схеми бази даних

Модель даних – сукупність правил формування структури даних в базах даних, операцій над ними, а також обмежень цілісності, що визначають припустимі зв'язки та значення даних.

Нормалізація відношень – покроковий зворотній процес побудови оптимальної структури таблиць у реляційній базі даних шляхом декомпозиції складних таблиць на більш прості.

Об'єкт – елемент інформаційної системи, інформація про який зберігається.

Об'єктне відношення – відношення, що зберігає дані про об'єкти (екземпляри сутності) предметної області.

Організаційно-методичне забезпечення – сукупність методів і засобів, що регламентують взаємодію працівників з технічними засобами й між собою в процесі розробки та експлуатації АС

Паралелізм – можливість одночасного опрацювання в СКБД багатьох транзакцій, які звертаються до одних і тих же даних.

Первинний ключ – атрибут (або група атрибутів), які однозначно ідентифікують кожний рядок у таблиці.

Подання – об'єкт, який не містить власних даних. Це свого роду віртуальна таблиця, що не існує як незалежний об'єкт в базі даних і вміст якої береться з інших таблиць шляхом виконання запиту.

Посилальна цілісність – забезпечення співвідношення значень зовнішнього ключа екземпляра дочірній сутності значенням первинного ключа в батьківській сутності.

Потенційний (альтернативний) ключ – ключ, який претендує на роль первинного та відповідає всім вимогам до первинних ключів.

Потужність відношення – число всіх кортежів, які утворюють відношення.

Предметна область – сукупність об'єктів, предметів реального світу, що розглядаються у контексті задачі, що вирішується.

Привілеї (повноваження) – права, що надаються конкретному користувачу при роботі з певними даними.

Програмне забезпечення – сукупність програм системи обробки інформації і програмних документів, необхідних для експлуатації цих програм.

Реляційна алгебра – система операцій маніпулювання відношеннями, кожний оператор якої в якості операндів має одно чи більше відношень.

Реляційна база даних – набір екземплярів кінцевих відношень.

Реплікація – підтримка актуальних копій деякого фрагмента розподіленої БД на декількох вузлах мережі.

Розподілена база даних – сукупність взаємопов’язаних баз даних, які знаходяться в різних вузлах комп’ютерної мережі.

Розподілена СКБД – програмний комплекс, призначений для керування розподіленими базами даних і який дозволяє зробити розподіленість системи прозорою для користувача.

Система баз даних – це інформаційна система, що містить у собі комплекс спеціальних засобів для підтримки динамічної моделі предметної області та забезпечення зберігання, пошуку та видавання інформації у вигляді, зручному користувачам.

Системний журнал транзакцій (Transaction log) – спеціальний файл, у якому записуються деталі всіх операцій оновлення, зокрема, нове та старе значення модифікованого об’єкта.

Системний збій – збій, який впливає на всі транзакції, що виконуються в даний момент, але не завдає фізичної шкоди самій базі даних.

Системний каталог (словник даних) – централізоване сховище відомостей про об’єкти, елементи даних, які їх складають, взаємозв’язки між об’єктами, їх джерела, значення, використання та формати подання, тобто метадані.

Ступінь відношення – кількість атрибутів (стовпців) відношення.

Сутність – це збиральне поняття, деяка абстракція реально існуючого об’єкта навколишнього світу, процесу або явища.

Таблиця (відношення) – деяка регулярна структура, яка складається з кінцевого набору однотипних записів.

Технічне забезпечення – комплекс технічних засобів, які забезпечують роботу автоматизованої системи, документація на ці засоби та технологічні процеси.

Тип сутності – набір однорідних об'єктів, які виступають як єдине ціле.

Топологія бази даних (структура розподіленої бази даних) – схема розподілення фізичної БД по мережі.

Транзакція – логічна одиниця роботи, що виконується без порушення логічної цілісності бази даних. Якщо в процесі виконання транзакції виникла помилка виконання, то система, що підтримує процес транзакції, гарантує повернення до первинного стану.

Тригер – попередньо визначена дія або послідовність дій, які автоматично здійснюються під час виконання операцій оновлення, додавання або вилучання даних.

Узагальнення – абстракція даних, яка дозволяє трактувати клас різних подібних об'єктів як один пойменований узагальнений тип об'єкта.

Шлюз – сукупність програмних засобів, які використовуються у гетерогенних розподілених СКБД і призначені для перетворення мови та моделі даних кожного типу СКБД, яка використовується, в мову та модель даних реляційної системи.

OLE (Object Linking and Embedding) – зв'язування та впровадження об'єктів – технологія, що дозволяє використовувати в додатку об'єкти, розроблені в іншому додатку. OLE-об'єктами можуть бути звуки, рисунки, діаграми, відеокліпи тощо.

SQL (Structured Query Language) – мова структурованих запитів, офіційний стандарт мови для роботи з реляційними базами даних.

ПРЕДМЕТНИЙ ПОКАЖЧИК

A

ALL, 140, 141, 153

ALTER TABLE, 132

ANSI, 127

ANY, 153, 154

API, 40

Application, 29, 64

B

BEGIN TRANSACTION, 182

BETWEEN, 141, 143

C

CHECK, 183, 184

COMMIT, 132, 138, 194

CREATE TABLE, 138, 139

CREATE VIEW, 125, 138, 179

CREATE INDEX, 121, 138, 145

D

DATA BASE ENGINE, 24

DDB, 45

DDL, 26, 126

DELETE, 127, 132, 138

DISTINCT, 162, 164

DML, 27, 127

DROP, 138, 176

E

ER-модель, 71

EXISTS, 162, 163, 167, 172

Запит

корельований, 169

некорельований, 163

Захист інформації, 189

Зв'язок, 71, 72

Значення

атомарне, 109

NULL, 139

Зовнішній ключ, 94, 144

I

Індексація, 121, 145

Інтерфейс користувача, 128

Інформаційне забезпечення, 7, 17, 20

K

Каталог, 20, 23

Клієнт-сервер, 39

Ключ

альтернативний, 101

вторинний, 93

зовнішній, 94, 129, 144

первинний, 72

складовий, 72

Контрольна точка, 192

Концептуальна модель, 71

Концептуальне проектування, 71

Концептуальна модель, 71

Корельовані підзапити в SQL, 161

Курсор, 124, 179

F

FROM, 176, 186

G

GRANT 186, 200

GROUP BY, 147, 156

H

HAVING, 155, 156

I

IN, 142

INDEX, 121, 145

INSERT, 127, 132, 138

J

JOIN, 51, 52, 98

L

LIKE, 142, 148

LOCK, 195

N

NOT EXIST, 162, 163, 167, 172

NULL, 139

O

ODBC, 41

OLE, 135

ON, 121, 145

R

ROLLBACK, 132, 138

REVOKE, 138, 184, 185

S

SELECT, 124, 138, 146, 187

SQL, 27, 42, 132

S-блокування, 195

Л

Локальна транзакція, 60

Локальне подання, 75

М

Метадані, 22, 30

Менеджер

драйверів, 42

завантаження, 68

Мова

опису даних, 26

маніпулювання даними, 27

Модель

даних, 54, 71, 91

концептуальна, 71

логічна, 91

реляційна, 91

фізична, 71

Н

Надлишковість даних, 77, 107

Некорельовані підзапити в SQL, 163

Нормалізація, 107, 108

Нормальні форми, 108

О

Об'єднання, 48, 51, 75, 83, 174

П

Перетин, 95

Підзапит

корельований, 169

некорельований, 163

Предметна область, 71, 74

T

TRANSACTION, 138, 194

TRANSACTION LOG, 191

U

UNION, 51, 174, 183

UNIQUE, 121, 140, 141

UPDATE, 28, 104, 132, 138

V

VIEW, 179

W

WHERE, 125, 147

X

X-блокування, 195

A

Агрегація, 84, 90

Адміністратор бази даних, 184

Аномалії оновлення, 104

Атомарність

даних, 109

транзакцій, 190

Атрибут, 48, 50, 71

Б

Багатокористувацька СКБД, 36

База даних

розподілена, 44

реляційна, 91, 93

Безпека, 197

Бізнес-правила, 99, 100

Бізнес-логіка, 99, 127

Блокування, 195

Привілеї, 138, 184, 198

Прикладна програма, 30, 32, 41

Подання, 76, 124, 138, 179

Посилальна цілісність, 94

Проектування бази даних, 67, 69, 107

Проекція, 97, 124

Прозорість

виконання, 55, 58

розподіленості, 55

транзакції, 55

Р

Реляційна модель даних, 91

Реляційна алгебра, 95

Реплікація, 65, 67

Розподілена БД, 45

Розподілена СКБД, 45

гомогенні, 53

гетерогенні, 53

С

Сервер, 37

Система баз даних, 39

Словник даних, 20, 30

СКБД

архітектура, 24

розподілені, 53

функціонування, 32

Сутність, 71

Схема бази даних, 27

Схеми інформаційних потоків, 18

В

Відновлення

бази даних, 26, 34, 189

системи, 26, 192

транзакції, 54

Відношення, 92, 93

Вторинний ключ, 93

Г

Гетерогенні РСКБД, 53

Глобальна транзакція, 60

Гомогенні РСКБД, 53

Д

Двофазна фіксація

транзакцій, 64

Домен, 92, 198

Е

Екземпляр сутності, 71

Ж

Життєвий цикл бази даних, 67

Журнал транзакцій, 191

З

Залежність

багатозначна, 106, 119

транзитивна, 106, 111

функціональна, 105, 107

Т

Таблиця, 29, 38

Технологія

клієнт-сервер, 39

ODBC, 41

Типи даних, 134

Тип сутності, 71

Тиразування, 50, 56, 60

Транзакція, 60, 64, 66, 127, 138,

Транзитивна залежність, 106

У

Узагальнення, 84

Ф

Файл-сервер, 38

Фрагментація

вертикальна, 51

горизонтальна, 50

даних, 48

Ц

Цілісність, 94, 138, 197, 205

Ш

Шлюз, 44, 54

Навчальне видання

КОРНІЄНКО Сергій Костянтинович

**ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОГО
ЗАБЕЗПЕЧЕННЯ АВТОМАТИЗОВАНИХ
СИСТЕМ**

Навчальний посібник

Дизайн і макет обкладинки *Миронова Н.О.*
Верстання *Гринь Д. В.*

Оригінал-макет підготовлено
в редакційно-видавничому відділі ЗНТУ

Підписано до друку 29.09.2014. Формат 60×84/16. Ум. друк. арк. 12,25.
Тираж 300 прим. Зам. № 545.

Запорізький національний технічний університет
Україна, 69063, м. Запоріжжя, вул. Жуковського, 64
Тел.: (061) 769–82–96, 220–12–14

Свідцтво суб'єкта видавничої справи ДК № 2394 від 27.12.2005.