**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**

**«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»**
**Факультет інформатики та обчислювальної техніки**

# БАЗИ ДАНИХ

**Курс лекцій кредитного модуля**
для студентів спеціальності
126 “Інформаційні системи та технології”
кафедри технічної кібернетики
всіх форм навчання

*Рекомендовано*
*Вченою радою факультету*
*інформатики та обчислювальної*
*техніки НТУУ «КПІ ім.І.Сікорського»*
*Протокол № ___ від __.__.____р.*

Київ - 2020

Бази даних. Курс лекцій. [Текст] / Уклад. К.Б.Остапченко.- К.: НТУУ-“КПІ ім.І.Сікорського”, 2020. - 150с.

Курс лекцій призначений для студентів спеціальності 126 «Інформаційні системи та технології» кафедри технічної кібернетики всіх форм навчання. В курсі наведена структура і тематика розділів лекційного матеріалу, теоретичні відомості, список рекомендованої літератури.

Укладач К.Б. Остапченко, к.т.н., доцент

*За редакцією укладача*

## Вступ

Кредитний модуль «**Бази даних**» (БД) є базовою дисципліною навчального плану підготовки фахівців із спеціальності 126 «Інформаційні системи та технології» і належить до циклу загальної підготовки бакалаврів освітньої програми ОПП «Інформаційне забезпечення робототехнічних систем».

Курс кредитного модуля призначений забезпечити підготовку фахівців у галузі проектування і застосування автоматизованих інформаційних систем, які входять до складу гнучких комп'ютеризованих систем, систем управління робототехнічними комплексами.

Цей курс базується на таких забезпечуючих дисциплінах: «Теорія алгоритмів», «Дискретна математика», «Комп‘ютерні мережі».

Цей курс забезпечує засвоєння студентами наступних дисциплін бакалаврату: «Проектування інформаційних систем», «Архітектура комп‘ютерних систем», «Роботизовані інтерактивні інфраструктури».

*Метою кредитного модуля* є формування у студентів базових уявлень, первинних знань, компетенцій та вмінь з основ розробки баз даних, виконання завдань з їх адміністрування, дослідження функціональних можливостей систем управління базами даних, а також засобів моделювання та проектування баз даних, достатніх для реалізації задач з розробки, впровадження та експлуатації інформаційних систем різного призначення.

У кредитному модулі вивчаються:

- базові поняття, терміни та визначення елементів баз даних, призначення та структура системи управління базами даних (СУБД), роль та місце баз даних в інформаційних системах;
- рівні подання даних, структури даних та методи їх моделювання задля формалізованого подання предметної області інформаційних систем;
- типи моделей організації даних в СУБД, їх суть, відмінності та можливе застосування;
- математичні засади побудови сучасних реляційних баз даних - реляційна алгебра та числення, нормалізовані методи проектування реляційних баз даних;
- конструкції мови структурованих запитів SQL взаємодії з реляційними базами даних, засоби їх захисту, методи забезпечення цілісності та збереження даних;
- подання внутрішніх структур даних СУБД в ЕОМ, сучасні тенденції побудови промислових СУБД.

Загальний обсяг підготовки складає 120 годин/4 кредити ECTS. Вид семестрового контролю – екзамен/письмовий.

### 1. Структура кредитного модуля

Структурно кредитний модуль БД складається з (рис.1): лекцій, комп‘ютерних практикумів, модульного контролю.

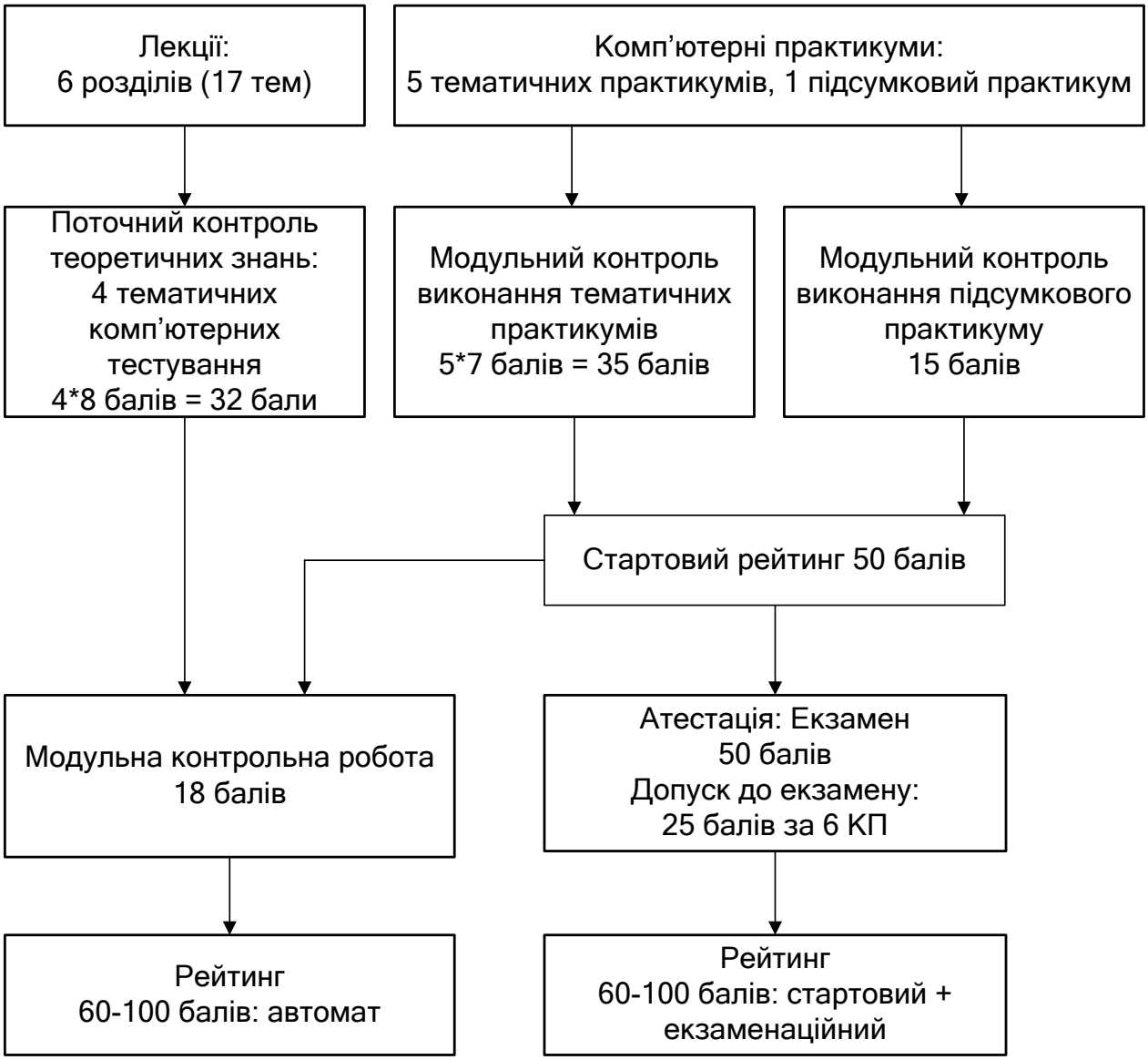



Рис. 1. Структура кредитного модуля

## 2. Зміст лекцій

| | |
|---|---|
| Розділ 1. Базові засади побудови та застосування систем баз даних | |
| 1 | Тема 1. Концепція систем баз даних та автоматизованих інформаційних систем<br>Питання 1. Визначення систем баз даних<br>Питання 2. Зв'язок програм і даних при використанні СБД<br>Питання 3. Відмінності баз даних від файлових систем<br>Питання 4. Етапи розвитку СУБД та їх інтеграції з АІС<br>Завдання на СРС: підготуватися до надання відповідей на контрольні запитання комп'ютерного тестування 1 |
| 2 | Тема 2. Архітектура автоматизованих інформаційних систем та систем управління базами даних<br>Питання 1. Склад і класифікація СУБД<br>Питання 2. Архітектура БД: рівні представлення даних, функції, компоненти<br>Питання 3. Функціональна схема побудови СУБД<br>Завдання на СРС: підготуватися до надання відповідей на контрольні запитання комп'ютерного тестування 1 |
| 3 | Тема 3. Особливості організації даних у сучасних інформаційних системах<br>Питання 1. Сучасна технологія обробки інформації «клієнт-сервер»<br>Питання 2. Сучасні напрямки досліджень в організації СУБД<br>Питання 3. Класи АІС : OLAP і OLTP-системи, призначення й структура<br>Завдання на СРС: підготуватися до надання відповідей на контрольні запитання комп'ютерного тестування 1 |
| | Тематичне комп'ютерне тестування 1 - (8 балів) |
| Розділ 2. Моделювання предметної області автоматизованих інформаційних систем | |
| 4 | Тема 1. Подання даних в автоматизованих інформаційних системах<br>Питання 1. Поняття про предметну область АІС: модель представлення даних в АІС, інформаційні і функціональні частини АІС<br>Питання 2. Класи моделей представлення даних<br>Питання 3. Концепція й етапи семантичного моделювання<br>Питання 4. Типи діаграм представлення понять семантичної моделі<br>Завдання на СРС: підготуватися до надання відповідей на контрольні запитання комп'ютерного тестування 2 |
| 5 | Тема 2. Проектування структури бази даних за допомогою семантичного моделювання<br>Питання 1. Модель представлення даних типу “ER-модель”<br>Питання 2. Проектування ER-моделі<br>Питання 3. Модифікації ER-моделі<br>Питання 4. Інформаційні компоненти бази даних<br>Питання 5. Процедури перетворення ER-моделі у компоненти БД<br>Завдання на СРС: підготуватися до надання відповідей на контрольні запитання комп'ютерного тестування 2 |
| | Тематичне комп'ютерне тестування 2 - (8 балів) |
| Розділ 3. Організація баз даних | |
| 6 | Тема 1. Моделі організації даних<br>Питання 1. Загальна класифікація логічних моделей структур даних<br>Питання 2. Порівняльна характеристика моделей організації даних: ієрархічної, мережної, реляційної, постреляційної, багатомірної, об'єктної<br>Завдання на СРС: підготуватися до надання відповідей на контрольні запитання комп'ютерного тестування 3 |
| 7 | Тема 2. Реляційна модель даних<br>Питання 1. Базові елементи і поняття РМ |

| | |
|---|---|
| | Питання 2. Структурна частина РМ<br>Питання 3. Цілісна частина РМ<br>Питання 4. Маніпуляційна частина РМ<br>Завдання на СРС: підготуватися до надання відповідей на контрольні запитання комп‘ютерного тестування 3 |
| 8 | Тема 3. Реляційна алгебра<br>Питання 1. Базові елементи і поняття РА<br>Питання 2. Теоретико-множинні операції РА<br>Питання 3. Спеціальні реляційні операції РА<br>Питання 4. Допоміжні операції РА<br>Завдання на СРС: підготуватися до надання відповідей на контрольні запитання комп‘ютерного тестування 3 |
| 9 | Тема 4. Реляційне числення<br>Питання 1. Базові поняття РЧ<br>Питання 2. Побудова запису виразу РЧ<br>Завдання на СРС: підготуватися до надання відповідей на контрольні запитання комп‘ютерного тестування 3 |
| 10 | Тема 5. Методи проектування реляційної бази даних<br>Питання 1. Проблеми проектування реляційної бази даних<br>Питання 2. Метод нормалізації реляційної бази даних<br>Питання 3. Нормальні форми відношень реляційної бази даних<br>Завдання на СРС: підготуватися до надання відповідей на контрольні запитання комп‘ютерного тестування 3 |
| | Тематичне комп‘ютерне тестування 3 - (8 балів) |
| Розділ 4. Мови реляційної бази даних | |
| 11 | Тема 1. Мовні засоби визначення даних у реляційній базі даних<br>Питання 1. Призначення, основні функції і стандартизація мов РБД<br>Питання 2. Опис даних і їхніх типів у РБД<br>Питання 3. Оператори визначення таблиць, представлень і індексів даних<br>Питання 4. Опис цілісності даних у РБД<br>Завдання на СРС: підготуватися до надання відповідей на контрольні запитання комп‘ютерного тестування 4 |
| 12 | Тема 2. Мовні засоби маніпулювання даними у реляційній базі даних<br>Питання 1. Оператори маніпулювання даними<br>Питання 2. Форми оператора організації запитів даних<br>Питання 3. Оператори керування транзакціями<br>Завдання на СРС: підготуватися до надання відповідей на контрольні запитання комп‘ютерного тестування 4 |
| 13 | Тема 3. Мовні засоби управління даними у реляційній базі даних<br>Питання 1. Підхід до організації безпеки у РБД<br>Питання 2. Оператори системного рівня надання привілеїв<br>Питання 3. Оператори об'єктного рівня надання привілеїв<br>Завдання на СРС: підготуватися до надання відповідей на контрольні запитання комп‘ютерного тестування 4 |
| Розділ 5. Внутрішня структура реляційної бази даних на ЕОМ | |
| 14 | Тема 1. Організація зовнішньої пам‘яті елементів реляційної бази даних<br>Питання 1. Підходи організації зовнішньої пам'яті РБД<br>Питання 2. Різновиду об'єктів зовнішньої пам'яті РБД<br>Питання 3. Методи організації індексів<br>Завдання на СРС: підготуватися до надання відповідей на контрольні запитання комп‘ютерного тестування 4 |
| 15 | Тема 2. Організація управління транзакціями і журналізації змін станів у |

| | |
|---|---|
| | реляційній базі даних<br>Питання 1. Рівні ізоляції виконання транзакцій<br>Питання 2. Методи організації виконання набору транзакцій<br>Питання 3. Призначення і принципи використання журналів у РБД<br>Питання 4. Організація процесів журналізації змін станів РБД<br>Завдання на СРС: підготуватися до надання відповідей на контрольні запитання комп‘ютерного тестування 4 |
| | Тематичне комп‘ютерне тестування 4 - (8 балів) |
| Розділ 6. Технологія проектування баз даних автоматизованих інформаційних систем | |
| 16 | Тема 1. Етапи та методи проектування автоматизованих інформаційних систем<br>Питання 1. Стандарти і моделі технологій проектування<br>Питання 2. Етапи розробки АІС |
| 17 | Тема 2. Засоби проектування баз даних<br>Питання 1. Засоби інформаційного моделювання<br>Питання 2. Засоби функціонального моделювання |
| | Модульна контрольна робота - (18 балів) |

### 3. Зміст комп‘ютерних практикумів

Основними завданнями циклу комп‘ютерного практикуму є:

- закріплення отриманих теоретичних знань в області адміністрування баз даних гнучких комп‘ютеризованих систем;
- оволодіння студентами навичками взаємодії з системою управління базою даних, вироблення раціональних прийомів програмування операцій з обробки даних за допомогою команд мови структурованих запитів.

Виконавши цикл робіт практикумів, студенти отримують навички роботи в інтегрованому середовищі системи автоматизованого проектування баз даних та з інструментальними засобами взаємодії з промисловим сервером бази даних.

| № | Назва комп‘ютерного практикуму | Кількість балів |
|---|---|---|
| КП1 | Моделювання предметної області і проектування бази даних інформаційної системи<br>Дидактичні матеріали: Розділ 2 Тема 2 | 7 |
| КП2 | Створення структури бази даних командами мови визначення даних<br>Дидактичні матеріали: Розділ 4 Тема 1 | 7 |
| КП3 | Обробка інформації в базі даних командами мови маніпулювання даними<br>Дидактичні матеріали: Розділ 4 Тема 2 | 7 |
| КП4 | Формування запитів на виведення інформації з бази даних<br>Дидактичні матеріали: Розділ 4 Тема 2 | 7 |
| КП5 | Управління доступом до бази даних командами мови управління даними<br>Дидактичні матеріали: Розділ 4 Тема 3 | 7 |
| КПП | Підсумковий комп’ютерний практикум: Проектування і адміністрування бази даних<br>Дидактичні матеріали: комплексне контрольне завдання з перевірки знань і навичок із розробки бази даних та адміністрування взаємодії із сервером бази даних | 15 |

## 4. Зміст модульного контролю

Основні цілі модульного контролю визначаються необхідністю засвоєння студентами знань з основних тем змістовних розділів кредитного модуля та розвитку у них навичок самостійної роботи при виконанні практичних завдань комп‘ютерних практикумів.

Зміст модульного контролю складається з поточного, який забезпечує оцінювання навчально-пізнавальної діяльності студента протягом семестру і визначається як стартовий рейтинг студента, та екзаменаційного, який призначений для оцінювання результатів навчання.

Стартовий рейтинг формується внаслідок оцінювання практичних навичок отриманних під час виконання робіт комп‘ютерних практикумів у формі тестування і захисту їх результатів.

Екзаменаційний рейтинг формується або на протязі навчання внаслідок оцінювання комплексу контрольних робіт у формі комп’ютерних тестувань або по завершенні навчання у формі письмового іспиту.

Усі комп‘ютерні тестування виконуються згідно переліку вивчених тем теоретичного матеріалу лекцій а модульна контрольна робота інтегрує всі знання тем комп‘ютерних практикумів:

| № | Назва контрольного заходу | Кількість балів |
|---|---|---|
| КТ1 | Тематичне комп‘ютерне тестування 1:<br>Базові засади побудови та застосування систем баз даних<br>Дидактичні матеріали: Розділ 1 | 8 |
| КТ2 | Тематичне комп‘ютерне тестування 2:<br>Моделювання предметної області автоматизованих інформаційних систем<br>Дидактичні матеріали: Розділ 2 | 8 |
| КТ3 | Тематичне комп‘ютерне тестування 3:<br>Реляційна модель бази даних<br>Дидактичні матеріали: Розділ 3 | 8 |
| КТ4 | Тематичне комп‘ютерне тестування 4:<br>Команди мови SQL та організація управління транзакціями і індексами<br>Дидактичні матеріали: Розділ 4-5 | 8 |
| МКР | Модульна контрольна робота:<br>Моделювання та проектування бази даних | 18 |

## 5. Інструментальні програмні та методичні засоби

При проведенні комп‘ютерних практикумів вивчається та застосовується програмне забезпечення наступного середовища та інструментарію:

- сервер бази даних промислового застосування Oracle 11g;
- система автоматизованого проектування баз даних Oracle Data Modeler;
- програмне середовище взаємодії та адміністрування сервера бази даних Oracle SQL Developer;
- інтерфейс підключення до сервера бази даних - протокол tcp, адреса tc.kpi.ua (зовн.ip:77.47.131.58, внутр.ip:10.18.80.10), порт 1521, сервіс base;
- ресурс завантаження рекомендованого інструментального засобу виконання комп’ютерних практикумів - https://www.oracle.com/technical-resources/;
- ресурси системи дистанційного навчання кафедри – http://test.tc.kpi.ua;
- інтерфейс підключення до віддаленого робочого місця – термінальний клієнт AnyDesk, адреса – 800495200, сеанс роботи - консоль, користувач – TKLAB\ik<номер групи>, пароль – kpi_ik<номер групи>.

# Розділ 1. Базові засади побудови та застосування систем баз даних

## Тема 1.1. Концепція систем баз даних та автоматизованих інформаційних систем

### *1.1.1. Визначення систем баз даних*

Хоча поняття «інформація» (від латинського informatio - роз'яснення, виклад) є фундаментальним і широко використовуваним, однозначного визначення воно не має. Є безліч його визначень, які використовуються в різних галузях знань. Наприклад, в теорії інформації, інформація - будь-яка сукупність сигналів, впливів або відомостей, які деяка система сприймає від навколишнього середовища (вхідна інформація), видає в навколишнє середовище (вихідна інформація), а також зберігає в собі (внутрішня, внутрісистемна інформація). З точки зору основних завдань: інформація - це те, що є в системі об'єктом зберігання, передачі і обробки. Відповідно, отримана інформація уточнює знання про стан системи. Звідси випливає, що відомості, які не міняють наших знань про стан системи, інформацією не є. Інформацію, призначену для обробки, називають даними. Їх часто представляють в формалізованому вигляді, тобто записаними у вигляді символів, чисел.

Відомо, що виникнення технологій баз даних припадає на початок 60-х років і пов‘язане з розвитком й впровадженням засобів обчислювальної техніки. Їх швидкому розвитку сприяли потреби в обробці інформації з використанням засобів їх накопичення. Інформаційні задачі обробки на відміну від обчислювальних мають такі особливості:

− збереження даних складної структури;

− відносно прості алгоритми обробки;

− великі обсяги оброблюваної інформації.

Інформаційна система виконує функції збирання, зберігання, розповсюдження і обробки інформації. Під *інформацією* в інформаційних системах розуміють будь-які відомості про будь-яку подію, сутність, процес і т.ін., які є об'єктом певних операцій: передачі, перетворення, зберігання або використання. *Дані* можна визначити, як інформацію зафіксовану у певній формі, яка придатна для подальшої обробки, зберігання і передачі. *Предметна область* – область застосування конкретної інформації.

Отже, поняття інформації, даних, знань споріднені. У багатьох ситуаціях часто буває достатньо інтуїтивного розуміння та інтерпретації цих категорій. Складність формального визначення термінів "інформація", "дані", "знання" полягає у загальновживаності цих термінів. Іншою причиною термінологічної плутанини є той факт, що межа між цими термінами для більшості фахівців досить умовна.

Тому в технології баз даних застосовують наступні визначення цих термінів.

*Дані* - це елементарні описи предметів, подій (явищ, фактів), дій і транзакцій (перетворень), що запам'ятовуються, класифікуються і зберігаються, але ніяк не організовуються і не інтерпретуються.

*Інформація* - це дані, які організовані таким чином, щоб вони набули певного значення і цінності для користувача.

Тоді, *знання* складаються з даних або інформації, що організовані й оброблені (перетворені) з метою передачі певного розуміння, накопиченого досвіду, результатів навчання й експертизи таким чином, щоб могли використовуватися для вирішення проблем або виконання дослідницьких дій (рис. 1).

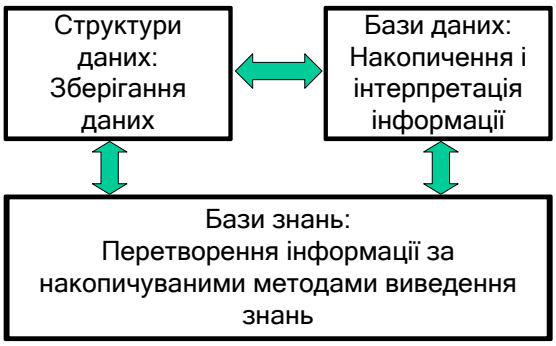



Рис. 1. Схема взаємодії «дані-інформація-знання»

Отже, дані можна розглядати як базове поняття, під яким мають на увазі опис різних подій, явищ, фактів, що відбивають сприйняття людиною реального світу. Традиційно фіксація даних здійснюється за допомогою конкретного засобу спілкування на конкретному носії.

Звичайно дані - факти, явища, події та їх інтерпретація - семантика фіксуються спільно у вигляді інформації, що передається від людини до людини. Однак, застосування ЕОМ для цілей ведення даних і обробки даних звичайно приводить до поділу на власне дані та їх інтерпретацію (рис. 2).

**Розклад руху літаків**

| Номер рейсу | Дні тиждня | Пункт відправлення | Час виліту | Пункт призначення | Час прибуття | Тип літака | Вартість квитка |
|---|---|---|---|---|---|---|---|
| 138 | 2_4_7 | Баку | 21.12 | Москва | 0.52 | ІЛ-86 | 115.00 |
| 57 | 3_6 | Єреван | 7.20 | Київ | 9.25 | ТУ-154 | 92.00 |
| 1234 | 2_6 | Казань | 22.40 | Баку | 23.50 | ТУ-134 | 73.50 |
| 242 | 1 по 7 | Київ | 14.10 | Москва | 16.15 | ТУ-154 | 57.00 |
| 86 | 2_3_5 | Мінськ | 10.50 | Сочі | 13.06 | ІЛ-86 | 78.50 |
| 137 | 1_3_6 | Москва | 15.17 | Баку | 18.44 | ІЛ-86 | 115.00 |

семантика

дані

інформація

Рис. 2. Інтерпретація інформації в базах даних

*Ведення* даних - термін, що поєднує дії по додаванню, видаленню чи зміні збережених даних про явище, подію, факт.

*Обробка* даних - термін, що характеризує дії по визначенню змісту даних та їх інтерпретації для формування інформації.

В основі рішення багатьох задач лежить обробка інформації, тобто виділення власне даних (фактів) і їх інтерпретація. Засобами обробки інформації на ЕОМ слугують *інформаційні системи (ІС)*.

Під *інформаційними системами* мають на увазі сукупність програмно-апаратних засобів обробки інформації, призначених для вирішення прикладних задач користувача у певній предметній області діяльності.

В результаті пам'ять ЕОМ використовується для збереження самих даних, а інтерпретація традиційно покладається на програму користувача, що утворює ІС (рис. 3). Тверда залежність між даними і їхніми програмами, що використовують, створює серйозні проблеми у веденні даних і робить використання їх менш гнучкими.

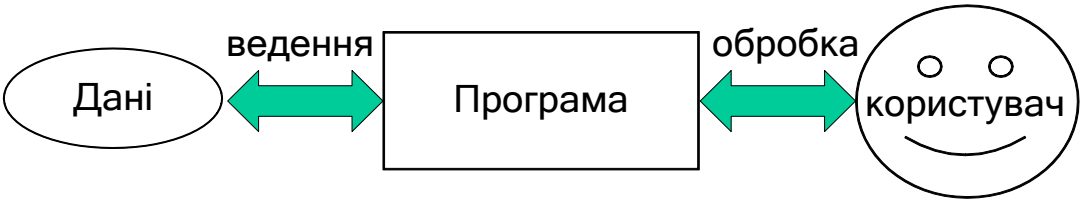



Рис. 3. Схема взаємодії користувача з даними в ИС

Активна діяльність по відшуканню прийнятних способів усуспільнення безупинно зростаючого обсягу інформації привела до створення на початку 60-х років XX століття спеціальних програмних комплексів, називаних *системи управління базами даних* (СУБД).

*Базами даних* (БД) стали називати електронні сховища інформації, доступ до яких здійснюється за допомогою одного чи декількох комп'ютерів.

*База даних* організується як масив зв'язаної інформації, тобто как єдині зібрання даних, між якими існують відношення, що встановлюють цілісний стан бази.

*Цілісність даних* у базі даних означає, что вся її інформація є доступною і погодженою згідно визначеного набору правил - обмежень.

*Системи баз* даних це комп'ютеризована система, основне призначення якої зберігати інформацію, надаючи користувачам засоби її вилучення та модифікації. До інформації може відноситися все, що заслуговує на увагу окремого користувача або організації, що використовує систему. Тобто, все необхідне для поточної роботи даного користувача або підприємства.

Дуже спрощена схема системи баз даних включає чотири головні компоненти: *дані*, *апаратне забезпечення*, *програмне забезпечення* та *користувачі*.

У загальному випадку дані в базі даних (у великих системах) є інтегрованими і поділеними.

Під поняттям *інтеграції* даних мається на увазі можливість представити базу даних як об'єднання декількох окремих файлів даних, яке повністю або частково виключає надмірність зберігання інформації.

Під поняттям *поділюваних* даних мається на увазі можливість використання декількома різними користувачами окремих елементів, що зберігаються в базі даних. Мається на увазі, що кожен з користувачів зможе отримати доступ до одних і тих же даних, можливо, навіть одночасно (паралельний доступ). Таке поділення даних, з паралельним або послідовним доступом, частково є наслідком того факту, що база даних має інтегровану структуру.

Між фізичною БД (даними, які зберігаються на комп'ютері) і користувачами системи розташовується рівень програмного забезпечення, який називають по-різному, як сервер бази даних (database server) або, як система управління базами даннях (СУБД, DataBase Management System - DBMS). Всі запити користувачів на отримання доступу до бази даних обробляються СУБД. Всі наявні засоби додавання, вибірки і поновлення даних також надає СУБД. Тобто, основне завдання СУБД - дати користувачеві бази даних можливість працювати з нею.

*Системи управління базами даних* – це сукупність мовних та програмних засобів для створення, ведення (наповнення, відновлення і видалення) та спільного використання баз даних.

Головна особливість СУБД – це наявність процедур для введення і збереження не тільки самих даних, але й описів структур даних - семантики. Структури даних (файли), споряджені описом збережених у них даних і

знаходжені під керуванням СУБД, стали називати банками даних, а потім базами даних.

Одним із наслідків згаданих вище характеристик бази даних (інтеграції та поділення) є те, що кожен конкретний користувач зазвичай має справу лише з невеликою частиною всієї бази даних. Причому оброблювані різними користувачами частини можуть довільним чином перекриватися і призводить до неузгодженості даних та їх суперечливому стану. Це вимагає від СУБД виконувати контроль паралельної роботи користувачів і робити її незалежною через механізм управління транзакціями.

*Транзакція* – це сукупність операцій СУБД, виконання яких не може бути перервано. Для того щоб зміни, внесені в БД у ході виконання кожної з вхідних у транзакцію операцій, були зафіксовані в базі, всі операції повинні завершитися успішно. У результаті такого впливу на СУБД, виконується переведення БД з одного цілісного стану в інше.

*Технологія* "*клієнт – сервер*" - це модель обчислень, яка передбачає розподіл функцій обробки в багатокористувальницькій базі даних по декількох комп'ютерах. Розподіл виконання функцій обробки між комп'ютерами здійснюється з використанням протоколу сервісних запитів, тобто один комп'ютер - клієнт запитує обслуговування в іншого комп'ютера - сервера, що реалізує обслуговування і відсилає його результати "клієнту".

Під *автоматизованою інформаційною системою* (АІС) розуміють комплекс апаратно-програмних засобів, що реалізують мультикомпонентну інформаційну систему, яка забезпечує сучасне керування процесами вирішення прикладних задач користувача у певній предметній області діяльності - прийняття рішень, проектування, виробництва і збуту, в режимі реального часу при транзакційній обробці даних.

### *1.1.2. Зв'язок програм і даних при використанні СБД*

Логічну структуру збережених у БД даних називають *моделлю представлення даних*. Знання про модель даних використовується для визначення правил взаємодії прикладних програм-додатків з базою даних для виведення і відтворення інформації користувачу. Ці правила оформлюються як основні дії, які здійснює програмне середовище СУБД.

Під *додатком (застосуванням)* часто мають на увазі програму, що забезпечує автоматизацію обробки інформації для прикладної задачі користувача в складі АІС.

При виконанні запиту на зчитування, оновлення або видалення даних, яке видане прикладною програмою користувача, СУБД виконує ряд дій, які включають:

1) інтерпретацію запиту;

2) пошук усіх описів даних, на які виданий запит;

3) формування команд, відповідно до яких операційна система пересилає із запам'ятовуючих пристроїв у буфер СУБД вміст всіх фізичних записів з необхідними даними;

4) виділення із цих записів потрібних даних, їхнє форматування, якщо це необхідно, створення заданого виду й послідовності виводу й пересилання в робочу область прикладної програми.

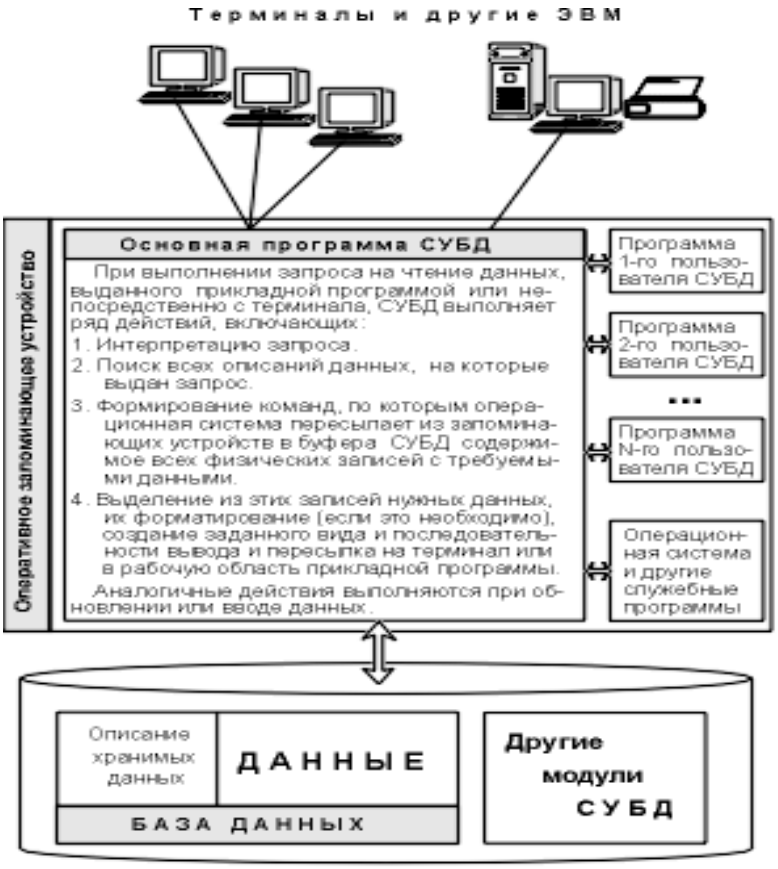



Рис. 4. Зв'язок програм з даними в СБД

Для роботи з базою даних іноді достатньо засобів саме СУБД у випадку проектування і обслуговування БД. Однак додатки розробляють у тих випадках, коли потрібно забезпечити зручність оперативної роботи з БД некваліфікованим користувачам і для організації зручного інтерфейсу інтерпретації даних.

*Користувачів* можна розділити на три великі і частково співпадаючі групи.

Перша група - *прикладні програмісти*, які відповідають за написання прикладних програм, що використовують базу даних. Прикладні програми отримують доступ до бази даних за допомогою видачі відповідного запиту до СУБД. Подібні програми можуть бути простими пакетними додатками або ж інтерактивними застосуваннями, призначеними для підтримки роботи кінцевих користувачів.

Друга група - *кінцеві користувачі*, які працюють з системою баз даних в інтерактивному режимі. Кінцевий користувач може отримувати доступ до бази даних, застосовуючи одне з інтерактивних застосуваннь, або ж інтерфейс, інтегрований в програмне забезпечення самої СУБД.

Третя група - *адміністратори*. При централізованому управлінні на підприємстві, яке використовує базу даних, є особи, які несуть основну відповідальність за дані та їх організацію на підприємстві - адміністратор даних, адміністратор бази даних.

*Адміністратор даних* - особа-кваліфікований користувач, яка відповідає за управління даними (планування БД, розробку і супроводження стандартів, прикладних алгоритмів і ділових процедур), вироблення вимог по супроводженню і обробці даних, а також за концептуальне і логічне проектування БД.

*Адміністратор* БД - особа-професійний користувач, яка відповідає за створення (фізичну реалізацію БД), за забезпечення безпеки і цілісності даних, ефективне використання і супровід БД. Адміністратор бази даних, на відміну від адміністратора даних, повинен бути професійним фахівцем в області інформаційних

технологій. Робота адміністратора БД полягає в створенні самих баз даних і організації технічного контролю, необхідного для здійснення рішень, прийнятих адміністратором даних.

### *1.1.3. Відмінності баз даних від файлових систем*

Із самого початку розвитку обчислювальної техніки утворилися два основних напрямки її використання:

- застосування обчислювальної техніки для виконання чисельних розрахунків;
- використання засобів обчислювальної техніки в АІС.

У самому широкому змісті АІС являє собою програмний комплекс, функції якого складаються в підтримці надійного збереження інформації в пам'яті ЕОМ, виконанні специфічних для даного додатка перетворень чи інформації обчислень, наданні користувачам зручного і легко освоюваного інтерфейсу.

Спочатку для збереження даних застосовувалися *файлові системи*. Для цих систем були характерні такі особливості:

− структура запису файла даних була відома тільки прикладній програмі, яка з ним працювала, а система управління файлами її не знала; кожна програма, яка працювала з файлом даних, повинна була мати у себе структуру даних, яка відповідала цьому файлу (така ситуація характеризувалась, як *залежність программ від даних*);

− система управління файлами повинна булла забезпечити авторизацію доступу до файлів для різних користувачів, але були відсутні централізовані методи управління доступом до інформації;

− для організації паралельної роботи користувачів з даними необхідне узгоджене управління, а система управління файлами при цьому працювала надто повільно.

Проте, перехід від використання централізованих систем керування файлами до СУБД вимагає розв‘язання наступних проблем:

- *захист файлів.* Оскільки файлові системи є загальним сховищем файлів, що належать, узагалі, різним користувачам, системи керування файлами повинні забезпечувати авторизацію доступу до файлів. У загальному виді підхід полягає в тому, що стосовно кожного зареєстрованого користувача даної обчислювальної системи для кожного існуючого файлу вказуються дії, які дозволені чи заборонені даному користувачу. В базах даних доступ повинен відбуватися не на рівні файлів, а на рівні окремих частин саме змісту файлів.
- *багатокористувальницькій доступ*. Якщо операційна система підтримує багатокористувальницькій режим, цілком реальна ситуація, коли два чи більш користувачів одночасно намагаються працювати з одним й тим файлом. Якщо хоча б один з них буде змінювати файл, то для коректної роботи цієї групи користувачів потрібно взаємна синхронізація. Файлові системи звичайно забезпечують збереження слабко структурованої інформації, залишаючи подальшу структуризацію прикладним програмам. АІС головним чином орієнтовані на збереження, вибір і модифікацію постійно змінюється інформації і вимагають складних структур даних і додаткових індивідуальних засобів керування даними. Неможливо обійтися загальною бібліотекою програм, що реалізує за допомогою тільки стандартної базової файлової системи більш складні методи збереження даних.
- *погодженість даних*. Є ключовим поняттям баз даних. Вже тільки вимога підтримки погодженості даних у декількох файлах не дозволяє обійтися

бібліотекою функцій файлової системи, така система повинна мати деякі власні дані (метадані) і навіть знання, що визначають цілісність даних.

- *уніфікація доступу і структуризація даних*. Набагато простіше, якби АІС дозволяла сформулювати інформаційний запит на прийнятній користувачам мові. При формулюванні запиту СУБД дозволить не задумуватися над тим, як буде виконуватися цей запит.

Переваги СУБД над файловими системами:

- *Мінімізація надлишковості даних*. Вимога мінімізації полягає у зберіганні в базі мінімальної кількості копій одних і тих же даних;

- *Несуперечливість даних* і контроль їх цілісності. Несуперечливість полягає у збереженні узгодженності даних при модифікації бази за допомогою засобів підтримки цілістності;

- *Незалежність прикладних програм від даних*. Незалежність полягає у можливості зміни структур даних від без зміни програм, що їх використовують;

- *Підвищена безпека.* Доступ до даних на рівні окремих блоків, елементів, а не всієї бази;

- Розвинені засоби резервного копіювання і відновлення. Захист даних від збоїв у роботі, як апаратних, так і програмних засобів;

- Багатокористувацький режим роботи. Одночасний паралельний режим роботи зі спільними даними.

### *1.1.4. Етапи розвитку СУБД та їх інтеграції з АІС*

АІС із централізованої БД (рис. 5) почали застосовуватися на початку 1960 рр. і працювали безпосередньо із БД, яка керувалася файловою системою операційної системи комп'ютерної системи, побудованої на базі Mainframe (універсального обчислювального комплексу) із централізованим доступом до всіх обчислювальних ресурсів. Функції СУБД виконувалися програмним забезпечення АІС, тобто були інтегровані до складу прикладних програмних застосувань. Створення незалежних від СУБД застосувань дозволило звертатися до БД безпосередньо. Переваги – економія зовнішньої та оперативної пам'яті, пришвидшення виконання застосувань, повний захист від модифікації. Недоліки такого підходу пов'язані з трудомісткістю доробки застосувань, відсутністю стандартних засобів СУБД по обслуговуванню БД.

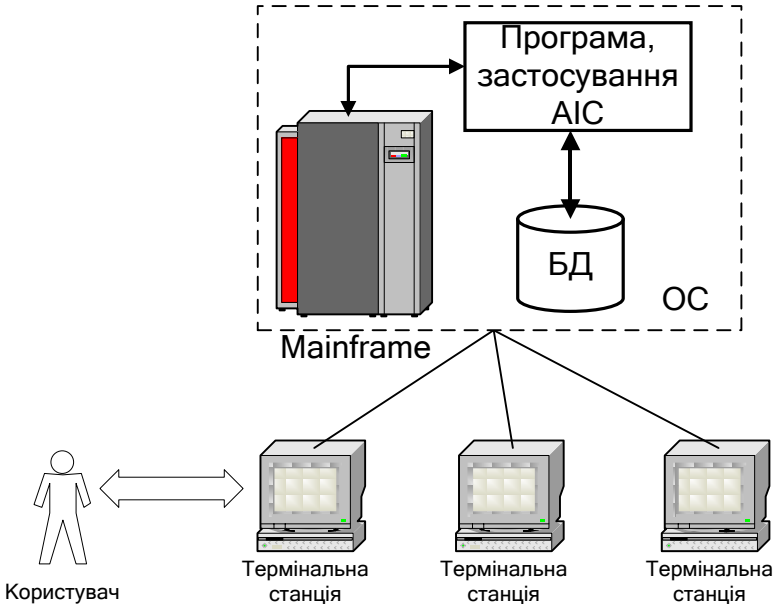



Рис. 5. АІС з централізованою базою даних

АІС із централізованої СУБД (рис. 6) почали застосовуватися на початку 1970 рр. і взаємодіяли із централізованою системою управління даними, які вже мали визначену структуру подання – модель даних. Така система управління виконували функції СУБД по веденню бази даних (маніпулюванню даними), розташованої у файловій системі ОС. Такий підхід дозволил підвищити швидкість роботи застосування, зменшити об'єм необхідної пам'яті, простоту розробки та модифікації застосувань. Недолік – додаткові витрати зовнішньої та оперативної пам‘яті на розташування та роботу СУБД.

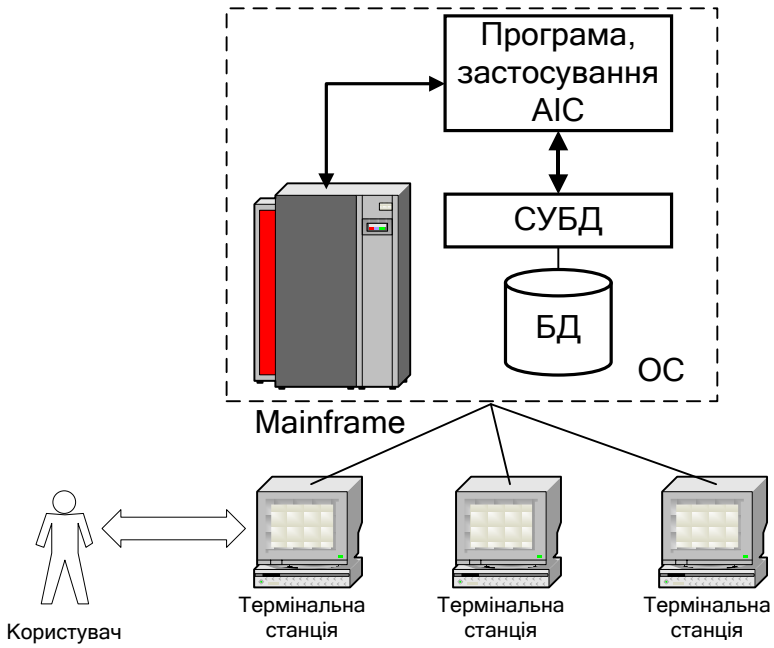



Рис. 6. АІС з централізованою СУБД

АІС із персональної СУБД (рис. 7) почали застосовуватися із появою персональних ЕОМ у середині 1980 рр. При такому підході застосування і ядро СУБД інтегруються в єдиний комплекс, іноді представляється як єдиний виконуваний модуль в ОС ПЕОМ.

Ядро СУБД – СУБД з усіченими функціями по веденню даних – зберігання і маніпулювання, інші функції не підтримуються, а передаються застосуванню АІС.

Застосування разом з ядром СУБД використовується для зменшення обсягу займаного простору і оперативної пам'яті, підвищення швидкості роботи програми, захисту програми від модифікації з боку користувача. Зазвичай ядро СУБД не містить засобів розробки застосувань.

Недолік – значні витрати зовнішньої та оперативної пам‘яті для збереження та використання ядра СУБД, недостатня швидкодія роботи застосувань, які працюють як інтерпретатори команд ядра СУБД.

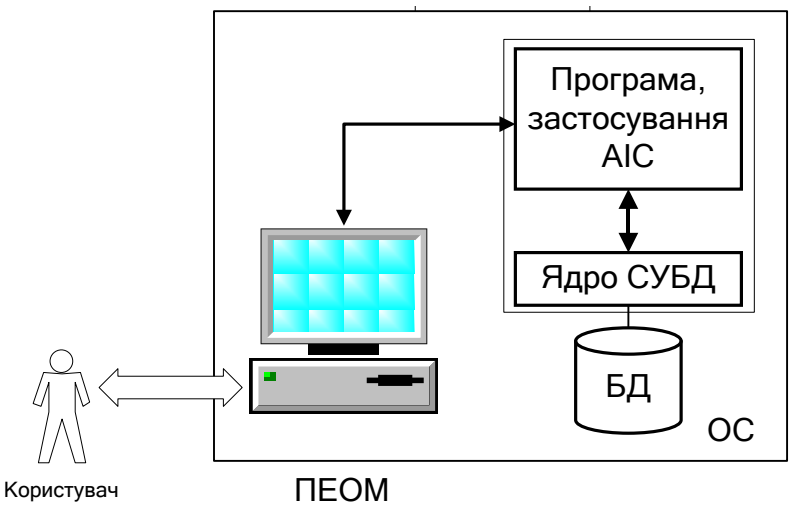



Рис. 7. АІС з персональною СУБД

АІС із файл-серверною БД (рис. 8) почали використовуватися із розвитком і застосуванням засобів мережної взаємодії персональних ЕОМ у 1990 рр. Такий підхід є подальшим розвитком архітектури АІС з персональною СУБД за рахунок перенесення місця накопичення і збереження даних на виділений файл-сервер і підключення різних застосувань АІС з різних місць до централізованої БД. Файлова система виділеного сервера має розширення стандартних функцій вводу-виводу даних за рахунок додавання можливості підключення і зчитування файлів даних іншими ПЕОМ у комп‘ютерній мережі. Також додані функції блокування файлів та окремих їх частин при одночасній роботи з ними різних ПЕОМ. Комп‘ютерна мережа використовується як транспортний засіб передачі файлів даних від файл-серверу до ПЕОМ і навпаки. Переваги і недоліки такого підходу наслідуються від АІС з персональною СУБД. Крім того, недоліком є висока інтенсивність передачі оброблюваних даних, надлишковість при передаванні даних мережою.

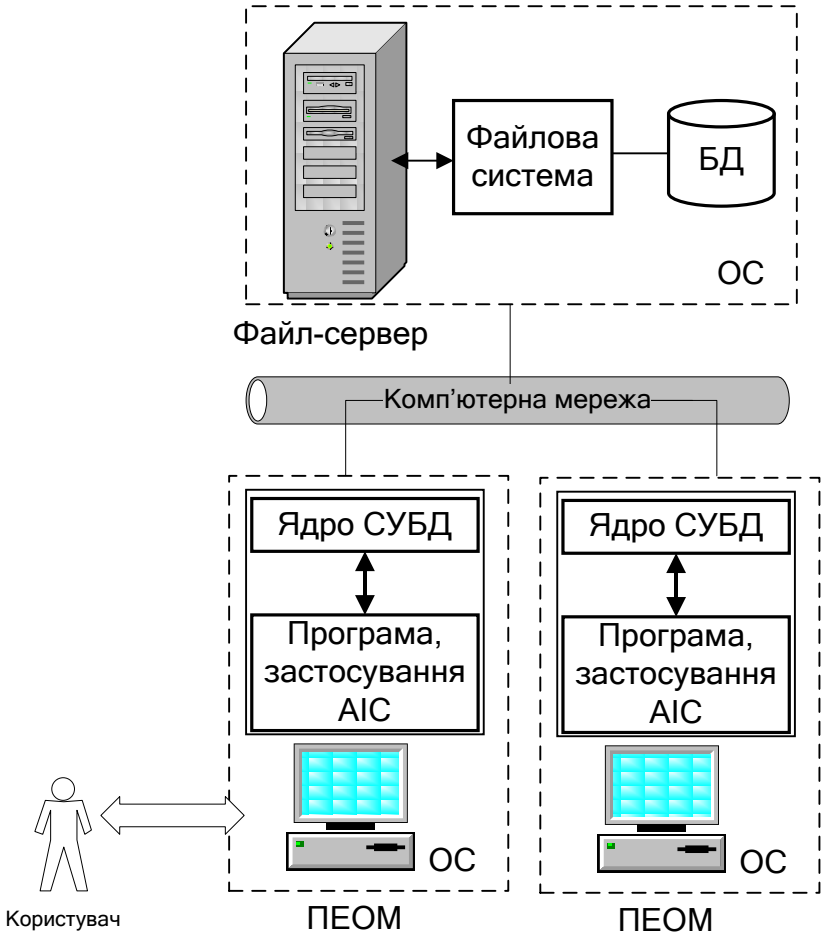



Рис. 8. АІС з файл-серверною БД

АІС із клієнт-серверною СУБД (рис. 9) почали використовуватися з розвитком технології «клієнт-сервер» із розширенням засобів мережної взаємодії ЕОМ в глобальній мережі Internet у 1995-2000 рр. Такий підхід є подальшим розвитком архітектури АІС з файл-серверною БД за рахунок перенесення принципів підходу АІС з централізованою СУБД на рівень корпоративної і глобальної мережі з організацією підключення різних застосувань АІС з різних місць і різних користувачів до централізованої СУБД. Комп'ютерна мережа використовується як транспортний засіб передачі запитів на отримання, маніпулювання даними від ПЕОМ до серверу БД і повернення результатів обробки такого запиту для подальшого інтерпретування застосуваннями АІС.

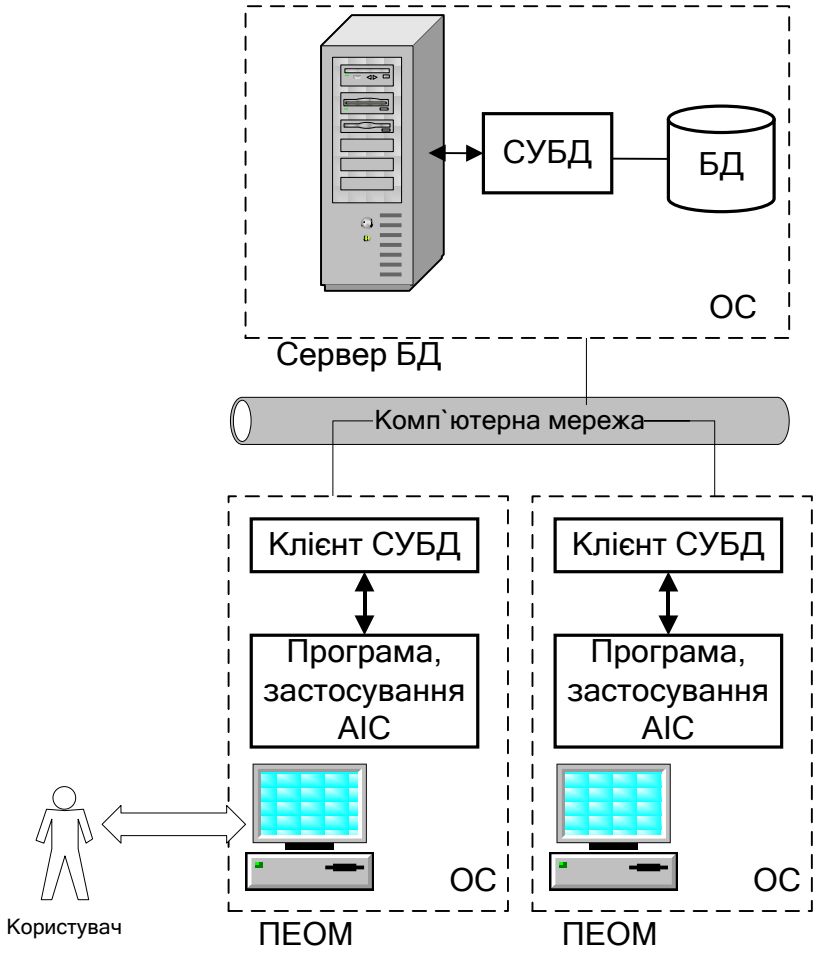



Рис. 9. АІС із клієнт-серверною СУБД

АІС із розподіленою СУБД (рис. 10) є розвитком підходу АІС із клієнт-серверною СУБД і поширення технології «клієнт-сервер» на взаємодію декількох СУБД у корпоративній або глобальній мережі передачі даних. Внаслідок такого підходу створюється можливість розподілу даних та їх обробки за декількома обчислювальними ресурсами, що впливає на надійність та швидкість функціонування розподілених систем.

При інтеграції комп'ютерів в розподіленій мережі виникає можливість розподілу функцій застосувань, що працюють як з єдиною БД, так і з розподіленою БД у мережі.

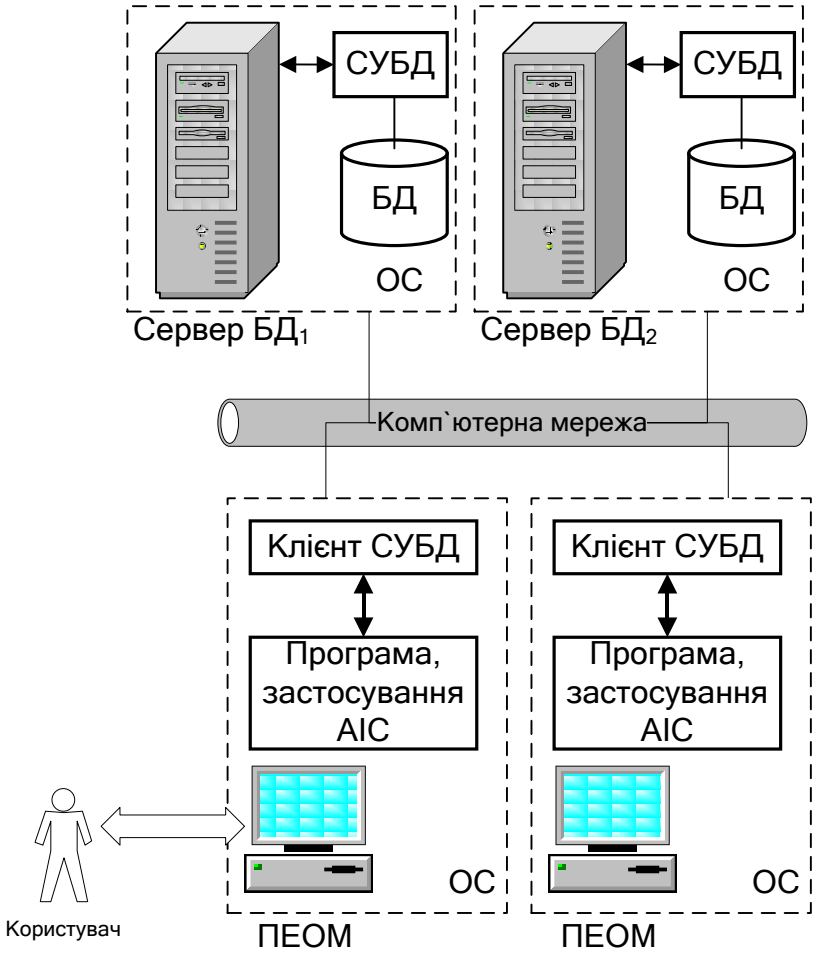



Рис. 10. АІС із розподіленою СУБД

## Тема 1.2. Архітектура автоматизованих інформаційних систем та систем управління базами даних

### *1.2.1. Склад і класифікація СУБД*

СУБД складається з таких компонентів:
- сервера БД (ядро СУБД);
- клієнта СУБД;
- підсистеми засобів обробки і управління БД;
- підсистеми засобів проектування і розробки додатків БД.

*Ядро СУБД* – містить сукупність базових механізмів СУБД, які використовуються при будь-яких варіантах конфігурації системи. Ядро СУБД виконує функцію посередника між підсистемами засобів розробки і обробки. Сучасні БД у більшості представляють користувачу дані у вигляді таблиць. Ядро СУБД отримує запити від інших компонентів в термінах таблиць, стовпців, рядків і перетворює ці запити в команди операційної системи, які виконують запис і читання з фізичних носіїв інформації. Крім того, ядро СУБД задіяне в управлінні транзакціями, блокуваннях, резервному копіюванні і відновленні.

*Підсистема засобів проектування і розробки* являє собою набір інструментів, які спрощують проектування і реалізацію баз даних і їх застосувань. Як правило, цей набір містить засоби для створення таблиць, форм, запитів й звітів. В СУБД є також мови програмування і інтерфейси до них.

*Підсистема обробки і управління* здійснює обробку компонентів застосування, які створені за допомогою засобів проектування (процесор форм; процесор запитів; генератор звітів; засоби обробки процедурні).

*Застосування БД* складається з форм, запитів, звітів, меню і прикладних програм. Форми, запити і звіти можна створювати за допомогою засобів, що постачаються у комплекті з СУБД. Прикладні програми повинні бути написані або на вхідній мові СУБД, або на одній зі стандартних мов, а потім за допомогою *клієнта СУБД* з'єднані з БД.

*Ознаками класифікації* СУБД є:

1) ступінь універсальності - проектно-залежні (спеціалізовані), проектно-незалежні (універсальні) СУБД;

2) масштаб (характер) використання - інтегровані (загальні), локальні, багато-користувальницькі, одно-користувальницькі СУБД;

3) місце зберігання даних - централізовані, розподілені СУБД;

4) ступінь зв‘язності (структурованості) даних - документальні (слабко структуровані), фактографічні (сильно структуровані) СУБД;

5) модель даних - ієрархічні, мережні, реляційні, об'єктні, багатовимірні СУБД.

До *основних класів* СУБД відносять:
- повнофункціональні СУБД;
- персональні СУБД;
- багато-користувальницькі СУБД;
- клієнт-серверні розподілені СУБД.

*Повнофункціональні СУБД* - традиційні СУБД із розвиненим інтерфейсом взаємодії із БД.

*Персональні СУБД* - по характеру використання позиціонуються як СУБД, що функціонують на ПЕОМ для підтримки роботи одного користувача (програмного додатку).

*Багато-користувальницькі СУБД* - містять у собі сервер БД і клієнтську частину для підтримки роботи багатьох користувачів у неоднорідному обчислювальному середовищі.

*Клієнт-Серверні розподілені СУБД* - призначені для організації центрів обробки інформації в мережах ЕОМ з розподіленими вузлами накопичення даних (розподіленими БД).

На поточному етапі розвитку автоматизованих інформаційних систем промислового застосування до баз даних висуваються *наступні вимоги*:

- *централізоване керування* при багатокористувальницькому доступі до даних – забезпечувати багаторазове та багатоаспектне використання одних і тих же даних різними користувачами;
- *скорочення надмірності даних* у БД – забезпечувати низьку вартість зберігання і використання даних;
- *забезпечення й впровадження стандартів* у поданні даних –забезпечувати простоту і гнучкість процесу створення і супроводу застосувань, що працюють з БД;
- *забезпечення цілісності й безпеки даних* – система повинна забезпечувати необхідний рівень захисту даних від перебоїв, впливу різних ситуацій або від зловмисного звернення до даних користувачів та мати засоби опису й підтримки обмежень цілостності, достатніх для забезпечення тих правил, що діють у предметній області;
- *забезпечення незалежності даних* – вимагається наявність механізмів підтримки логічної та фізичної незалежності даних. Будь які зміни в БД повинні виконуватися незалежно від застосувань, що вже існують і використовують БД;
- *підтримка розподіленої обробка даних* у мережах ЕОМ – забезпечувати можливість простого й ефективного збільшення обсягів даних без порушення наявних способів їхнього використання, а також легкого виконання реорганізації та реструктуризації даних.

### *1.2.2. Архітектура БД: рівні представлення даних, функції, компоненти*

#### *1.2.2.1. Рівні подання даних у СУБД*

СУБД повинна надавати доступ до даних будь-яких користувачів, включаючи й тих, які практично не мають або не хочуть мати уявлення про:

- фізичне розміщення в пам'яті даних і їхніх описів;
- механізми пошуку запитуваних даних;
- проблеми, що виникають при одночасному запиті тих самих даних багатьма користувачами або їх прикладними програмами;
- способи забезпечення захисту даних від некоректних відновлень або несанкціонованого доступу;
- підтримку бази даних в актуальному стані та інших функцій СУБД.

При виконанні основних із цих функцій СУБД повинна використовувати різні описи (подання) даних. В 1975 році підкомітет SPARC (Standards Planning and Requirements Committee) американського національного інституту стандартів (American National Standards Institute, ANSI) висунув проект трирівневої архітектури подання даних в СУБД, який пізніше затверджений ISO (рис. 11):

- *зовнішній рівень* - Індивідуальні уявлення предметної області окремими користувачами бази даних, виконані за допомогою тексту, графіки та інших, зрозумілих усім засобів;
- *концептуальний рівень* - Узагальнене уявлення всіх користувачів і додатків бази даних, як правило, виконане з використанням інфологічної моделі;
- *внутрішній рівень* - Опис бажаного способу організації бази даних в середовищі зберігання обраної СУБД - опис фізичної реалізації, що дозволяє домогтися оптимальної продуктивності СУБД і забезпечення економного використання запам'ятовуючих пристроїв.

Природно, що проект бази даних треба починати з аналізу предметної області й виявлення вимог до неї окремих користувачів - співробітників організації, для яких створюється база даних. Проектування бази доручається людині (групі осіб) – *адміністратору даних* (АД).

Поєднуючи різні представлення про вміст бази даних, отримані в результаті опитування користувачів, і свої представлення про дані, які можуть знадобитися в майбутніх програмних додатках, АД спочатку створює узагальнений неформальний опис створюваної бази даних у трирівневій архітектурі подання. Цей опис, виконаний з використанням природної мови, математичних формул, таблиць, графіків і інших засобів, зрозумілий всім користувачам, що працюють над проектуванням бази даних, називають *інфологічною моделлю подання даних.*

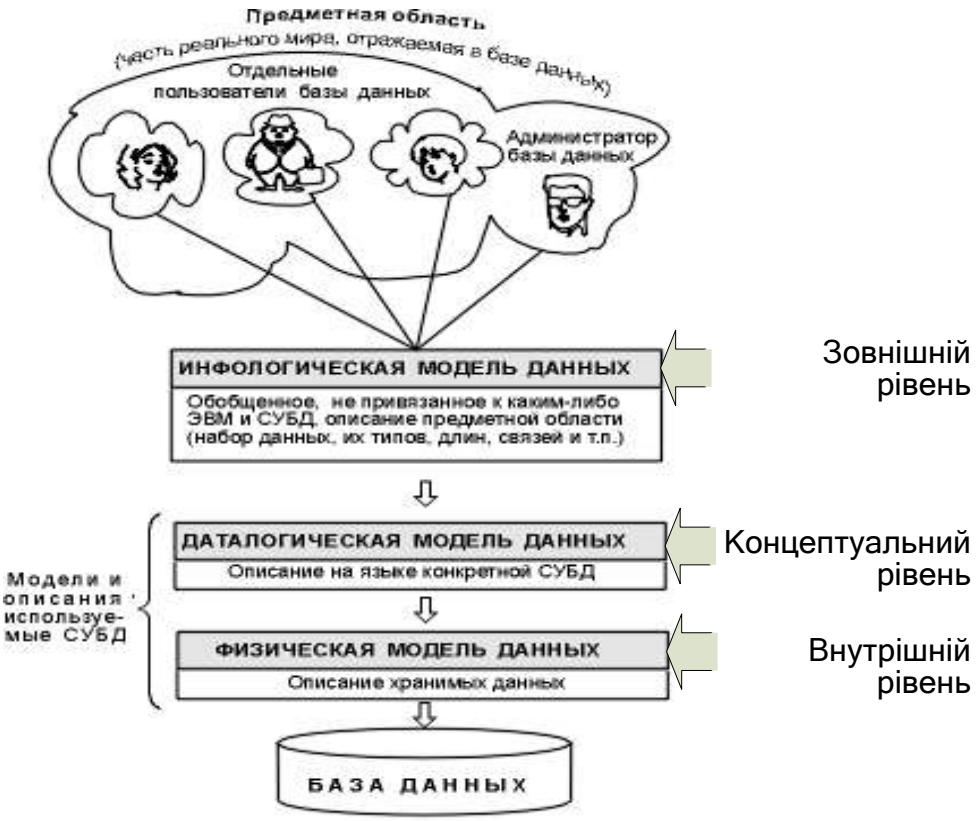



Рис. 11. Архітектура даних у СУБД

Така людино-орієнтована модель повністю незалежна від фізичних параметрів середовища зберігання даних. Тому інфологічна модель не повинна змінюватися доти, поки якісь зміни в реальному світі не зажадають зміни в ній деякого визначення, щоб ця модель продовжувала відтворювати предметну область.

Інші моделі, є комп'ютерно-орієнтованими. З їхньою допомогою СУБД дає можливість програмам і користувачам здійснювати доступ до збережених даних лише за їхніми іменами, не піклуючись про фізичне розташування цих даних. Потрібні дані відшукуються СУБД на зовнішніх пристроях за *фізичною моделлю подання даних*.

Тому що, зазначений доступ здійснюється за допомогою конкретної СУБД, то моделі повинні бути описані *мовою опису даних* цієї СУБД. Такий опис, створюваний *адміністратором БД* (АБД) за інфологічною моделлю даних, називають *даталогічною моделлю подання даних*.

Трирівнева архітектура (інфологічний, даталогічний і фізичний рівні) дозволяє забезпечити незалежність збережених даних від їхніх програм, що використовують.

АБД може при необхідності переписати збережені дані на інші носії інформації й (або) реорганізувати їхню фізичну структуру, змінивши лише фізичну модель даних. АБД може підключити до системи будь-яке число нових користувачів (нових додатків), доповнивши, якщо треба, даталогическую модель. Зазначені зміни фізичної й даталогической моделей не будуть замічені існуючими користувачами системи (виявляться "прозорими" для них), так само як не будуть замічені й нові користувачі. Отже, незалежність даних забезпечує можливість розвитку системи баз даних без руйнування існуючих додатків.

*1.2.2.2. Функції СУБД*

Традиційних можливостей файлових систем виявляється недостатньо для побудови навіть простих інформаційних систем. Виявлено кілька ***вимог***, які не покриваються можливостями систем керування файлами:

- підтримка логічно погодженого набору файлів;
- забезпечення мови маніпулювання даними;
- відновлення інформації після різного роду збоїв;
- реально паралельна робота декількох користувачів.

Вважається, що якщо прикладна інформаційна система опирається на деяку систему керування даними, яка володіє цими властивостями, то ця система керування даними і є системою управління базами даних.

Однією з головних відмінностей СУБД від файлових систем або систем управління даними є вимога цілісності БД.

*Цілісність* – це властивість БД, яка означає, що в ній міститься повна, несуперечлива і адекватна (узгоджена) до відтвореної предметної області інформація за рахунок реалізації механізму керування транзакціями.

В СУБД ***властивостями транзакції*** є:

- атомарність «*Atomicity*» (виконуються всі, що входять в транзакцію, операції або жодна з них );
- узгодженість (несуперечність) «*Consistency*» (відсутній взаємний вплив паралельно виконуваних транзакцій, кожна транзакція становить перехід від одного цілісного стану бази до іншого, тобто новий стан бази задовольняє усім встановленим обмеженням цілісності. Результат запиту має бути узгоджений зі станом БД на момент початку його виконання, в обробку беруться дані на момент початку його відпрацювання);
- ізольованість «*Isolation*» (на результати кожної транзакції не повинні впливати інші паралельно виконувані транзакції, результати не завершеної транзакції не видимі іншим транзакцій до тих пір, поки перша не закінчиться);
- стійкість (довговічність) «*Durability*» (відсутня можливість втрати даних виконаних транзакцій, тобто зміни внесені успішно до бази залишаються збереженими незалежно від працездатності системи баз даних).

До числа *основних функцій* СУБД прийнято відносити наступні:

1) *безпосереднє керування даними в зовнішній пам'яті*. Ця функція включає забезпечення необхідних структур зовнішньої пам'яті як для зберігання даних, що безпосередньо входять у БД, так і для службових цілей, наприклад, для прискорення доступу до даних у деяких випадках (звичайно для цього використовуються індекси). У деяких реалізаціях СУБД активно використовуються можливості існуючих файлових систем, в інших робота провадиться аж до рівня пристроїв зовнішньої пам'яті. В розвинених СУБД користувачі в кожному разі не зобов'язані знати, чи використовує СУБД файлову систему, і якщо використовує, то як організовані файли. Зокрема, СУБД підтримує власну систему іменування об'єктів БД.

2) *керування буферами оперативної пам'яті.* СУБД звичайно працюють із БД значного розміру; принаймні, цей розмір звичайно істотно більше доступного обсягу оперативної пам'яті. Зрозуміло, що якщо при звертанні до будь-якого елемента даних буде провадитися 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. При цьому, навіть якщо операційна система робить загальносистемну буферизацію, цього недостатньо для цілей СУБД, яка має у своєму розпорядженні набагато більшу інформацію про буферизацію тієї або іншої частини БД. Тому в розвинених СУБД підтримується власний набір буферів оперативної пам'яті із власною дисципліною заміни буферів. Існує окремий напрямок розвитку СУБД, який орієнтований на постійну присутність в оперативній пам'яті всієї БД. Цей напрямок ґрунтується на припущенні, що в обсяг оперативної пам'яті комп'ютерів може бути настільки великий, що дозволить не турбуватися про буферизацію (так звані In-Memory Data Base).

3) *керування транзакціями.* Транзакція - це послідовність операцій над БД, яка розглядається (сприймається) СУБД як єдине ціле. Тобто транзакція успішно виконується, і СУБД фіксує зміни БД, зроблені цією транзакцією, у зовнішній пам'яті, або жодне із цих змін ніяк не відбивається на стані БД. Поняття транзакції необхідно для підтримки логічної цілісності БД. Але поняття транзакції набагато більш важливе в багато-користувальницьких СУБД. Та особливість, що кожна транзакція починається при цілісному стані БД і залишає цей стан цілісним після свого завершення, робить дуже зручним використання поняття транзакції як одиниці активності користувача стосовно БД. При відповідному керуванні паралельно виконуваних транзакцій з боку СУБД кожний з користувачів може в принципі відчувати себе єдиним користувачем СУБД.

З керуванням транзакціями в багато-користувальницької СУБД пов'язані важливі поняття *серіалізації транзакцій* і *серіального (послідовного) плану виконання сукупності транзакцій*. Під серіалізацією паралельно виконуваних транзакцій розуміється такий порядок планування їхньої роботи, при якому сумарний ефект сукупності транзакцій еквівалентний ефекту їх деякого послідовного виконання. Серіальний план виконання сукупності транзакцій - це такий план, що приводить до серіалізації транзакцій. Зрозуміло, що якщо вдається домогтися дійсно серіального плану виконання сукупності транзакцій, то для кожного користувача, з ініціативи якого утворена транзакція, присутність інших транзакцій буде непомітно (якщо не вважати деякого сповільнення в роботі у порівнянні з одно-користувальницьким режимом).

Існує кілька базових алгоритмів серіалізації транзакцій. У централізованих СУБД найпоширеніші алгоритми, засновані на синхронізаційних захопленнях об'єктів БД. При використанні будь-якого алгоритму серіалізації можливі ситуації конфліктів між двома або більше транзакціями через доступ до об'єктів БД. У цьому випадку для підтримки серіалізації необхідно виконати відкіт (ліквідувати всі зміни, зроблені в БД) одній або іншій транзакції. Це один з випадків, коли користувач багато-користувальницької СУБД може реально (і досить неприємно) відчути присутність у системі транзакцій інших користувачів.

4) *журналізація й відновлення БД після збоїв.* Однією з основних вимог до СУБД є надійність зберігання даних у зовнішній пам'яті. Під надійністю зберігання розуміється те, що СУБД повинна бути спроможна відновити останній погоджений стан БД після будь-якого апаратного або програмного збою. Звичайно розглядаються два можливих види апаратних збоїв: так звані м'які збої, які можна трактувати як раптову зупинку роботи комп'ютера (наприклад, аварійне вимикання живлення), і жорсткі збої, які характеризуються втратою інформації на носіях зовнішньої пам'яті.

Прикладами програмних збоїв можуть бути: аварійне завершення роботи СУБД через помилку в програмі або в результаті деякого апаратного збою або аварійне завершення користувацької програми, у результаті чого деяка транзакція залишається незавершеною. Першу ситуацію можна розглядати як особливий вид м'якого апаратного збою; при виникненні останньої потрібно ліквідувати наслідки тільки однієї транзакції. В кожному разі для відновлення БД потрібно мати деяку додаткову інформацію. Інакше кажучи, підтримка надійності зберігання даних у БД вимагає надмірностей у зберіганні даних, причому та частина даних, що використовується для відновлення, повинна зберігатися особливо надійно. Найпоширенішим методом підтримки такої надлишкової інформації є ведення журналу змін БД.

*Журнал* - це особлива частина БД, недоступна користувачам СУБД і підтримувана з особливою старанністю (іноді підтримуються дві копії журналу, що розташовані на різних фізичних носіях), до якої надходять записи про всі зміни основної частини БД. У різних СУБД зміни БД фіксуються у журналі на різних рівнях: іноді запис у журналі відповідає деякій логічній операції зміни БД (наприклад, операції видалення рядка з таблиці БД), іноді - мінімальної внутрішньої операції модифікації сторінки зовнішньої пам'яті; у деяких системах одночасно використовуються обидва підходи.

У всіх випадках дотримуються стратегії "попереджуючого" запису у журнал (так званого протоколу Write Ahead Log - WAL). Ця стратегія полягає в тім, що запис про зміну будь-якого об'єкта БД повинен потрапити в зовнішню пам'ять журналу раніше, ніж змінений об'єкт потрапить у зовнішню пам'ять основної частини БД. Відомо, що якщо в СУБД коректно дотримується протокол WAL, то за допомогою журналу можна вирішити всі проблеми відновлення БД після будь-якого збою.

Найпростіша ситуація відновлення - індивідуальний відкот транзакції. Для цього не потрібен загальносистемний журнал змін БД (redo log). Досить для кожної транзакції підтримувати локальний журнал (undo) операцій модифікації БД, виконаних у цій транзакції, і робити відкот транзакції, шляхом виконання зворотних операцій, ідучи від кінця локального журналу. У деяких СУБД так і роблять, але в більшості систем локальні журнали не підтримують, а індивідуальний відкот транзакції виконують по загальносистемному журналі, для чого всі записи від однієї транзакції зв'язують зворотним списком (від кінця до початку).

При м'якому збої в зовнішній пам'яті основної частини БД можуть перебувати об'єкти, які модифіковані транзакціями, що не закінчилися до моменту збою, а також можуть бути відсутні об'єкти, які модифіковані транзакціями, які до моменту збою успішно завершилися (через використання буферів оперативної пам'яті, вміст яких при м'якому збої пропадає). При дотриманні протоколу WAL у зовнішній пам'яті журналу повинні гарантовано перебувати записи, що відносяться до операцій модифікації обох видів об'єктів. Метою процесу відновлення після м'якого збою є стан зовнішньої пам'яті основної частини БД, що виникло б при фіксації в зовнішній пам'яті змін всіх транзакцій, що завершилися, і яке не містило б ніяких слідів незакінчених транзакцій. Для того, щоб цього домогтися, спочатку роблять відкіт незавершених транзакцій (undo), а потім повторно відтворюють (redo) ті операції завершених транзакцій, результати яких не відображені в зовнішній пам'яті. Цей процес містить багато тонкостей, пов'язаних із загальною організацією керувань буферами й журналом.

Для відновлення БД після жорсткого збою використовують журнал і архівну копію БД. *Архівна копія* - це повна копія БД до моменту початку заповнення журналу (є багато варіантів більше гнучкого трактування змісту архівної копії). Звичайно, для нормального відновлення БД після жорсткого збою необхідно, щоб журнал не пропав. До схоронності журналу в зовнішній пам'яті в СУБД пред'являються

особливо підвищені вимоги. Тоді відновлення БД полягає в тому, що, виходячи з архівної копії, за допомогою журналу відтворюється робота всіх транзакцій, які закінчилися до моменту збою. У принципі, можна навіть відтворити роботу незавершених транзакцій і продовжити їхню роботу після завершення відновлення. Однак у реальних системах це звичайно не робиться, оскільки процеси відновлення після жорсткого збою є досить тривалими.

5) *підтримка мов БД.* Для роботи з базами даних використовуються спеціальні мови, у цілому називані *мовами баз даних*. У ранніх СУБД підтримувалося трохи спеціалізованих за своїми функціями мов. Найчастіше виділялися дві мови - *мова визначення схеми* БД (*SDL - Schema Definition Language*) і *мова маніпулювання даними (DML - Data Manipulation Language).* SDL служив головним чином для визначення логічної структури БД, тобто тієї структури БД, яку вона представляє користувачам. DML містив набір операторів маніпулювання даними, тобто операторів, що дозволяють заносити дані в БД, видаляти, модифікувати або вибирати існуючі дані. На даний час термін SDL замінено на - мову визначення даних (*DDL - Data Definition Language*).

У сучасних СУБД звичайно підтримується єдина інтегрована мова, що містить всі необхідні засоби для роботи із БД, починаючи від її створення, і користувальницький інтерфейс, що забезпечує базовий, з базами даних. Стандартною мовою реляційних СУБД є мова SQL (Structured Query Language).

Насамперед, мова SQL об‘єднує засоби DDL і DML, тобто дозволяє визначати схему БД і маніпулювати даними. При цьому іменування об'єктів БД (для реляційної БД - іменування таблиць і їхніх стовпців) підтримується на мовному рівні в тому розумінні, що компілятор мови SQL робить перетворення імен об'єктів у їхні внутрішні ідентифікатори на підставі спеціально підтримуваних службових таблиць-каталогів. Внутрішня частина СУБД (ядро) взагалі не працює з іменами таблиць і їхніх стовпців.

Мова SQL містить спеціальні засоби визначення обмежень цілісності БД. Знову ж, обмеження цілісності зберігаються в спеціальних таблицях-каталогах, і забезпечення контролю цілісності БД провадиться на мовному рівні, тобто при компіляції операторів модифікації БД компілятор SQL на підставі наявних у БД обмежень цілісності генерує відповідний програмний код.

Спеціальні оператори мови SQL дозволяють визначати так звані представлення БД, що фактично є збереженими в БД запитами (результатом будь-якого запиту до реляційної БД є таблиця-відношення) з іменованими стовпцями. Для користувача подання є такою ж таблицею, як будь-яка базова таблиця, збережена в БД, але за допомогою подань можна обмежити, або навпаки, розширити видимість БД для конкретного користувача. Підтримка подань провадиться також на мовному рівні.

Нарешті, авторизація доступу до об'єктів БД провадиться також на основі спеціального набору операторів SQL. Ідея полягає в тому, що для виконання операторів SQL різного виду користувач повинен мати різні повноваження. Користувач, що створив таблицю БД, має повний набір повноважень для роботи із цією таблицею. У число цих повноважень входить повноваження на передачу всіх або частини повноважень іншим користувачам, включаючи повноваження на передачу повноважень. Повноваження користувачів описуються в спеціальних таблицях-каталогах, контроль повноважень підтримується на мовному рівні.

*1.2.2.3. Компоненти СУБД*

Організація СУБД і состав її компонентів відповідає розглянутому набору функцій.

В сучасній СУБД логічно можна виділити наступні *компоненти* (рис. 12):

- найбільш внутрішню її частину - ядро СУБД (часто називають Data Base Engine);
- компілятор мови БД (наприклад, SQL);
- підсистему підтримки часу виконання;
- набір утиліт обслуговування роботи системи.

У деяких системах ці частини виділяються явно (фізично), в інших – неявно (логічно), але такий поділ можна провести у всіх СУБД.

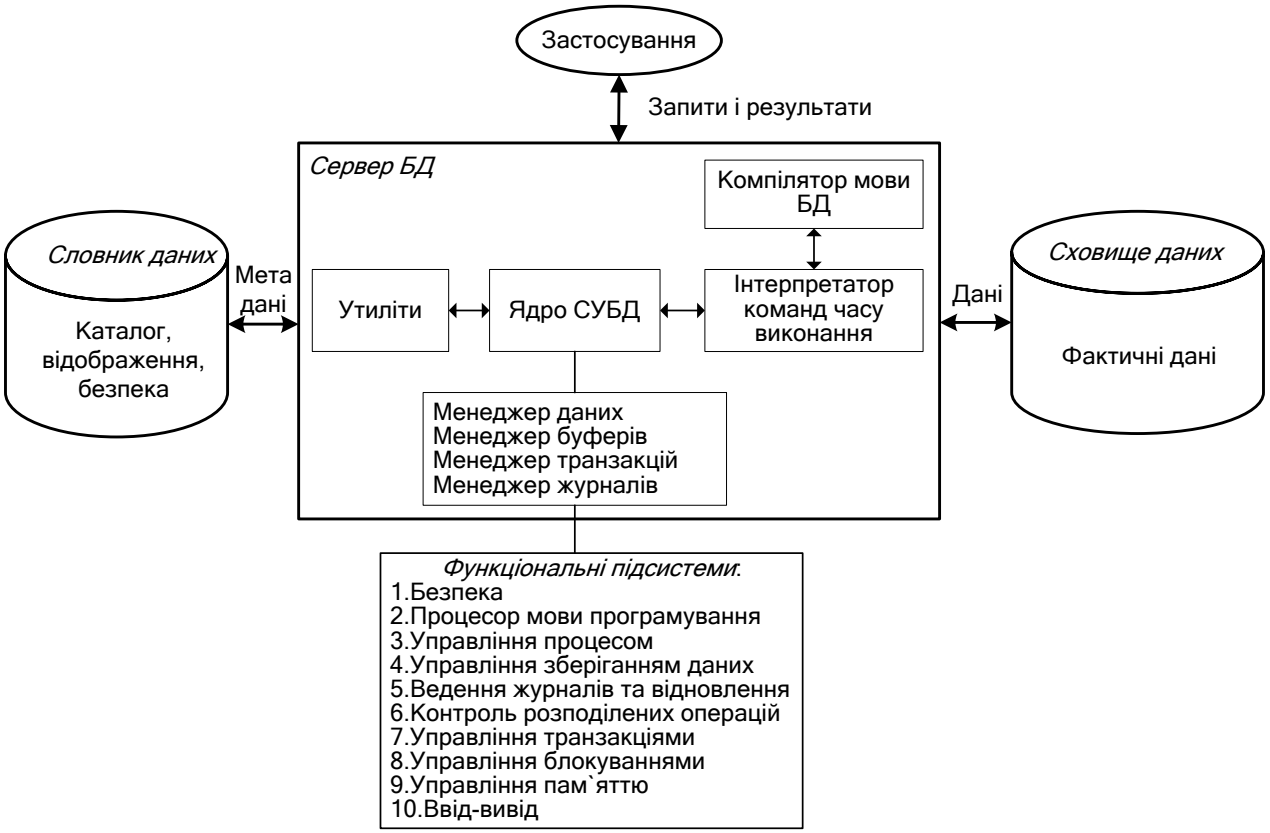



Рис. 12. Організаційна схема СУБД

*Ядро СУБД* відповідає за керування даними в зовнішній пам'яті, керування буферами оперативної пам'яті, керування транзакціями й журналізацію. Відповідно, можна виділити такі компоненти ядра (логічно, хоча в деяких системах ці компоненти виділяються явно), як менеджер даних, менеджер буферів, менеджер транзакцій і менеджер журналу.

*Менеджер буферів* – призначений для рішення задач ефективної буферізації оперативної пам'яті.

*Менеджер даних* – призначений для управління зовнішньою пам'ятью, забезпечення створення структур для даних, що зберігаються і допоміжних структур (індекси і т.ін.).

*Менеджер транзакцій* – підтримує механізми фіксації і відкату транзакцій, пов’язаний з менеджером буферів оперативної пам'яті і забезпечує зберігання всієї інформації, яка потрібна після збоїв системи.

*Менеджер журналів* – забезпечує реєстрацію відомостей про виконання транзакцій, про працюючих користувачів, про виконання застосування, про доступи до різних структур даних і т.ін.

Функції цих компонентів взаємозалежні, і для забезпечення коректної роботи СУБД всі ці компоненти повинні взаємодіяти по ретельно продуманих і перевірених

протоколах. Ядро СУБД має власний інтерфейс, недоступний користувачам прямо й використовуваний у програмах, вироблених компілятором SQL (або в підсистемі підтримки виконання таких програм) і утилітах БД. Ядро СУБД є основною резидентною частиною СУБД. При використанні архітектури " клієнт-сервер" ядро є основної складової серверної частини системи.

Основною функцією *компілятора мови БД* є компіляція операторів мови БД у деяку виконувану програму. Основною проблемою СУБД є те, що мови цих систем (а це, як правило, SQL) є непроцедурними, тобто в операторі такої мови визначається деяка дія над БД, але ця специфікація (визначення) не є процедурою, а лише описує в деякій формі умови здійснення бажаної дії. Тому компілятор повинен вирішити, яким образом виконувати оператор мови перш, ніж зробити програму. Застосовуються досить складні методи оптимізації операторів. Результатом компіляції є виконувана програма, що представляється в деяких системах у машинних кодах, але більш часто у виконуваному внутрішньому машинно-незалежному коді. В останньому випадку реальне виконання оператора виробляється із залученням *підсистеми підтримки часу виконання*, що представляє собою, по суті справи, інтерпретатор цієї внутрішньої мови.

Нарешті, в окремі *утиліти БД* звичайно виділяють такі процедури, які занадто накладно виконувати з використанням мови БД, наприклад, завантаження й вивантаження БД, збір статистики, глобальна перевірка цілісності БД і т.ін. Утиліти програмуються з використанням інтерфейсу ядра СУБД, а іноді навіть із проникненням усередину ядра.

СУБД можна розглядати як операційну систему (або підсистему), розроблену спеціально для керування доступом до даних; її *основні функції* - зберігання, вибірка й забезпечення безпеки даних. Подібно операційній системі, СУБД управляє доступом одночасно працюючих користувачів бази даних до деякого набору ресурсів. *Підсистеми СУБД* дуже схожі з відповідними підсистемами ОС і сильно інтегровані з надаваними базовою ОС сервісними функціями доступу на машинному рівні до таких ресурсів, як пам'ять, центральний процесор, пристрої й файлові структури.

СУБД підтримують власний список авторизованих користувачів і їхніх привілеїв; управляють кэшем пам'яті й сторінковим обміном; управляють блокуванням поділюваних ресурсів; приймають і планують виконання запитів користувача; управляють використанням табличного простору.

Таким чином, можна виділити дві важливі частини архітектури СУБД - *ядро*, що є програмним забезпеченням, і *словник* даних, що складається зі структур даних системного рівня, використовуваних ядром, що управляє базою даних.

Фундаментальне розходження між СУБД і файловими системами полягає в способі доступу до даних. СУБД дозволяє звертатися до фізичного даних у більше абстрактній, логічній формі, забезпечуючи легкість і гнучкість при розробці додатків. Додатка, що використовують СУБД, звертаються до даних через комп'ютер бази даних без безпосередньої залежності від фактичного джерела даних, ізолюючи додаток від деталей фізичних структур даних. така незалежність даних можлива завдяки словника даних СУБД, що зберігає метадані (дані про дані) для всіх об'єктів, розташованих у базі даних. *Словник даних* СУБД - перелік об'єктів БД, що зберігається в спеціальній області БД і ведеться винятково ядром СУБД. Запити читання й відновлення БД обробляються ядром з використанням інформації зі словника. Інформація в словнику призначена для підтвердження існування об'єктів, забезпечення доступу до них і опису фактичного фізичного розташування в пам'яті.

*1.2.3. Функціональна схема побудови СУБД*

Функціонально до складу СУБД включають *сервер БД* і *програмне забезпечення мережної взаємодії* клієнта й сервера (рис. 13).

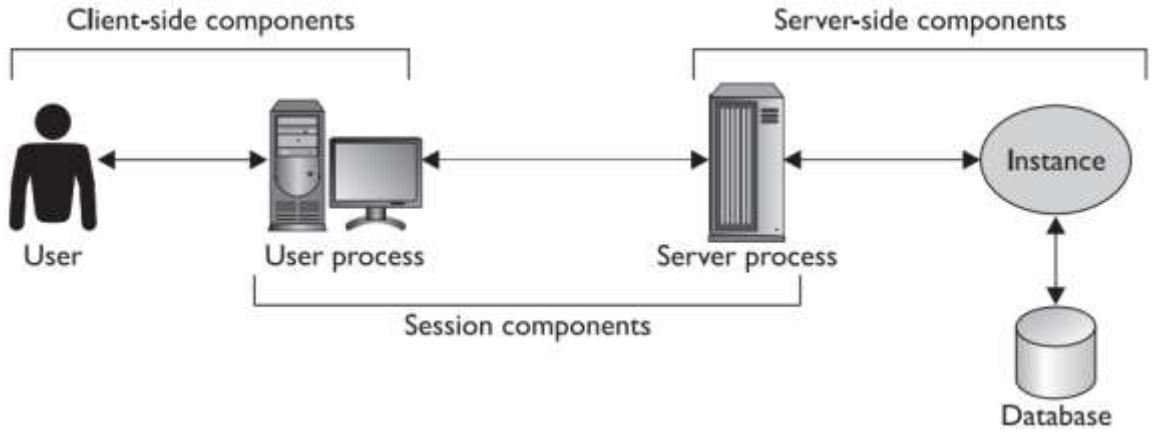



Рис. 13. Схема взаємодії СУБД з користувачем

*Сервер БД* становить із себе області пам'яті для роботи СУБД, функціональні процеси й накопичувачі даних на зовнішніх носіях – дисках (рис. 14). Сукупність областей пам'яті й процесів утворять екземпляр БД, запущений на ЕОМ сервера БД.

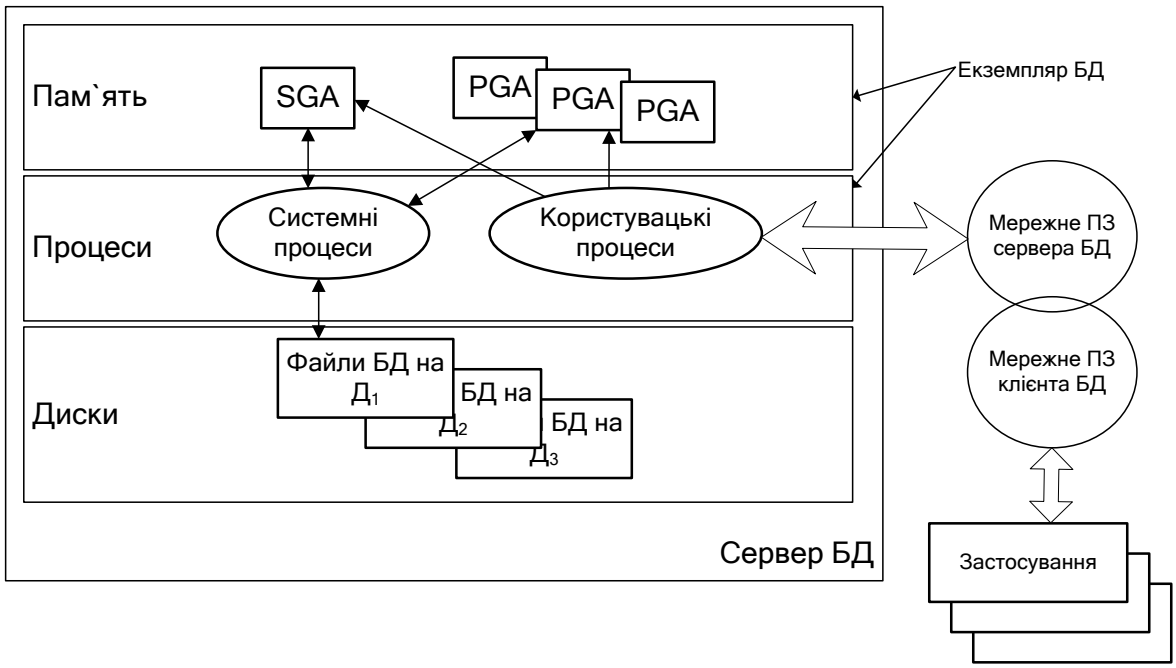



Рис. 14. Структурна схема сервера БД

*Системна область* SGA складається з Кешів:
- буферів даних;
- журнального Кеша;
- Кеша словника даних;
- області SQL з курсорами;

*Системні процеси* (рис. 15):

1) Основні - DBWR (запис блоків даних Writer-Reader), LGWR (запис журналу Logging), SMON (контроль транзакцій), PMON (контроль користувальницьких процесів);

2) Допоміжні - ARCH (архівування), RECO (відновлення), LCK (розподілений доступ).

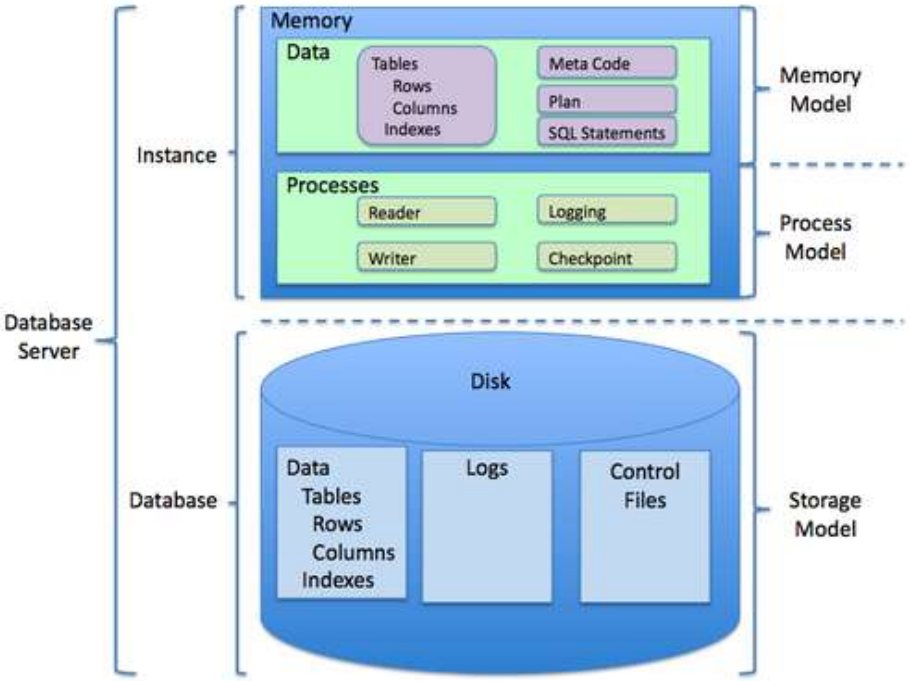



Рис. 15. Системні компоненти сервера БД

*Загальний простір файлів* БД сегментоване (поділено на простори) і складається з:

- сегментів таблиць;
- сегментів індексів;
- сегментів відкоту;
- тимчасових сегментів.

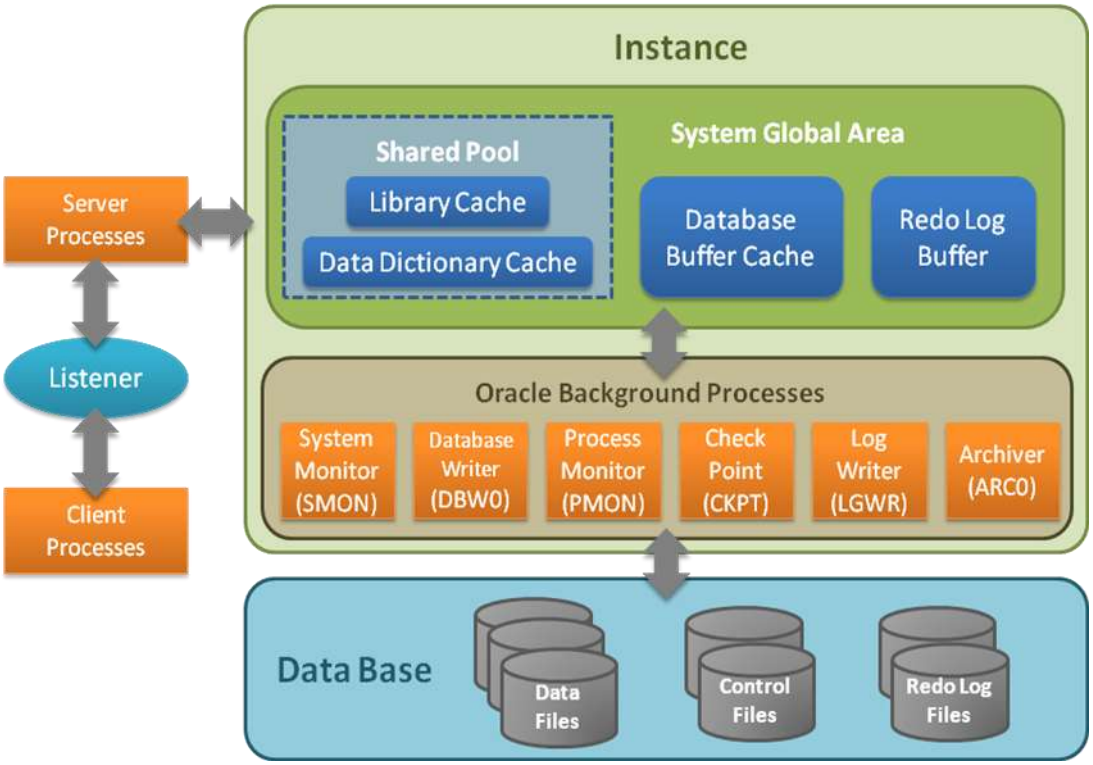



Рис. 16. Функціональна схема сервера БД

Отже, *база даних* - зібрання даних, між якими існують змістовні зв'язки. Фізичне розташування й реалізація бази даних прозорі для прикладних програм; фізичну базу даних можна переміщати й реорганізовувати й це не зробить впливу на працездатність програм.

*Фізично база даних* - не більш ніж набір файлів десь на диску. Розташування цих файлів несуттєво для функціонування (хоча важливо для продуктивності) бази даних.

*Логічно база даних* - це безліч користувальницьких розділів, кожний з яких ідентифікується ім'ям користувача (з паролем), унікальним у даної БД.

Існують три основні групи файлів на диску, що становлять базу даних:

1)Файли бази даних;

2)Керуючі файли;

3)Журнальні файли.

Найбільш важливі з них - файли бази даних, де розташовуються властиво дані. Керуючі й журнальні файли підтримують функціонування архітектури (рис. 17). Для доступу до даних БД всі трьох набору файлів повинні бути присутнім, бути відкритими й доступними.

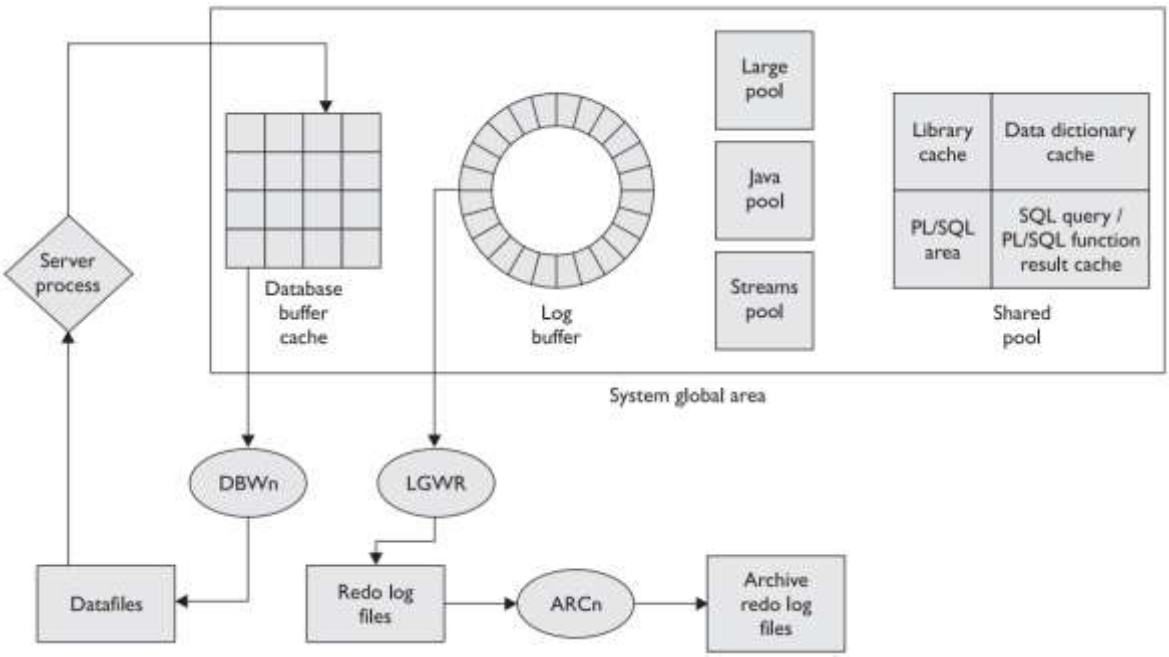



Рис. 17. Архітектура взаємодії процесів сервера БД

## Тема 1.3. Особливості організації даних у сучасних інформаційних системах

### *1.3.1. Сучасна технологія обробки інформації «клієнт-сервер»*

Ефективність функціонування АІС багато в чому залежить від її архітектури взаємодії з БД, тобто схеми розподілу функцій - ведення, обробки даних та їх подання.

Структура інформації найчастіше дуже складна, і хоча структури даних різні в різних інформаційних системах, між ними часто буває багато спільного. На початковому етапі використання обчислювальної техніки для управління інформацією проблеми структуризації даних вирішувалися індивідуально в кожній інформаційній системі. Проводилися необхідні надбудови над файловими системами у вигляді бібліотек обслуговуючих програм, подібно до того, як це робиться в компіляторах.

Але оскільки інформаційні системи вимагають складних структур даних, ці додаткові індивідуальні засоби керування даними були істотною частиною інформаційних систем і практично повторювалися від однієї системи до іншої. Дуже скоро стало зрозуміло, що неможливо обійтися загальною бібліотекою програм, що реалізує над стандартною базовою файловою системою більш складні методи зберігання даних. Прагнення виділити і узагальнити загальну частина інформаційних систем, відповідальну за управління складно структурованими даними, стало спонукальною причиною створення СУБД. Таким чином, СУБД стали вирішувати безліч проблем, які важко або взагалі неможливо вирішити при використанні файлових систем.

В результаті функціональні частини АІС стали розміщувати на одному або на декількох обчислювальних ресурсах. В залежності від кількості ресурсів можливі такі варіанти використання програмних засобів: додатка-застосування і СУБД.

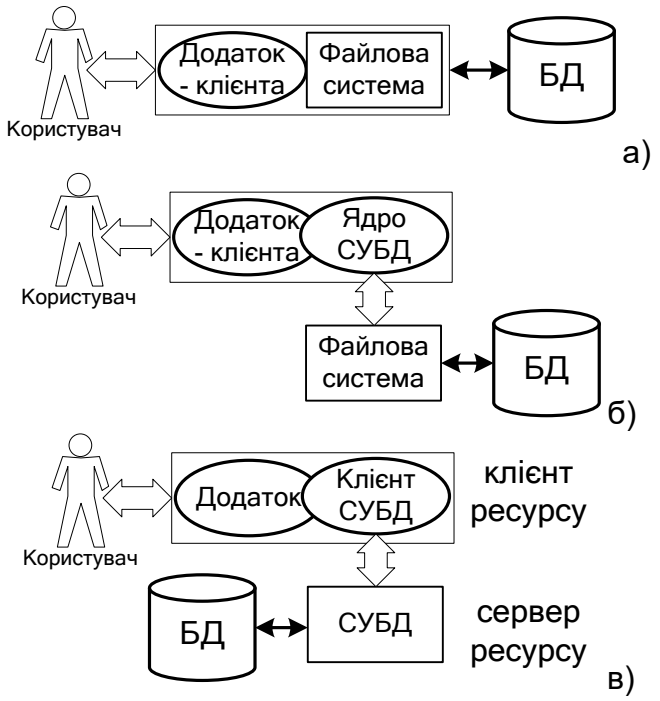



Рис. 18. Варіанти використання обчислювальних ресурсів

Створення незалежних застосувань дозволяє звертатися до БД без СУБД. Такий варіант дозволяє підвищити швидкість роботи застосування, зменшити об'єм

використанної пам'яті. Недоліки такого підходу пов'язані з трудомісткістю доробки застосувань, відсутністю стандартних уніфікованих засобів по обслуговуванню БД.

У другому варіанту взаємодія користувача і БД виконується за допомогою інтегрованого використання застосування і ядра СУБД. Такий підхід дозволяє підвищити швидкість розробки застосування, зменшити об'єм необхідної пам'яті для роботи.

У третьому варіанті взаємодія користувача і БД виконується за допомогою застосування з‘єднаного напряму через користувацький інтерфейс з СУБД. Такий підхід дозволяє забезпечити незалежність застосування від способів зберігання і звернення до даних і уніфікувати правила взаємодії з БД для багатох користувачів.

В даний час перспективною є обчислювальна архітектура типу «клієнт-сервер» за принципом виділення обслуговуючого і обслуговується вузла комп'ютерної мережі ресурсів.

*Обслуговуючим вузлом - сервером* називається комп'ютер або програма, що управляє ресурсом в комп'ютерній мережі.

*Обслуговуваним вузлом - клієнтом* називається комп'ютер або програма, яка використовує цей ресурс. Якщо керованим ресурсом є база даних, то відповідний сервер називається *сервером бази даних*, а клієнт - *додатком* бази даних у вигляді AIC.

*Перевага архітектури* «клієнт-сервер»: вдале поєднання централізованого зберігання, обслуговування, колективного доступу до даних з індивідуальною обробкою інформації в додатку.

Реальне поширення архітектури "клієнт-сервер" стало можливим завдяки розвитку обчислювальних засобів, комп‘ютерних мереж та широкого впровадження в практику концепції відкритих систем.

### *1.3.1.1. Концепція відкритих систем*

Основним змістом підходу *відкритих систем* є спрощення комплексування обчислювальних систем за рахунок міжнародної та національної стандартизації апаратних і програмних інтерфейсів. Головною спонукальною причиною розвитку концепції відкритих систем став повсюдний перехід до використання спочатку локальних, а потім корпоративних, глобальних комп'ютерних мереж та й ті проблеми комплексування апаратно-програмних засобів, які викликав цей перехід. У зв'язку з бурхливим розвитком технологій глобальних комунікацій відкриті системи набувають ще більшого значення і масштабності.

Ключовою фразою відкритих систем, спрямованої в бік користувачів, є незалежність від конкретного постачальника. Орієнтуючись на продукцію компаній, які дотримуються стандартів відкритих систем, споживач, який отримує будь-який продукт такої компанії, не потрапляє до неї в залежність. Він може продовжити нарощування потужності своєї системи шляхом придбання продуктів будь-яких інший компанії, що дотримуються стандартів. Причому це стосується як апаратних, так і програмних засобів і не є необґрунтованою декларацією.

Практичною опорою системних і прикладних програмних засобів відкритих систем є стандартизована операційна система. В даний час такою системою є UNIX. Фірмам-постачальникам різних варіантів ОС UNIX в результаті тривалої роботи вдалося прийти до угоди про основні стандарти цієї операційної системи. Зараз всі поширені версії UNIX в основному сумісні по частині інтерфейсів, що надаються прикладним (а в більшості випадків і системним) програмістам. Як здається, незважаючи на появу системи Windows, що претендує на стандарт, саме UNIX залишиться основою відкритих систем у найближчі роки.

Технології та стандарти відкритих систем забезпечують реальну і перевірену практикою можливість виробництва системних і прикладних програмних засобів з властивостями *мобільності* (*portability*) і *інтероперабельності* (*interoperability*).

*Мобільності* означає порівняльну простоту перенесення програмної системи в широкому спектрі апаратно-програмних засобів, які відповідають стандартам.

*Інтероперабельність* означає спрощення комплексування нових програмних систем на основі використання готових компонентів зі стандартними інтерфейсами.

Використання підходу відкритих систем вигідно і виробникам, і користувачам. Перш за все відкриті системи забезпечують природне рішення проблеми поколінь апаратних і програмних засобів. Виробники таких засобів не змушені вирішувати всі проблеми заново; вони можуть принаймні тимчасово продовжувати комплексування системи, використовуючи існуючі компоненти.

При цьому виникає новий рівень конкуренції. Всі виробники зобов'язані забезпечити деяке стандартне середовище, але змушені домагатися її якомога кращої реалізації. Звичайно, через якийсь час існуючі стандарти почнуть грати роль стримування прогресу, і тоді їх доведеться переглядати.

Перевагою для користувачів є те, що вони можуть поступово замінювати компоненти системи на більш досконалі, що не втрачаючи працездатності системи. Зокрема, в цьому криється рішення проблеми поступового нарощування обчислювальних, інформаційних та інших потужностей комп'ютерної системи.

В *основе концепції відкритих систем* лежить відома ідея *поділу ресурсів*. Розвиток цієї ідеї призводить до функціонального виділення компонентів мережі: *сервера* і *клієнта*. Прийнято називати *клієнтом* мережі, той компонент, який запитує послугу у деякого сервера і сервером - компонент, який надає послуги деяким клієнтам.

Сервер мережі надає послуги-ресурси робочим станціям і/або іншим серверам. Робоча станція призначена для безпосередньої роботи користувача і володіє ресурсами, відповідними локальним потребам даного користувача. Сервер мережі повинен володіти ресурсами, відповідними його функціональному призначенню і потребам мережі.

Прикладами сервера ресурсу можуть служити:

- сервер телекомунікацій, що забезпечує послуги зв'язку даної мережі з зовнішнім світом;
- обчислювальний сервер, що дає можливість виконувати обчислення, які неможливо здійснити на робочих станціях;
- дисковий сервер, що володіє розширеними ресурсами зовнішньої пам'яті і надає їх у використання робочим станціями і, можливо, іншим серверам;
- файловий сервер, що підтримує загальне сховище файлів для всіх робочих станцій;
- поштовий сервер, що підтримує передачу повідомлень користувачів робочих станцій і , можливо, до інших серверів;
- сервер баз даних це фактично звичайна СУБД, що приймає запити із мережі і повертає результати їх обробки на робочі станції.

У загальному випадку, щоб прикладна програма, що виконується на робочій станції, могла запросити послугу у деякого сервера, як мінімум потрібно деякий інтерфейсний програмний шар, що підтримує такого роду взаємодію (неприродно вимагати, щоб прикладна програма безпосередньо користувалася примітивами транспортного рівня мережі). З цього, власне, і випливають основні принципи системної архітектури "клієнт-сервер".

Система розбивається на дві частини (рис. 19), які можуть виконуватися в різних вузлах мережі, - клієнтську та серверну частини. Прикладна програма або кінцевий користувач взаємодіють з клієнтською частиною системи, яка в

найпростішому випадку забезпечує просто над мережевий інтерфейс. Клієнтська частина системи при необхідності звертається по мережі до серверної частини. Зауважимо, що в розвинених системах мережеве звернення до серверної частини може і не знадобитися, якщо система може передбачати потреби користувача, і в клієнтської частини містяться дані, здатні задовольнити його наступний запит.

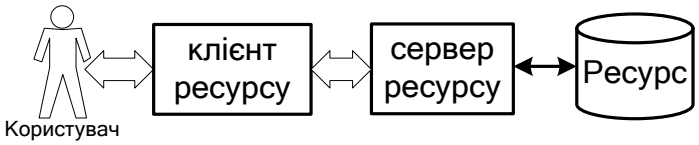



Рис. 19. Архітектура «клієнт-сервер»

Інтерфейс серверної частини визначений і фіксований. Тому можливе створення нових клієнтських частин існуючої системи (приклад інтероперабельності на системному рівні).

Основною проблемою систем, заснованих на архітектурі "клієнт-сервер", є те, що відповідно до концепції відкритих систем від них потрібна мобільність в якомога ширшому класі апаратно-програмних рішень відкритих систем. Навіть якщо обмежитися UNIX-орієнтованими локальними мережами, в різних мережах застосовується різна апаратура та протоколи зв'язку. Спроби створення систем, що підтримують всі можливі протоколи, призводить до їх перевантаження мережевими деталями на шкоду функціональності.

Ще більш складний аспект цієї проблеми пов'язаний з можливістю використання різних представлень даних в різних вузлах неоднорідною локальної мережі. У різних комп'ютерах може існувати різна адресація, подання чисел, кодування символів і т.і. Це особливо істотно для серверів високого рівня: телекомунікаційних, обчислювальних, баз даних.

Спільним рішенням проблеми мобільності систем, заснованих на архітектурі "клієнт-сервер" є опора на програмні пакети, що реалізують протоколи віддаленого виклику процедур (RPC - Remote Procedure Call). При використанні таких коштів звернення до сервісу в віддаленому вузлі виглядає як звичайний виклик процедури. Засоби RPC, в яких, природно, міститься вся інформація про специфіку апаратури локальної мережі та мережевих протоколів, переводить виклик в послідовність мережевих взаємодій. Тим самим, специфіка мережного середовища і протоколів прихована від прикладного програміста.

При виклику віддаленої процедури програми RPC виробляють перетворення форматів даних клієнта в проміжні машинно-незалежні формати і потім перетворення у формати даних сервера. При передачі відповідних параметрів виробляються аналогічні перетворення.

Якщо система реалізована на основі стандартного пакета RPC, вона може бути легко перенесена в будь-яку відкриту середу.

*1.3.1.2. Клієнти і сервери баз даних*

Термін "сервер баз даних" зазвичай використовують для позначення всієї СУБД, заснованої на архітектурі "клієнт-сервер", включаючи і серверну, і клієнтську частини. Такі системи призначені для зберігання і забезпечення доступу до баз даних.

Хоча зазвичай одна база даних цілком зберігається в одному вузлі мережі і підтримується одним сервером, сервери баз даних являють собою просте й дешеве наближення до розподілених баз даних, оскільки загальна база даних доступна для всіх користувачів локальної мережі.

Доступ до бази даних від прикладної програми або користувача виробляється шляхом звернення до клієнтської частини системи. В якості основного інтерфейсу між клієнтської і серверної частинами виступає мова баз даних SQL.

Ця мова по суті справи представляє собою поточний стандарт інтерфейсу СУБД у відкритих системах. Збірна назва SQL-сервер відноситься до всіх серверів баз даних, заснованих на SQL. Дотримуючись вимог при програмуванні можна створювати прикладні інформаційні системи, мобільні в класі SQL-серверів.

Сервери баз даних, інтерфейс яких базується виключно на мові SQL, мають свої переваги і свої недоліки. Очевидна перевага - стандартність інтерфейсу. У межі, хоча поки це не зовсім так, клієнтські частини будь-якої SQL-орієнтованої СУБД могли б працювати з будь-яким SQL-сервером незалежно від того, хто його зробив.

Недолік теж досить очевидний. При такому високому рівні інтерфейсу між клієнтської і серверної частинами системи на стороні клієнта працює занадто мало програм СУБД. Це нормально, якщо на стороні клієнта використовується малопотужна робоча станція. Але якщо клієнтський комп'ютер має достатню потужність, то часто виникає бажання покласти на нього більше функцій управління базами даних, розвантаживши сервер, який є вузьким місцем всієї системи.

Одним з перспективних напрямків СУБД є гнучке конфігурування системи, при якому розподіл функцій між клієнтською і призначеної для користувача частинами СУБД визначається при установці системи (рис. 20). Згадувані вище протоколи віддаленого виклику процедур особливо важливі в системах управління базами даних, заснованих на архітектурі "клієнт-сервер".

По-перше, використання механізму віддалених процедур дозволяє дійсно перерозподіляти функції між клієнтської і серверної частинами системи, оскільки в тексті програми віддалений виклик процедури нічим не відрізняється від віддаленого виклику, і отже, теоретично будь-який компонент системи може розташовуватися і на стороні сервера, і на стороні клієнта.

По-друге, механізм віддаленого виклику приховує відмінності між взаємодіючими комп'ютерами. Фізично неоднорідна локальна мережа комп'ютерів приводиться до логічно однорідної мережі взаємодіючих програмних компонентів. В результаті користувачі не зобов'язані серйозно піклуватися про разову закупівлю сумісних серверів і робочих станцій.

У типовому на сьогоднішній день випадку на стороні клієнта СУБД працює тільки таке програмне забезпечення, яке не має безпосереднього доступу до баз даних, а звертається для цього до сервера з використанням мови SQL.

На даний час використовують два варіанта архітектури взаємодії застосувань користувачів з БД: дворівневу та трирівневу схеми.

*Дворівнева архітектура* передбачає класичну схему технології «клієнт-сервер» - клієнтські програми за допомогою сполучного програмного забезпечення серверу БД здійснюють зв'язок з базою даних із розділенням функцій ведення даних і їх обробки з поданням кінцевому користувачеві.

*Трирівнева архітектура* передбачає розподіл функцій обробки і подання за різними ресурсами та виділення у процесі взаємодії кінцевого користувача клієнтських веб-додатків, серверу додатків, побудованого за допомогою "проміжного" програмного забезпечення та серверу БД.

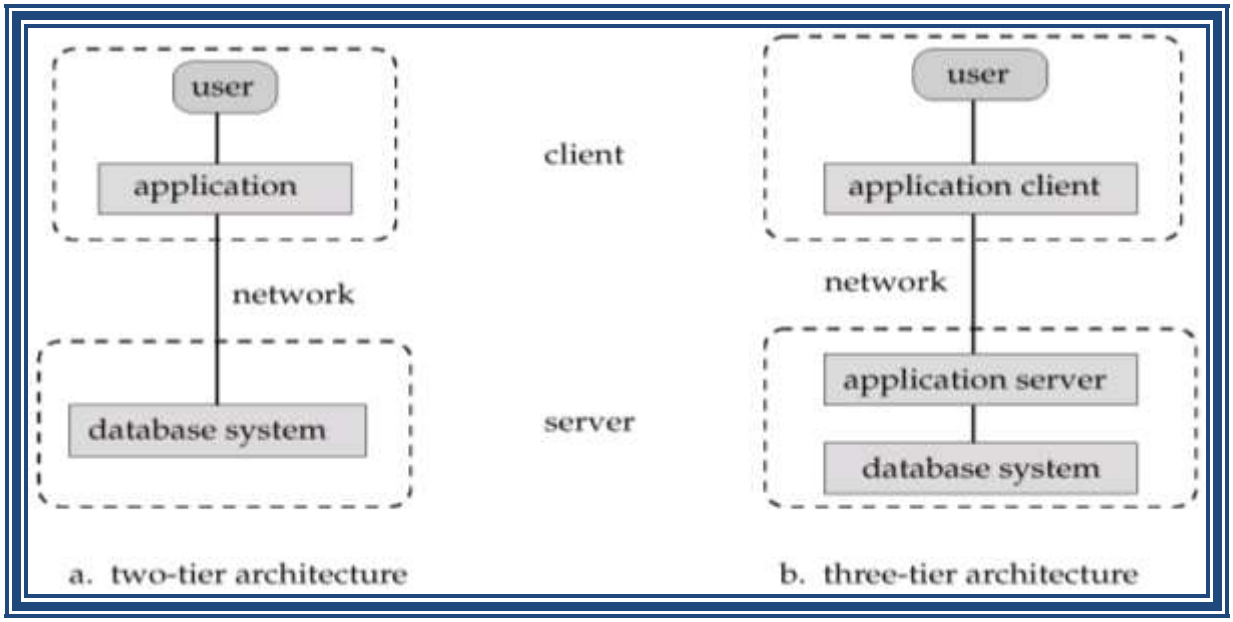



Рис. 20. Варіанти архітектури «клієнт-сервер» у базах даних

У деяких випадках хотілося б включити до складу клієнтської частини системи деякі функції для роботи з "локальним кешем" бази даних, тобто з тією її частиною, яка інтенсивно використовується клієнтської прикладною програмою. У сучасній технології це можна зробити тільки шляхом формального створення на стороні клієнта локальної копії сервера бази даних і розгляду всієї системи як набору взаємодіючих серверів.

З іншого боку, іноді хотілося б перенести більшу частину прикладної системи на сторону сервера, якщо різниця в потужності клієнтських робочих станцій і сервера надто велика. В общем-то, при використанні RPC це зробити неважко. Але потрібно, щоб базове програмне забезпечення сервера дійсно дозволяло це. Зокрема, при використанні ОС UNIX проблеми практично не виникають.

Очевидно, що вимоги до апаратури і програмного забезпечення клієнтських і серверних комп'ютерів розрізняються залежно від виду використання системи.

Якщо поділ між клієнтом і сервером досить жорстке (як в більшості сучасних СУБД), то користувачам, які працюють на робочих станціях або персональних комп'ютерах, абсолютно все одно, яка апаратура та операційна система працюють на сервері, аби він справлявся з виникаючим потоком запитів.

### *1.3.2. Сучасні напрямки досліджень в організації СУБД*

Хоча віднесення СУБД до того чи іншого класу в даний час може бути виконано тільки умовно (наприклад, іноді об'єктно-орієнтовану СУБД відносять до систем наступного покоління), можна відзначити три напрямки в області СУБД наступного покоління:

1.*Напрямок Postgres*. Основна характеристика: максимальне слідування (наскільки це можливо з урахуванням нових вимог) відомим принципам організації СУБД.

2.*Напрямок Exodus/Genesis*. Основна характеристика: створення, власне, не системи, а генератора систем, найбільш повно відповідають потребам додатків. Рішення досягається шляхом створення наборів модулів зі стандартизованими інтерфейсами, причому ідея поширюється, аж до самих базисних шарів системи.

3.*Напрямок Starburst*. Основна характеристика: досягнення розширення системи і її пристосування до потреб конкретних програм шляхом використання

стандартного механізму управління правилами. По суті справи, система являє собою деякий інтерпретатор системи правил і набір модулів-дій, що викликаються відповідно до цих правил. Можна змінювати набори правил (існує спеціальна мова завдання правил) або змінювати дії, підставляючи інші модулі з тим же інтерфейсом.

В цілому, СУБД наступного покоління - це прямі спадкоємці реляційних систем. Проте, різні напрямки систем третього покоління варто розглянути окремо, оскільки вони володіють деякими різними характеристиками.

*1.3.2.1. Напрям Postgres: орієнтація від розширеної реляційної моделі до об'єктно-орієнтованих БД*

Одним з основних положень реляційної моделі даних є вимога нормалізації відносин: поля кортежів можуть містити лише атомарні значення. Для традиційних додатків реляційних СУБД - банківських систем, систем резервування тощо - це зовсім не обмеження, а навіть перевага, що дозволяє проектувати економні по пам'яті БД з гранично зрозумілою структурою. Запити з сполуками в таких системах порівняно рідкісні, для динамічної підтримки цілісності використовуються відповідні засоби SQL.

Однак з появою ефективних реляційних СУБД їх стали намагатися використовувати і в менш традиційних прикладних системах - САПР, системах штучного інтелекту і т.д. Такі системи зазвичай оперують складно структурованими об'єктами, для реконструкції яких з плоских таблиць реляційної БД доводиться виконувати запити, майже завжди вимагають з'єднання відносин. Відповідно до вимог розробників нетрадиційних додатків з'явився напрям досліджень баз складних об'єктів. Основний зміст цього напрямку полягає в тому, що в руки проектувальників даються настільки ж потужні й гнучкі засоби структуризації даних, як ті, які були притаманні ієрархічним і мережевих систем баз даних.

Важливою відмінністю є те, що в системах баз даних, які підтримують складні об'єкти, зберігається чітка межа між логічним і фізичним уявленнями таких об'єктів. Зокрема, для будь-якого складного об'єкта (довільної складності) повинна забезпечуватися можливість переміщення або копіювання його як єдиного цілого з однієї частини бази даних в іншу її частину або навіть в іншу базу даних. Це дуже велика область досліджень, в якій порушуються питання моделей даних, структур даних, мов запитів, управління транзакціями, журналізації і т.д. Багато в чому ця область стикається з областю об'єктно-орієнтованих БД.

Близьке, засноване на інших принципах напрямок представлено системами баз даних, заснованих на реляційної моделі, в якій не обов'язково підтримується перша нормальна форма відносин. Нагадаємо, що вимога атомарности значень, які можуть зберігатися в елементах кортежів відносин, є базовою вимогою класичної реляційної моделі. Приведення вихідного табличного представлення предметної області до "плоскому" виду є обов'язковим першим кроком в процесі проектування реляційної бази даних на основі принципів нормалізації. З іншого боку, абсолютно очевидно, що таке "сплощення" таблиць хоча і є необхідною умовою отримання неізбиточной і "правильної" схеми реляційної бази даних, в подальшому потенційно викликає виконання численних з'єднань, наявність яких може звести нанівець всі переваги "хорошою" схеми бази даних.

В "ненормалізованих" реляційних моделях даних допускається зберігання в якості елемента кортежу кортежів (записів), масивів (регулярних індексованих множин даних), регулярних множин елементарних даних, а також відносин. При цьому така вкладеність може бути, по суті, необмеженою. Якщо уважно продумати ці ідеї, то стане зрозуміло, що вони призводять (тільки) до логічно відокремленим (від фізичного представлення) можливостям ієрархічної моделі даних. До теперішнього часу фактично повністю сформовано теоретичне обгрунтування розширення

реляційних баз даних з відмовою від нормалізації і переході на ненормалізованих модель, наприклад, у вигляді об'єктно-орієнтованої моделі.

Абстрактні типи даних. Однією з найбільш відомих СУБД третього покоління є система Postgres. У Postgres реалізовані багато цікавих засоби: підтримується темпоральна модель зберігання і доступу до даних і в зв'язку з цим абсолютно переглянутий механізм журналізації змін, відкатів транзакцій і відновлення БД після збоїв; забезпечується потужний механізм обмежень цілісності; підтримуються ненормалізовані відношення (робота в цьому напрямку почалася ще в середовищі Ingres), хоча і в досить дивний спосіб: в поле відношення може зберігатися динамічно виконуваний запит до БД.

Одна властивість системи Postgres зближує її з властивостями об'єктно-орієнтованих СУБД. У Postgres допускається зберігання в полях відносин даних абстрактних, визначених користувачами типів. Це забезпечує можливість впровадження поведінкового аспекту в БД, тобто вирішує ту ж задачу, що і об'єктно-орієнтовані БД, хоча, звичайно, семантичні можливості моделі даних Postgres значно слабше, ніж у об'єктно-орієнтованих моделей даних. Основна різниця полягає в тому, що системи класу Postgres не передбачають наявності мови програмування, однаково розуміється як зовнішньою системою програмування, так і системою управління базами даних. Якщо з використанням такої системи програмування визначаються типи даних, що зберігаються в базі даних, то СУБД виявляється не в змозі контролювати безпеку цих визначень, тобто відсутня гарантія, що при виконанні процедур абстрактних типів даних не буде зруйнована сама база даних.

Підтримка історичної інформації і темпоральних запитів. Звичайні БД зберігають миттєвий знімок моделі предметної області. Будь-яка зміна в поточному моменті часу деякого об'єкта призводить до недоступності стану цього об'єкта в попередній момент часу. Найцікавіше, що насправді в більшості розвинених СУБД попередній стан об'єкта зберігається в журналі змін, але можливості доступу з боку користувача немає.

Звичайно, можна явно ввести в збережені відношення явний тимчасової атрибут і підтримувати його значення на рівні додатків. Більш того, в більшості випадків так і роблять. Недарма в стандарті SQL з'явилися спеціальні типи даних date і time. Але в такому підході є кілька недоліків: СУБД не знає семантики часового поля відношення і не може контролювати коректність його значень; з'являється додаткова надмірність зберігання (попереднє стан об'єкта даних зберігається і в основний БД, і в журналі змін); мови запитів реляційних СУБД не пристосовані для роботи із часом.

Існує окремий напрямок досліджень і розробок в області темпоральних БД. У цій області досліджуються питання моделювання даних, мови запитів, організація даних у зовнішній пам'яті і т.д. Основна теза темпоральних систем полягає в тому, що для будь-якого об'єкта даних, створеного в момент часу t1 і знищеного в момент часу t2, в БД зберігаються (і доступні користувачам) всі його стану в тимчасовому інтервалі [t1, t2].

Дослідження і побудови прототипів темпоральних СУБД зазвичай виконуються на основі деякої реляційної СУБД. Темпоральна СУБД - це надбудова над реляційної системою. Звичайно, це не найкращий спосіб реалізації з точки зору ефективності, але він простий і дозволяє робити досить глибокі дослідження.

Прикладом кардинального (але, може бути, передчасного) вирішення проблеми темпоральних БД може служити СУБД Postgres. Головними особливостями системи управління пам'яттю в Postgres є, по-перше, те, що в ній не ведеться звичайна журналізация змін бази даних і миттєво забезпечується коректний стан бази даних після перезавантаження системи із втратою стану оперативної пам'яті. По-друге, система управління пам'яттю підтримує історичні дані.

Запити можуть містити часові характеристики об'єктів. Реалізаційно ці два аспекти пов'язані.

Основне рішення полягає в тому, що при модифікаціях кортежу зміни проводяться не на місці його зберігання, а заводиться новий запис, куди поміщаються змінені поля. Ця запис містить, крім того, дані, що характеризують транзакцію, що виробляла зміни (в тому числі і час її завершення), і підшивається до списку до постійно змінюваних кортежів. В системі підтримується унікальна ідентифікація транзакцій і є спеціальна таблиця транзакцій, що зберігається в стабільній пам'яті. Таким чином, після збоїв просто не слід звертати увагу на хвостові записи списків, що відносяться до незакінчений транзакціях. Синхронізація підтримується на основі звичайного двохфазного протоколу захоплень.

Окремий компонент системи здійснює архівацію об'єктів бази даних. Він виробляє збірку розрослися списків змінювалися кортежів і записує їх в область архівного зберігання. До цієї області теж можуть адресуватися запити, але вже тільки на читання.

Система орієнтована на використання оптичних дисків з разової записом і стабільної оперативної пам'яті (хоча б невеликого обсягу). При наявності таких технічних засобів вона виграє по ефективності навіть при роботі в традиційному режимі в порівнянні зі схемою з журналізацией. Однак можлива робота і на традиційній апаратурі, тоді ефективність системи дещо поступається традиційним схемам.

Відповідні можливості роботи з історичними даними закладені в мову Postquel. Можлива вибірка інформації, що зберігалася в базі даних в зазначений час, в зазначеному часовому інтервалі і т.д. Крім того, є можливість створювати версії відношень і допускається їх подальша модифікація з урахуванням змін основних варіантів.

*1.3.2.2. Напрям Exodus/Genesis: генерація систем баз даних, орієнтованих на додатки*

Гіпотеза цього напрямку полягає в наступному: ніколи не стане можливим створити універсальну систему управління базами даних, яка буде достатня і не надлишкова для застосування в будь-якому додатку. Тому дуже заманливо проводити не закінчені універсальні СУБД, а щось на кшталт компіляторів компіляторів (compiler compiler), що дозволяють зібрати систему баз даних, орієнтовану на конкретний додаток (або клас додатків). Бажано вміти генерувати систему баз даних, можливості (і відповідні накладні витрати) якої в достатній мірі відповідають потребам програми. На сьогоднішній день на комерційному ринку такі "генераційні" системи відсутні (наприклад, при виборі сервера системи Oracle ви не можете відмовитися від будь-яких непотрібних для вашого застосування його властивостей або зажадати наявності деяких додаткових властивостей). Однак існують як мінімум два експериментальних прототипу - Genesis і Exodus.

Обидві ці генераційні системи засновані насамперед на принципах модульності і точного дотримання встановлених інтерфейсів. По суті справи, системи складаються з мінімального ядра (розвиненої файлової системи в разі Exodus) і технологічного механізму програмування додаткових модулів. У проекті Exodus цей механізм грунтується на системі програмування E, яка є простим розширенням C ++, що підтримує стабільне зберігання даних у зовнішній пам'яті. Замість готової СУБД надається набір "напівфабрикатів" з узгодженими інтерфейсами, з яких можна згенерувати систему, що максимально відповідає потребам додатків.

### *1.3.2.3. Напрям Starburst: оптимізація запитів, що керована правилами*

Оптимізатор запитів - це найбільш громіздкий, складний і критичний компонент СУБД. Всі розробники систем управління базами даних згодні з тим, що на оптимізації запитів економити не можна. Чим більша кількість варіантів виконання запиту аналізується і чим більш точні оцінки вартості плану виконання запиту застосовуються, тим більш імовірно, що запит буде виконаний ефективно.

Головна неприємність, пов'язана з оптимізаторами запитів, полягає в тому, що відсутня прийнята технологія їх програмування. Зазвичай оптимізатор являє собою аморфний набір відносно незалежних процедур, які жорстко пов'язані з іншими компонентами компілятора. З цієї причини дуже важко змінювати стратегії оптимізації або якісно їх розширювати (робити це доводиться, оскільки оптимізація взагалі і оптимізація запитів, зокрема, в принципі є емпіричною дисципліною, а хороші емпіричні алгоритми з'являються тільки з часом).

Є компромісні рішення, що не виводять за межі традиційної технології виробництва компіляторів. В основному всі вони пов'язані із застосуванням тих чи інших інструментальних засобів, що забезпечують автоматизацію побудови компіляторів. Серед них відзначимо технологію, застосовану в сімействі компіляторів gcc. Основним виробничим гідністю gcc є застосування єдиної мови як засіб внутрішнього представлення програми. Високорівнева ліспоподобний мову RTL використовується на всіх фазах компіляції gcc, що дозволяє застосовувати одні і ті ж перетворюють процедури на різних стадіях оптимізації програми (аж до стадії машинно-залежних оптимізацій).

У пакеті gcc забезпечується набір універсальних, настроюються процедур перетворення графів внутрішнього представлення програми. У певному сенсі gcc можна розглядати як спеціалізований мова для написання компіляторів (компіляторів будь-яких мов, а не лише процедурних мов програмування або декларативних мов баз даних). Як стверджується, gcc дозволяє підвищити продуктивність праці розробників компіляторів в 2-3 рази.

Однак найбільш революційний підхід був застосований в експериментальній постреляціонной системі компанії IBM Starburst. Система Starburst заснована на застосуванні виробничої системи. Ця система є, по суті, віртуальною машиною, в якій виконуються всі компоненти СУБД, починаючи від компілятора мови баз даних (розширеного варіанту мови SQL) і закінчуючи підсистемою безпосереднього виконання запитів. Сама СУБД являє собою набір продукційних правил, кожне з яких викликається продукционной системою при виникненні відповідної події і виконує деяку дію, яке, в свою чергу, може привести до виникнення події, що активізує інше правило. Правила представляються на спеціальній мові. Підтримується набір визначених правил низького рівня, що забезпечують інтерфейс з підсистемою управління зовнішньою пам'яттю (звичайно, з міркувань ефективності ця підсистема написана не продукційною мовою).

Очевидно, що така організація системи забезпечує максимальну гнучкість. Наприклад, щоб впровадити в оптимізатор запитів деяку нову стратегію виконання (наприклад, розширити застосовуваний набір методів виконання еквісоединенія) досить додатково написати одне або кілька нових правил, пов'язаних з подією вимоги виконати з'єднання. Тим самим, Starburst може використовуватися як потужний і гнучкий засіб дослідження методів оптимізації запитів. Звичайно, сумнівно, що технологія, покладена в основу Starburst, дозволить цій системі конкурувати з такими виконаними в традиційній манері комерційними СУБД, як DB2, Oracle, Informix і т.д.

### *1.3.3. Класи АІС : OLAP і OLTP-системи, призначення й структура*

В області інформаційних технологій існують два взаємно доповнюють один одного напряму обробки даних:

- технології, орієнтовані на оперативну (трансакціонної) обробку даних. Ці технології лежать в основі АІС, призначених для оперативної обробки даних. Називаються такі системи - OLTP (OnLine Transaction Processing) системи;
- технології, орієнтовані на аналіз даних і прийняття рішень. Ці технології лежать в основі АІС, призначених для аналізу накопичених даних. Називаються такі системи - OLAP (OnLine Analytical Processing) системи.

АІС, орієнтовані на оперативну обробку транзакцій, і АІС, призначені для ділового аналізу, використовуються зовсім по-різному і служать різним цілям.

АІС оперативної обробки транзакцій - це основа щоденного функціонування підприємства: прийняття замовлень клієнтів, облік сировини, складський облік, облік оплати продукції, тобто головним чином облікові функції.

АІС ділового аналізу використовуються для прийняття рішень на основі збору і аналізу великого обсягу інформації. Їх головні користувачі - менеджери, службовці планового відділу та відділу маркетингу.

Якщо в АІС оперативної обробки транзакцій основні операції обробки даних - це невеликі за обсягом транзакції, що складаються з простих запитів, як на вибірку, так і на вставку і оновлення невеликої кількості рядків в БД, то в АІС ділового аналізу практично 100% операцій є операціями вибірки (читання) рядків, причому кількість відбираються (аналізованих) рядків може досягати кілька мільйонів.

#### *1.3.3.1. OLTP-системи*

OLTP-системи, будучи високоефективним засобом реалізації оперативної обробки, виявилися малопридатні для задач аналітичної обробки. Це викликано наступним:

1) Засобами традиційних OLTP-систем можна побудувати аналітичний звіт і навіть прогноз будь-якої складності, але заздалегідь регламентований. Будь-який крок в сторону, будь нерегламентоване вимога кінцевого користувача, як правило, вимагає знань про структуру даних і досить високої кваліфікації програміста.

2) Багато необхідні для оперативних систем функціональні можливості є надлишковими для аналітичних задач і в той же час можуть не відображати предметної області. Для вирішення більшості аналітичних завдань потрібне використання зовнішніх спеціалізованих інструментальних засобів для аналізу, прогнозування та моделювання. Жорстка ж структура баз не дозволяє досягти прийнятної продуктивності в разі складних вибірок і угрупувань і, отже, вимагає великих затрат часу для організації шлюзів.

3) На відміну від транзакційних, в аналітичних системах не потрібні і, відповідно, не передбачаються розвинені засоби забезпечення цілісності даних, їх резервування і відновлення. Це дозволяє не тільки спростити самі кошти реалізації, а й знизити внутрішні накладні витрати і, отже, підвищити продуктивність при вибірці даних.

#### *1.3.3.2. OLAP-системи*

Основне призначення OLAP-систем: динамічний багатовимірний аналіз історичних і поточних даних, стабільних у часі; аналіз тенденцій; моделювання і прогнозування майбутнього. Такі системи, як правило, орієнтовані на обробку довільних, заздалегідь не регламентованих запитів. В якості основних характеристик цих систем можна відзначити наступні:

- підтримка багатовимірного представлення даних, рівноправність всіх вимірювань, незалежність продуктивності від кількості вимірювань;

- прозорість для користувача структури, способів зберігання і обробки даних;
- автоматичне відображення логічної структури даних в зовнішні системи;
- динамічна обробка розрідженних матриць ефективним способом.

Термін OLAP часто ототожнюють з системами підтримки прийняття рішень (СППР) - DSS (Decision Support Systems). А як синонім терміну «рішення» використовують Data Warehousing - «сховища даних». Під цим розуміється набір організаційних рішень, програмних і апаратних засобів для забезпечення аналітиків інформацією на основі даних з систем обробки транзакцій нижнього рівня і інших джерел.

Сховища даних дозволяють обробляти дані, накопичені за тривалі періоди часу. Ці дані є різнорідними (і не обов'язково структурованими). Для сховищ даних притаманний багатовимірний характер запитів. Величезні обсяги даних, складність структури як даних, так і запитів - все це вимагає використання спеціальних методів доступу до інформації.

В інших джерелах поняття СППР вважається більш широким. Сховища даних і засоби оперативної аналітичної обробки можуть служити одними з компонентів архітектури СППР.

OLAP завжди включає в себе інтерактивну обробку запитів і подальший багатопрохідний аналіз інформації, який дозволяє виявити різноманітні, не завжди очевидні тенденції, що спостерігаються в предметної області.

Іноді розрізняють OLAP в вузькому сенсі - як системи, які забезпечують тільки вибірку даних в різних розрізах, і OLAP в широкому сенсі, або просто OLAP, що включають в себе:

- підтримку декількох користувачів, що редагують БД;
- функції моделювання, в тому числі обчислювальні механізми отримання похідних результатів, і навіть агрегування і об'єднання даних;
- прогнозування, виявлення тенденцій і статистичний аналіз.

Кожен з цих типів систем вимагає специфічної організації даних, а також спеціальних програмних засобів, що забезпечують ефективне виконання поставлених завдань.

OLAP-засоби забезпечують проведення аналізу ділової інформації за багатьма параметрами, таких як вид товару, географічне положення покупця, час оформлення угоди і продавець, кожен з яких допускає створення ієрархії уявлень. Так, для часу можна користуватися річними, квартальними, місячними і навіть тижневими і денними проміжками; географічне розбиття може проводитися по містах, штатах, регіонах, країнам або, якщо буде потрібно, по цілим півкулях.

OLAP-системи можна розбити на три класи:

1) Найбільш складними і дорогими з них є засновані на патентованих технологіях сервери багатовимірних БД. Ці системи забезпечують повний цикл OLAP-обробки і або включають в себе, крім серверного компонента, власний інтегрований клієнтський інтерфейс, або використовують для аналізу даних зовнішні програми роботи з електронними таблицями. Продукти цього класу найбільшою мірою відповідають умовам застосування в рамках великих інформаційних сховищ. Для їх обслуговування потрібен цілий штат співробітників, що займаються як установкою і супроводом системи, так і формуванням уявлень даних для кінцевих користувачів. Зазвичай подібні пакети досить дороги.

2) Реляційні OLAP-системи (ROLAP). Тут для зберігання даних використовуються старі реляційні СУБД, а між БД і клієнтським інтерфейсом організовується визначається адміністратором системи шар метаданих. Через цей проміжний шар клієнтський компонент може взаємодіяти з реляційної БД як з багатовимірної. Подібно засобам першого класу, ROLAP-системи добре пристосовані для роботи з великими інформаційними сховищами, вимагають значних витрат на

обслуговування фахівцями інформаційних підрозділів і передбачають роботу в розрахованому на багато користувачів режимі.

ROLAP-засоби реалізують функції підтримки прийняття рішень в надбудові над реляційним процесором БД.

Такі програмні продукти повинні відповідати ряду вимог, зокрема:

- мати потужний оптимізований для OLAP генератор SQL-виразів, що дозволяє застосовувати багатопрохідні SQL-оператори SELECT і / або корельовані підзапити;

- мати досить розвиненими засобами для проведення нетривіальною обробки, що забезпечує ранжування, порівняльний аналіз і обчислення процентних співвідношень в рамках класу;

- генерувати SQL-вирази, оптимізовані для цільової реляційної СУБД, включаючи підтримку доступних в ній розширень цієї мови;

- надавати механізми опису моделі даних за допомогою метаданих і давати можливість використовувати ці метадані для побудови запитів в реальному масштабі часу;

- включати в себе механізм, що дозволяє оцінювати якість побудови зведених таблиць з точки зору швидкості обчислення, бажано з накопиченням статистики по їх використанню.

3) OLAP-систем - інструменти генерації запитів і звітів для настільних ПК, доповнені OLAP-функціями або інтегровані з зовнішніми засобами, які виконують такі функції. Ці вельми розвинені системи здійснюють вибірку даних з вихідних джерел, перетворюють їх і поміщають в динамічну багатовимірну БД, що функціонує на ПК кінцевого користувача. Зазначений підхід, що дозволяє обійтися як без дорогого сервера багатовимірної БД, так і без складного проміжного шару метаданих, необхідного для ROLAP-засобів, забезпечує в той же час достатню ефективність аналізу. Ці кошти для настільних ПК найкраще підходять для роботи з невеликими, просто організованими БД. Потреба у кваліфікованому обслуговуванні для них нижче, ніж для інших OLAP-систем, і приблизно відповідає рівню звичайних середовищ обробки запитів.

*1.3.3.3. Завдання, які вирішуються OLTP- і OLAP-системами*

Завдання, ефективно вирішуються кожної з систем, визначені на основі порівняльних характеристик OLTP- і OLAP-систем.

Завдання, які вирішуються OLTP- і OLAP-системами

| Характеристика | OLTP | OLAP |
|---|---|---|
| Частота оновлення даних | Висока частота, невеликі «порції» | Мала частота, великі «порції» |
| Джерела даних | В основному внутрішні | Стосовно аналітичній системі, в основному зовнішні |
| Вік даних | Поточні (кілька місяців) | Історично (за роки) і прогнозовані |
| Рівень агрегації даних | Деталізовані дані | В основному агреговані дані |
| Можливості аналітичних операцій | Регламентовані звіти | Послідовність інтерактивних звітів, динамічна зміна рівнів агрегації і зрізів даних |
| Призначення системи | Фіксація, оперативний пошук і обробка даних, регламентована аналітична обробка | Робота з історичними даними, аналітична обробка, прогнозування, моделювання |

Порівняння OLTP і OLAP систем

| Характеристика | OLTP | OLAP |
|---|---|---|
| Переважаючі операції | Введення даних, пошук | Аналіз даних |
| Характер запитів | Багато простих транзакцій | Складні транзакції |
| Збережені дані | Оперативні, деталізовані | Охоплюють великий період часу, агреговані |
| Вид діяльності | Оперативна, тактична | Аналітична, стратегічна |
| Тип даних | Структуровані | Різнотипні |

Відмінності між OLAP і OLTP системами

| Характеристика | OLTP | OLAP |
|---|---|---|
| Вміст | Поточні дані | Дані, накопичені за довгий період часу |
| Структура даних | Структура таблиць відповідає структурі транзакцій | Структура таблиць зрозуміла і зручна для написання запитів (куби фактів - схема «зірка») |
| Типовий розмір таблиць | Тисячі рядків | Мільйони рядків |
| Схема доступу | Зумовлена для кожного типу оброблюваних транзакцій | Довільна; залежить від того, яка саме завдання стоїть перед користувачем в даний момент і які відомості потрібні для її вирішення |
| Кількість рядків, до яких звертається один запит | Десятки | Від тисяч до мільйонів |
| З якими даними працює додаток | З окремими рядками | З групою рядків (підсумкові запити) |
| Інтенсивність звернень до бази даних | Велика кількість бізнес-транзакцій в хвилину або в секунду | На виконання запитів потрібен час: хвилини або навіть години |
| Тип доступу | Вибірка, вставка і оновлення | Вибірка даних (майже 100% операцій) |
| Чим визначається продуктивність | Час виконання транзакції | Час виконання запиту |

Як видно, робоче навантаження OLTP і OLAP баз даних настільки різна, що дуже важко або навіть неможливо підібрати одну СУБД, яка найкращим чином задовольняла б вимогам додатків обох типів (важливо, щоб всі запити ділового аналізу, що тривають тривалий час, не знижували продуктивності операційної обробки транзакцій).

**Література до розділу**

[1, с.7-26]; [2, с.13-25]; [3, с.6-28]; [4,с.7-14]; [6, с.43-99]

# Розділ 2. Моделювання предметної області автоматизованих інформаційних систем

## Тема 2.1. Подання даних в автоматизованих інформаційних системах

### *2.1.1. Поняття про предметну область АІС: цикл розробки АІС, інформаційні і функціональні частини АІС*

Предметна область АІС - це частина реального світу, представлена сукупністю взаємодіючих інформаційних об'єктів, які знаходять своє відображення в базі даних.

Для логічного й успішного створення бази даних необхідно пройти весь цикл розробки інформаційної системи. Кожен етап циклу складається з визначених процедур, які необхідно виконати, якщо потрібно, щоб проект бази даних був ефективним. Встановлено, що завдяки якісно створеному проекту бази даних, стає можливим реалізація надійної і високопродуктивної системи. У процесі розробки інформаційної системи доводитися вирішувати безліч питань – від контролю надмірності даних до удосконалювання взаємодії з кінцевим користувачем. Вирішуючи кожну з цих проблем в ретельно продуманому проекті бази даних, підвищується її продуктивність та й інформаційної системи в цілому.

Від зародження концепції до введення в експлуатацію розробка бази даних здійснюється в строгій відповідності із циклом розробки інформаційної системи. Цей цикл заснований на системному підході до розробки бази даних за принципом «зверху-униз», який забезпечує перетворення інформаційних потреб предметної області системи в робочу базу даних. *Предметна область* – обумовлена область застосування конкретної інформації про об‘єкт автоматизації, для якого розробляється інформаційна система.

Цикл розробки включає такі п'ять основних етапів:

1) Моделювання;
2) Проектування;
3) Кодування і документування;
4) Налагодження і впровадження;
5) Експлуатація.

Етап «*Моделювання*» припускає вивчення й аналіз потреб об'єкта автоматизації в інформації, де буде впроваджуватися інформаційна система з базою даних. З'ясування інформаційних потреб виробляється в ході взаємодії розроблювача майбутньої системи із замовником на основі бесід з кінцевими користувачами задля встановлення термінів, понять, характеристик об'єктів (сутностей), що утворять предметну область. Також, корисним для розробки майбутньої системи є вивчення документації, у якій формулюються задачі функціонування об'єкта автоматизації (підприємства, виробництва, організації, бізнесу і т.п.) і прикладної системи. Результати аналізу представляються у виді семантичної/інфологічної моделі (Semantic/Logical Model), тобто словесні описи перетворяться в графічні представлення інформаційних потреб і правил роботи об'єкта автоматизації. Форма і семантика інфологічної моделі повинна бути придатна для обговорення і спільного удосконалювання з аналітиками й експертами замовника.

Етап «*Проектування*» припускає розробку структури бази даних. Сутності і зв'язки, представлені в інфологічної моделі, перетворяться в інформаційні компоненти бази даних (таблиці, стовпці, ключі, обмеження і т.п.), які утворять так називану серверну/даталогічну модель (Server/Data Model) бази даних.

Етап «*Кодування і документування*» припускає створення дослідного зразка системи, написання і виконання команд для створення таблиць і допоміжних об'єктів

бази даних, програмування інтерфейсів системи,а також розробку документації користувача, текстів довідок і посібників з експлуатації системи.

Етап «*Налагодження і впровадження*» передбачає передачу системи користувачу замовника, спостереження за її продуктивністю, розширення можливостей і подальше її удосконалення.

Етап «*Експлуатація*» передбачає самостійне використання системи замовником на власних засобах, можливо під наглядом виконавця-розробника, але без втручання в технологічні процеси функціонування системи з метою гарантійного обслуговування.

Таким чином, в АІС виділяють з точки зору її організації дві основні складові: функціональну та інформаційну частини.

Функціональна частина АІС - являє собою програмне забезпечення, що реалізує логіку обробки, інтерпретації даних предметної області та їх подання кінцевому користувачеві.

Інформаційна частина АІС - являє собою модель даних, яка реалізує способи і механізми подання інформації.

### *2.1.2. Класи моделей подання даних*

Моделювання – це першооснова процесу будь-якої розробки інформаційної системи. Метою моделювання є формалізація інформаційних потреб користувачів у виді подання правил взаємодії даних у рамках розроблювальних прикладних систем. При моделюванні необхідно враховувати такі ключові фактори, як:

- *продуктивність*. Ефективність проектування має більше значення для кінцевої продуктивності системи, ніж усі наступні зусилля з корекції і настроюванню системи.

- *інтегрованість складових*. Прикладні системи звичайно створюються командами розробників. При відсутності спільного (єдиного, узгодженого) вихідного завдання на проект кожен розробник працює виходячи з власних представлень про проектовану систему. Якісний же проект забезпечує не тільки однаковість зовнішнього вигляду і поводження системи, але й успішну інтеграцію готових прикладних частин системи одна з одною.

- *інтеграція з іншими системами*. Нову систему часто приходиться інтегрувати з вже існуючими системами і навіть з тими, котрі ще тільки створюються. Якісне проектування дозволяє поширювати вищевказані переваги на корпоративні (складні, комплексні) системи.

- *документація і взаємодія*. Одна з основних задач розробника полягає також в тому, щоб довести кожне конструкторське рішення до інших через документування.

- *масштабованість/розширюваність*. Питання оцінки продуктивності системи повинні розглядатися на етапі проектування, а не під час експлуатації. Розробка проекту в невеликому контрольованому (тестовому) середовищі не дозволяють перевірити його поводження в реальних ситуаціях з великими обсягами даних, що певним чином впливає на можливість встановлення недоліків проекту.

- *використання ефективних готових рішень*. Залучення в процес моделювання вже раніше апробованих рішень з відомими параметрами продуктивності дозволять скоротити загальний час проектування інформаційної системи.

Таким чином, основу проектування інформаційної системи складають моделі даних різного призначення.

*Модель даних* – це деяка абстракція, в якій знаходять своє відображення найбільш важливі аспекти функціонування визначеної предметної області, а

другорядні – ігноруються. Модель даних являє собою деяку цільову модель предметної області.

Модель даних являє собою набір концептуальних інструментів для опису даних, відносин між ними, семантики даних і обмежень їх цілісності, тобто формальну систему, що включає:

- структурні аспекти (визначають правила породження допустимих для даної предметної області видів структур даних);
- аспекти підтримки цілісності даних (визначають правила співіснування даних, які можуть бути реалізовані засобами цієї моделі);
- аспекти маніпулювання даними (визначають можливі операції над такими структурами даних).

Тобто, модель визначає, в який спосіб відбувається об‘єднання даних у структури різної складності, які існують обмеження на значення даних і як здійснюється оперування цими даними.

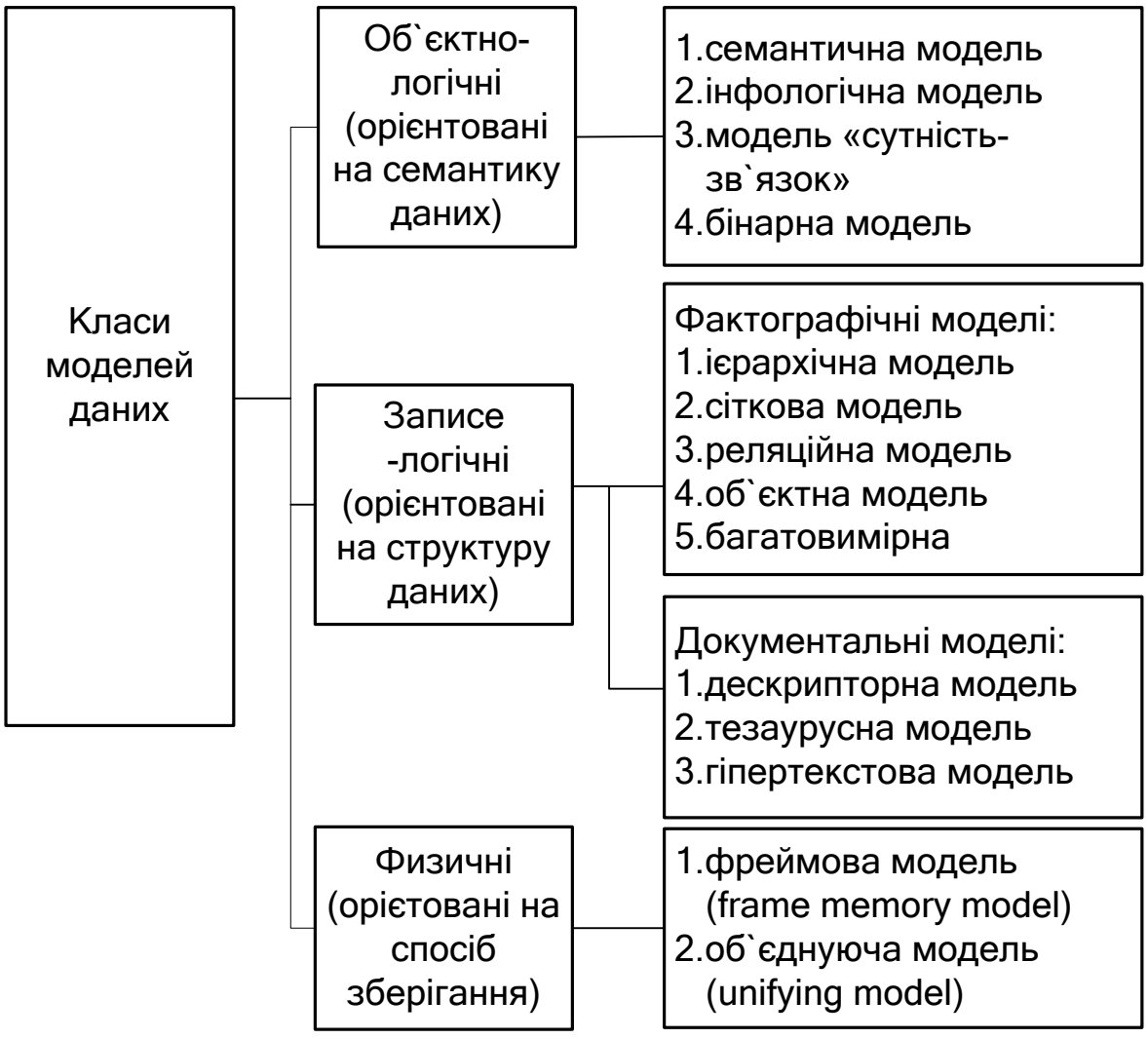



Рис. 1. Класифікація моделей подання даних

Тому моделями майбутньої інформаційної системи на різних етапах процесу проектування можуть бути такі їх різновиди (рис. 1):

- у виді описів у представленні користувача системи (технічне завдання/умови на розробку проекту);
- у виді інформаційних сутностей предметної області інформаційної системи (семантична модель);

- у виді табличної моделі бази даних (серверна модель) і файлів, блоків даних (фізична модель) на сервері бази даних.

Проте, кінцевим результатом моделювання предметної області інформаційної системи є створення моделі такого виду, яка б вирішувала задачі формалізації всіх правил роботи об'єкта автоматизації, була зрозуміла кінцевому користувачу замовника майбутньої інформаційної системи, а також містила досить докладну інформацію, на основі якої проектувальник розробника (адміністратор бази даних, програміст) міг би створити працездатну базу даних й ефективну інформаційну систему.

### *2.1.3. Концепція й етапи семантичного моделювання*

При проектуванні структури БД найчастіше застосовується метод семантичного моделювання.

*Семантичне моделювання* - область досліджень, присвячених способам уявлення смислового значення даних. Семантичне моделювання являє собою моделювання структури даних, спираючись на зміст цих даних.

Семантична модель створюється на основі словесних описів і документів, що відтворюють інформаційні потреби користувача і правила роботи системи у певній предметній області об‘єкта автоматизації. Така модель є концептуальним (інфологічним) представленням предметної області інформаційної системи.

Етапи семантичного моделювання:

1) Виявлення безлічі семантичних понять, за допомогою яких представляється предметна область.

2) Визначення набору формальних об'єктів, які можуть бути використані для подання семантичних понять.

3) Формування набору формальних правил цілісності, призначених для роботи з формальними об'єктами.

4) Визначення набору формальних операторів для маніпулювання формальними об'єктами.

Поняття концепції семантичного моделювання:

1.Сутність - деякий відмітний об'єкт предметної області, інформацію про який необхідно відображати в АІС.

2.Атрибут - елемент даних, що описує певну властивість сутності.

3. Зв'язок - сутність, що служить для забезпечення взаємодії між двома або більше сутностями, розглядається як двунаправленная асоціація.

4. Тип - деяка сукупність помітних об'єктів предметної області, що мають спільні властивості.

Процес моделювання є ітераційним й припускає повернення до попередніх етапів для перегляду раніше прийнятих рішень і включає наступні кроки:

- виділення семантичних об'єктів (понять) предметної області системи;
- класифікація семантичних об'єктів з виділенням сутностей і зв'язків з подальшим їх описом;
- побудова діаграми з урахуванням всіх сутностей і зв'язків;
- формування унікального ідентифікатора сутностей;
- додавання не ключових атрибутів до сутностей;
- приведення моделі до стану забезпечення правил цілістності.

Перевагами такої моделі є:

- можливість обміну ідеями при розробці;
- ефективність збору і документування інформаційних потреб системи;
- зрозуміле графічне представлення системи;
- простота розробки і внесення удосконалень.

### *2.1.4. Типи діаграм подання понять семантичної моделі*

Модель даних відображує уявлення про реальний світ. Проте важливо, аби обсяг знань і семантика даних, відтворені в моделі, були адекватні способу використання даних. Вважатимемо, що модель даних це сукупність структури даних, операцій над ними (операції маніпулювання даними) та обмежень цілісності. Іншими словами, модель визначає, в який спосіб відбувається об'єднання даних у структури різної складності, які існують обмеження на значення даних і як здійснюється оперування цими даними.

Основою для будь-якої структури даних є відображення елементарної одиниці даних у вигляді такої трійки: <об'єкт, властивість об'єкта, значення властивостей>. Сукупність взаємопов'язаних між собою елементарних одиниць даних може відображуватися різноманітними способами, що приводить до формування різних структур з різним ступенем деталізації.

Моделі даних поділяються на два класи: сильно та слабко типізовані.

У сильно типізованих моделях усі дані мають належати до певної категорії, або типу. Якщо дані не підпадають під жодну з категорій, їх потрібно типізувати штучно. Деякі моделі будуються у такий спосіб, що категорії визначаються наперед і не можуть змінюватися динамічно. Отримаємо різновиди *діаграм сутностей* (рис. 2).

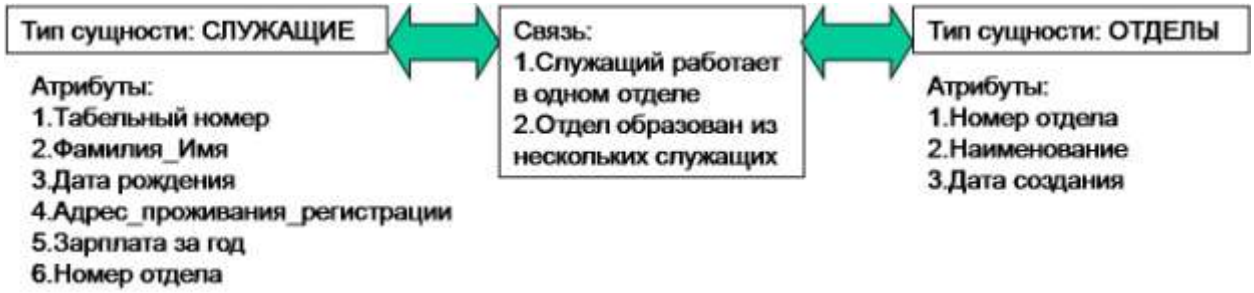



Рис. 2. Діаграма сутностей

Сильно типізовані моделі мають значні переваги, бо дають змогу побудувати абстракції властивостей даних і дослідити їх у термінах категорій. Більшість моделей, що використовуються в автоматизованих системах, зокрема й базах даних, належать до сильно типізованих.

Для слабко типізованих моделей належність даних до тієї чи іншої категорії немає жодного значення. Категорії використовуються настільки, наскільки це доцільно в кожному конкретному випадку. Окремі дані можуть існувати як незалежно, так і у зв'язку з іншими. Інформація про категорії (якщо вони використовуються) розглядається як додаткова. Отримаємо різновиди *діаграм екземплярів сутностей* (рис. 3).

На відміну від сильно типізованих моделей, слабко типізовані забезпечують інтеграцію даних і категорій. Найкращі можливості такої інтеграції надаються численням предикатів, яке у багатьох моделях даних використовується для зображення знань, що не підтримуються базовими засобами моделювання.

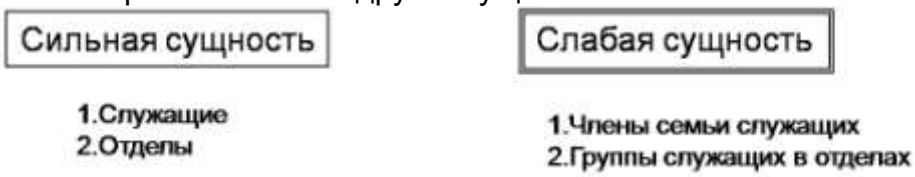

Рис. 3. Діаграма екземплярів сутностей

## Тема 2.2. Проектування структури бази даних за допомогою семантичного моделювання

### *2.2.1. Модель представлення даних типу "ER-модель"*

Наиболее известным и получившим широкое распространение среди методов семантического моделирования является метод построения модели «сущность-связь», предложенной в 1976 году Ченом.

*Сущности*: обычные (сильные) - которые не являются слабыми, слабые - существование которых зависит от других сущностей.

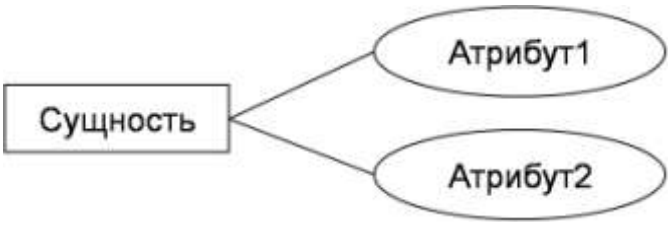



*Тип сущности* определяется наименованием сущности.

Тип сущности *Y* является подтипом сущности *X*, если подтип наследует все свойства и связи родителя - супертипа.


Сущности и связи характеризуются некоторыми свойствами. Все сущности одного и того же типа обладают общими свойствами. Сущность одного типа отличается от сущности другого типа набором собственных свойств.

*Свойства сущности* представляются атрибутами, соединенными с ней линиями.




Выделяют: простые, составные, ключевые, многозначные, производные свойства.

*Составное свойство* - атрибут, значение которого состоит (образуется) из простых свойств.

*Ключевое свойство* - атрибут, значение которого уникально в контексте сущности.

*Многозначное свойство* - атрибут, значение которого представляется набором (перечнем, списком) значений.

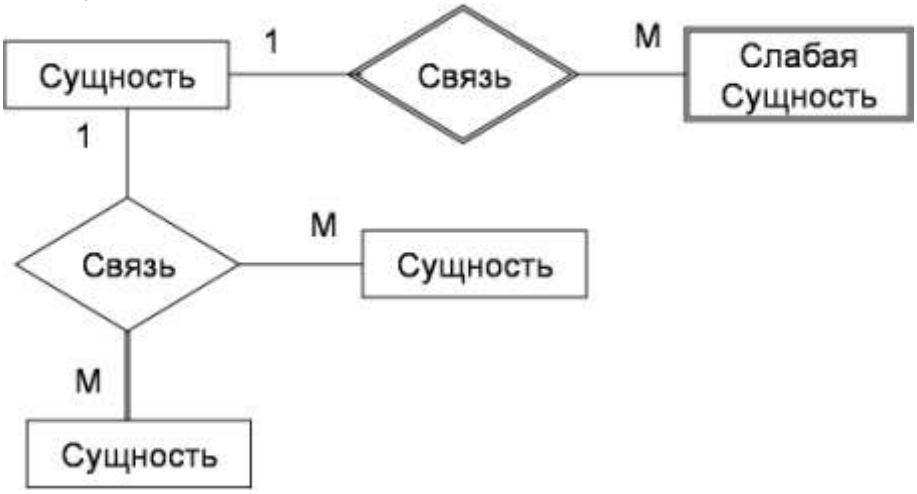
*Производное свойство* - атрибут, значение которого образуется в результате обработки (агрегирования) набора значений другого атрибута.


*Связь* определяется как ассоциация, объединяющая несколько сущностей.

Сущности, участвующие в связи, называются *сторонами связи*, а их экземпляры, включенные в связь, называются ее *участниками*. Количество экземпляров участников связи определяет *степень связи*.

Если каждый экземпляр сущности участвует в связи с экземплярами других участников связи, то такое участие - *класс связи* считается *полным* (обязательным), в противном случае - *частичным* (необязательным).

По степеням направлений связи определяется *тип связи*: «один к одному», «один ко многим», «много ко многим».




*2.2.2. Проектування ER-моделі*

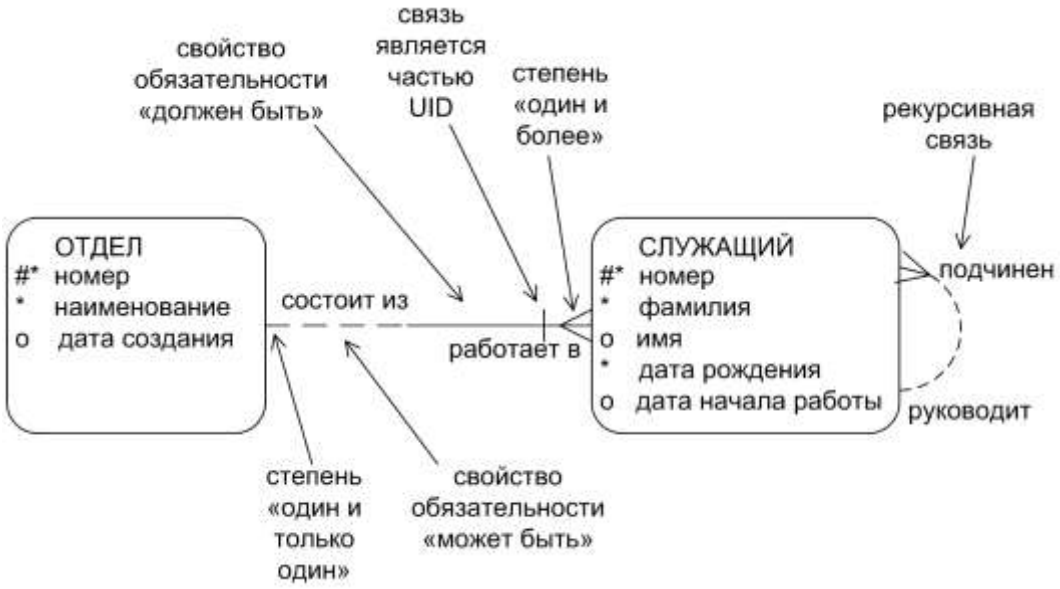
ER-диаграмма является методом представления концептуальной (логической) структуры данных предметной области АИС в графическом виде для более наглядного отображения основных компонент проекта базы данных.


Подтипы сущностей ER-диаграммы


*2.2.3. Модифікації ER-моделі*

Модификации - это специальный вид модели Чена, в котором определены ограничения на условия установления связей между сущностями, на представление свойств в сущностях и в связях.

Например, бинарные модели, в которых возможны связи только между двумя сущностями и связи не имеют атрибутов , концептуальные модели - в которых, свойства отображаются атрибутами в сущностях.




*2.2.4. Інформаційні компоненти бази даних*

Построенная в соответствии с правилами ER-модель может считаться только *пред-проектом* базы данных. Т.к., обычная ER-диаграмма недостаточно точна (полна) для построения базы (в части обеспечения целостности данных).

*Условия целостности* базы данных, т.е. ограничения направленные на ее сохранение, позволяющие добиться того, чтобы пользователи выполняли только те операции, которые оставляют базу данных всегда правильной и согласованной.

Поэтому, сущности и связи, представленные в ER-модели, должны быть преобразованы в другие информационные объекты - компоненты базы данных - таблицы, столбцы, ограничения целостности, которые образуют так называемую *серверную (реляционную) модель*.

Условиям ограничений целостности соответствуют в серверной модели:

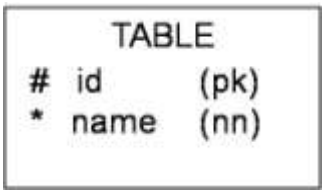
- *ключи* трех типов (первичный, уникальный и внешний),
- *свойство обязательности* (определенности) значений атрибутов-столбцов, образующих ключи.

*Первичный ключ*

Каждая строка таблицы уникально опознается по одному или нескольким столбцам – так называемому *первичному ключу* PRIMARY KEY (PK). Определение первичного ключа PK должно быть таким, чтобы он не повторялся и не мог иметь неопределенные значения NULL.

Первичный ключ, состоящий из нескольких столбцов, называется *составным* (сложным) первичным ключом. Комбинация значений столбцов в составном первичном ключе должны быть уникальной, хотя значения в каждом отдельном столбце могут повторяться.

Никакая часть первичного ключа не может содержать неопределенные значения NULL. В качестве первичного ключа может использоваться любой уникальный ключ таблицы, а в целом для таблицы определяется только единственный первичный ключ.

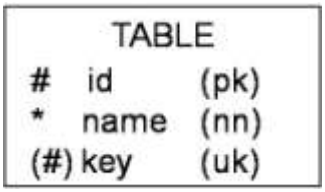


*Уникальный ключ*

*Уникальный ключ* UNIQUE (UK) – это столбец или сочетание столбцов с неповторяющимися значениями.

Таблица может иметь несколько уникальных ключей, которые могут претендовать на роль первичного ключа. Однако, уникальные ключи могут содержать неопределенные значения NULL. Поэтому, для того чтобы уникальный ключ использовался в качестве первичного для однозначного определения строк таблицы, необходимо, чтобы его значения обладали свойством обязательности NOT NULL.

В результате определение первичного ключа равнозначно определению ключа как UNIQUE и NOT NULL, а остальные ключи остаются *альтернативными уникальными ключами*.




*Внешний ключ*

*Внешний ключ* FOREIGN KEY (FK) – это столбец или набор столбцов в одной таблице, содержащий ссылку на первичный ключ PK или уникальный ключ UK в той же или другой таблице.

Внешние ключи основаны на значениях данных и являются не физическими, а чисто логическими указателями. Их значения должны либо совпадать со значениями соответствующего первичного или уникального ключа, либо быть неопределенными NULL.

Если внешний ключ является частью первичного ключа таблицы, то он не может быть неопределенным.

Для каждого внешнего ключа может быть задан подходящий набор правил обновления ON UPDATE и удаления ON DELETE значений внешнего ключа, которые обеспечивают их согласованность.


*Типы ограничений целостности*

Таким образом, в базе данных устанавливаются такие типы ограничений целостности:

1) *целостность сущностей* – ни одна часть первичного ключа не может иметь значение NULL и, кроме того, ключи должны быть уникальны;

2) *целостность ссылок* – значения внешнего ключа должны совпадать с первичным ключом или быть NULL;

3) *целостность столбцов* – значения данных в столбце должны соответствовать заданному типу данных (числовому NUMBER, строчному с фиксированной длиной CHAR, строчному с переменной длиной VARCHAR2, дате DATE) и обладать при необходимости свойством обязательности значений NOT NULL;

4) *целостность, задаваемая пользователем* – значения данных соответствуют правилам информационной системы.

*2.2.5. Процедури перетворення ER-моделі у компоненти БД*

Преобразование ER-модели предметной области в проект базы данных (серверную модель) заключается в выполнение таких шагов:

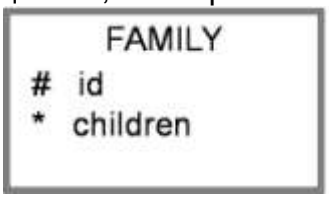
1) преобразование сущностей в таблицы;
2) преобразование атрибутов в столбцы;
3) преобразование уникальных идентификаторов в первичные ключи;
4) преобразование связей в дополнительные столбцы-внешние ключи.

*Преобразование сущностей*

**Сильная** сущность преобразуется в таблицу с собственным PK.


```
Table EMPLOYEE
(id, name NOT NULL, fname),
PRIMARY KEY(id);
```

**Слабая** сущность - в таблицу с сурогатным (замещающим) PK или составным PK из ключевого атрибута слабой сущности и ключевого атрибута сильной сущности, с которой связана слабая сущность.



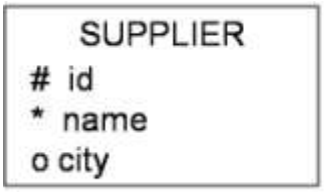

```
Table FAMILY
 (id, children NOT NULL, empl_id),
 PRIMARY KEY (empl_id, id),
 FOREIGN KEY(empl_id) REFERENCES EMPLOYEE;
```

**Подтип** сущности преобразуется в таблицу, с таким же PK, что и у таблицы для сущности-супертипа, но со своим набором атрибутов, при этом PK является также и FK.


```
Table PROGRAMER
 (empl_id, lang),
 PRIMARY KEY(empl_id),
 FOREIGN KEY(empl_id) REFERENCES EMPLOYEE;
```

*Преобразование атрибутов*

*Преобразование атрибутов* в столбцы означает присвоение столбцам имени атрибутов из ER-модели, при этом обязательные атрибуты, отмеченные символом “*”, преобразуются в столбцы со свойством NOT NULL. Многозначные атрибуты преобразуются в дополнительную таблицу, связанную с исходной таблицей внешним ключом.




```
Table SUPPLIER
 (id, name NOT NULL),
 PRIMARY KEY(id);

Table SUPP_CITY (supp_id, city),
 FOREIGN KEY(supp_id) REFERENCES SUPPLIER;
```

*Преобразование UID*

*Преобразование уникальных идентификаторов* в первичные ключи означает такие действия:

1) преобразовать уникальный идентификатор, отмеченный в ER-модели символом “#”, в столбцы первичного ключа и обозначить их как PK. Уникальный идентификатор, содержащий только один атрибут, преобразуется в одностолбцовый

PK, который будет обладать свойствами обязательности NOT NULL и уникальности UNIQUE;

2) пометить все альтернативные уникальные идентификаторы уникальным ключом UK со свойством NOT NULL;

3) если уникальный идентификатор сущности включает связь, что соответствует определению слабой сущности, то необходимо добавить дополнительный столбец с указанием внешнего ключа для каждой слабой связи и пометить его как PK и FK.


```
Table DEPARTMENT
(id, name NOT NULL),
PRIMARY KEY(id),
UNIQUE (name);
```

*Преобразование связей*

*Преобразование связей* во внешние ключи выполняется для каждого типа связей отдельно. Однако, общим является образование на одной из сторон связи дополнительного столбца с ограничением целостности внешний ключ, который служит ссылкой на значение первичного ключа противоположной стороны.

Дополнительным шагом может служить назначение правил обеспечения целостности внешних ключей при удалении или обновлении: каскадное и ограниченное.

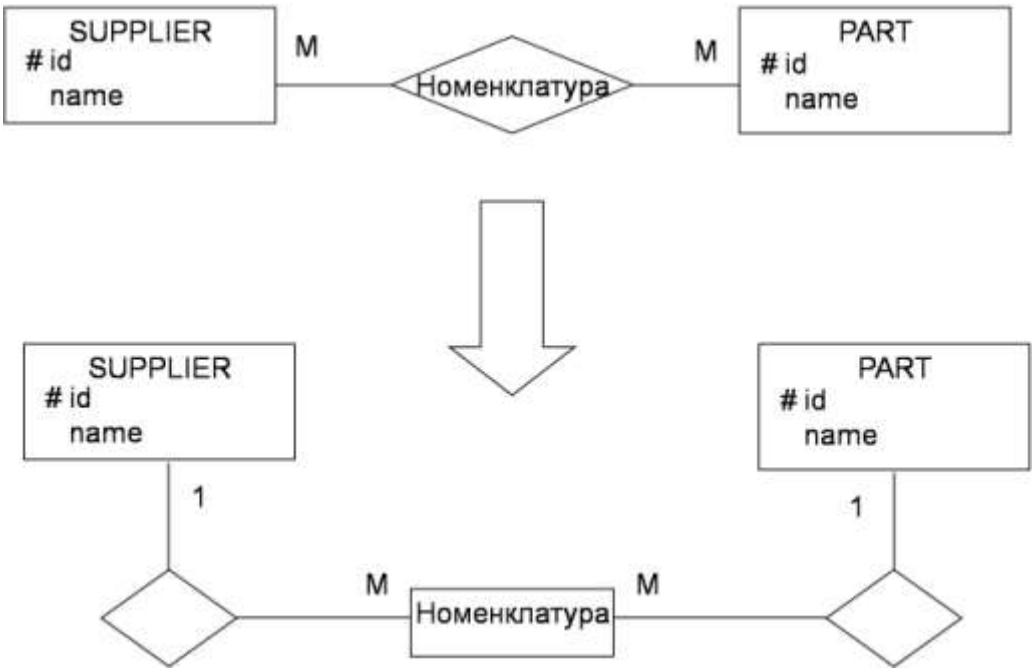
*Преобразование связи типа «M-M».*


```
Table SUPPLIER
(id, name)
PRIMARY KEY(id);

Table PART
(id, name)
PRIMARY KEY(id);
```

```
Table SUPP_PART
(supp_id NOT NULL, part_id NOT NULL),
PRIMARY KEY(supp_id,part_id),
FOREIGN KEY(supp_id) REFERENCES SUPPLIER,
FOREIGN KEY(part_id) REFERENCES PART;
```

Содержание процедуры преобразования связи «M-M» показана на примере:




*Преобразование связи типа необязательная «1-М»*

Преобразование связи с соблюдением требования отсутствия необязательных столбцов.


```
Table DEPARTMENT
(id, name),
PRIMARY KEY(id);

Table EMPLOYEE
(id, name),
PRIMARY KEY(id);

Table EMPL_DEPT
(empl_id NOT NULL, dept_id NOT NULL),
PRIMARY KEY(empl_id),
FOREIGN KEY (empl_id) REFERENCES EMPLOYEE,
FOREIGN KEY (dept_id) REFERENCES DEPARTMENT;
```

Таблица EMPL_DEPT обеспечивает требование отсутствия внешних ключей как необязательных столбцов.

*Преобразование связи типа необязательная «1-М»*

Преобразование связи без соблюдения требования отсутствия необязательных столбцов.


```
Table DEPARTMENT
(id, name),
PRIMARY KEY(id);

Table EMPLOYEE
(id, name, dept_id),
PRIMARY KEY(id),
FOREIGN KEY (dept_id) REFERENCES DEPARTMENT;
```

В таблице EMPLOYEE разрешены внешние ключи как необязательные столбцы.

*Преобразование связи типа обязательная (частично) «1-М»*


```
Table DEPARTMENT
(id, name),
PRIMARY KEY(id);

Table EMPLOYEE
(id, name, dept_id NOT NULL),
PRIMARY KEY(id),
FOREIGN KEY (dept_id) REFERENCES DEPARTMENT;
```

*Преобразование связи типа необязательная «1-1»*

Преобразование связи с соблюдением требования отсутствия необязательных столбцов.


```
Table EMPLOYEE
(id, name),
PRIMARY KEY(id);

Table CHILDREN
(id, name),
PRIMARY KEY(id);

Table CHIL_EMPL
(empl_id NOT NULL, chil_id NOT NULL),
UNIQUE(empl_id),
UNIQUE(chil_id),
FOREIGN KEY (empl_id) REFERENCES EMPLOYEE,
FOREIGN KEY (chil_id) REFERENCES CHILDREN;
```

Таблица CHIL_EMPL обеспечивает требование отсутствия внешних ключей как необязательных столбцов.

*Преобразование связи типа необязательная «1-1»*

Преобразование связи без соблюдения требования отсутствия необязательных столбцов.


```
Table EMPLOYEE
(id, name),
PRIMARY KEY(id);

Table CHILDREN
(id, name, empl_id),
PRIMARY KEY(id),
FOREIGN KEY(empl_id) REFERENCES EMPLOYEE,
UNIQUE(empl_id);
```

В таблице CHILDREN разрешены внешние ключи как необязательные столбцы.

Внешний ключ образуется на любой необязательной стороне.

*Преобразование связи типа обязательная (частично) «1-1»*


```
Table EMPLOYEE
(id, name),
PRIMARY KEY(id);

Table CHILDREN
(id, name, empl_id NOT NULL),
PRIMARY KEY(id),
FOREIGN KEY(empl_id) REFERENCES EMPLOYEE,
UNIQUE(empl_id);
```

Внешний ключ образуется на обязательной стороне.

*Преобразование связи типа обязательная (полностью) «1-1»*


Table EMPLOYEE
(id, empl_name, chil_name),
PRIMARY KEY(id);

или

Table CHILDREN
(id, chil_name, empl_name),
PRIMARY KEY(id);

Выполняется объединение атрибутов в одну таблицу с общим первичным ключом.

# Розділ 3. Організація баз даних

## Тема 3.1. Моделі організації даних

### *3.1.1. Загальна класифікація логічних моделей структур даних*

*Модель данных* представляет собой набор концептуальных инструментов для описания данных, отношений между ними, семантики данных и ограничений их целостности.

В системах управления базами данных используют *логические модели*, опирающиеся на понятие *записи (записе-логические модели)*, которые ориентированы на описание структуры данных в информации, а не на описание структуры самой информации (*объектов* ее предметной области) пригодных для использования в АИС.


### *3.1.2. Порівняльна характеристика моделей організації даних: ієрархічної, мережної, реляційної, постреляційної, багатомірної, об'єктної*

*Иерархическая модель* создана в 60-х годах, типичным представителем является СУБД IMS (Information Management System) корпорации IBM. Структурно данные представляются в форме дерева - упорядоченного графа.

*Сетевая модель* создана в начале 70-х годов, типичным представителем является СУБД IDS (Integrated Data Store) корпорации General Electric, IDMS компании Computer Associates. Структурно данные представляются в форме сети - ориентированного графа.

*Реляционная модель* предложена в 1970 году в работе Э.Кодда. Структурно данные представляются в естественном виде - таблице (отношении). Таблица представляется как множество строк, каждая из которых имеет одинаковую структуры и состоит из колонок (полей). Типичный представитель - СУБД Oracle, DB2 (IBM).

*Постреляционная модель* представляет собой расширение РМ, которое снимает ограничение неделимости данных, хранящихся в полях таблицы. ПРМ допускает многозначные поля, значения которых состоят из подзначений.

*Многомерная модель* предложена в 1993 году Э.Коддом и связана с возможностью концептуального представления и обработки многомерных данных. Структурно информация представляется в форме многогранного куба - гиперкуба.

*Объектная модель* стандартизирована рекомендациями ODMG-93 (Object Database Management Group) и представляется в виде дерева, узлами которого являются объекты - типы (классы), определяемые пользователем.

### *3.1.2.1. Иерархическая модель данных*

ИМ состоит из упорядоченного набора данных одного типа «дерево».

Тип «*дерево*» состоит из типа «запись» и упорядоченного набора подчиненных типов «дерево».

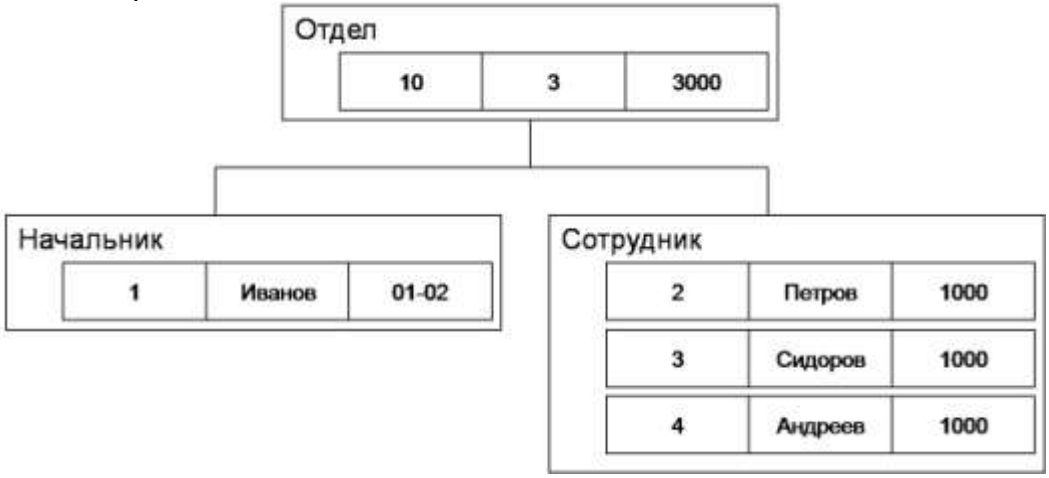
Тип «*запись*» состоит из совокупности элементарных типов данных - целое, строка, указатель и т.д.

В целом тип «дерево» представляет собой иерархически упорядоченный набор типов «запись», который начинается с корневого типа и складывается из подчиненных типов.

*Корневым* называется тип, который может иметь подчиненные типы и сам не является подтипом.

*Подчиненный* тип (подтип) является потомком по отношению к типу, который выступает для него в роли предка (родителя). Потомки одного и того же типа является близнецами по отношению друг к другу.


*Иерархическая БД* представляет собой упорядоченную совокупность деревьев - экземпляров данных типа «дерево», содержащих записи - экземпляры данных типа «запись». Поля записи хранят собственно значения, составляющие основное содержание базы. Обход всех элементов иерархической БД производится сверху вниз и слева направо.




*Основные операции манипулирования* в ИМ:

- поиск указанного дерева (экземпляра);
- переход от одного дерева к другому;
- переход от одной записи к другой внутри дерева;
- переход от одной записи к другой в порядке обхода иерархии;

- вставка новой записи в указанную позицию дерева;
- удаление текущей записи.

*Ограничения целостности*:

- автоматически поддерживается целостность ссылок между предками и потомками - никакой потомок не может существовать баз своего предка, а у некоторых предков может не быть потомков;
- не поддерживается целостность по ссылкам между записями, не входящими в одну иерархию (дерево).

*Преимущества*:

- эффективное использование памяти ЭВМ;
- высокие показатели времени выполнения основных операций над данными;
- удобство при работе с иерархически упорядоченными данными.

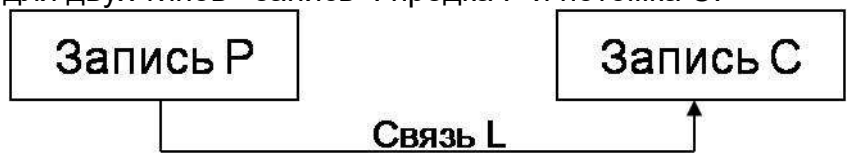
*Недостатки*:

- громоздкость при работе с данными с логически сложными связями;
- сложность понимания для неподготовленного пользователя.

### *3.1.2.2. Сетевая модель данных*

СМ позволяет отображать разнообразные взаимосвязи элементов данных в виде произвольного графа, обобщая тем самым иерархическую модель. В сетевой структуре данных потомок может иметь любое количество предков (а не один, как у ИМ).

*Сетевая модель* состоит из набора типов «запись» и набора типов «связь». Тип «*запись*» состоит из совокупности элементарных типов данных, а тип «связь» определяется для двух типов «запись»: предка P и потомка C.




Для типа связи L с типом записи предка P и типом записи потомка C должны выполняться следующие условия:

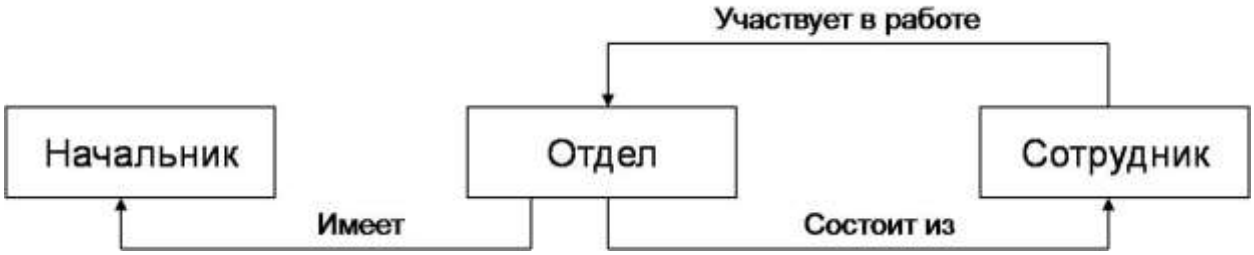
- каждый экземпляр типа P является предком только в одном экземпляре L;
- каждый экземпляр типа C является потомком не более, чем в одном экземпляре L.

На формирование типов связей не накладывается особых ограничений:

- потомок в одном типе связи может быть предком в другом типе связи;
- предок может быть предком в любом числе типов связей;
- предок может быть потомком в любом числе типов связей;
- может быть образовано любое число типов связей между типом записи предка и типом записи потомка, но с разными правилами их образования-родства;
- тип связи может быть рекурсивной;
- предок и потоком могут меняться местами в разных связях.

*Сетевая БД* представляется собой набор экземпляров записей и набор экземпляров связей каждого типа, определенных в схеме базы.

Экземпляр типа «*связь*» состоит из одного экземпляра типа «запись»-предка и упорядоченного набора экземпляров типа «запись»-потомков.




*Основные операции манипулирования* в СМ:

- поиск записи (экземпляра);
- переход от предка к первому потомку;
- переход к следующему потомку по некоторой связи;
- переход от потомка к предку по некоторой связи;
- создание новой записи;
- включение записи в связь и исключение из связи;
- удаление текущей записи.

*Ограничения целостности*:

- в целом, их поддержка не требуется, за исключением целостности ссылок между предками и потомками.

*Преимущества*:

- эффективное использование памяти ЭВМ;
- оперативность управления данными;
- возможность построения эффективных прикладных систем на низком уровне взаимодействия с ЭВМ;
- удобство формирования произвольных связей.

*Недостатки*:

- высокая сложность и жесткость схемы базы данных;
- сложность понимания и обработки информации для неподготовленного пользователя;
- слабый контроль целостности связей из-за возможности установления произвольных связей между записями.

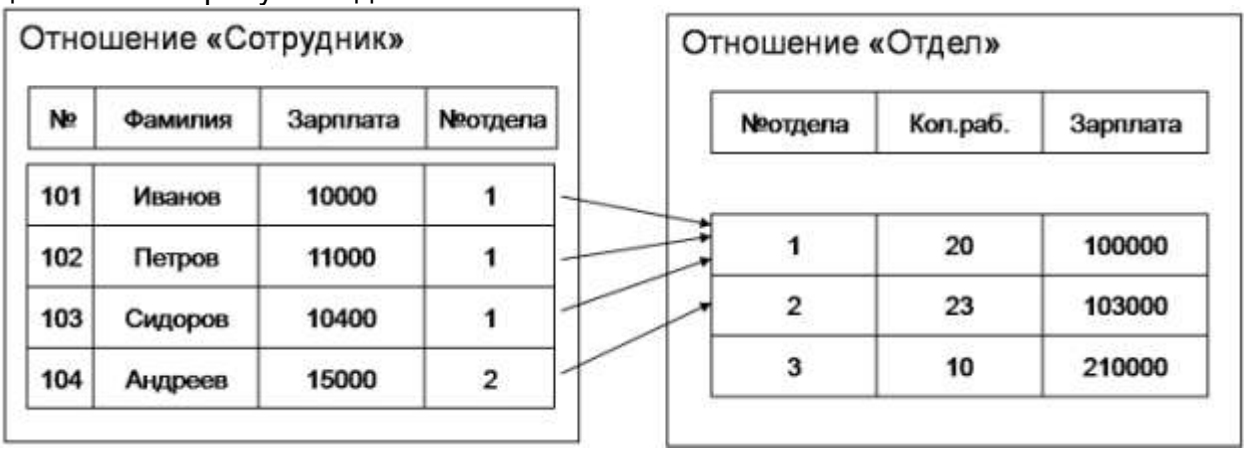
### *3.1.2.3. Реляционная модель данных*

РМ основана на понятии отношения из теории множеств.

*Отношение* представляет собой множество элементов, называемых кортежами.

*Кортежи* отражают информацию о состоянии объекта предметной области. Все кортежи в отношении имеют одинаковую структуру, которая определяется как набор *атрибутов* отношения.


Т.к. в рамках одного отношения не удается описать сложные логические структуры данных из предметной области, применяют связывание отношений по однотипным атрибутам одного назначения.




*Преимущества*:

- простота и удобство физической реализации;
- понятность представления данных для неподготовленного пользователя.

*Недостатки*:

- отсутствие стандартных средств идентификации отдельных записей;
- сложность описания иерархических и сетевых связей.

### *3.1.2.4. Постреляционная модель данных*

ПРМ представляет собой расширенную РМ, снимающую ограничение неделимости данных, представленных кортежами отношения.

Допускаются *многозначные атрибуты* в отношениях - атрибуты, значения которых состоят из подмножества значений. Набор значений многозначного атрибута рассматривается как отдельное отношение, встроенное в основное отношение.


Кроме обеспечения вложенности атрибутов, ПРМ поддерживает:

- ассоциированные многозначные атрибуты - множественные группы;
- переменную длину атрибутов и их количество в отношении;
- возможность хранения ненормализованных (повторяющихся, избыточных) данных.

Совокупность ассоциированных атрибутов называется *ассоциацией*, в которой первое значение первого атрибута соответствует первому значению всех последующих атрибутов и т.д.

Для обеспечения функции контроля целостности значений атрибутов рассматривается возможность создания процедур, автоматически вызываемых до и после обращения к данным (коды корреляции и конверсии). *Коды корреляции* выполняются сразу после обращения к данным, перед их обработкой, а *коды конверсии* - выполняются после обработки.

*Преимущества*:

- возможность компактного представления данных меньшим количеством отношений;
- высокая наглядность представления информации и повышение эффективности ее обработки.

*Недостатки*:

- сложность решения проблемы обеспечения целостности и непротиворечивости хранимых данных.

### *3.1.2.5. Многомерная модель данных*

В концепции развития АИС выделяют следующие направления:

- создание систем оперативной (транзакционной) обработки информации (OLTP - OnLine Transaction Processing);
- создание систем аналитической обработки информации (OLAP - OnLine Analytical Processing).

Реляционная модель данных предназначалась для АИС оперативной обработки информации и в этой области стали весьма эффективной. В системах аналитической обработки РМ показала себя менее эффективной и недостаточно гибкой.

В 1993 году Э.Кодд сформулировал 12 основных требований к системам класса OLAP, среди которых выделил возможность концептуального представления и обработки многомерных данных.

Многомерная модель данных предназначена для создания узкоспециализированных СУБД, ориентированных на интерактивную аналитическую обработку информации.

ММ основывается на следующих свойствах информации:

- *агрегативности* - представление информации на различных уровнях ее обобщения (детальности);
- *историчности* - обеспечение высокого уровня статичности (неизменности) собственно данных и их взаимосвязей при обязательности привязки ко времени;
- *прогнозируемости* - задание функций прогнозирования и применения их к различным временным интервалам.

Многомерность модели данных означает не многомерность их визуализации, а многомерное логическое представление структуры информации при описании и манипулировании данными.

Реляционное представление

Отношение «Продажи»

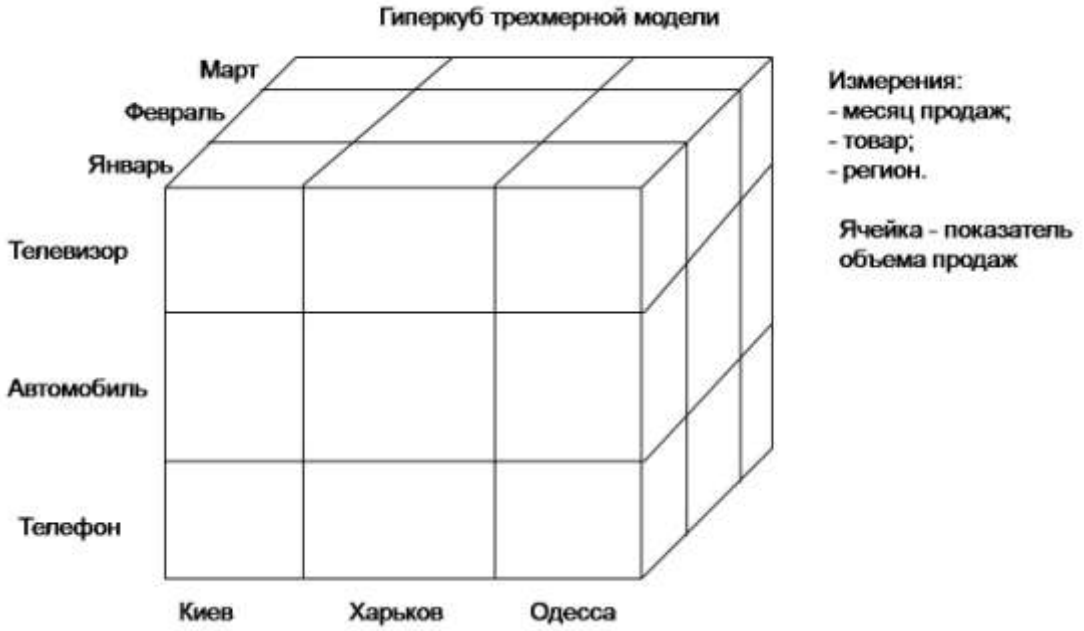
| Товар | Месяц | Объем |
|---|---|---|
| Телевизор | январь | 1000 |
| Телевизор | февраль | 1030 |
| Телевизор | март | 1100 |
| Автомобиль | январь | 1500 |
| Автомобиль | февраль | 1500 |
| Телефон | январь | 1600 |
| Телефон | февраль | 1650 |

Многомерное представление

Отношение «Продажи»

| Товар \ Месяц | Январь | Февраль | Март |
|---|---|---|---|
| Телевизор | 1000 | 1030 | 1100 |
| Автомобиль | 1500 | 1500 | |
| Телефон | 1600 | 1650 | |

Если размерность модели больше 2, то логическая структура информации об объектах представляется в виде многомерных гиперкубов, а пользователь в этом случае работает со срезами (двумерными таблицами) или диаграммами таких гиперкубов.




*Измерение* - это множество однотипных данных, образующих одну из граней гиперкуба. Наиболее часто используемым измерением является временной интервал - день, месяц, квартал, год и т.п. В ММ измерение играет роль индекса, служащего для идентификации конкретных значений в ячейках гиперкуба.

*Ячейка* (или показатель) - это поле, значение которого однозначно определяется фиксированным набором измерений. Тип поля чаще всего определяется как числовой. В зависимости от того, как формируется значение поля, ячейки бывают переменными или формульными.

*Переменные ячейки* имеют значения, которые загружаются из внешнего источника или формируются программно.

*Формульные ячейки* содержат значения, которые вычисляются по заранее заданным формулам на основе данных из других ячеек.

Существует две схемы организации данных в многомерной модели:

- поликубическая;
- гиперкубическая.

В *поликубической схеме* предполагается использование нескольких гиперкубов с различными измерениями и различными их размерностями.

В *гиперкубической схеме* предполагается использование нескольких гиперкубов с одинаковыми измерениями и размерностями, т.е. ячейки гиперкубов определяются одним и тем же набором измерений.

Для манипулирования данными в ММ используются следующие специальные операции: формирование среза, вращение, агрегация, детализация.

*Срез* - подмножество ячеек гиперкуба, полученное в результате фиксации одного или нескольких измерений.

*Вращение* - изменение порядка следования измерений при визуальном представлении данных.

*Агрегация* - переход к более общему представлению информации из гиперкуба.

*Детализация* - переход к более детальному представлению информации из гиперкуба.

Применение операций агрегации и детализации предполагает установления в гиперкубе иерархии отношений между измерениями. Например, для временного интервала - день, месяц, квартал, год, для географического измерения - город, регион, страна.

*Преимущества*:

- удобство и эффективность аналитической обработки больших объемов данных, связанных со временем.

*Недостатки*:

- громоздкость для решения простых задач оперативной обработки информации.

### *3.1.2.6. Объектная модель данных*

Основным отличием ОМ является способность идентификации отдельных записей при представлении данных. Между записями модели и функциями их обработки устанавливается взаимосвязь, аналогичная соответствующим средствам объектно-ориентированных языков программирования.

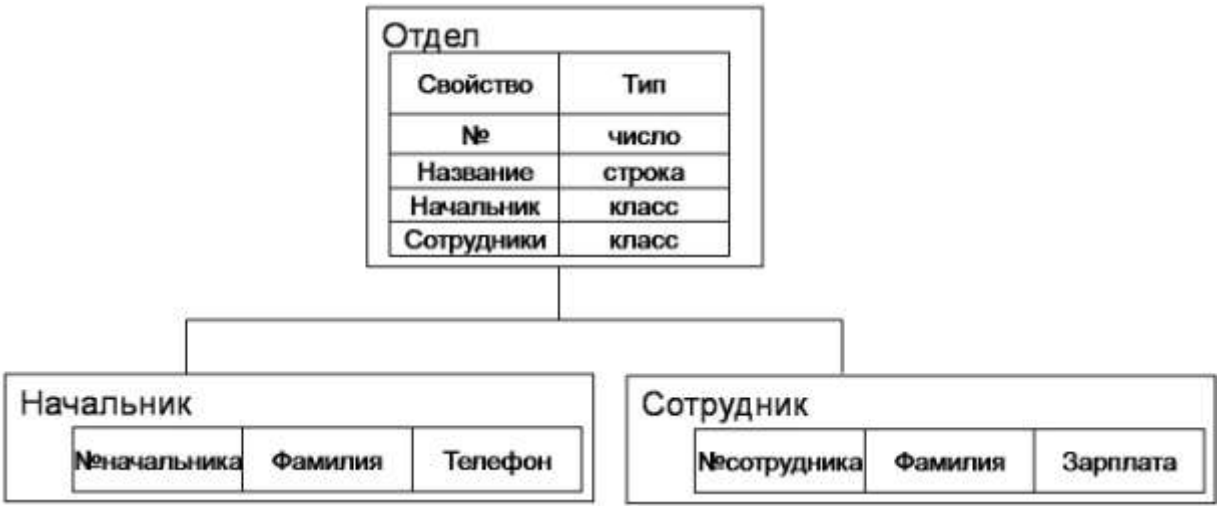
Структурно ОМ представляется в виде дерева, узлами которого являются *объекты*. Свойства объектов описываются стандартными типами или типом, определенным пользователем - *классом объекта*. Значением свойства типа «класса» является объект-экземпляр другого класса. Каждый объект-экземпляр класса является потомком объекта, в котором он определен как свойство, т.е. Принадлежит своему классу и имеет одного родителя.

Логическая структура ОМ подобна ИМ, но отличается методами манипулирования данными - логическим операциями, усиленными объектно-ориентированными механизмами инкапсуляции, наследования и полиморфизма.

*Инкапсуляция* - ограничивает область видимости свойств рамками того объекта, где они определены.

*Наследование* - распространяет область видимости свойств на всех потомков объекта.

*Полиморфизм* - способность одного и того же кода работать с разными типами данных.




*Преимущества*:

- возможность отображения информации о сложных взаимосвязях объектов реального мира;

- возможность идентификации отдельных записей базы данных и определять функции их обработки.

*Недостатки*:

- высокая понятийная сложность для неподготовленного пользователя;

- неудобство обработки информации;

- низкая скорость выполнения запросов.

## Тема 3.2. Реляційна модель даних

### *3.2.1. Базові елементи і поняття РМ*

Основи реляційної моделі даних вперше викладені Е.Коддом в 1970р. Подальший розвиток РМ отримала в роботах К.Дейта та інших дослідників.

Згідно К.Дейта РМ складається з трьох частин:

- структурної частини;
- цілісна частини;
- маніпуляційної частини.

*Структурна частина* визначає, які об'єкти розглядаються РМ.

*Цілісна частина* описує обмеження спеціального виду, які повинні виконуватися для будь-яких об'єктів РМ.

*Маніпуляційна частина* надає механізми маніпулювання і представлення реляційних даних - *реляционную алгебру* і *реляційне числення*.

В цілому, РМ деякої предметної області визначається як набір об'єктів-відношень (*relation*), що змінюються в часі. При створенні АІС сукупність відношень дозволяє зберігати дані про об'єкти предметної області та моделювати зв'язки між ними.

### *3.2.2. Структурна частина РМ*

Основными понятиями РМ являются: тип данных, домен, атрибут, отношение, кортеж, первичный ключ.

***Тип данных*** в РМ полностью соответствует понятию типа в языках программирования. Основным требование РМ к используемым типам данных является их простота, т.е. в реляционных операциях с данными не должна учитываться внутренняя структура данных.

Типа данных делятся на три вида: простые, структурированные, ссылочные.

*Простые* (атомарные) типы не имеют внутренней структуры для заданного уровня семантики данных. К простым относят типы: логический, строковый, численный и их расширения - целый, вещественный, дата, время, денежный, интервальный и т.д.

*Структурированные* типы предназначены для задания сложных структур данных, обладающих общей семантикой. Свойством структурированного типа данных является то, что он имеют внутреннюю структуру, используемую на том же уровне абстракции (семантики), что и сами типы данных - компоненты этой структуры. В качестве структурированных типов рассматривают: массивы, структуры (записи).

Массив представляет собой функцию с конечной областью определения, представленную как отображение $F : I \to R$, где $I$ - множество индексов массива (натуральных чисел), а $R$ - множество элементов массива - значений из области определения: $F(i) = r_i$, $i \in I$, $r_i \in R$.

Структура (запись) представляет собой кортеж (упорядоченный набор элементов) $r = (r_1, r_2, ..., r_n)$ из декартового произведения множеств типов $T_i$, где элемент $r_i \in T_i$.

*Ссылочный* тип (указатель) предназначен для обеспечения возможности указания на другие данные. Применяется для обработки сложных изменяющихся структур - деревьев, графов, рекурсивных структур.

***Домен*** - это семантическое понятие и рассматривается как подмножество значений некоторого типа данных, имеющих определенных смысл: $D = \{d_i | d_i \in T : f(d_i) = true\}$, где

$d_i$ - значение домена,

$T$ - тип данных,

$f$ - функция, определяющая семантику домена.

Свойства домена:

- имеет уникальное имя;
- определен на некотором типе данных или другом домене;
- может иметь некоторое логическое условие, определяющее подмножество данных для данного домена с определенной смысловой нагрузкой.

***Атрибут отношения*** - это пара вида $(A_i, D_i)$, где $A_i$ - наименование атрибута, $D_i$ - домен атрибута.

Свойства атрибута:

- имена атрибутов в отношении уникальны;
- имена атрибутов могут совпадать с именами доменов;
- вместо домена может применяться тип данных.

***Отношение*** *R* - это именованное множество, состоящее из заголовка (схемы отношения) и тела (экземпляра отношения), определенное на множестве доменов $D = \{D_i | i = 1, n\}$.

*Схема отношения* - это именованное множество атрибутов $\{(A_i, D_i) | i = 1, n\}$.

*Тело отношения* - это множество кортежей, соответствующих схеме отношения.

***Кортеж*** - это множество пар вида $(A_i, v_i)$, где $A_i$ - наименование атрибута, $v_i$ - значение атрибута, принадлежащее домену $D_i$.

*Степень (арность) отношения* - количество атрибутов в схеме отношения.

*Мощность отношения* - количество кортежей в теле отношения.

*Реляционной базой данных* называют набор отношений, а *схемой реляционной базы* - набор схем отношений. Схема отношения - статична, т.е. не меняется во время работы базы, а тело отношения - изменяемое.

Свойства отношений:

- отсутствие кортежей-дубликатов;
- отсутствие упорядоченности кортежей;
- отсутствие упорядоченности атрибутов;
- атомарность значений атрибутов.

***Первичным ключом*** называется атрибут или атрибуты отношения, однозначно идентифицирующие каждый из его кортежей. Первичный ключ должен быть минимальным. Такие ключи считаются не избыточными.

Первичные ключи используются для достижения следующих целей:

- исключения дублирования значений;
- упорядочивания кортежей;
- ускорения работы с кортежами;
- организации связывания отношений.

Связывание отношений обеспечивается внешним ключом.

***Внешний ключ*** - это не ключевой атрибут отношения, значениями которого являются значения ключевого атрибута другого отношения.


*3.2.3. Цілісна частина РМ*

В целостной части фиксируется два базовых требования к организации данных в РМ

- целостность сущностей;
- целостность ссылок (связей).

В РМ объекту-сущности предметной области соответствуют кортежи отношений.

*Целостность сущностей* означает, что любой кортеж любого отношения отличим от другого кортежа этого отношения, т.е. отношение должно обладать первичным ключом.

Сложные сущности предметной области представляются в РМ в виде нескольких кортежей разных отношений.

*Целостность ссылок* означает, что для каждого значения ссылки из дочернего отношения (ссылающегося отношения) в родительском отношении (на которое ведет ссылка) должен присутствовать кортеж с таким же значением первичного ключа, т.е. дочернее отношение должно обладать внешним ключом, значение которого должно соответствовать значению первичного ключа родительского отношения или быть неопределенным.

Поддержка целостности ссылок обеспечивается *правилами контроля* значений пары «первичный ключ-внешний ключ»:

- запрет изменения или удаления кортежа, на который существуют ссылки (*блокирующее правило*);
- автоматическое изменение или удаление кортежей, на которые существуют ссылки, вместе с кортежами, которые ссылаются (*каскадное правило*);
- автоматическое удаление кортежей, на которые существуют ссылки с заменой внешний ключей ссылающихся кортежей неопределенными значениями (*правило неопределенных связей*).

*Свойства контроля целостности* связей:

- каждому кортежу родительского отношения соответствует нуль или более кортежей дочерних отношений;
- в дочернем отношении нет кортежей с внешним ключом, для которых отсутствуют кортежи в родительском отношении;
- каждый кортеж с внешним ключом из дочернего отношения имеет связь только с одним кортежем в родительском отношении.

*3.2.4. Маніпуляційна частина РМ*

Манипуляционная часть определяет два фундаментальных механизма манипулирования данными в РМ: *реляционная алгебра* (РА) и *реляционное исчисление* (РИ).

РА определяет *способ получения результата* при манипулирования данными, а РИ - *форму представления результата* манипулирования.

РА основана на теории множеств, РИ - на исчислении предикатов 1 порядка математической логики.

Особенности механизмов манипулирования данными в РМ:

1.*Замкнутость*. Оба механизма являются замкнутыми относительно понятия отношение, т.е. выражения РА и формулы РИ определяются над отношениями и результатом их вычисления также является отношение. В результате любое выражение или формула может быть использована в других выражениях или формулах.

2.*Эквивалентность*. Механизмы РА и РИ эквивалентны, т.е. для любого допустимого выражения РА можно построить формулу РИ, производящую такой же результат, и наоборот.

3.*Выразительная мощность*. Определяется степенью полноты, т.е. способностью представить очень сложные запросы с помощью одного выражения РА или одной формулы РИ. Принято, что РА и РИ обладают большой выразительной мощностью, а язык манипулирования реляционной БД считается реляционно-полным, если любой оператор РА и РИ может быть выражен средствами этого языка.

## Тема 3.3. Реляційна алгебра

### *3.3.1. Базові елементи і поняття РА*

*Реляционная алгебра* представляет собой набор операторов, использующих отношения в качестве аргументов, и возвращающие отношение в качестве результата:

$R=f(R_1,R_2,\ldots,R_n)$.

В реляционных выражениях РА допустима вложенность выражений сколь угодно сложной структуры:

$R=f(f_1(R_1,R_2,\ldots,R_n), f_2(R_1,R_2,\ldots,R_n),\ldots)$.

Каждое отношение в РА имеет уникальное имя. Имя отношения, полученного в результате выполнения оператора РА, определяется в левой его части.

*Операторы РА* конструируются на базе системы операций, в состав которой входят:

- теоретико-множественные операции;
- специальные реляционные операции;
- дополнительные вспомогательные операции.

РА, предложенная Коддом, включающая теоретико-множественные и специальные реляционные операции, обладает несколькими недостатками: избыточностью операций (часть из них выражается через другие) и недостаточной выразительностью при построении сложных реляционных выражений для манипулирования данными.

Поэтому, Дейтом расширен состав операций за счет вспомогательных.

*Теоретико-множественные операции* (согласно РА Кодда) включают классические операции теории множеств:

- объединение;
- пересечение;
- вычитание;
- произведение.

*Специальные реляционные операции* (согласно РА Кодда) - это развитие теоретико-множественных применительно к манипулированию данными:

- выборка (селекция);
- проекция;
- соединение;
- деление.

*Вспомогательные операции* (согласно расширения РА Дейтом):

- переименования атрибутов в отношении;
- расширение отношения за счет образование новых вычисляемых атрибутов;
- вычисление итоговых функций для группы кортежей;
- присвоение результата реляционного выражения;
- вставка кортежей в отношение;
- обновление кортежей отношения;
- удаление кортежей из отношения.

Операции РА могут выполняться над одним отношением (*унарные операции*) или над двумя отношениями (*бинарные операции*).

При выполнении бинарных операций, участвующие в операции отношения должны быть совместимы по типу.

*Совместимыми по типу* называют отношения, имеющие идентичные схемы (заголовки), что означает совпадение множества имен атрибутов и типов соответствующих доменов. Для приведения к совместимости по типу часто применяют вспомогательную операцию переименования атрибута.

*3.3.2. Теоретико-множинні операції РА*

***Объединением*** двух совместимых по типу отношений $R_1$ и $R_2$ называется отношение $R$ с тем же заголовком, что и у отношений $R_1$ и $R_2$, и телом, состоящим из кортежей, принадлежащих или $R_1$, или $R_2$, или обоим отношениям:

$R = R_1$ *UNION* $R_2$.

Свойство объединения: результирующее отношение содержит все кортежи исходных отношений с исключением повторений, т.е. кортежи-дубликаты отсутствуют; результирующее отношение не наследует первичные ключи исходных отношений.

| № | Фамилия | Зарплата |
|---|---|---|
| 101 | Иванов | 10000 |
| 102 | Петров | 10400 |
| 103 | Сидоров | 15000 |

| № | Фамилия | Зарплата |
|---|---|---|
| 101 | Иванов | 10000 |
| 102 | Андреев | 12000 |
| 104 | Сидоров | 15000 |

*UNION*

| № | Фамилия | Зарплата |
|---|---|---|
| 101 | Иванов | 10000 |
| 102 | Петров | 10400 |
| 103 | Сидоров | 15000 |
| 102 | Андреев | 12000 |
| 104 | Сидоров | 15000 |

***Пересечением*** двух совместимых по типу отношений $R_1$ и $R_2$ называется отношение $R$ с тем же заголовком, что и у отношений $R_1$ и $R_2$, и телом, состоящим из кортежей, принадлежащих одновременно $R_1$ и $R_2$:

$R = R_1$ *INTERSECT* $R_2$.

Свойство пересечения: результирующее отношение содержит все одинаковые кортежи исходных отношений, но без их повторения; результирующее отношение не наследует первичный ключ исходных отношений; *A INTERSECT B == A MINUS* (*A MINUS B*).

*INTERSECT*

| № | Фамилия | Зарплата |
|---|---|---|
| 101 | Иванов | 10000 |

***Вычитанием*** двух совместимых по типу отношений $R_1$ и $R_2$ называется отношение $R$ с тем же заголовком, что и у отношений $R_1$ и $R_2$, и телом, состоящим из кортежей, принадлежащих $R_1$ и не принадлежащих $R_2$:

$R = R_1$ *MINUS* $R_2$.

Свойство вычитания: результирующее отношение содержит только те кортежи первого отношения, которые не входят во второе отношение.

*MINUS*

| № | Фамилия | Зарплата |
|---|---|---|
| 102 | Петров | 10000 |
| 103 | Сидоров | 15000 |

***Произведением*** двух отношений $R_1$ и $R_2$ называется отношение $R$, заголовок которого образуется сцеплением заголовков, а тело состоит из по парного сцепления всех кортежей исходных отношений:

$R = R_1$ *TIMES* $R_2$.

Свойства произведения: мощность результирующего отношения равна произведению мощностей исходных отношений, т.к. каждый кортеж первого отношения сцепляется с каждым кортежем второго; перемножать можно любые два отношения не совместимых по типу; если исходные отношения совместимы по типу, то перед выполнением операции произведения необходимо выполнить переименование атрибутов.

| №сотр | Фамилия |
|---|---|
| 101 | Иванов |
| 102 | Петров |
| 103 | Сидоров |

| №отд | Отдел |
|---|---|
| 1 | Технический |
| 2 | Информационный |

TIMES

| №сотр | Фамилия | №отд | Отдел |
|---|---|---|---|
| 101 | Иванов | 1 | Технический |
| 101 | Иванов | 2 | Информационный |
| 102 | Петров | 1 | Технический |
| 102 | Петров | 2 | Информационный |
| 103 | Сидоров | 1 | Технический |
| 103 | Сидоров | 2 | Информационный |

Свойства теоретико-множественных операций:

- все операции являются *ассоциативными*, т.е.

(*A* op *B*) op *C* = *A* op (*B* op *C*),

где op - операции объединения, пересечения, вычитания, произведения;

- операции объединения, пересечения и произведения являются *коммутативными*, т.е.

*A* op *B* = *B* op *A*.

*3.3.3. Спеціальні реляційні операції РА*

***Выборкой*** (***селекцией***) на отношении $R_1$ по условию (формуле) *f* называется отношение *R* с тем же заголовком, что и у исходного отношения, и телом, состоящим из кортежей, удовлетворяющих заданному условию:

$R = R_1$ *WHERE f*.

Кортеж отношения удовлетворяет условию, если его значение становится истинным на значениях атрибутов кортежа.

Условие *f* представляет собой реляционное логическое выражение, в которое входят атрибуты отношения $R_1$, скалярные константы, связанные операциями сравнения и логическими операциями AND, OR, NOT с использованием скобок при формировании сложных выражений.

В простом выражении условие *f* имеет следующий вид:

$X \Theta Y$

где $\Theta$ - один из операторов сравнения =, >, <,≥, ≤, ≠ ;

*X*, *Y* - атрибуты отношения $R_1$ или скалярные значения.

Выборки построенные с использованием простейшего выражения называются $\Theta$-*выборками*:

*R WHERE X* $\Theta$ *Y.*

СОТРУДНИКИ WHERE Зарплата < 15000

| № | Фамилия | Зарплата |
|---|---|---|
| 101 | Иванов | 10000 |
| 102 | Петров | 10400 |
| 103 | Сидоров | 15000 |

WHERE

| № | Фамилия | Зарплата |
|---|---|---|
| 101 | Иванов | 10000 |
| 102 | Петров | 10400 |

Свойства выборки:

- формирование подмножества кортежей исходного отношения, т.е. образование горизонтального среза данных;

- *A WHERE* $f_1$ AND $f_2$ == (*A WHERE* $f_1$) *INTERSECT* (*A WHERE* $f_2$);
- *A WHERE* $f_1$ OR $f_2$ == (*A WHERE* $f_1$) *UNION* (*A WHERE* $f_2$);
- *A WHERE* NOT $f_1$ == *A MINUS* (*A WHERE* $f_1$).

***Проекцией*** отношения $R_1$ на атрибуты $X$, $Y$, … , $Z$ называется отношение $R$ с заголовком ($X$,$Y$,…,$Z$) и телом, содержащим соответствующие заголовку фрагменты кортежей отношения $R_1$ с исключением повторяющихся значений кортежей:

$R = R_1 [X,Y,…,Z]$.

СОТРУДНИКИ[Город]

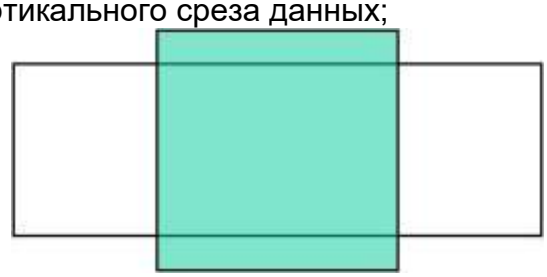
| № | Фамилия | Город |
|---|---|---|
| 101 | Иванов | Киев |
| 102 | Петров | Киев |
| 103 | Сидоров | Харьков |

[Город]

| Город |
|---|
| Киев |
| Харьков |

Свойства проекции:

- образование вертикального среза данных;



- повторение одинаковых атрибутов не допустимо;
- отсутствие списка атрибутов эквивалентно указанию всех атрибутов, т.е. $R$ == $R_1$[*] - тождественная проекция;
- пустой список атрибутов означает пустую проекцию $R$ == $R_1$[ ], результатом которой является пустое множество кортежей.

***Соединением*** двух отношений $R_1$ и $R_2$ по условию $f$ называется отношение $R$, которое образуется путем произведения исходных отношений с последующим применением выборки по заданному условию:

$R$ = ($R_1$ *TIMES* $R_2$)*WHERE f*.

Частные виды операции соединения:

- Θ-соединение - соединение по простому логическому выражению условия;
- эквисоединение - соединение по условию равенства атрибутов;
- естественное соединение - соединение по равенству всех общих атрибутов.

Θ-*соединением* отношений $R_1$ и $R_2$ по атрибутам $X$ и $Y$ называется отношение $R$ = ($R_1$ *TIMES* $R_2$)*WHERE X* Θ *Y*,

где X - атрибут отношения $R_1$;

Y - атрибут отношения $R_2$;

Θ - один из операторов сравнения =, >, <,≥, ≤, ≠ .

Пример выполнения операции соединения по простому условию: необходимо определить какие покупатели и какие товары имеют право их приобретать в зависимости от значения их статуса.

| №пок | Покупатель | X-статус |
|---|---|---|
| 101 | Иванов | 4 |
| 102 | Петров | 1 |
| 103 | Сидоров | 2 |

| №тов | Товар | Y-статус |
|---|---|---|
| 101 | Болт | 3 |
| 102 | Гайка | 2 |
| 103 | Винт | 1 |

Результат операции соединения:

(ПОКУПАТЕЛИ *TIMES* ТОВАРЫ)*WHERE X* > *Y*.

| | №пок | Покупатель | X-статус | №тов | Товар | Y-статус |
|---|---|---|---|---|---|---|
| | 101 | Иванов | 4 | 101 | Болт | 3 |
| Θ | 101 | Иванов | 4 | 102 | Гайка | 2 |
| | 101 | Иванов | 4 | 103 | Винт | 1 |
| | 103 | Сидоров | 2 | 103 | Винт | 1 |

*Эквисоединением* отношений $R_1$ и $R_2$ по атрибутам $X$ и $Y$ называется отношение $R = (R_1\ TIMES\ R_2)WHERE\ X = Y$,

где X - атрибут отношения $R_1$;

Y - атрибут отношения $R_2$.

Пример выполнения операции эквисоединения: необходимо определить какие сотрудники работают и в каких отделах.

| №сотр | Фамилия | №отд_сотр |
|---|---|---|
| 101 | Иванов | 1 |
| 102 | Петров | 2 |
| 103 | Сидоров | 1 |

| №отд | Отдел |
|---|---|
| 1 | Технический |
| 2 | Информационный |

Результат операции эквисоединения:

(СОТРУДНИКИ *TIMES* ОТДЕЛЫ)*WHERE* №отд_сотр = №отд.

| | №сотр | Фамилия | №отд_сотр | №отд | Отдел |
|---|---|---|---|---|---|
| Экви | 101 | Иванов | 1 | 1 | Технический |
| | 102 | Петров | 2 | 2 | Информационный |
| | 103 | Сидоров | 1 | 1 | Технический |

Недостаток: в результирующем отношении появляются два атрибута с идентичными значениями.

*Естественным соединением* отношений $R_1(A_1,A_2,\ldots,A_n,X_1,X_2,\ldots,X_p)$ и $R_2(X_1,X_2,\ldots,X_p,B_1,B_2,\ldots,B_m)$ называется отношение $R$ с заголовком $(A_1,A_2,\ldots,A_n,X_1,X_2,\ldots,X_p,B_1,B_2,\ldots,B_m)$ и телом, содержащим множество кортежей $(a_1,a_2,\ldots,a_n,x_1,x_2,\ldots,x_p,b_1,b_2,\ldots,b_m)$, таких, что $(a_1,a_2,\ldots,a_n,x_1,x_2,\ldots,x_p)\in R_1$, а $(x_1,x_2,\ldots,x_p,b_1,b_2,\ldots,b_m)\in R_2$:

$R = R_1\ JOIN\ R_2$.

Свойства естественного соединения:

- условием соединения является равенство значений всех общих атрибутов;
- общие атрибуты имеют одинаковое наименование и определены на одном и том же домене;
- в синтаксисе операции явно не указываются атрибуты соединения. Естественное соединение производится по всем общим атрибутам;
- в результирующем отношении общие атрибуты не повторяются, т.е.

$R = R_1\ JOIN\ R_2$ ==

$((R_1\ TIMES\ R_2)WHERE\ (R_1.X_1 = R_2.X_1$ AND $R_1.X_2 = R_2.X_2$ AND...))$[A_1,A_2,\ldots,A_n,X_1,X_2,\ldots,X_p,B_1,B_2,\ldots,B_m]$;

- операция является ассоциативной, т.е.

(*A JOIN B*) *JOIN C* = *A JOIN* (*B JOIN* C).

СОТРУДНИКИ *JOIN* ОТДЕЛЫ

*JOIN*

| №сотр | Фамилия | №отд | Отдел |
|---|---|---|---|
| 101 | Иванов | 1 | Технический |
| 102 | Петров | 2 | Информационный |
| 103 | Сидоров | 1 | Технический |

***Делением*** отношения $R_1(A_1,A_2,\ldots,A_n,X_1,X_2,\ldots,X_k)$ на отношение $R_2(X_1,X_2,\ldots,X_k)$ называется отношение $R$ с заголовком $(A_1,A_2,\ldots,A_n)$ и телом, содержащим множество кортежей $(a_1,a_2,\ldots,a_n)$, таких, что для всех кортежей $(x_1,x_2,\ldots,x_k)\in R_2$ в отношении $R_1$ найдется кортеж $(a_1,a_2,\ldots,a_n,x_1,x_2,\ldots,x_k)$:

$R = R_1$ *DEVIDEBY* $R_2$.

Свойства деления:

- условием выполнения является наличие в первом исходном отношении всех атрибутов из второго отношения;
- общие атрибуты имеют одинаковое наименование и определены на одном и том же домене;
- в результирующее отношение общие атрибуты не входят;
- кортеж результата операции является фрагментом кортежа первого отношения, который встречается в сочетании со всеми кортежами второго отношения;
- операция не является ассоциативной;
- $R_1$ *DEVIDEBY* $R_2$ == $R_1[X]$ *MINUS* (($R_1[X]$ *TIMES* $R_2$) *MINUS* $R_1$)$[X]$.

Пример выполнения операция деления: необходимо определить какие поставщики поставляют все типы деталей.

| Поставщик | №детали |
|---|---|
| 1 | 1 |
| 1 | 2 |
| 1 | 3 |
| 2 | 1 |
| 2 | 2 |
| 3 | 1 |

| №детали |
|---|
| 1 |
| 2 |
| 3 |

ПОСТАВКИ *DEVIDEBY* ДЕТАЛИ

*DEVIDEBY*

| Поставщик |
|---|
| 1 |

*3.3.4. Допоміжні операції РА*

***Переименованием*** атрибута $A_1$ отношения $R_1(A_1,A_2,\ldots,A_n)$ называется отношение $R$ с заголовком $(B_1,A_2,\ldots,A_n)$ и телом, содержащим тоже множество кортежей, что и отношение $R_1$ :

$R = R_1$ *RENAME* $A_1$ *AS* $B_1$.

Свойства переименования:

- изменяется только имя атрибута, а не домена, с сохранением множества его значений;
- мощность и значения кортежей отношения не изменяются;
- возможно множественное переименование атрибутов одного отношения, т.е. за одну операцию выполняется изменение имен нескольких атрибутов:

$R_1$ *RENAME* $A_1$ *AS* $B_1$, $A_2$ *AS* $B_2$,…, $A_n$ *AS* $B_n$.

| №сотр | Фамилия | №отд |
|---|---|---|
| 101 | Иванов | 1 |
| 102 | Петров | 2 |
| 103 | Сидоров | 1 |

СОТРУДНИКИ *RENAME* №отд *AS* №отд_сотр

*RENAME*

| №сотр | Фамилия | №отд_сотр |
|---|---|---|
| 101 | Иванов | 1 |
| 102 | Петров | 2 |
| 103 | Сидоров | 1 |

***Расширением*** отношения $R_1(A_1,A_2,\ldots,A_n)$ на атрибут $X$ с помощью выражения $e$ называется отношение $R$ с заголовком $(A_1,A_2,\ldots,A_n,X)$ и телом, содержащим множество кортежей $(a_1,a_2,\ldots,a_n,x)$ таких, что $(a_1,a_2,\ldots,a_n)\in R_1$, а $x=e$:

$R$ = *EXTEND* $R_1$ *ADD* $e$ *AS* $X$.

Свойства расширения:

- новый атрибут не должен входить в состав исходного отношения и не может использоваться в выражении;
- мощность и значения кортежей исходного отношения не изменяются;
- для каждого кортежа исходного отношения рассчитывается значение нового атрибута по правилам, заданным выражением, на основе значений атрибутов данного кортежа;
- рассчитываемое выражение конструируется из арифметических операций, операций сравнения;
- возможно множественное расширение отношения, т.е. за одну операцию выполняется добавление нескольких атрибутов:

*EXTEND* $R_1$ *ADD* $e_1$ *AS* $X_1$, $e_2$ *AS* $X_2$,…, $e_m$ *AS* $X_m$;

- $R_1$ *RENAME* $A_1$ *AS* $B_1$ == (*EXTEND* $R_1$ *ADD* $A_1$ *AS* $B_1$)[$B_1,A_2,\ldots,A_n$].

*EXTEND* ТОВАРЫ *ADD* Цена*Количество *AS* Стоимость

| №тов | Товар | Цена | Количество |
|---|---|---|---|
| 101 | Болт | 10 | 3 |
| 102 | Гайка | 15 | 2 |
| 103 | Винт | 5 | 1 |

EXTEND

| №тов | Товар | Цена | Количество | Стоимость |
|---|---|---|---|---|
| 101 | Болт | 10 | 3 | 30 |
| 102 | Гайка | 15 | 2 | 30 |
| 103 | Винт | 5 | 1 | 5 |

***Подведением итогов*** отношения $R_1(A_1,A_2,\ldots,A_n,X_1,X_2,\ldots,X_k)$ по атрибутам $X_1,X_2,\ldots,X_k$ с помощью выражения $e$ называется отношение $R$ с заголовком $(X_1,X_2,\ldots,X_k,Y)$ и телом, содержащим множество кортежей $(x_1,x_2,\ldots,x_k,y)$, таких, что $(x_1,x_2,\ldots,x_k)\in R_1$, а $y=E$:

$R$ = *SUMMARIZE* $R_1$ *BY* $X_1,X_2,\ldots,X_k$ *ADD* $e$ *AS* $Y$.

Свойства подведения итогов:

- новый атрибут не должен входить в состав исходного отношения, списка атрибутов и не может использоваться в выражении;
- мощность кортежей результирующего отношения не больше (чаще меньше) мощности исходного отношения за счет выполнения над ним проекции и групповых вычислений значений заданного выражения;
- для каждого подмножества кортежей исходного отношения, соответствующих проекции, построенной по списку атрибутов, рассчитывается групповое значение для нового атрибута по правилам, заданным выражением, на основе значений атрибутов подмножества кортежей;
- рассчитываемое выражение конструируется из арифметических операций, операций сравнения и итоговых функций количества *COUNT*, суммы *SUM*, среднего значения *AVG*, минимального значения *MIN*, максимального значения *MAX*;
- возможно множественное подведение итогов отношения, т.е. за одну операцию вычисляется несколько групповых значений:

*SUMMARIZE* $R_1$ *BY* $X_1,X_2,\ldots,X_k$ *ADD* $e_1$ *AS* $Y_1$, $e_2$ *AS* $Y_2$,…, $e_m$ *AS* $Y_m$.

| Поставщик | №детали | Цена |
|---|---|---|
| 1 | 1 | 10 |
| 1 | 2 | 20 |
| 1 | 3 | 30 |
| 2 | 1 | 10 |
| 2 | 2 | 25 |
| 3 | 1 | 15 |

*SUMMARIZE* ПОСТАВКИ *BY* Поставщик *ADD COUNT AS* Количество, *SUM AS* Стоимость

*SUMMARIZE*

| Поставщик | Количество | Стоимость |
|---|---|---|
| 1 | 3 | 60 |
| 2 | 2 | 35 |
| 3 | 1 | 15 |

***Присвоением результата*** отношения $R_1$ называется совместимое по типу отношение $R$ с тем же заголовком и телом, что и у отношения $R_1$:

$R := R_1$.

Свойства присвоения:

- результирующее отношение полностью эквивалентно по структуре и содержанию исходному отношению;
- значения кортежей результирующего отношения, которые были до выполнения операции, полностью теряются.

***Вставкой кортежей*** из отношения $R_1$ называется совместимое по типу отношение $R_2$ с тем же заголовком, что и у отношения $R_1$ и телом, состоящим из кортежей , принадлежащих или $R_1$, или $R_2$, или обоим отношениям:

*INSERT* $R_1$ *INTO* $R_2$.

Свойства вставки:

- результирующее и исходное отношения полностью эквивалентны по структуре;
- значения кортежей результирующего отношения, которые были до выполнения операции, полностью сохраняются;
- действия операции эквивалентны объединению с последующим присвоением результата одному из исходных отношений до объединения:

*INSERT* $R_1$ *INTO* $R_2$ == $R_2 := R_1$ *UNION* $R_2$;

- в качестве исходного отношения может выступать реляционное выражение, построенное на операции расширения и проекции:

*INSERT* (*EXTEND* НОВЫЕ_ОТДЕЛЫ *ADD* 3 AS №отд,
‘Административный’ AS Отдел)[№отд,Отдел] *INTO* ОТДЕЛЫ.

| №отд | Отдел |
|---|---|
| 1 | Технический |
| 2 | Информационный |

*INSERT*

| №отд | Отдел |
|---|---|
| 1 | Технический |
| 2 | Информационный |
| 3 | Административный |

***Обновлением кортежей*** в отношении $R_1(A_1,A_2,\ldots,A_n)$ называется совместимое по типу отношение $R$ с тем же заголовком, что и у отношения $R_1$ и телом, содержащим множество кортежей $(e,a_2,\ldots,a_n)$ таких, что $(a_2,\ldots,a_n)\in R_1$, а e - скалярное выражение обновления:

$R$ = *UPDATE* $R_1$ *SET* $A_1$:=$e$.

Свойства обновления:

- результирующее и исходное отношения полностью эквивалентны по структуре;
- изменяются значения всех кортежей исходного отношения и присваиваются результирующему;
- в качестве исходного отношения может выступать реляционное выражение, построенное на операции выборки, ограничивающий подмножество обновляемых кортежей:

$R$ = *UPDATE* $R_1$ *WHERE f SET* $A_1$:=$e$;

- возможно множественное обновление атрибутов одного отношения, т.е. за одну операцию выполняется изменение значений нескольких атрибутов:

$R$ = *UPDATE* $R_1$ *SET* $A_1$:=$e_1$, $A_2$:=$e_2$,…, $A_n$:=$e_n$.

*UPDATE* ПОСТАВКИ *WHERE* №детали=1 *SET* Цена=5

| Поставщик | №детали | Цена |
|---|---|---|
| 1 | 1 | 10 |
| 1 | 2 | 20 |
| 1 | 3 | 30 |
| 2 | 1 | 10 |
| 2 | 2 | 25 |
| 3 | 1 | 15 |

*UPDATE*

| Поставщик | №детали | Цена |
|---|---|---|
| 1 | 1 | 5 |
| 1 | 2 | 20 |
| 1 | 3 | 30 |
| 2 | 1 | 5 |
| 2 | 2 | 25 |
| 3 | 1 | 5 |

***Удалением кортежей*** из отношения $R_1(A_1,A_2,…,A_n)$ называется совместимое по типу отношение $R$ с тем же заголовком, что и у отношения $R_1$ и телом, не содержащим множество кортежей отношения $R_1$:

$R$ = *DELETE* $R_1$.

Свойства удаления:

- результирующее и исходное отношения полностью эквивалентны по структуре;
- мощность результирующего отношения уменьшается на мощность исходного;
- в качестве исходного отношения может выступать реляционное выражение, построенное на операции выборки, ограничивающей подмножество удаляемых кортежей:

$R$ = *DELETE* $R_1$ *WHERE* $f$;

- *DELETE* $R_1$ == $R_1$ *MINUS* $R_1$;
- *DELETE* $R_1$ *WHERE* $f$ == $R_1$ *MINUS* ($R_1$ *WHERE* $f$).

*DELETE* ПОСТАВКИ *WHERE* №детали=1

*DELETE*

| Поставщик | №детали | Цена |
|---|---|---|
| 1 | 2 | 20 |
| 1 | 3 | 30 |
| 2 | 2 | 25 |

## Тема 3.4. Реляційне числення

### *3.4.1. Базові поняття РЧ*

Основное отличие РИ от РА заключается в том, процесс получения искомого результата в РА описывается явным образом путем указания набора операторов, которые необходимо выполнить для получения искомого отношения, а в РИ указываются свойства этого отношения без конкретизации процедуры его получения.

РА образует процедурный подход к получению искомого результата, а РИ - декларативный (описательный).

Рассмотрим отличие РИ от РА на примере следующего задания формирования запроса: получить фамилию и номер сотрудников, которые числятся начальниками отделов с количеством сотрудников больше 10 для следующих отношений:

| №сотр | Фамилия | №отд_сотр |
|---|---|---|
| 101 | Иванов | 1 |
| 102 | Петров | 2 |
| 103 | Сидоров | 1 |

| №отд | Отдел | Количество | №нач_отд |
|---|---|---|---|
| 1 | Технический | 20 | 101 |
| 2 | Информационный | 10 | 102 |

*Реляционные выражения РА*:

R1:=СОТРУДНИКИ TIMES ОТДЕЛЫ WHERE №сотр=№нач_отд

R2:=R1 WHERE Количество > 10

R3:= R1[Фамилия,№сотр]

*Запрос РИ*:

Выдать «Фамилия» и «№сотр» из «СОТРУДНИКИ» таких, что существуют «ОТДЕЛЫ» с таким же значением в «№нач_отд» и значением «Количество» > 10.

*Базисными понятиями* РИ являются понятия:

- переменная отношения;
- правильно построенная формула, опирающаяся на переменные;
- предикаты и кванторы.

В зависимости от того, что является областью определения переменной, различают РИ кортежей и РИ доменов. В РИ кортежей допустимым значением переменной является кортеж, в РИ доменов - значение домена.

В РИ кортежей формула запроса складывается из *декларативной части* - описания необходимых переменных, и *формульной* - задающей запросное выражение, определяющее искомый результат.

Для определения *кортежной переменной отношения* используется оператор:

RANGEVAR *Variable* RANGES OVER *Relation* | *List*,

где *Variable* - кортежная переменная,

*Relation* - реляционное отношение,

*List* - список реляционных выражений в скобках, разделенных запятой.

Сокращенная запись этого оператора:

RANGE *Variable* IS *Relation* | *List*.

Свойства переменной:

- элементами списка выступают отношения или реляционные выражения, построенные на отношениях или переменных отношений;
- все элементы списка совместимы по типу;
- область допустимых значений переменной образуется объединением значений (кортежей) всех элементов списка.

Примеры объявлений переменных:

RANGEVAR *Empl* RANGES OVER СОТРУДНИКИ;

RANGEVAR *Dept* RANGES VAR ОТДЕЛЫ;

RANGEVAR *Empl_Dept* RANGES OVER (СОТРУДНИКИ TIMES ОТДЕЛЫ WHERE №сотр=№нач_отд);

*3.4.2. Побудова запису виразу РЧ*

Для определения *формулы запроса* используется форма Бэкуса-Наура (запись выражения реляционного исчисления кортежей):

*Target_List* WHERE *wff,*

где *Target_List* - список целевых элементов запроса, разделенных запятой;

*wff* - правильно построенная формула (ППФ), служащая для выражения условия истинности значений кортежных переменных.

Значением выражения формулы является отношение, заголовок которого представлен целевым списком, а тело - набором кортежей, определенных по ППФ.

Общий смысл записи выражения исчисления заключается в перечислении атрибутов искомого (целевого) отношения, атрибуты которого должны удовлетворять условию истинности ППФ.

В терминах РА: целевой список - это операция проекции по атрибутам, а ППФ - операция выборки кортежей.

*Целевой список* состоит из элементов:

*Variable*,*Variable.Attribute*,*Variable.Attribute* AS *New_Attribute*,

каждый из которых может иметь следующий вид:

- *Variable* - переменная отношения, ранее определенная в декларативной части. Эквивалентно перечислению всех атрибутов переменой отношения;
- *Variable.Attribute* - атрибут переменной отношения;
- *Variable.Attribute* AS *New_Attribute* - ссылка (указание) на новое имя атрибута переменной отношения.

*Правильно построенная формула* обеспечивает средство формулировки условия отбора кортежей и состоит из логического выражения, построенного на других ППФ или простых условий сравнения атрибутов переменных отношений, с использованием логических операций NOT, AND, OR и кванторов EXISTS, FORALL.

Простым условием сравнения атрибутов считается ППФ:

(*Variable*1.*Attribute*1 Θ *Variable*2.*Attribute*2),

где Θ - операция сравнения.

Использование логических операций должно учитывать возможность получение неопределенного значения у атрибута переменной отношения.

В РИ кортежей применяется трехзначная логика определения значений логических операций:

| AND | False | True | Null |
|---|---|---|---|
| False | *False* | *False* | *False* |
| True | *False* | *True* | *Null* |
| Null | *False* | *Null* | *Null* |

| OR | False | True | Null |
|---|---|---|---|
| False | *False* | *True* | *Null* |
| True | *True* | *True* | *True* |
| Null | *Null* | *True* | *Null* |

| NOT | |
|---|---|
| False | *True* |
| True | *False* |
| Null | *Null* |

ППФ (Null = Null) дает неопределенный результат Null.

ППФ (Null ≠ Null) дает неопределенный результат Null.

Для проверки существования неопределенного значения применяется предикат IS:

*Variable*.*Attribute* IS NULL,

*Variable*.*Attribute* IS NOT NULL.

Квантор существования EXISTS *Variable* (*wff*) принимает истинное значение, если существует по крайней мере одно значение переменной *Variable*, для которой вычисление ППФ *wff* дает значение истина:

EXISTS $v(f)$ = *False* OR $f(v_1)$ OR $f(v_2)$ OR … OR $f(v_n)$.

RANGE Сотр1 IS СОТРУДНИКИ;
RANGE Сотр2 IS СОТРУДНИКИ;
Сотр1.№сотр,Сотр1.Фамилия WHERE
EXISTS Сотр2(Сотр1.№отд=Сотр2.№отд);

Квантор всеобщности FORALL *Variable* (*wff*) принимает истинное значение, если для всех значений переменной *Variable* вычисление ППФ *wff* дает значение истина:

FORALL $v(f)$ = *True* AND $f(v_1)$ AND $f(v_2)$ AND … AND $f(v_n)$.
RANGE Сотр1 IS СОТРУДНИКИ;
RANGE Сотр2 IS СОТРУДНИКИ;
Сотр1.№сотр,Сотр1.Фамилия,Сотр1.Зарплата WHERE
FORALL Сотр2(Сотр1.Зарплата>Сотр2.Зарплата);

Взаимосвязь кванторов:

FORALL $v(f)$ == NOT EXISTS $v$(NOT $f$).

*Запрос*:

Выдать «Фамилия» и «№сотр» из «СОТРУДНИКИ» таких, что существуют «ОТДЕЛЫ» с таким же значением в «№нач_отд» и значением «Количество» > 10.

*Результат*:

RANGE Сотр IS СОТРУДНИКИ;
RANGE Отд IS ОТДЕЛЫ;
Сотр.Фамилия,Сотр.№сотр WHERE
EXISTS Отд (Сотр.№отд=Отд.№отд AND Отд.Количество>10);

## Тема 3.5. Методи проектування реляційної бази даних

### *3.5.1. Проблеми проектування реляційної бази даних*

*Основные проблемы*, возникающие при проектировании структур данных в отношениях реляционной модели:

- избыточность данных;
- аномалии при манипулировании данными.

***Избыточность данных*** означает повторение (дублирование) данных в базе.

Различают *простое* и *избыточное* дублирование. Наличие первого является технологической необходимостью при поддержании связывания отношений в реляционной базе, а второе - признаком плохой структурированности отношений, что может приводить к проблемам при обработке данных.

Избыточное дублирование усложняет механизмы поддержания целостности - полноты и непротиворечивости данных в базе, порождает проблемы аномалии при манипулировании с дублированными данными.

*Пример избыточного дублирования данных*

| №сотр | Фамилия | №отд | Отдел | Зарплата |
|---|---|---|---|---|
| 101 | Иванов | 1 | Технический | 10000 |
| 102 | Петров | 2 | Информационный | 20000 |
| 103 | Сидоров | 1 | Технический | 10000 |

*Пример простого дублирования данных*

| №сотр | Фамилия | №отд |
|---|---|---|
| 101 | Иванов | 1 |
| 102 | Петров | 2 |
| 103 | Сидоров | 1 |

| №отд | Отдел | Зарплата |
|---|---|---|
| 1 | Технический | 10000 |
| 2 | Информационный | 20000 |

***Аномалиями*** называют такую ситуацию в отношениях базы данных, которая приводит к противоречиям в данных или существенно усложняет их обработку.

Различают три основных *вида аномалии*:

- аномалия обновления;
- аномалия удаления;
- аномалия ввода.

*Аномалия обновления* - означает противоречивость данных, вызванная их избыточностью и частичным обновлением.

Проявляется в том, что изменение значения одного кортежа может повлечь просмотр всего отношения и внесение изменений в другие кортежи.

Например, обновление зарплаты некоторому отделу вызовет обновление кортежей всех сотрудников данного отдела.

*Аномалия удаления* - означает непреднамеренную потерю данных, вызванную удаление других данных.

Проявляется в том, что при удалении значения одного кортежа может исчезнуть информация, которая не связана напрямую с удаляемыми данными кортежа.

Например, удаление кортежа о единственном сотруднике некоторого отдела вызовет потерю информации о самом отделе, в котором он работал.

*Аномалия ввода* - означает невозможность внесения данных, вызванная отсутствием других данных.

Проявляется в том, что данные о новом кортеже нельзя внести в отношение до тех пор, пока информация не станет полной или вставка потребует просмотра всего отношения.

Например:

-ввод нового сотрудника в существующий отдел потребует определения зарплаты данного отдела,

-ввод нового сотрудника в новый отдел потребует полноту информации,

-ввод только нового отдела невозможно без знания информации о его сотрудниках.

*3.5.2. Метод нормалізації реляційної бази даних*

При проектировании базы данных выделяют три *основных подхода* к созданию структур данных АИС:

1.Сбор данных об объектах предметной области АИС в рамках одного отношения с последующей декомпозицией его на несколько взаимосвязанных отношений с помощью процедуры метода нормализации - классический подход.

2.Формулирование знаний о предметной области АИС (определение типов исходных данных, их взаимосвязей) и требований их обработки с помощью систем автоматизации проектирования и разработки баз данных с последующим формированием готовой схемы базы данных - подход на основе CASE-систем функционального и семантического моделирования предметной области АИС.

3.Структурирование информации об объектах предметной области АИС с целью использования в объектно-ориентированных системах в процессе системного анализа на основе совокупности правил и рекомендаций - поход адаптации универсальных проблемно-ориентированных систем на объект автоматизации.

Проектирование РБД представляет собой процесс последовательного приближения набора схем отношений к *нормализованному состоянию*.

**Нормализация** - процесс приведения реляционных отношений к стандартному виду, обеспечивающему решение проблем избыточности и аномалии.

Решение проблем достигается процедурой разделения (декомпозиции) исходного отношения на несколько новых отношений, обладающих лучшими свойствами по сравнению с исходным отношением. Эта процедура декомпозиции является основной процедурой *метода нормализации отношений*.

**Метод нормализации отношений** основан на понятии зависимости между атрибутами отношения, выявление которой необходимо для проектирования РБД.

*Выявление зависимостей* основано на анализе семантики атрибутов.

Выделяют *три вида зависимостей* между атрибутам отношения:

- функциональные;
- транзитивные;
- многозначные.

В отношении атрибут *B* **функционально зависит** от атрибута *A*, если каждому значению *A* соответствует только одно значение *B*:

$A \to B$ .

Это означает, что во всех кортежах отношения для одного значения *A* будет соответствовать только одно определенное значение *B*.

Атрибут *A* в функциональной зависимости $A \to B$ называется *детерминантом*, т.к. его значения определяют значения второго атрибута зависимости.

Атрибуты *A* и *B* могут быть составными.

Функциональная зависимость $A \to B$ называется *полной*, если атрибут *B* не зависит функционально от любого подмножества атрибута *A*, т.е. значение *B* определяется только от всего значения *A*, а не от его части.

В отношении между атрибутами A и B устанавливается *функциональная взаимозависимость*, если существуют функциональные зависимости $A \to B$ и $B \to A$, т.е. между значениями атрибутов имеется взаимно однозначное соответствие:

$A \leftrightarrow B$ .

В отношении атрибут С **транзитивно зависит** от атрибута А, если существует некоторый атрибут В такой, что выполняются условия существования функциональных зависимостей $A \to B$, $B \to C$ , и отсутствует обратная функциональная зависимость $C \to A$:

$A \to B \to C$.

В отношении атрибут *B* **многозначно зависит** от атрибута *A*, если каждому значению *A* соответствует множество значений *B*:

$A \Rightarrow B$ .

Многозначные зависимости могут быть вида «*один ко многим*» $A \Rightarrow B$, «*многие к одному*» $A \Leftarrow B$, «*многие ко многим*» $A \Leftrightarrow B$.

*Ключом* отношения являются атрибуты-детерминанты функциональных зависимостей, т.к. его значения однозначно определяют значения каждого атрибута.

*Неключевым атрибутом* называется любой атрибут отношения, не входящий в состав ключа.

Два и более атрибутов отношения называются *взаимно независимыми*, если ни один из этих атрибутов не является функционально зависимым от других.

Т.к. существование различных зависимостей между атрибутами отношения являются причинами возникновения избыточности и аномалий, то метод нормализации предполагает расчленение отношения с функциональными зависимостями атрибутов на несколько связанных отношений со связями типа «1:1», «1:М», «М:М», т.е. связи между образованными отношениями должны отражать зависимости между атрибутами исходного отношения.

**Процедура декомпозиции** исходного отношения $R(\underline{A},B,C,D)$ (приведение к виду, обеспечивающее выполнение некоторого правила нормализации - устранение зависимостей $F_1{:}B \to C$, $F_2{:}B \to D$):

1.Создается новое отношение, атрибутами которого будут атрибуты из исходного отношения, входящие в противоречащую правилу функциональную зависимость. Детерминант функциональной зависимости становится ключом: $R_1(\underline{B},C)=R(\underline{A},B,C,D)[B,C]$, $R_2(\underline{B},D)=R(\underline{A},B,C,D)[B,D]$.

2.Атрибут, стоящий в правой части функциональной зависимости, исключаются из исходного отношения: $R`(\underline{A},B)=R(\underline{A},B,C,D)[A,B]$.

3.Если один и тот же детерминант входит в несколько функциональных зависимостей, то все функционально зависящие от него атрибуты помещаются в качестве неключевых атрибутов в общее отношение, ключом которого буде детерминант: $R_{12}(\underline{B},C,D)=R(\underline{A},B,C,D)[B,C,D]$.

4.Если в исходном отношении существует более одной функциональной зависимости, нарушающие заданное правило нормализации, то п.1,2,3 повторяются для каждой такой зависимости.

*Пример не нормализованного отношения и его аномалии*.

Дано отношение:

СОТРУДНИКИ-ОТДЕЛЫ-ПРОЕКТЫ (№сотр, зарплата, №отд, №проекта, задание)

Первичный ключ отношения: №сотр, №проекта

Функциональные зависимости:

№сотр $\to$ зарплата

№сотр $\to$ №отд

№отд $\to$ зарплата

№сотр,№проекта $\to$ задание

Свойства отношения:

- первичный ключ составной;

- атрибуты зарплата, №отд функционально зависят атрибута №сотр, который является частью ключа;
- аномалия ввода при вставке нового кортежа, описывающего нового сотрудника, который еще не выполняет никакого проекта (первичный ключ не может содержать неопределенные значения);
- аномалия удаления из-за возможности разрушения связи данного сотрудника с отделом при удалении информации о связи сотрудника с проектом;
- аномалия обновления при переводе сотрудника в другой отдел с вынужденной модификацией всех кортежей, описывающей этого сотрудника.

*3.5.3. Нормальні форми відношень реляційної бази даних*

Процесс проектирования РБД с использованием метода нормальных форм является итерационным и заключается в последовательном переводе отношений из одной нормальной формы к другой. При этом каждая следующая форма обладает лучшими (с точки зрения избыточности и аномалий) свойствами, чем предыдущая.

Под **нормальной формой отношения** понимают некоторый определенный набор ограничений (условий), устраняющий отдельные зависимости атрибутов отношения (как причины появления избыточности и аномалий).

Таким образом, каждая следующая нормальная форма обладает следующими *свойствами*:

- ограничивает определенный тип зависимостей атрибутов в отношении,
- устраняет соответствующие зависимости аномалии при выполнении операций манипулирования над отношениями в РБД,
- сохраняет свойства предыдущих форм.

В теории РБД выделяют следующую *последовательность НФ*:

- 1 НФ (обеспечение атомарности атрибутов);
- 2 НФ (устранение ФЗ неключевых атрибутов от других таких же);
- 3 НФ (устранение транзитивных ФЗ);
- 3 расширенная НФ - форма Бойса-Кодда (при составном ключе);
- 4 НФ (устранение многозначных зависимостей);
- 5 НФ (устранение зависимостей соединений проекций отношения).

**Первая Нормальная Форма**.

Отношение находится в 1НФ, если все его атрибуты являются атомарными.

По определению Кодда любое отношение РМ находится в 1НФ.

**Вторая Нормальная Форма**.

Отношение находится во 2НФ, если оно находится в 1НФ и каждый неключевой атрибут полностью функционально зависит от ключа (составного).

Таком образом, неключевые атрибуты не могут функционально зависеть от части ключа. Фактически этим определением для простых ключей утверждается обязательное их существование, а при составных - его минимальность.

В результате перевода отношения во 2НФ позволяет исключить явную избыточность данных с сохранением некоторого неявного их повторения.

**Третья Нормальная Форма**.

Отношение находится в 3НФ, если оно находится во 2НФ и каждый неключевой атрибут не является транзитивно зависимым от ключа.

Таким образом, атрибуты, не входящие в первичный ключ, не могут зависеть от других атрибутов, не входящих в этот ключ. Фактически утверждается взаимная независимость неключевых атрибутов и их полная зависимость от первичного ключа.

В результате перевода отношения в 3НФ позволяет исключить полностью избыточное дублирование данных, приводящее к аномалиям ввода, удаления и обновления.

На практике приведение отношений к 3НФ достаточно для построения хороших в определенном ранее смысле РБД.

**Нормальная Форма Бойса-Кодда**.

Если в отношении существует зависимость атрибутов составного ключа от неключевых атрибутов, то Бойсом и Коддом предложена расширенная (усиленная) версия 3НФ.

Отношение находится в НФБК, если оно находится в 3НФ и в нем отсутствуют зависимости атрибутов ключей (составных) от неключевых атрибутов.

Таким образом, каждый детерминант функциональных зависимостей отношения является ключем отношения, т.е. отсутствует транзитивное замыкание (транзитивная взаимозависимость).

**Четвертая Нормальная Форма.**

Отношение находится в 4НФ, если оно находится в 3НФ и в нем не содержится многозначных зависимостей.

Таким образом, если в отношении существует многозначная зависимость $A \Rightarrow B$, то все остальные атрибуты отношения функционально зависят только от $A$.

Проблема многозначных зависимостей связана с наличием в отношении многозначных атрибутов, а ее решение достигается размещением многозначных атрибутов в дополнительном отношении вместе с ключом, от которого атрибут зависит - ключом исходного отношения.

**Пятая Нормальная Форма.**

Результатом нормализации предыдущим форм является образования вместо одного исходного отношения двух новых. Если существует зависимость соединения (повторение значений атрибутов в нескольких кортежах, например, когда ключом отношения являются все атрибуты), то выполнить декомпозицию на два отношения не представляется возможным.

Отношение $R(X,Y,\ldots,Z)$ удовлетворяет *зависимости соединения* $*(X,Y,\ldots,Z)$ в том и только в том случае, если $R$ восстанавливается без потерь путем соединения своих проекций по $X,Y,\ldots,Z$.

Зависимость соединения является обобщением ФЗ и МЗ.

Отношение находится в 5НФ в том и только том случае, если любая зависимость соединения в отношении следует из существования некоторого возможного ключа в отношении.

Результатом 5НФ является декомпозиция отношения на три и более отношений.

5НФ - последняя форма, которую можно получить путем декомпозиции, но проверка ее условий является не тривиальной задачей и поэтому не используется на практике.

# Розділ 4. Мови реляційної бази даних

## Тема 4.1. Мовні засоби визначення даних у реляційній базі даних

### *4.1.1. Призначення, основні функції і стандартизація мов РБД*

Прототип современного языка SQL для взаимодействия с БД появился в середине 70-х и был разработан в рамках проекта экспериментальной реляционной СУБД System_R.

*Язык Structured Query Language* (SQL) предназначен для авторизации доступа, формулирования запросов и манипулирования данными средствами определения и манипулирования схемой БД, определения ограничений целостности и управления транзакциями, определения структур физического уровня, поддерживающих эффективное выполнение запросов.

*Основные функции* языка:

- запросы и манипулирование данными;
- определение и манипулирование схемой БД;
- определение ограничений и программных механизмов (триггеров) поддержки целостности данных в БД;
- организация представлений данных;
- определение и управление физическими структурами, поддерживающих эффективное выполнение запросов к БД;
- авторизация доступа к схеме БД и ее данным;
- управление механизмом выполнения транзакций;
- интеграция в традиционные языки программирования (встроенный SQL);
- поддержка интерактивного взаимодействия с БД (динамический SQL).

*Стандартизация языков РБД*

Появление теории реляционных баз данных и предложенного Коддом языка запросов ”Alpha", основанного на реляционном исчислении, инициировало разработку ряда языков запросов, которые можно отнести к двум классам:

- алгебраические языки, позволяющие выражать запросы средствами специализированных операторов, применяемых к отношениям;
- языки исчисления предикатов представляют собой набор правил для записи выражения, определяющего новое отношение из заданной совокупности существующих отношений.

Разработка, в основном, шла в отделениях фирмы IBM (языки ISBL, SQL, QBE) и университетах США (PIQUE, QUEL).

Сегодня из всех этих языков полностью сохранились и развиваются QBE (Query-By-Example - запрос по образцу) и SQL. В начале 80-х годов SQL стал фактическим стандартом таких языков для профессиональных реляционных СУБД, а 1989 году он стал международным стандартом языка баз данных.

Версии стандарта языка SQL:

1989 год - первый стандарт SQL/89;

1992 год - второй стандарт SQL/92;

1999 год - третий стандарт SQL/99;

2004 год - четвертый стандарт SQL2004.

*Стандарт SQL/89* во многих частях имеет чрезвычайно общий характер и допускает очень широкое толкование. В этом стандарте полностью отсутствуют такие разделы как манипулирование схемой БД и динамический SQL. Многие существенные аспекты языка в соответствии со стандартом определяются в реализации.

*Стандарт SQL/92* существенно более полный и охватывает практически все необходимые для реализации аспекты: манипулирование схемой БД, управление

транзакциями и сессиями (сессия - это последовательность транзакций, в пределах которой сохраняются временные отношения), подключение к БД, динамический SQL.

*Стандарт SQL/99* содержит механизмы триггеров и возможность использования абстрактных типов данных.

*Стандарт SQL/2004* содержит механизмы опубликования данных и взаимодействия с БД через Internet и поддержки языка UML.

Логическое деление языка SQL на:

- язык определения данных DDL;
- язык манипулирования данными DML;
- язык управления данными DCL.

### *4.1.2. Опис даних і їхніх типів у РБД*

Існує достатня розмаїтість типів даних у реляційній базі для адекватного представлення значень різних властивостей. Основними типами даних є символ, число, дата і бінарне число. Наприклад, у СУБД Oracle встановлені такі подання основних типів:

| Тип даних | Опис |
|---|---|
| VARCHAR2(*розмір*) | Символьні значення змінної довжини, що не перевищує заданого *розміру.* Мінімальна довжина дорівнює 1, максимальна – 4000. Значення розміру обязательно. |
| CHAR(*розмір*) | Символьні значення фіксованої довжини, рівної *розміру.* Довжина за замовчуванням складає 1, максимальна – 255. Значення розміру за замовчуванням 1. |
| NUMBER | Число з крапкою, що плаває, з точністю 38 значущих цифр. |
| NUMBER(*p,s*) | Числове значення, що має максимальну точність *p* від 1 до 38 і максимальний масштаб *s*; точність – це загальна кількість десяткових цифр, а масштаб – кількість цифр праворуч від десяткової крапки. |
| DATE | Значення дати і часу між 1 січня 4712 до н.е. і 31 грудня 4712 н.е. |
| LONG | Символьні значення змінної довжини розміром до 2 гігабайтів. У таблиці допускається тільки один стовпець типу LONG. В останній версіях СУБД Oracle використовується тип CLOB для символьних значень розміром до 4 гігабайт. |
| RAW і LONG RAW | Аналогічні, відповідно, типам данных VARCHAR2 і LONG, але використовуються для збереження байт-орієнтованих або двійкових даних, що не повинні інтерпретуватися сервером Oracle. В останній версіях СУБД Oracle використовується тип BLOB для двійкових даних розміром до 4 гігабайт. |

Узагальнений синтаксис специфікації подання типу даних у команді створення таблиці:

<*стовпець*> <*тип_данных*> [DEFAULT <*вираз*>] [<*обмеження_стовпця*>]

де

<*стовпець*> – ім'я стовпця,

<*тип_даних*> – тип даних і довжина стовпця,

DEFAULT <*вираз*> – визначає <*вираз*> як значення за замовчуванням, використовуване при відсутності значення в команді вставки нового рядка,

<*обмеження_стовпця*> – правило цілісності даних як частина визначення стовпця (визначення обмежень цілісності).

### *4.1.3. Оператори визначення таблиць, представлень і індексів даних*

Спецификация оператора определения таблицы данных:

<table_definition>::= CREATE TABLE <table_name>(<table_element> [{,<table_element>}...])

<table_name>::= <schema>.<name>

<table_element>::= <column_definition> | <table_constraint_definition>

<column_definition>::= <column_name> <data_type> [<default_clause>] [<column_constraint>...]

<default_clause>::= DEFAULT { <literal> | USER | NULL }

<column_constraint>::= [CONSTRAINT <constraint_name>] NOT NULL | [<unique_specification>] | <references_specification> | CHECK (<search_condition>)

<table_constraint_definition>::= <unique_constraint_definition> | <referential_constraint_definition> | <check_constraint_definition>

<unique_constraint_definition>::= [CONSTRAINT <constraint_name>] <unique_specification> (<column_list>)

<unique_specification>::= UNIQUE | PRIMARY KEY

<column_list>::= <column_name> [{,<column_name>}...]

<referential_constraint_definition>::= FOREIGN KEY (<column_list>) <references_specification>

<references_specification>::= REFERENCES <referenced_table> [ON DELETE CASCADE]

<referenced_table>::= <table_name> [(<column_list>)]

<check_constraint_definition>::= CHECK (<search_condition>)

Спецификация оператора определения таблицы данных по шаблону:

<table_definition>::= CREATE TABLE <table_name>(<column_name> [{,<column_name>}...]) AS <select_subquery>

Спецификация оператора внесения изменений в определение таблицы данных:

<table_change_definition>::= ALTER TABLE <table_name> <column_modification> | <constraint_modification>

<column_modification>::= <add_column> | <modify_column> | <delete_column>

<add_column>::= ADD (<column_definition> [{, <column_definition>}…])

<modify_column>::= MODIFY (<column_definition> [{, <column_definition>}…])

<delete_column>::= DROP COLUMN <column_name>

<constraint_modification>::= <add_constraint> | <delete_constraint> | <change_constraint>

<add_constraint>::= ADD <table_constraint_definition>

<delete_constraint>::= DROP <table_constraint_definition> | CONSTRAINT <constraint_name> [CASCADE]

<change_constraint>::= DISABLE | ENABLE CONSTRAINT <constraint_name> [CASCADE]

Спецификация оператора удаления таблицы данных:

<table_delete_definition>::= DROP TABLE <table_name> [CASCADE CONSTRAINTS]

Спецификация оператора переименования таблицы данных:

<table_rename_definition>::= RENAME TABLE <table_name> TO <table_name>

Спецификация оператора определения индекса таблицы данных:

<index_definition>::= CREATE INDEX <index_name> ON <table_name> (<column_name> [{,<column_name>}...])

Спецификация оператора удаления индекса таблицы данных:

<index_delete_definition>::= DROP INDEX <index_name>

Спецификация оператора определения представления данных:

<view_definition>::= CREATE VIEW <view_name>(<column_name> [{,<column_name>}...]) AS <select_subquery> [WITH CHECK OPTION]

Спецификация оператора удаления представления данных:

<view_delete_definition>::= DROP VIEW <view_name>

### *4.1.4. Опис цілісності даних у РБД*

Обмеження цілісності діють на рівні бази даних в цілому як настанови (правила). Вони використовуються в наступних цілях:

1) Для реалізації правил забезпечення цілісності даних на рівні таблиці при операціях вставки, оновлення і видалення рядків. Якщо обмеження задане, то успішне виконання операції без його дотримання неможливо;

2) Для запобігання видалення таблиці, якщо вона залежить від інших таблиць;

3) Для оснащення прикладними (специфічними до процесів обробки даних) правилами інформаційних систем, які взаємодіють з базою даних.

Типи обмежень цілісності даних наведені в таблиці:

| Обмеження | Опис |
|---|---|
| NOT NULL | Означає, что даний стовпець не може містити невизначених значень. |
| UNIQUE | Указує, что чи стовпець набір стовпців містить значення, що повинні бути унікальними для всіх рядків таблиці. |
| PRIMARY KEY | Унікально ідентифікує кожен рядок таблиці. |
| FOREIGN KEY | Встановлює і підтримує відношення між даним стовпцем і стовпцем таблиці, на яку робиться посилання, за допомогою правила зовнішнього ключа. |
| CHECK | Задає довільну умову, що повинне перевірятися. |

Звичайно обмеження створюються одночасно зі створенням таблиці. Але додавати їх можна і після створення таблиці. Крім того, обмеження можуть бути тимчасово заборонені, тобто призупені у дії.

Обмеження можуть бути задані на одному з двох рівнів опису у команді створення таблиці або окремою командою, що вносить зміни до вже існуючій таблиці:

| Рівень | Опис |
|---|---|
| Стовпець | Посилається на один стовпець і описується в межах характеристик відповідного стовпця. Дозволяє задати правило цілісності будь-якого типу. |
| Таблиця | Посилається на один або декілька стовпців і описується окремо від визначеня стовпців у даній таблиці. Дозволяє задати будь-які обмеження, крім NOT NULL. |

Узагальнений синтаксис специфікації подання обмеження на рівні стовпця у команді створення таблиці:

*<стовпець>* <*тип_данных*> [DEFAULT <*вираз*>]

[[CONSTRAINT <*ім'я_обмеження*>] <*тип_обмеження*>] …

На рівні стовпця (у кінці визначення стовпця) може бути задано декілька обмежень до цього стовпця, розділених пробілом.

Узагальнений синтаксис подання обмеження на рівні таблиці у команді створення таблиці:

*<стовпець>* <*тип_данных*> [DEFAULT <*вираз*>], ...

[[CONSTRAINT <*ім'я_обмеження*>] <*тип_обмеження*> (<*стовпець*>)], …

На рівні таблиці (у кінці визначення таблиці) може бути задано декілька обмежень, розділених комою, як до різних стовпців, так і до одного стовпця.

## Тема 4.2. Мовні засоби маніпулювання даними у реляційній базі даних

### *4.2.1. Оператори маніпулювання даними*

Язык DDL содержит операторы, предназначенные для выполнения действий по модификации данных, их выборке и управлению транзакциями.

Модификация данных может выполняться с помощью операторов DELETE (удалить), INSERT (вставить) и UPDATE (обновить). Область данных, над которыми производится манипулирование, а также организация запросов на выборку данных определяется с помощью оператора SELECT, а управление транзакциями выполняется операторами COMMIT (фиксация), ROLLBACK (откат).

В стандарте языка SQL определены ***две группы операторов*** манипулирования данными (в зависимости от способа их применения).

*Операторы первой группы* связаны с понятием курсора и используются только в режиме встраивания в традиционные языки программирования. Операторы этой группы работают с некоторым курсором, объявление которого должно содержаться в том же модуле или программе со встроенным SQL.

*Операторы второй группы* предназначены для индивидуального использования и могут применяться как в режиме встраивания, так и в интерактивном режиме. Каждый из операторов этой группы является абсолютно независимым от какого бы то ни было другого оператора.

В стандарте SQL определены два ***способа взаимодействия с БД*** из прикладной программы, написанной на традиционном языке программирования.

*Первый способ* состоит в том, что все операторы SQL, с которыми может работать данная прикладная программа, собраны в один модуль и оформлены как процедуры этого модуля. Для этого SQL содержит специальный подъязык - язык модулей. При использовании такого способа взаимодействия с БД прикладная программа содержит вызовы процедур модуля SQL с передачей им фактических параметров и получением ответных параметров.

*Второй способ* состоит в использовании так называемого встроенного SQL, когда с использованием специального синтаксиса в программу на традиционном языке программирования встраиваются операторы SQL. В этом случае с точки зрения прикладной программы оператор SQL выполняется "по месту". Явная параметризация операторов SQL отсутствует, но во встроенных операторах SQL могут использоваться имена переменных основной программы, и за счет этого обеспечивается связь между прикладной программой и СУБД.

*Структура модуля SQL* определяется следующими синтаксическими правилами:

<module>::= MODULE [<module_name>] LANGUAGE <language_identifier> AUTHORIZATION <authorization_identifier> [<declare_cursor>...] <procedure> ...

*Процедуры в модуле SQL* определяются в следующем синтаксисе:

<procedure>::= PROCEDURE <procedure_name> (<parameter_declaration>…);
<SQL_statement>;...
<parameter_declaration>::= <parameter_name> <data_type> |SQLCODE
<SQL_statement>::= <OPEN_cursor> | <FETCH_cursor> | <CLOSEcursor>
| <INSERT_statement>
| <UPDATE_statement_positioned> | <UPDATE_statement_searched>
| <DELETE_statement_ positioned > | <DELETE_statement_searched>
| <SELECT_statement>
| <COMMIT_statement> | <ROLLBACK_statement>

statement_positioned - область данных оператора определяется и ограничена курсором;

statement_searched - область данных оператора определяется самим оператором.

Поскольку в стандарте SQL специфицированы правила встраивания операторов под конкретные языки, то сформулируем только общие синтаксические правила встраивания, используемые для любого языка (***встроенный*** SQL):

<embedded_SQL_statement>::= EXEC SQL

{<declare_cursor> | <embedded_exception_declaration> | <SQL_statement>} [END EXEC ;]

<embedded_SQL_declare_section>::= EXEC SQL BEGIN DECLARE SECTION [END EXEC ;]

[<host_variable_definition>…]

EXEC SQL END DECLARE SECTION [END EXEC ;]

<embedded_variable_name>::= :<host_identifier>

<embedded_exception_declaration>::= WHENEVER <condition> <exception_action>

<condition>::= SQLERROR | NOT FOUND

<exception_action>::= CONTINUE | GOTO <target>

<target>::= :<host_identifier> <unsigned_integer>

***Курсор*** - это понятие языка SQL, позволяющее с помощью набора специальных операторов получить построчный доступ к результату запроса к БД.

*Оператор объявления курсора*:

<declare_cursor>::= DECLARE <cursor_name> CURSOR FOR <cursor_specification>

<cursor_specification>::= <query_expression> [<order_by_clause>]

<query_expression>::= <query_term> |

<query_expression> { UNION|INTERSECT|MINUS} [ALL] <query_term>

<query_term>::= <select_query>|(<query_expression>)

<select_query>::= (SELECT [ALL|DISTINCT] <select_list> <table_expression>

<table_expression>::= <from_clause> [<where_clause>] [<group_by_clause>] [<having_clause>]

<order_by_clause>::= ORDER BY <sort_specification> [{,<sort_specification>}...]

<sort_specification>::= {<unsigned_integer> | <column_specification>} [ASC | DESC]

*Оператор открытия курсора*:

<OPEN_cursor>::= OPEN <cursor_name>

*Оператор чтения очередной строки данных курсора*:

<FETCH_cursor>::= FETCH <cursor_name> INTO <variable_name> [{,<variable_name>}...]

*Оператор закрытия курсора*:

<CLOSE_cursor>::= CLOSE <cursor_name>

*SQL-операторы позиционной модификации данных* (1 группа):

- удаление текущей строки данных курсора

<DELETE_statement_positioned>::= DELETE FROM <table_name>

WHERE CURRENT OF <cursor_name>

- изменение данных текущей строки курсора

<UPDATE_statement_positioned>::= UPDATE <table_name> SET <set_clause> [{,<set_clause>}... ] WHERE CURRENT OF <cursor_name>

<set_clause>::= <column_name>={<value_expression> | NULL | DEFAULT}

*Одиночные SQL-операторы* (2 группа):

- поисковое удаление

<DELETE_statement_searched>::= DELETE FROM <table_name> [WHERE <search_condition>]

- поисковое изменение

<UPDATE_statement_searched>::= UPDATE <table_name> SET <set_clause> [{,<set_clause>}... ] [WHERE <search_condition>]

- вставка данных

<INSERT_statement>::= INSERT INTO <table_name> [(<column_name> [{,column_name}…])] {VALUES(<insert_value> [{,insert_value}…]) | <select_query>}

<insert_value>::= {<value> | NULL | DEFAULT}

- управления транзакцией:

<COMMIT_statement>::= COMMIT

<ROLLBACK_statement>::= ROLLBACK [<label_name>]

<SAVE_statement>::=SAVE TO <label_name>

*4.2.2. Форми оператора організації запитів даних*

*Оператор одиночной выборки*:

<SELECT_statement>::= SELECT [ALL DISTINCT] <select_list> INTO <target_list> <table_expression>

<target_list>::= <variable_name> [{,<variable_name>}...]

*Запрос выборки, определяемый в курсоре*:

<query_expression>::= {<query_term> | <query_expression> {UNION |INTERSECT | MINUS} [ALL] <query_term>} [<order_by_clause>]

<query_term>::= <SELECT_query>|(<query_expression>)

<order_by_clause>::= ORDER BY <sort_specification> [{,<sort_specification>}...]

<sort_specification>::= {<unsigned_integer> | <column_specification>} [ASC | DESC]

*Разделы (ключевые фразы) запроса выборки*:

- раздел параметров выборки

<SELECT_query>::= SELECT [ALL|DISTINCT] <select_list> <table_expression>

<select_list>::= * | <sublist> [{,<sublist>}...]

<sublist>::= <table_name>.* | <correlation_name>.* | <SELECT_query> | <value_expression> [ [AS] <column_name>]

<value_expression>::= [<table_name>.] <column_name> | [<correlation_name>.] <column_name> | {COUNT | MAX | MIN | SUM | AVG} ([ALL | DISTINCT] <value_expression>)

<table_expression>::= <from_clause> [<where_clause>] [<group_by_clause>] [<having_clause>]

- раздел табличного выражения - источника данных выборки

<from_clause>::= FROM <table_reference> [ {,<table_reference>}...]

<table_reference>::= <table_name> [[AS] <correlation_name> [(<column_list>)]] | <derived_table> [AS] <correlation_name> [(<column_list>)] | <joined_table>

<column_list>::= <column_name>[{,<column_name>}...]

<derived_table>::= <SELECT_subquery>

<joined_table>::= <cross_join> | qualified_join> | (<joined table>)

<cross_join>::= <table_reference> CROSS JOIN <table_reference>

<qualified_join>::= <table_reference> [NATURAL [<join_type>] JOIN <table reference> [<join_specification>]

<join_type>::= INNER | {LEFT | RIGHT | FULL} [OUTER] | UNION

<join_specification>::= ON <search_condition> | USING (<column_list>)

- раздел условия отбора данных

<where_clause>::= WHERE <search_condition>

<search_condition>::= <boolean_term> | <search_condition> OR <boolean_term>

<boolean_term>::= <boolean_factor> | <boolean_term> AND <boolean_factor>

<boolean_factor>::= [NOT] <boolean_primary> [IS [NOT] <truth_value>]

<boolean_primary>::= <predicate> | (<search_condition>)
<truth_value> ::= TRUE | FALSE | UNKNOWN
<предикат_сравнения>::= <value_expression> {= | <> | < | > | <= | >=} {<value_expression> | [{ALL | SOME ANY}] <SELECT_subquery>}
<предикат_интервала>::= <value_expression> [NOT] BETWEEN <value_expression> AND <value_expression>
<предикат_списка>::= <value_expression> [NOT] IN {(value [{,value}...]) | <SELECT_subquery>}
<предикат_совпадения>::= <value_expression> [NOT] LIKE <character_string>
<предикат_неопределенного_значения>::= <value_expression> IS [NOT] NULL
<предикат_существования>::= EXISTS <SELECT_subquery>
<предикат_уникальности>::= UNIQUE <SELECT_subquery>
- раздел группирования данных
<group_by_clause>::= GROUP BY <column_name> [{,<column _name >}...]
- раздел условия отбора групп данных
<having_clause>::= HAVING <search_condition>
Подзапрос выборки:
<SELECT_subquery>::= (<SELECT_query>)
Примеры запросов выборки данных из таблиц
region(id,name), department(id,name,region_id):
SELECT * FROM region
SELECT region.* FROM region
SELECT id,name FROM region WHERE name like ‘K%’ ORDER BY name DESC
SELECT DISTINCT name FROM region
SELECT count(id),region_id FROM department GROUP BY region_id HAVING count(id)>10
SELECT count(id),region_id FROM department WHERE name like ‘K%’ GROUP BY region_id HAVING count(id)>10
SELECT r.id AS kod,r.name AS region,d.count_dept AS department
FROM (SELECT count(id) as count_dept,region_id FROM department GROUP BY region_id) d, region r WHERE r.id=d.region_id

### *4.2.3. Оператори керування транзакціями*

Сервер бази даних забезпечує погодженість даних на основі механізму транзакцій. Транзакції забезпечують велику гнучкість, більш широкий спектр засобів управління при зміні даних, а також їхню погодженість у випадку помилок при виконанні команд або збої системи.

Транзакції складаються з команд DML, що вносять у дані одну погоджену зміну. Наприклад, переказ коштів між двома рахунками обов'язково включає дебетування одного рахунку і кредитування іншого на ту ж саму суму. Успішне або невдале виконання повинно бути одночасним для обох дій у рахунках. Тобто, залишати в базі дані про кредитування без даних про дебетування не припустимо.

Тип транзакції визначається належністю команди, що її утворює, до типу мови структурованих запитів. Типи транзакцій наведені у таблиці:

| Тип | Опис |
|---|---|
| Маніпулювання даними (DML) | Складається з довільної кількості команд DML, сприйнятих сервером бази даних як одна логічна одиниця роботи |
| Визначення даних (DDL) | Складається з однієї команди DDL |
| Керування даними (DCL) | Складається з однієї команди DCL |

Транзакція починається, коли сервер бази даних одержує першу команду, що змінює стан бази, і закінчується, коли відбувається щось одне з наступних подій:

1) Виконання команди управління транзакцією COMMIT або ROLLBACK;

2) Виконання команди DDL (наприклад, CREATE) або команда DCL;

3) Виявлення певних помилок, наприклад, взаємного блокування;

4) Завершення користувачем сеансу роботи з базою даних;

5) Програмно-апаратний збій або аварійне зупинення системи.

Після завершення однієї транзакції автоматично починається наступна після успішного виконання команди DML, DDL або DCL. Результати виконання команд DDL і DCL фіксуються автоматично. Отже, ці команди неявно завершують транзакцію.

4.2.3.1.Команди керування транзакціями. Для явного управління логікою транзакцій використовуються команди COMMIT, SAVEPOINT і ROLLBACK.

| Команда | Опис |
|---|---|
| COMMIT | Завершує поточну транзакцію, роблячи постійними всі зроблені зміни даних |
| SAVEPOINT *ім'я* | Встановлює у поточній транзакції маркер збереження (savepoint) – точку відкоту |
| ROLLBACK [TO SAVEPOINT *ім'я*] | Припиняє поточну транзакцію, скасовуючи всі зроблені зміни в даних до встановленого маркеру |

Неявна обробка транзакций виконується у випадках, наведених у таблиці:

| Статус | Причина |
|---|---|
| Автоматична фіксація | Команда DDL чи DCL |
| | Нормальне завершення сеансу роботи в базі даних без явного виконання команди COMMIT чи ROLLBACK |
| Автоматичний відкат | Аварійне припинення сеансу роботи або системний збій у базі даних |

4.2.3.2.Фіксація змін. Кожна зміна даних, що виконана в ході транзакції, є тимчасовою, доти транзакція не буде зафіксована.

До виконання фіксації транзакції дані будуть знаходиться в наступному стані:

1) команди по обробці (маніпулюванню) даних змінюють тільки буфер бази даних. Отже, попередній стан даних може бути відновлено;

2) поточний користувач може переглянути результати своїх команд по обробці даних, виконуючи запити до таблиць бази;

3) інші користувачі не можуть бачити результати команд обробки даних, виконуваних поточним користувачем. СУБД забезпечує погодженість читання, і кожен користувач бачить дані такими, якими вони були на момент виконання останньої команди COMMIT;

4) змінені рядки таблиць блокуються і інші користувачі не мають можливості їх змінювати.

Усі внесені зміни фіксуються за допомогою команди COMMIT, у результаті чого дані переходять у наступний стан:

1) змінені дані записуються в базу даних. Попередній стан даних втрачається;

2) усі користувачі можуть бачити результати виконаних команд;

3) блокування знімається зі змінених рядків, інші користувачі одержують можливість вносити в них чергові зміни;

4) усі маркери збереження видаляються.

Приклад створення нового відділу, до якого переводится службовець з номером 2. Усі зміни даних по завершенні переведення фіксуються:

```
INSERT INTO s_dept (id, name, region_id) VALUES (54, 'Education', 1) ;
```

```
UPDATE s_emp SET dept_id=54 WHERE id=2 ;
COMMIT ;
```

4.2.3.3.Скасування змін. Усі незафіксовані зміни скасовуються командою ROLLBACK, у результаті чого дані переходять у наступний стан:

1) зроблені зміни втрачаються. Відновлюється попередній стан даних;

2) розблокуються рядки, з якими виконувалися операції, і інші користувачі одержують можливість вносити в них зміни.

Приклад скасування змін при видаленні записів. Під час видалення запису з таблиці test помилково стерті усі дані з цієї таблиці, які потрібно відновити:

```
DELETE FROM test ;
ROLLBACK ;
```

4.2.3.4.Скасування змін до маркеру збереження. Команда SAVEPOINT дозволяє створювати в поточній транзакції точки відкоту, що розбивають її на одиниці меншого розміру. Після цього за допомогою команди ROLLBACK TO можна скасовувати незафіксовані зміни до зазначеного маркеру. Створювати можна кілька маркерів збереження в одній транзакції. Повторне ж створення маркеру збереження з тим же ім'ям, викликає видалення створення попереднього такого маркеру.

Приклад скасування змін до точки відкоту. Створюється маркер збереження update_done між двома командами для можливого повернення до стану виконання першої команди:

```
UPDATE s_emp SET comission=comission*1.1 WHERE dept_id=54 ;
SAVEPOINT update_done ;
INSERT INTO s_region (id, name) VALUES (8, 'Central')
ROLLBACK TO update_done ;
```

4.2.3.5.Відкот на рівні команди. Якщо при виконанні команди виявлена помилка, то частина транзакції може бути скасована шляхом неявного відкоту. Якщо в ході транзакції помилка виникла в одній команді DML, то скасовується результат тільки цієї команди, тобто здійснюється відкот на рівні команди. Однак зміни, внесені під час цієї транзакції попередніми командами DML, зберігаються. Вони можуть бути зафіксовані або скасовані у явній формі.

СУБД виконує неявну команду COMMIT до і після виконання будь-якої команди мови визначення даних DDL. Тому, навіть, якщо команда DDL виконана невдало, скасувати зміни, зроблені попередньою командою, неможливо, тому що сервер вже їх зафіксував.

Завершувати явно транзакції слід командою COMMIT або ROLLBACK.

## Тема 4.3. Мовні засоби управління даними у реляційній базі даних

### *4.3.1. Підхід до організації безпеки у РБД*

Существенной особенностью языка SQL, появившейся в нем с самого начала, является обеспечение защиты доступа к данным средствами самого языка.

В соответствии с идеологией языка SQL контроль прав доступа данного пользователя к таблицам БД производится на основе механизма привилегий. Фактически, этот механизм состоит в том, что для выполнения любого действия над таблицей пользователь должен обладать соответствующей привилегией.

Основная идея такого подхода состоит в том, что по отношению к любой таблице БД и любому столбцу таблицы вводится предопределенный набор привилегий, а с каждой транзакцией неявно связывается идентификатор пользователя, от имени которого она выполняется.

После создания новой таблицы все привилегии, связанные с этой таблицей и всеми ее столбцами, принадлежат только пользователю-создателю таблицы. В число привилегий входит привилегия передачи всех или части привилегий другому пользователю, включая привилегию на передачу привилегий. Существует также привилегия изъятия всех или части привилегий у пользователя, которому они ранее были переданы. Эта привилегия также может передаваться.

В многопользовательской среде система безопасности СУБД должна обеспечить:

- идентификацию пользователей БД;
- управление доступом пользователей к БД;
- предоставление доступа к определенным объектам БД;
- регистрацию выданных и полученных привилегии в словаре БД.

Безопасность БД можно разделить на две части:

- безопасность системы;
- безопасность данных.

*Безопасность системы* охватывает доступ к БД и пользование базой на системном уровне, т.е. имя и пароль пользователя, дисковое пространство, выделяемое пользователю, и системные операции, разрешенные пользователю.

*Безопасность данных* включает доступ к объектам базы данных, их использование и действия, которые может выполнять пользователь с этими объектами.

Все привилегии пользователей БД подразделены на следующие уровни:

- привилегии системного уровня;
- привилегии объектного уровня.

*Привилегии системного уровня* управляют правами выполнения команд SQL и других системных операций, и назначаются администратором базы данных другим пользователям БД.

Например, прежде чем создать таблицу необходимо иметь системную привилегию с правом создания таблицы.

*Привилегии объектного уровня* обеспечивают возможность выполнения определенных типов действия с указанным объектом. Владелец объекта имеет полный контроль над объектом и может выполнять любые действия с ним; он не обязан иметь привилегии объектного уровня, а должен иметь привилегию системного уровня с правом создания этого объекта.

Фактически владелец объекта-пользователь БД, который может предоставлять привилегии объектного уровня другим пользователям.

### *4.3.2. Оператори системного рівня надання привілеїв*

*Оператор подключение к БД*.

С выполнения этого оператора начинается сеанс взаимодействия с указанным сервером БД:

<connect_statement>::= CONNECT <connection_target>

<connection_target>::= <SQL-server name> [AS <connection_name>] [USER <user_name>] | DEFAULT

Имя соединения либо задается явно, либо выбирается по умолчанию.

Если имя соединения задается по умолчанию, то образуется SQL-сессия с именем по умолчанию, связанная с умалчиваемым SQL-сервером. Способ распознавания сервера по умолчанию определяется в реализации.

Если оператор установления соединения успешно инициирует SQL-сессию, то:

- текущее соединение и текущая транзакция становятся потенциальными; информация о контексте SQL-сервера сохраняется и не влияет на операции, выполняемые в инициированном соединении;

- образованные SQL-сессия и SQL-соединение становятся текущими;

- идентификатор авторизации SQL-сессии устанавливается в соответствии с указанным именем пользователя.

*Оператор выбора подключения*:

<set_connection_statement>::= SET CONNECTION <connection_object>

<connection_object>::= DEFAULT | <connection_name>

Текущие подключение и сессия становятся потенциальными; информация о контексте сервера сохраняется и не влияет на выполнение операций через выбранное подключение.

Выбранное подключение и соответствующая SQL-сессия становятся текущими; информация о контексте сессии восстанавливается к тому состоянию, в котором она находилась к моменту перехода в потенциальную.

*Оператор отказа от подключения*.

Служит для разрыва сеанса подключения к SQL-серверу и имеет следующий синтаксис:

<disconnect_statement>::= DISCONNECT <disconnect_object>

<disconnect_object>::= <connection_object> | ALL | CURRENT

Администратор базы данных *предоставляет системные привилегии* пользователям с помощью оператора GRANT. Пользователь может пользоваться предоставленными привилегиями сразу после их получения.

<system_privilege_definition>::= GRANT <privilege> [{,<privilege>}...] TO <grantee> [{,<grantee>}...] [WITH ADMIN OPTION]

<privilege>::= CREATE <object_name> | ALTER <object_name> | DROP <object_name> | <system_action>

<grantee>::= <authorization_identifier>

<authorization_identifier>::= <user_name> | <role_name>

Объектами системных привилегий выступают объекты БД (таблицы, представления, индексы, программные модули и т.д.), а также режимы организации работы этих объектов (сессии, схемы пользователей, дисковые пространства и т.д.).

Субъектом предоставления привилегий выступает пользователь (схема пользователя) или роль.

<user_definition>::= CREATE USER <user_name> IDENTIFIED BY <password> [<table_space_definition>]

Роль - это именованная группа взаимосвязанных привилегии, которая могут быть предоставлена пользователю. Пользователь может иметь доступ к нескольким ролям, а одна и та же роль может быть предоставлена многим пользователям.

<role_definition>::= CREATE ROLE <role_name>

*Основные привилегии системного уровня*:

- CREATE (ALTER) SESSION
- CREATE (ALTER, DROP) USER
- CREATE (ALTER, DROP) ROLE
- CREATE (ALTER, DROP )TABLE
- CREATE (ALTER, DROP) VIEW
- CREATE (ALTER, DROP) PROCEDURE

*Отмена системных привилегий* производится с помощью оператора REVOKE. Команда REVOKE отменяет указанные привилегии для всех указанных пользователей или ролей, которым, были предоставлены эти привилегии:

<system_privilege_refuse>::= REVOKE <privilege> [{,<privilege>}...] FROM <grantee> [{,<grantee>}...]

<privilege>::= <system_action> | ALL PRIVILEGES

<grantee>::= <authorization_identifier> | PUBLIC

<privilege> – системная привилегия или предоставленная роль,

ALL PRIVILEGES – все возможные системные привилегии,

PUBLIC – удаляет указанные привилегий или роли у всех пользователей.

*4.3.3. Оператори об'єктного рівня надання привілеїв*

Администратор базы данных или пользователь-владелец объекта БД *предоставляет объектные привилегии* другим пользователям с помощью оператора GRANT:

<object_privilege_definition>::= GRANT <privileges> ON <object_name> TO <grantee> [{,<grantee>}...] [WITH GRANT OPTION]

<privileges>::= {ALL PRIVILEGES | <action> [{,<action>}...]}

<action>::= SELECT | INSERT | DELETE | UPDATE [(<column_list>)] | REFERENCES [(<column_list>]

<column_list>::= <column_name> [{,<column_name>}...]

<grantee>::= PUBLIC | <authorization_identifier>

*Отмена объектных привилегий* производится с помощью оператора REVOKE. Команда REVOKE отменяет указанные привилегии для всех указанных пользователей или ролей, которым, были предоставлены эти привилегии:

<object_privilege_refuse>::= REVOKE <privilege> [{,<privilege>}...] ON <object_name> FROM <grantee> [{,<grantee>}...] [CASCADE CONSTRAINTS]

<privilege>::= <system_action> | ALL PRIVILEGES

<grantee>::= <authorization_identifier> | PUBLIC

CASCADE – используется для отмены ограничений, наложенных на объект с помощью привилегии REFERENCES для сохранения ссылочной целостности.

У таблиці наведені можливі види привілеїв на об'єкти бази даних.

| Об'єктні привілеї | Таблиця | Представлення | Послідовність | Процедура/ Функція |
|---|---|---|---|---|
| ALTER | + | | + | |
| DELETE | + | + | | |
| EXECUTE | | | | + |
| INDEX | + | | | |
| INSERT | + | + | | |
| REFERENCES | + | | | |
| SELECT | + | + | + | |
| UPDATE | + | + | | |

Примеры предоставления системных и объектных привилегий:

CREATE ROLE manager ;

GRANT create table, create view TO manager ;

GRANT manager TO user1, user2;

```
CREATE ROLE manager1 ;
GRANT manager TO manager1;
GRANT select  ON s_emp TO scott, managers ;
GRANT update (name, region_id) ON s_dept TO scott, managers ;
GRANT select, insert ON s_dept TO scott WITH GRANT OPTION ;
GRANT select ON slice.s_dept TO PUBLIC ;
REVOKE select, insert ON s_dept FROM scott ;
```

# Розділ 5. Внутрішня структура реляційної бази даних на ЕОМ

## Тема 5.1. Організація зовнішньої пам‘яті елементів реляційної бази даних

### *5.1.1. Підходи організації зовнішньої пам'яті РБД*

Реляционные БД обладают следующими *важными особенностями*, влияющими на организацию внешней памяти:

1.Реализация двух уровней в системе организации внешней памяти.

Уровень языкового управления на базе языка SQL. Уровень непосредственного управления данными во внешней памяти (управление буферами оперативной памяти, управление транзакциями, журнализация изменений БД).

Функции подсистемы нижнего уровня - поддержка во внешней памяти набор базовых структур.

Функции подсистемы верхнего уровня - конкретная интерпретация набора базовых структур.

2.Поддержка отношений. Информация, связанная с именованием объектов базы данных и их конкретными свойствами, поддерживается подсистемой языкового уровня.

3.Регулярность структур данных. Т.к. основным объектом реляционной модели данных является отношение, то набор объектов внешней памяти может иметь простую регулярную структуру.

4.Возможность эффективного выполнения операторов языкового уровня над отношениями с помощью поддержки дополнительных управляющих структур - индексов.

5.Для обеспечения надежного хранения баз данных реализована поддержка избыточности данных через реализацию журнала изменений базы данных.

В теории СУБД рассматриваются два альтернативных подхода к их организации с точки разделения функций между различными компонентами - СУБД System R и Ingres.

В СУБД System R существовала интегрированная подсистема управления данными, транзакциями и журнализацией, в то время как в Ingres управление данными, было отделено от управления транзакциями и журнализацией.

Подход System R позволяет использовать более эффективные методы за счет совместного решения проблем физической и логической синхронизации, использовании общих протоколов при управлении буферами и журнализации и т.д. Но при этом подсистема нижнего уровня становится монолитом и при самой удачной ее структуризации компоненты остаются связанными общими протоколами взаимодействия. Непродуманные локальные изменения одного компонента могут привести к фатальным последствиям для всей системы.

Подход Ingres позволяет упростить структуру системы и сделать ее более гибкой, но это возможно только за счет упрощения алгоритмов: применения более простых методов управления транзакциями; жестких протоколов журнализации и т.д.

### *5.1.2. Різновиду об'єктів зовнішньої пам'яті РБД*

*Разновидности объектов* во внешней памяти базы данных:

1.Строки отношений - основная часть базы данных;

2.Управляющие структуры - индексы, повышающие эффективность выполнения запросов;

3.Журнальная информация, поддерживающая надежность хранения данных;

4.Служебная информация, необходимая для удовлетворения внутренних потребностей подсистемы нижнего уровня (информация о свободной и занятой памяти, статистика использования данных и т.п.).

Существуют два принципиальных подхода к физическому хранению отношений:

- кортежное хранение отношений с единицей физического хранения - кортеж;
- столбцовое хранение отношений с единицей физического хранения - столбец.

*Кортежное хранение* обеспечивает быстрый доступ к целому кортежу с дублированием во внешней памяти общих значений разных кортежей одного отношения. При считывании части кортежа может потребоваться лишние обмены с внешней памятью.

Наиболее распространенный подход хранения отношения по строкам реализуется с помощью страничного механизма накопления данных.


К *основным характеристикам* этой организации можно отнести следующие:

1.Каждый кортеж обладает уникальным идентификатором, который не изменяется все время существования кортежа.

2.Каждый кортеж хранится целиком в одной странице. Максимальная длина кортежа любого отношения ограничена размерами страницы.

Хранение "длинных" данных выполняется в отдельных файлах или наборе страниц внешней памяти с заменой его в кортеже на имя соответствующего файла или на страничную ссылку. В настоящее время все чаще используется метод, когда "длинные" данные организуются в виде B-деревьев последовательностей байтов.

3.В одной странице данных хранятся кортежи только одного отношения. Существуют, однако, варианты с возможностью хранения в одной странице кортежей нескольких отношений. Это вызывает некоторые дополнительные расходы по части служебной информации (при каждом кортеже нужно хранить информацию о соответствующем отношении), но резко сокращается число обменов с внешней памятью при выполнении соединений.

4.Изменение схемы хранимого отношения с добавлением нового столбца не вызывает потребности в физической реорганизации отношения. Достаточно лишь изменить информацию в описателе отношения и расширять кортежи только при занесении информации в новый столбец.

5.Проблема распределения памяти в страницах данных связана с проблемами синхронизации и журнализации и не всегда тривиальна. Например, если в ходе выполнения транзакции некоторая страница данных опустошается, то ее нельзя перевести в статус свободных страниц до конца транзакции, поскольку при откате транзакции удаленные при прямом выполнении транзакции и восстановленные при ее откате кортежи должны получить те же самые идентификаторы.

6.Распространенным способом повышения эффективности СУБД является кластеризация отношения по значениям одного или нескольких столбцов. Полезным для оптимизации соединений является совместная кластеризация нескольких отношений.

7.С целью использования возможностей распараллеливания обменов с внешней памятью иногда применяют схему декластеризованного хранения отношений: кортежи с общим значением столбца декластеризации размещают на разных дисковых устройствах, обмены с которыми можно выполнять в параллель.

*Столбцовое хранение* обеспечивает снижение затрат внешней памяти из-за отсутствия дубликатов значений столбца и считывания больше необходимой информации за одно обращение к внешней памяти. Эффективно такое хранение и для оптимизации операций соединения, но при сборке всего кортежа или его части потребуются дополнительные действия.

Основная идея состоит в совместном хранении всех значений одного (или нескольких) столбцов. Для каждого кортежа отношения его хранение организовано аналогичным образом с тем отличием, что его значения состоят из ссылок на места расположения соответствующих значений столбцов.

Наибольшее распространение получили в распределенных СУБД с так называемым вертикальным разделением отношения, когда в разных узлах сети хранятся разные проекции данного отношения.


*5.1.3. Методи організації індексів*

Их основное назначение состоит в обеспечении эффективного прямого доступа к кортежу отношения по ключу. Обычно индекс определяется для одного отношения, и ключом является значение атрибута.

Общей идеей любой организации индекса, поддерживающего прямой доступ по ключу и последовательный просмотр в порядке возрастания или убывания значений ключа является хранение упорядоченного списка значений ключа с привязкой к каждому значению ключа списка идентификаторов кортежей.

Наиболее популярным подходом к организации индексов в базах данных является использование техники B-деревьев.

С точки зрения внешнего логического представления *B-дерево* - это сбалансированное сильно ветвистое дерево во внешней памяти.

*Сбалансированность* означает, что длина пути от корня дерева к любому его листу одна и та же.

*Ветвистость дерева* - это свойство каждого узла дерева ссылаться но большое число узлов-потомков.

С точки зрения физической организации B-дерево представляется как мультисписочная структура страниц внешней памяти, т.е. каждому узлу дерева соответствует блок внешней памяти (страница). Внутренние и листовые страницы обычно имеют разную структуру.

Структура внутренней страницы имеет следующий вид:

<N1 ключ(1)> <N2 ключ(2)> … <Nn ключ(n)>

При этом выдерживаются следующие свойства:

- ключ(1) <= ключ(2) <= ... <= ключ(n);

- в странице дерева Nm находятся ключи k со значениями
  ключ(m) <= k <= ключ(m+1).

Листовая страница обычно имеет следующую структуру:

| <ключ(1) сп(1)> <ключ(2) сп(2)> … <ключ(t) сп(t)> |
|---|

Листовая страница обладает следующими свойствами:

- ключ(1) < ключ(2) < ... < ключ(t);
- сп(r) -упорядоченный список идентификаторов кортежей, включающих значение ключ(r);
- листовые страницы связаны одно- или двунаправленным списком.

Поиск в B-дереве - это прохождение от корня к листу в соответствии с заданным значением ключа. Поскольку деревья сильно ветвистые и сбалансированные, то для выполнения поиска по любому значению ключа потребуется одно и то же (и обычно небольшое) число обменов с внешней памятью.

Более точно, в сбалансированном дереве, где длины всех путей от корня к листу одни и те же, если во внутренней странице помещается n ключей, то при хранении m записей требуется дерево глубиной $\log_n(m)$. Если n достаточно велико (обычный случай), то глубина дерева невелика, и производится быстрый поиск.

Основным достоинством B-деревьев является автоматическое поддержание свойства сбалансированности при выполнении операций занесения и удаления записей.

Альтернативным и все более популярным подходом к организации индексов является использование метода *хэширования*.

Общей идеей методов хэширования является применение к значению ключа некоторой функции свертки (хэш-функции), вырабатывающей значение меньшего размера. Свертка значения ключа затем используется для доступа к записи.

В классическом случае свертка ключа используется как адрес в таблице, содержащей ключи и записи. Основным требованием к хэш-функции является равномерное распределение значение свертки. При возникновении коллизий (одна и та же свертка для нескольких значений ключа) образуются цепочки переполнения.

Главным ограничением этого метода является фиксированный размер таблицы. Если таблица заполнена слишком сильно или переполнена, но возникнет слишком много цепочек переполнения, и главное преимущество хэширования - доступ к записи почти всегда за одно обращение к таблице - будет утрачено. Расширение таблицы требует ее полной переделки на основе новой хэш-функции (со значением свертки большего размера).

Поэтому обычно вводят промежуточные таблицы-справочники, содержащие значения ключей и адреса записей, а сами записи хранятся отдельно. Тогда при переполнении справочника требуется только его переделка, что вызывает меньше накладных расходов.

## Тема 5.2. Організація управління транзакціями і журналізації змін станів у реляційній базі даних

### *5.2.1. Рівні ізоляції виконання транзакцій*

Поддержание механизма транзакций - показатель уровня развитости СУБД.

Корректное поддержание транзакций одновременно является основой обеспечения целостности баз данных, а также составляют базис изолированности пользователей во многопользовательских системах.

Понятие транзакции непосредственно связано с понятием целостности данных.

Различают *два вида ограничений*, обеспечиваемых механизмом транзакций: немедленно проверяемые и откладываемые. Первые обеспечивают целостность при каждом выполнении операции с базой данных, а вторые - при завершении транзакции.

У каждой транзакции при выполнении имеется *режим доступа*:

- только чтение;
- чтение и запись.

В многопользовательских БД механизм транзакции обеспечивает определенный уровень изоляции пользователей, т.е. создание достоверной и надежной иллюзии того, что каждый из пользователей работает с БД как с однопользовательской.

Уровень изоляции транзакции определяет степень, с которой на операции этой транзакции влияют операции параллельно выполняющихся транзакций, и операции данной транзакции влияют на операции других транзакций.

Уровень изоляции определяет вид явлений-ситуаций, которые могут произойти при параллельном выполнении транзакций. Возможны следующие виды явлений:

1.*"Потерянные изменения"*: Транзакция T1 изменяет объект базы данных A. До завершения транзакции T1 транзакция T2 также изменяет объект A. Транзакция T2 завершается оператором ROLLBACK по причине нарушения ограничений целостности. Тогда при повторном чтении объекта A транзакция T1 не видит изменений этого объекта, произведенных ранее. Чтобы избежать такой ситуации в транзакции T1 требуется, чтобы до завершения транзакции T1 никакая другая транзакция не могла изменять объект A. Отсутствие потерянных изменений является минимальным требованием к СУБД по части синхронизации параллельно выполняемых транзакций.

2.*"Грязное чтение"*: Транзакция T1 изменяет объект базы данных A. Затем транзакция T2 читает объект A. Если после этого T1 выполнит ROLLBACK, то окажется, что T2 прочитала строку, которая никогда не существовала. Чтобы избежать ситуации чтения "грязных" данных, до завершения транзакции T1, изменившей объект A, никакая другая транзакция не должна читать объект A.

3.*"Неповторяющееся чтение"*: Транзакция T1 читает объект базы данных A. До завершения транзакции T1 транзакция T2 изменяет или удаляет объект A и выполняет COMMIT. Если после этого T1 попытается повторно прочитать объект A, то либо получит его измененное состояние, либо обнаружит, что объект удален. Чтобы избежать неповторяющихся чтений, до завершения транзакции T1 никакая другая транзакция не должна изменять объект A. В большинстве систем это является максимальным требованием к синхронизации транзакций, хотя еще не гарантирована реальная изолированность пользователей.

4.*"Фантом"*: Транзакция T1 читает набор кортежей N, которые удовлетворяют условию поиска S. Затем транзакция T2 выполняет оператор SQL, который добавляет одим или более кортежей, удовлетворяющих условию S, и завершается

COMMIT. Если после этого транзакция T1 повторит чтение с тем же самым условием S, она получит другой набор кортежей. Чтобы избежать появления кортежей-фантомов, требуется наивысший "логический" уровень синхронизации транзакций.

Уровни изоляции в СУБД различаются по возможности недопущения явлений-ситуаций при параллельном (конкурентном) выполнении транзакций:

- READUNCOMMITTED,
- READCOMMITTED,
- REPEATABLEREAD,
- SERIALIZABLE.

Все четыре уровня изоляции гарантируют, что каждая SQL-транзакция либо выполнится полностью, либо не выполнится совсем и что ни одно изменение не будет потеряно.

| *Уровень изоляции* | *C2* | *C3* | *C4* |
|---|---|---|---|
| READUNCOMMITTED | Возможно | Возможно | Возможно |
| READCOMMITTED | Невозможно | Возможно | Возможно |
| REPEATABLEREAD | Возможно | Невозможно | Возможно |
| SERIALIZABLE | Невозможно | Невозможно | Невозможно |

По умолчанию устанавливается уровень изоляции SERIALIZABLE. При выполнении конкурирующих транзакций на этом уровне изоляции гарантируется их сериализуемость.

*Сериализованное* выполнение - это такое выполнение операций конкурирующих транзакций, которое производит тот же самый окончательный эффект, что и некоторое последовательное выполнение этих транзакций, т.е. такое выполнение, при котором каждая транзакция полностью завершается до начала следующей.

### *5.2.2. Методи організації виконання набору транзакцій*

Основная реализационная проблема состоит в выборе метода сериализации набора транзакций, который не слишком ограничивал бы их параллельность.

Существуют два базовых подхода к сериализации транзакций - основанный на *синхронизационных захватах* объектов базы данных и на использовании *временных меток*. Суть обоих подходов состоит в обнаружении конфликтов транзакций и их устранении.

Между транзакциями могут существовать следующие *виды конфликтов*:

1.W-W - транзакция 2 пытается изменять объект, измененный не закончившейся транзакцией 1;

2.R-W - транзакция 2 пытается изменять объект, прочитанный не закончившейся транзакцией 1;

3.W-R - транзакция 2 пытается читать объект, измененный не закончившейся транзакцией 1.

Наиболее распространенным в централизованных СУБД (включающих системы, основанные на архитектуре "клиент-сервер") является подход, основанный на соблюдении двухфазного протокола *синхронизационных захватов* объектов БД.

Схема протокол состоит в том, что перед выполнением любой операции в транзакции T над объектом базы данных A от имени транзакции T запрашивается синхронизационный захват объекта A в соответствующем режиме (в зависимости от вида операции).

*Основными режимами синхронизационных захватов* являются:

1.совместный режим - S (Shared), означающий разделяемый захват объекта; требуемый для выполнения операции чтения объекта;

2.монопольный режим - X (eXclusive), означающий монопольный захват объекта; требуемый для выполнения операций занесения, удаления и модификации.

Правила совместимости захватов:

| *Состояние объекта* | X | S |
|---|---|---|
| нет захвата | да | да |
| X | нет | нет |
| S | нет | да |

Результат захвата "нет" соответствует описанным ранее возможным случаям конфликтов транзакций по доступу к объектам базы данных (WW, RW, WR). Совместимость S-захватов соответствует тому, что конфликт RR не существует.

Транзакция, запросившая синхронизационный захват объекта БД, уже захваченный другой транзакцией в несовместимом режиме, блокируется до тех пор, пока захват с этого объекта не будет снят.

В контексте реляционных баз данных возможны следующие альтернативы объектов для синхронизационного захвата:

*файл* - физический (с точки зрения базы данных) объект, область хранения нескольких отношений и, возможно, индексов;

*отношение* - логический объект, соответствующий множеству кортежей данного отношения;

*страница данных* - физический объект, хранящий кортежи одного или нескольких отношений, индексную или служебную информацию;

*кортеж* - элементарный физический объект базы данных.

Одним из *наиболее чувствительных недостатков* метода сериализации транзакций на основе синхронизационных захватов является возможность возникновение тупиков (*deadlocks*) между транзакциями.

Основой обнаружения тупиковых ситуаций является построение графа ожидания транзакций. Граф ожидания транзакций - это ориентированный двудольный граф, в котором существует два типа вершин - вершины, соответствующие транзакциям, и вершины, соответствующие объектам захвата. В этом графе существует дуга, ведущая из вершины-транзакции к вершине-объекту, если для этой транзакции существует удовлетворенный захват объекта. В графе существует дуга из вершины-объекта к вершине-транзакции, если транзакция ожидает удовлетворения захвата объекта.

В системе существует ситуация тупика, если в графе ожидания транзакций имеется хотя бы один цикл.

Для распознавание тупика периодически производится построение графа ожидания транзакций, и в этом графе ищутся циклы.

Разрушение тупика начинается с выбора в цикле транзакций так называемой транзакции-жертвы, т.е. транзакции, которой решено пожертвовать, чтобы обеспечить возможность продолжения работы других транзакций.

Критерием выбора является стоимость транзакции. Стоимость транзакции определяется на основе многофакторная оценка, в которую с разными весами входят время выполнения, число накопленных захватов, приоритет.

После выбора транзакции-жертвы выполняется откат этой транзакции, который может носить полный или частичный характер. При этом освобождаются захваты и может быть продолжено выполнение других транзакций.

*Альтернативный метод* сериализации транзакций, хорошо работающий в условиях редких конфликтов транзакций и не требующий построения графа ожидания транзакций. основан на использовании временных меток.

Основная идея метода состоит в следующем: если транзакция T1 началась раньше транзакции T2, то система обеспечивает такой режим выполнения, как если бы T1 была целиком выполнена до начала T2.

Для этого каждой транзакции T предписывается временная метка t, соответствующая времени начала T. При выполнении операции над объектом A

транзакция Т помечает его своей временной меткой и типом операции (чтение или изменение).

Перед выполнением операции над объектом А транзакция Т1 выполняет следующие действия:

1.Проверяет, не закончилась ли транзакция Т, пометившая этот объект. Если Т закончилась, Т1 помечает объект А и выполняет свою операцию.

2.Если транзакция Т не завершилась, то Т1 проверяет конфликтность операций. Если операции неконфликтны, при объекте А остается или проставляется временная метка с меньшим значением, и транзакция Т1 выполняет свою операцию.

3.Если операции Т1 и Т конфликтуют, то если t(T)>t(T1), производится откат Т и Т1 продолжает работу.

4.Если же t(T)<t(T1), то Т1 получает новую временную метку и начинается заново.

К недостаткам метода временных меток относятся потенциально более частые откаты транзакций, чем в случае использования синхронизационных захватов. Это связано с тем, что конфликтность транзакций определяется более грубо. Кроме того, в распределенных системах не очень просто вырабатывать глобальные временные метки с отношением полного порядка.

### *5.2.3. Призначення і принципи використання журналів у РБД*

Журнализация изменений - функция СУБД, которая сохраняет информацию, необходимую для восстановления базы данных в предыдущее согласованное состояние в случае логических или физических отказов.

В простейшем случае журнализация изменений заключается в последовательной записи во внешнюю память всех изменений, выполняемых в базе данных. Записывается следующая информация:

- порядковый номер, тип и время изменения;
- идентификатор транзакции;
- объект, подвергшийся изменению (номер хранимого файла и номер блока данных в нём, номер строки внутри блока);
- предыдущее состояние объекта и новое состояние объекта.

Формируемая таким образом информация называется журнал изменений базы данных. Журнал содержит отметки начала и завершения транзакции, и отметки принятия контрольной точки.

В СУБД с отложенной записью блоки данных внешней памяти снабжаются отметкой порядкового номера последнего изменения, которое было выполнено над этим блоком данных. В случае сбоя системы эта отметка позволяет узнать какая версия блока данных успела достичь внешней памяти.

СУБД с отложенной записью периодически выполняет контрольные точки. Во время выполнения этого процесса все незаписанные данные переносятся на внешнюю память, а в журнал пишется отметка принятия контрольной точки. После этого содержимое журнала, записанное до контрольной точки может быть удалено.

Журнал изменений может не записываться непосредственно во внешнюю память, а аккумулироваться в оперативной. В случае подтверждения транзакции СУБД дожидается записи оставшейся части журнала на внешнюю память. Таким образом гарантируется, что все данные, внесённые после сигнала подтверждения, будут перенесены во внешнюю память, не дожидаясь переписи всех измененных блоков из дискового кэша. СУБД дожидается записи оставшейся части журнала также при выполнении контрольной точки.

В случае логического отказа или сигнала отката одной транзакции журнал сканируется в обратном направлении, и все записи отменяемой транзакции извлекаются из журнала вплоть до отметки начала транзакции. Согласно

извлеченной информации выполняются действия, отменяющие действия транзакции, а в журнал записываются компенсирующие записи. Этот процесс называется откат (rollback).

В случае физического отказа, если ни журнал, ни сама база данных не повреждена, то выполняется процесс прогонки (rollforward). Журнал сканируется в прямом направлении, начиная от предыдущей контрольной точки. Все записи извлекаются из журнала вплоть до конца журнала. Извлеченная из журнала информация вносится в блоки данных внешней памяти, у которых отметка номера изменений меньше, чем записанная в журнале. Если в процессе прогонки снова возникает сбой, то сканирование журнала вновь начнется сначала, но фактически восстановление продолжится с той точки, откуда оно прервалось.

Структура журнала обычно является сугубо частным делом конкретной реализации. Отметим только самые общие свойства:

1.Журнал обычно представляет собой чисто последовательный файл с записями переменного размера, которые можно просматривать в прямом или обратном порядке.

2.Обмены производятся стандартными порциями (страницами) с использованием буфера оперативной памяти. В грамотно организованных системах структура (и тем более, смысл) журнальных записей известна только компонентам СУБД, ответственным за журнализацию и восстановление.

3.Поскольку содержимое журнала является критичным при восстановлении базы данных после сбоев, к ведению файла журнала предъявляются особые требования по части надежности. В частности, обычно стремятся поддерживать две идентичные копии журнала на разных устройствах внешней памяти.

***Организация служебной информации***

Для корректной работы подсистемы управления данными во внешней памяти необходимо поддерживать информация, которая используется только этой подсистемой и не видна подсистеме языкового уровня. Набор структур служебной информации зависит от общей организации системы, но обычно требуется поддержание следующих служебных данных:

1.*Внутренние каталоги*, описывающие физические свойства объектов базы данных, например, число атрибутов отношения, их размер и, возможно, типы данных; описание индексов, определенных для данного отношения и т.д.

2.*Описатели свободной и занятой памяти* в страницах отношения. Такая информация требуется для нахождения свободного места при занесении кортежа.

3.*Связывание страниц одного отношения*. Если в одном файле внешней памяти могут располагаться страницы нескольких отношений, то нужно каким-то образом связать страницы одного отношения. Тривиальный способ использования прямых ссылок между страницами часто приводит к затруднениями при синхронизации транзакций (например, особенно трудно освобождать и заводить новые страницы отношения). Поэтому стараются использовать косвенное связывание страниц с использованием служебных индексов. В частности, известен общий механизм для описания свободной памяти и связывания страниц на основе B-деревьев.

### *5.2.4. Організація процесів журналізації змін станів РБД*

Одним из основных требований к развитым СУБД является надежность хранения баз данных. Это требование предполагает, в частности, возможность восстановления согласованного состояния базы данных после любого рода аппаратных и программных сбоев. Очевидно, что для выполнения восстановлений необходима некоторая дополнительная информация. В подавляющем большинстве

современных реляционных СУБД такая избыточная дополнительная информация поддерживается в виде журнала изменений базы данных.

*Общей целью журнализации* изменений баз данных является обеспечение возможности восстановления согласованного состояния базы данных после любого сбоя. Поскольку основой поддержания целостного состояния базы данных является механизм транзакций, журнализация и восстановление тесно связаны с понятием транзакции.

*Общими принципами* восстановления являются следующие:

- результаты зафиксированных транзакций должны быть сохранены в восстановленном состоянии базы данных;
- результаты незафиксированных транзакций должны отсутствовать в восстановленном состоянии базы данных.

Возможны следующие *ситуации*, при которых требуется производить восстановление состояния базы данных:

1.Индивидуальный откат транзакции. Тривиальной ситуацией отката транзакции является ее явное завершение оператором ROLLBACK. Возможны также ситуации, когда откат транзакции инициируется системой. Примерами могут быть возникновение исключительной ситуации в прикладной программе (например, деление на ноль) или выбор транзакции в качестве жертвы при обнаружении синхронизационного тупика. Для восстановления согласованного состояния базы данных при индивидуальном откате транзакции нужно устранить последствия операторов модификации базы данных, которые выполнялись в этой транзакции.

2.Восстановление после внезапной потери содержимого оперативной памяти (мягкий сбой). Такая ситуация может возникнуть при аварийном выключении электрического питания, при возникновении неустранимого сбоя процессора (например, срабатывании контроля оперативной памяти) и т.д. Ситуация характеризуется потерей той части базы данных, которая к моменту сбоя содержалась в буферах оперативной памяти.

3.Восстановление после поломки основного внешнего носителя базы данных (жесткий сбой). Эта ситуация при достаточно высокой надежности современных устройств внешней памяти может возникать сравнительно редко, но тем не менее, СУБД должна быть в состоянии восстановить базу данных даже и в этом случае. Основой восстановления является архивная копия и журнал изменений базы данных.

Во всех трех случаях основой восстановления является избыточное хранение данных. Эти избыточные данные хранятся в журнале, содержащем последовательность записей об изменении базы данных.

Возможны *два основных варианта ведения журнальной информации*. В первом варианте для каждой транзакции поддерживается отдельный локальный журнал изменений базы данных этой транзакцией. Эти локальные журналы используются для индивидуальных откатов транзакций и могут поддерживаться в оперативной (правильнее сказать, в виртуальной) памяти. Кроме того, поддерживается общий журнал изменений базы данных, используемый для восстановления состояния базы данных после мягких и жестких сбоев.

Этот подход позволяет быстро выполнять индивидуальные откаты транзакций, но приводит к дублированию информации в локальных и общем журналах. Поэтому чаще используется второй вариант - поддержание только общего журнала изменений базы данных, который используется и при выполнении индивидуальных откатов.

# Розділ 6. Технологія проектування баз даних автоматизованих інформаційних систем

## Тема 6.1. Сучасні підходи до розроблення і впровадження інформаційних систем

Процес проектування і розробки інформаційної системи (ІС) не може бути подібним до процесу приготування їжі за кулінарною книгою, необхідно бути завжди готовим до труднощів, пов'язаних з освоєнням нових технологій.

Основні проблеми, що постають перед програмною інженерією, пов'язані з інтеграцією створеного раніше програмного забезпечення (ПЗ) у нові розробки (legacy challenge), роботою в розподіленому гетерогенному середовищі (heterogeneity challenge) та обмеженнями часу, що відводиться на розроблення інформаційних продуктів (delivery challenge).

*Основні розділи програмної інженерії:*

- аналіз вимог до ІС, яку треба створити;
- детальний проект ІС;
- кодування;
- тестування системи;
- процес супроводження програмного продукту;
- керування конфігурацією;
- забезпечення якості розроблення;
- забезпечення відповідності розроблення вимогам її замовників та забезпечення відповідності кодів проекту;
- процес удосконалення отриманого програмного продукту.

Еталонна модель програмної інженерії визначається взаємодією трьох факторів: процесів, продуктів та ресурсів.

*Життєвий цикл програмного забезпечення*. Поняття життєвого циклу програмного забезпечення (ЖЦ ПЗ) є одним з базових у програмній інженерії (ПІ).

*Життєвий цикл ПЗ* - певна послідовність фаз або стадій від моменту прийняття рішення про необхідність створення ПЗ до повного вилучення ПЗ з експлуатації.

На кожній фазі відбувається певна сукупність процесів, кожний з яких породжує певний продукт, використовуючи необхідні ресурси. Стандарт міжнародної організації ISO/IEC 12207:1995 "Information Technology - Software Life Cycle Processes" визначає структуру ЖЦ, що містить процеси, дії і задачі, які мають бути виконані під час створення ПЗ.

Стандарт визначає *ПЗ* як набір комп'ютерних програм, процедур і, можливо, пов'язаних із ними документації й даних. Процес - це сукупність взаємопов'язаних дій, що перетворюють вхідні дані у вихідні.

Процес поділяється на набір дій, а дії - на набір задач. Процеси, дії та задачі ініціюються іншими процесами і виконуються у міру необхідності, причому немає заздалегідь визначених послідовностей виконання.

Усі продукти програмної інженерії становлять певні описи - тексти вимог до розроблення, узгодження домовленостей, документацію, тексти програм, інструкції щодо експлуатації тощо. Головні ресурси програмної інженерії, що визначають ефективність розроблень, - це час та вартість.

Відповідно до стандарту ISO/IEC 12207 усі процеси ЖЦ ПЗ поділяються на три групи (рис. 1):

- *основні процеси* (придбання, доставка, розроблення, експлуатація, супровід);
- *організаційні* процеси (управління, удосконалення, навчання);
- *допоміжні* процеси (документування, забезпечення якості, верифікація, атестація, аудит, загальна оцінка тощо).

Процеси придбання й доставки - це процедури, що передбачають виконання замовлення та постачання продукту замовнику.

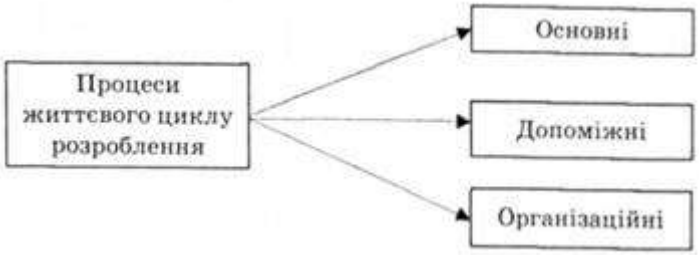



*Рис.* 1. Процеси життєвого циклу розроблення ПЗ

Процес розроблення передбачає дії, що виконуються розробником, і охоплює роботи зі створення ПЗ та його компонентів відповідно до вимог, включаючи оформлення проектної Й експлуатаційної документації, підготовку матеріалів, необхідних для перевірки працездатності і відповідної якості програмних продуктів, матеріалів, потрібних для організації навчання персоналу.

*Основні процеси включають:*

- *процес придбання, що* ініціює життєвий цикл ІС та визначає її покупця;
- *процес розроблення,* що визначає дії організації - розробника інформаційного продукту;
- *процес постачання,* що визначає дії під час передачі розробленого продукту покупцеві;
- *процес експлуатації,* що означає дії з обслуговування системи під час її використання - консультації користувачів, вивчення їхніх побажань тощо;
- *процес супроводження,* що означає дії з керування модифікаціями, підтримки актуального стану та функціональної придатності, інсталяції та вилучення версій систем у користувача.

Процес розроблення ПЗ має забезпечити шлях від усвідомлення потреб замовника до передачі йому готового продукту (рис. 2).

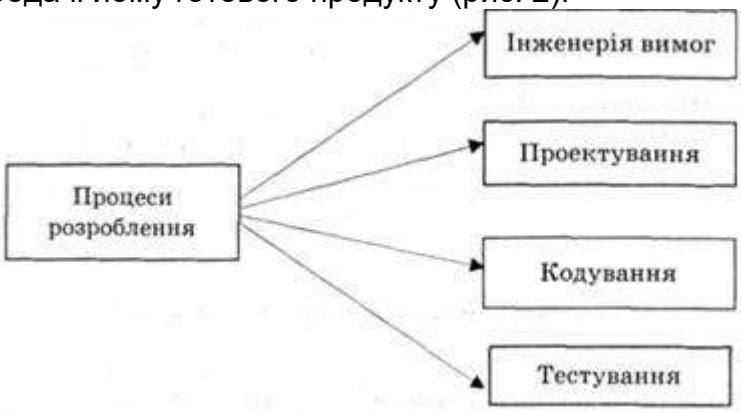



*Рис.* 2. Процеси розроблення програмного забезпечення

Він складається з таких етапів:

- *визначення вимог* - збір та аналіз вимог замовника виконавцем та подання їх у нотації, що зрозуміла як замовнику, так і виконавцю;
- *проектування* - перетворення вимог до розроблення у послідовність проектних рішень щодо способів реалізації вимог: формування загальної архітектури програмної системи та принципів її прив'язки до конкретного середовища функціонування; визначення детального складу модулів кожної з архітектурних компонент;
- *реалізація* - перетворення проектних рішень у програмну систему, що реалізує означені рішення;
- *тестування* - перевірка кожного з модулів та способів їх інтеграції; тестування програмного продукту в цілому (так звана верифікація); тестування відповідності функцій працюючої програмної системи вимогам, що були до неї поставлені замовником (так звана валідація);
- *експлуатація та супроводження* готової системи. Підготовча робота починається з вибору моделі ЖЦ ПЗ, що відповідає масштабові, значимості і складності проекту. Процес розроблення має відповідати обраній моделі. Розробник повинен вибрати, адаптувати до умов проекту і використовувати погоджені із замовником стандарти, методи й засоби розроблення, а також скласти план виконання робіт.

*Аналіз вимог до системи* розглядає функціональні можливості, вимоги користувача, вимоги до надійності і безпеки, вимоги до зовнішніх інтерфейсів тощо. Вимоги до системи оцінюються відповідно до критеріїв реалізації і можливості перевірки при тестуванні.

*Проектування архітектури системи* полягає у визначенні компонентів її устаткування, ПЗ й операцій, що виконуються персоналом.

*Аналіз вимог до ПЗ* визначає: функціональні можливості, включаючи характеристики продуктивності і середовища функціонування компонента; зовнішні інтерфейси; специфікації надійності і безпеки; ергономічні вимоги; вимоги до даних; вимоги до інсталяції та введення системи; вимоги до документації користувачів; вимоги до експлуатації і супроводу.

*Проектування архітектури ПЗ* включає такі задачі (для кожного компонента ПЗ):

а) трансформацію вимог до ПЗ в архітектуру, що визначає структуру ПЗ і склад його компонентів;

б) розроблення і документування програмних інтерфейсів ПЗ і БД;

в) розроблення попередньої версії документації користувачів;

г) розроблення і документування попередніх вимог до тестів і плану інтеграції ПЗ.

*Детальне проектування ПЗ* включає такі задачі:

а) опис компонентів ПЗ й інтерфейсів між ними на нижчому рівні, що достатній для їх подальшого самостійного кодування і тестування;

б) розроблення і документування детального проекту бази даних;

в) відновлення (за необхідності) документації;

г) розроблення і документування вимог до тестів і плану тестування компонентів ПЗ;

д) відновлення плану інтеграції ПЗ. Кодування і тестування ПЗ охоплюють такі задачі:

а) розроблення (кодування) і документування кожного компонента ПЗ і бази даних, а також сукупності тестових процедур і даних для їхнього тестування;

б) тестування кожного компонента ПЗ і БД на відповідність вимогам. Результати тестування компонентів мають бути документовані;

в) відновлення (за необхідності) документації користувачів;

г) відновлення плану інтеграції ПЗ.

*Інтеграція ПЗ* передбачає збирання розроблених компонентів ПЗ відповідно до плану інтеграції і тестування компонентів. Для кожного з компонентів розробляються набори тестів і тестові процедури, що призначені для перевірки кваліфікаційних вимог при наступному кваліфікаційному тестуванні. *Кваліфікаційна вимога* - це набір критеріїв або умов, який необхідно виконати, щоб кваліфікувати програмний продукт на відповідність своїм специфікаціям і можливість його використовувати в умовах експлуатації.

*Кваліфікаційне тестування ПЗ* проводиться розробником у присутності замовника для демонстрації того, що ПЗ дійсно відповідає своїм специфікаціям. Кваліфікаційне тестування здійснюється для кожного компонента ПЗ щодо всіх вимог при використанні різних тестів. При цьому також перевіряються повнота технічної документації та її адекватність самим компонентам ПЗ.

*Інтеграція системи* полягає в об'єднанні всіх її компонентів, включно з ПЗ й устаткуванням. Після інтеграції система у свою чергу піддається кваліфікаційному тестуванню на відповідність сукупності вимог до неї. При цьому також готуються оформлення і перевірка повного комплекту документації на систему.

Встановлення ПЗ здійснюється розробником відповідно до плану в тому операційному середовищі і на тому обладнанні, що передбачені замовленням.

*Приймання ПЗ* передбачає оцінку результатів кваліфікаційного тестування ПЗ та системи і документування результатів оцінювання, що проводиться замовником за допомогою розробника. Розробник здійснює остаточну передачу ПЗ замовнику відповідно до договору, забезпечуючи при цьому необхідне навчання і підтримку.

*Процес експлуатації* охоплює дії і задачі оператора - організації, що експлуатує систему. Цей процес включає такі етапи: 1) підготовчу роботу; 2) експлуатаційне тестування; 3) експлуатацію системи; 4) підтримку користувачів.

*Підготовча робота* включає проведення оператором планування дій і робіт, що виконуються у процесі експлуатації, й установку експлуатаційних стандартів та визначення процедур локалізації і розв'язання проблем, які виникають у процесі експлуатації.

*Експлуатаційне тестування* проводиться для кожної чергової версії програмного продукту, після чого вона передається в експлуатацію.

*Експлуатація системи* здійснюється у призначеній для цього ОС відповідно до документації користувачів.

*Підтримка користувачів* полягає в наданні допомоги і консультацій при виявленні помилок у процесі експлуатації ПЗ.

*Процес супроводу* передбачає дії і задачі, що виконуються службою супроводу. Цей процес активізується при модифікаціях програмного продукту і відповідної документації або модернізації, адаптації ПЗ. Супровід - це внесення змін у ПЗ з метою виправлення помилок, підвищення продуктивності або адаптації до умов праці, що змінилися.

Зміни, внесені в наявне ПЗ, не повинні порушувати його цілісність. Процес супроводу включає перенесення ПЗ в інше середовище (міграцію) і закінчується зняттям ПЗ з експлуатації. Цей процес охоплює такі дії: 1) підготовчу роботу; 2) аналіз проблем і запитів на модифікацію ПЗ; 3) модифікацію ПЗ; 4) перевірку і приймання; 5) міграцію ПЗ в інше середовище; 6) зняття ПЗ з експлуатації.

*Підготовча робота служби супроводу* включає такі задачі: планування дій і робіт, які виконуються у процесі супроводу та визначення процедур локалізації і розв'язання проблем, що виникають у процесі супроводу.

*Аналіз проблем і запитів на модифікацію ПЗ"* що виконуються службою супроводу, включає такі задачі:

- аналіз повідомлення про проблему або запит на модифікацію ПЗ. При цьому визначаються такі характеристики можливої модифікації: тип (коригувальна, поліпшуюча, профілактична); масштаб (розміри модифікації, вартість і термін її реалізації); критичність (вплив на продуктивність, надійність або безпеку);
- оцінка доцільності та можливих варіантів проведення модифікації;
- затвердження обраного варіанта модифікації.

Модифікація ПЗ передбачає визначення компонентів ПЗ, їхніх версій і документації, що підлягають модифікації, внесення необхідних змін відповідно до правил *процесу розроблення.* Підготовлені зміни тестуються і перевіряються за критеріями, що передбачені документацією. При підтвердженні коректності змін у програмах відбувається коригування документації.

*Перевірка і приймання* полягають у перевірці цілісності модифікованої системи і затвердженні внесених змін.

При *перенесенні ПЗ в інше середовище* використовуються наявні або розробляються нові засоби перенесення, потім виконується конвертування програм і даних у нове середовище. З метою полегшення переходу передбачається паралельна експлуатація ПЗ у старому і новому середовищі впродовж певного періоду, під час якого проводиться необхідне навчання користувачів з новим ПЗ.

*Зняття ПЗ з експлуатації* здійснюється за рішенням замовника за участю організації експлуатації, служби супроводу і користувачів. При цьому програмні продукти і відповідна документація підлягають архівуванню відповідно до договору.

**Моделі життєвого циклу ПЗ**

Модель ЖЦ ПЗ залежить від специфіки, масштабу і складності проекту та особливостей умов, за яких система створюється та функціонує.

***Модель ЖЦ ПЗ*** - це структура, що визначає послідовність виконання і взаємозв'язок процесів, дій, задач протягом ЖЦ.

Модель ЖЦ конкретного ПЗ інформаційної системи визначає характер процесу створення цього ПЗ, що означає сукупність упорядкованих у часі, об'єднаних у стадії робіт.

***Стадія створення ПЗ*** - це частина процесу створення ПЗ, що обмежена певними часовими рамками і завершується випуском конкретного продукту (моделей ПЗ, програмних компонентів, документації).

Найбільшого поширення набули дві моделі: каскадна (водоспадна), створена в 1970-1985 рр., та спіральна, створена в 1986-1990 рр.

***Каскадна модель*** життєвого циклу (модель водоспаду, англ. ***waterfall model)*** була запропонована у 1970 р. У. Ройсом. Принципова особливість каскадної моделі - перехід на наступну стадію здійснюється тільки після повного завершення роботи на поточній стадії, повернення на пройдені стадії не передбачається. Кожна стадія закінчується одержанням результатів, що є вхідними даними для наступної стадії, та випуском повного комплекту документації. Вимоги до ПЗ, визначені на стадії формування вимог, документуються у вигляді технічного завдання і фіксуються на весь час розроблення. Критерієм якості розробки за такої моделі є точність виконання специфікацій технічного завдання.

На рис. 3 зображена ***каскадна модель*** життєвого циклу програмної системи. Цінність цієї моделі полягає в тому, що вона фіксує послідовність етапів розроблень та можливість повернення до попередніх етапів роботи.

Основна увага розробників зосереджується на досягненні найкращих значень технічних характеристик ПЗ, а саме: продуктивності, обсягу пам'яті тощо.

***Переваги*** застосування каскадної моделі:

о на кожній стадії формується закінчений набір проектної документації, яка відповідає критеріям повноти й узгодженості;

о виконання робіт у логічній послідовності дає змогу планувати терміни завершення всіх робіт і відповідні витрати.

Ця модель добре зарекомендувала себе при побудові ПЗ, для яких на самому початку розроблення можна досить точно і повно сформулювати усі вимоги. Під цю категорію потрапляють складні системи з великою кількістю задач обчислювального характеру, системи реального часу тощо.

***Недоліки*** цієї моделі викликані насамперед тим, що реальний процес створення ПЗ ніколи цілком не укладався в жорстку схему. Процес створення ПЗ часто має ітераційний характер: результати чергової стадії викликають зміни у проектних рішеннях, що прийняті на попередніх стадіях.


*Рис.* 3. Каскадна модель життєвого циклу

Отже, постійно виникає потреба в поверненні до попередніх стадій і уточненні або перегляді раніше прийнятих рішень.

У результаті реальний процес створення ПЗ набуває іншого вигляду. Цю схему часто називають ***моделлю з проміжним контролем,*** тому що коригування між стадіями розроблення забезпечують більшу надійність порівняно з каскадною моделлю, проте збільшують весь період розроблення ПЗ.

Основний недолік каскадної моделі - високий ризик створення системи, що не задовольняє потреби користувачів. Практика переконує, що на початковій стадії проекту точно сформулювати всі вимоги до майбутньої системи не вдається. Це викликано двома причинами: 1) користувачі не в змозі відразу викласти усі свої вимоги і не можуть передбачати, як вони зміняться в ході розроблення; 2) у зовнішньому середовищі за час розроблення можуть відбутися зміни, що вплинуть на вимоги до системи. За каскадної моделі вимоги до ПЗ фіксуються у вигляді технічного завдання на весь час її створення, а узгодження одержуваних результатів з користувачами виробляється тільки в точках, запланованих після завершення кожної стадії (при цьому можливе коригування результатів згідно із зауваженнями користувачів, якщо вони не стосуються вимог технічного завдання). Отже,

користувачі можуть внести важливі зауваження тільки після того, як робота над системою буде повністю завершена. У разі неточного викладу вимог або їх зміни після тривалого періоду створення ПЗ користувачі одержать систему, що не відповідає їх потребам.

Для подолання перерахованих проблем у середині 1980-х років була запропонована спіральна модель ЖЦ ПЗ (рис. 4).


*Рис.* 4. Модель спірального процесу розроблення ІС

***Спіральна модель (spiral model)*** була розроблена у середині 1980-х років Барі Боемом. Вона ґрунтується на класичному циклі Демінга PDCA (plan-do-check-act). При використанні цієї моделі ПЗ створюється в кілька ітерацій (витків спіралі) методом прототипування.

Нині ця модель досить поширена. Найвідоміші приклади її реалізації - це RUP (Rational Unified Process) фірми Rational і MSF (Microsoft Solution Framework). Створення ПЗ за такої моделі має ітераційниЙ характер і рухається по спіралі, проходячи стадії, де на кожному витку уточнюються характеристики майбутнього інформаційного продукту.

Суттєва особливість спіральної моделі ЖЦ ПЗ полягає в тому, що прикладне ПЗ створюється не відразу, а частково, з використанням методу прототипування. ***Прототип*** - це програмний компонент, що реалізує окремі функції і зовнішні інтерфейси ПЗ. Створення прототипів здійснюється кількома ітераціями. Кожна ітерація відповідає створенню фрагмента або версії ПЗ, уточнюються цілі і характеристики проекту, оцінюється якість отриманих результатів і плануються роботи наступної ітерації. На кожній ітерації виробляється ретельна оцінка ризику перевищення термінів і вартості проекту, щоб визначити необхідність виконання ще однієї ітерації, ступінь повноти і точності розуміння вимог до системи, а також доцільність припинення проекту. Спіральна модель позбавляє користувачів і розробників ПЗ від необхідності повного й точного формулювання вимог до системи на початковій стадії, оскільки вони уточнюються на кожній ітерації. У такий спосіб уточнюються і послідовно конкретизуються деталі проекту і зрештою вибирається обґрунтований варіант, який і реалізується.

ІтераційниЙ процес розроблення відображає об'єктивно спіральний цикл створення системи. Неповне завершення робіт на кожній стадії дає змогу переходити на наступну стадію, не чекаючи повного завершення роботи на поточній. При ітеративному способі розроблення відсутню стадію можна буде виконати на наступній ітерації. Головне завдання - якнайшвидше показати користувачам системи працездатний продукт, активізуючи процес уточнення і доповнення вимог.

Спіральна модель не виключає використання каскадного підходу на кінцевих стадіях проекту в тих випадках, коли вимоги до системи стають цілком чіткими.

Основна проблема спірального циклу - визначення моменту переходу на наступну стадію. Для її вирішення необхідно ввести часові обмеження на кожну зі стадій життєвого циклу.

Перехід здійснюється відповідно до плану, навіть якщо не вся запланована робота закінчена. План складається на основі статистичних даних, отриманих у попередніх проектах, і особистого досвіду розробників.

**Інженерія вимог**

Стадія формування вимог до ПЗ - це найважливіша стадія, оскільки вона визначає успіх усього проекту. Ця стадія складається з таких етапів:

1) планування робіт включає визначення мети розробки, попередню економічну оцінку проекту, створення плану-гра-фіка виконання робіт, навчання спільної робочої групи;

2) проведення обстеження діяльності об'єкта (організації) автоматизації, у рамках якого здійснюються: попереднє виявлення вимог до майбутньої системи; визначення структури організації; визначення переліку цілей організації; аналіз розподілу функцій за підрозділами і між співробітниками; виявлення функціональних взаємодій між підрозділами, інформаційних потоків усередині підрозділів і між ними, зовнішніх стосовно організації об'єктів і зовнішніх інформаційних взаємодій; аналіз наявних засобів автоматизації діяльності організації;

3) побудову моделей діяльності організації, що передбачає обробку матеріалів обстеження;

4) побудову двох видів моделей:

- моделі ***"як є",*** що відображає наявний на момент обстеження стан справ і допомагає зрозуміти, як саме функціонує певне підприємство, а також виявити вузькі місця і сформулювати пропозиції щодо поліпшення ситуації;
- моделі ***"як має бути",*** що відображає схему про нові технології роботи підприємства. Кожна з моделей містить повну функціональну й інформаційну модель діяльності організації, а також у разі потреби модель, що описує динаміку поведінки організації.
- відмовостійкість;
- кількість клієнтів, що одночасно мають доступ до системи;
- вимоги безпеки;
- час очікування відповіді на звернення до системи;
- виконавські властивості системи (обмеження щодо ресурсів пам'яті, швидкість реакції на звернення до системи тощо).

Наступний крок аналізу вимог - встановлення їх пріоритетності, бо вимоги, висунуті різними носіями інтересів у системі, можуть конфліктувати між собою. Крім того, кожна з вимог потребує для свого втілення певних ресурсів, надання яких може залежати також від визначеного для неї пріоритету.

Ще одним важливим завданням аналізу є передбачення здатності адаптації до можливих змін у вимогах та забезпечення можливостей внесення змін без суттєвого перегляду всієї системи. У процесі аналізу вимог мають бути перевірені їх правдивість та відповідність інтересам замовника.

## Тема 6.2. Засоби проектування баз даних

### Автоматизація проектування ІС

На етапі проектування ІС побажання замовників перетворюються у проектні рішення у формі певної системи програмування.

***Проект ІС*** - це проектно-конструкторська та технологічна документація, в якій подається опис рішень створення та експлуатації ІС у конкретному програмно-технічному середовищі.

В основі проектування будь-якого продукту лежить парадигма подолання складності завдання шляхом його декомпо-зиції на окремі компоненти.

***Технологія проектування ІС*** - це поєднання методології та інструментальних засобів проектування ІС.

Методологія проектування передбачає наявність концепції, принципів проектування, засобів проектування. ***Метод проектування ПЗ*** - це організована сукупність процесів створення моделей, що описують різні аспекти ІС з використанням нотацій. ***Метод*** - це сукупність:

- ***концепцій і теоретичних основ*** (наприклад, структурний або об'єктно орієнтований підхід);
- ***нотацій,*** що використовуються для побудови моделей статичної структури і динаміки поведінки ІС (діаграми потоків даних і діаграми "сутність - зв'язок" для структурного підходу, діаграми варіантів використання, діаграми класів в об'єктно орієнтованому підході);
- ***процедур, що*** визначають практичне застосування методу (послідовність і правила побудови моделей, критерії для оцінювання результатів).

***Технологія проектування ПЗ*** - це сукупність технологічних операцій проектування (рис. 5) у певній послідовності і взаємозв'язку. Апарат технологічних мереж проектування - це зручний інструмент формалізації технології проектування ІС. Основа його формалізації - визначення технологічної операції проектування у вигляді множини документів (описувач множини фактів), параметрів (описувач одного факту), програм (опис алгоритмів рішення задачі), універсальних множин (повна множина фактів одного типу), на яких задані перетворювачі, ресурси, засоби проектування на конкретному вході/виході.

*Рис.* 5. Контекст технологічної операції проектування

Методи реалізуються через конкретні технології і методики, стандарти й інструментальні засоби, що забезпечують виконання процесів ЖЦ ПЗ. Розрізняють ***методи оригінального проектування,*** коли створюється оригінальна ІС, та типового проектування, коли ІС компонується з готових типових рішень. Комбінація різних методів проектування зумовлює характер технології проектування ІС. Найвідоміші технології проектування ІС - це ***канонічна*** (ручна технологія індивідуального проектування) та ***індустріальна,*** що у свою чергу поділяється на ***автоматизовану*** (з використанням CASE-тех-нологій) і ***типову*** (модельно орієнтовану або параметрично орієнтовану).

Більшість існуючих САвЕ-засобів засновано на методах структурного або об'єктно орієнтованого аналізу і проектування, що використовує специфікації у вигляді діаграм або текс-

Перехід від моделі "як є" до моделі "як має бути" може відбуватися двома способами:

1) удосконалюванням діючих технологій на основі оцінки їхньої ефективності;
2) радикальною зміною технологій і перепроектуванням бізнес-процесів.

***Стадія проектування*** включає такі етапи:

- ***розроблення системного проекту.*** На цьому етапі дається відповідь на питання: що має робити майбутня ІС?, а саме: визначаються архітектура системи, її функції, зовнішні умови функціонування, інтерфейси й розподіл функцій між користувачами і системою, вимоги до програмних та інформаційних компонентів, склад виконавців і терміни розроблення. Основу системного проекту становлять моделі ІС, що проектуються на основі моделі "як має бути", а результатом діяльності автоматизації є технічне завдання;

- ***розроблення технічного проекту,*** яке охоплює проектування системи, що включає проектування архітектури системи і детальне проектування.

Моделі ІС уточнюються і деталізуються до необхідного рівня. На кожній стадії проектування може виконуватися кілька процесів, що визначаються у стандарті ШО/ІЕС 12207. Кожна програма - це певний перетворювач, поведінку і властивості якого визначають у процесі створення системи так, щоб вирішити певну проблему.

***Вимоги до програмної системи*** - це властивості, які слід мати системі для адекватного виконання своїх функцій.

У сучасних ІТ фаза життєвого циклу, на якій фіксуються вимоги до розроблення програмного забезпечення, ***визначальна*** для його якості, термінів та вартості робіт. Саме на цій фазі мають бути зафіксовані реальні потреби користувачів у функціональних, операційних та сервісних можливостях, які має реалізувати розробник. Отже, на цій фазі відбувається домовленість між замовником та виконавцем, яка визначає подальші дії виконавця.

Ціна помилок і нечітких неоднозначних формулювань на цій фазі дуже висока, адже час та засоби витрачаються на непотрібну замовникові програму. Внесення необхідних коректив при цьому може вимагати серйозних переробок, а інколи й повного перепроектування і, відповідно, перепрограмування. За статистичними даними відсоток помилок у постановці завдань перевищує відсоток помилок кодування, і це є наслідком суб'єктивного характеру процесу формулювання вимог та майже повної відсутності засобів його формалізації. Дійовими особами процесу формулювання вимог є:

- ***носії інтересів замовників*** (досить часто замовника репрезентують кілька професійних груп, які можуть мати не тільки відмінні, але навіть суперечні потреби);

- ***оператори,*** що обслуговують функціонування системи;
- ***розробники*** системи.

Процес формулювання вимог складається з двох етапів - збирання та аналізу вимог.

***Джерела відомостей про вимоги:***

- мета та завдання системи, як їх формулює замовник;
- діюча система або колектив, що виконує її функції;
- загальні знання щодо проблемної галузі замовника;
- відомчі стандарти замовника, що стосуються організаційних вимог, середовища функціонування майбутньої системи, її виконавських та ресурсних можливостей.

***Методи збирання вимог:***

- інтерв'ю з носіями інтересів замовника та операторами;
- спостереження за роботою діючої системи;
- фіксація сценаріїв усіх можливих випадків використання системи, виконуваних при цьому системою функцій, ролей осіб, що запускають ці сценарії або взаємодіють з системою під час її функціонування.

Множина зібраних вимог може бути розподілена між двома основними категоріями:

1) такі, що відображають можливості, які повинна забезпечити система, - ***функціональні;***

2) такі, що відображають обмеження, пов'язані з функціонуванням системи, - ***нефункціональні.***

Є кілька класів нефункціональних вимог, суттєвих для більшості ІС, які виражають обмеження, актуальні для багатьох проблемних галузей:

вимоги конфіденційності;

тів для опису зовнішніх вимог, зв'язків між моделями системи, динаміки поведінки системи й архітектури програмних засобів. CASE-технологія дозволяє у наочній формі моделювати ПрО, аналізувати її модель на всіх стадіях розроблення і супроводу ІС і розробляти застосування відповідно до інформаційних потреб користувачів.

Сучасна технологія проектування ПЗ ІС має забезпечувати:

- відповідність стандартові ІБО/ІЕС 12207;
- гарантоване досягнення цілей розробки БС у межах бюджету з дотриманням якості й установленого часу;
- можливість декомпозиції проекту на складові з наступною інтеграцією цих частин;
- мінімальний час одержання працездатного ПЗ ІС;
- незалежність проектних рішень від засобів реалізації 1С (СУБД, операційних систем, мов і систем програмування);
- підтримка CASE-засобів, що забезпечують автоматизацію процесів, виконуваних на всіх стадіях ЖЦ.

Реальне застосування будь-якої технології проектування ПЗ 1С не можливе без розробки стандартів, яких мають дотримуватися всі учасники проекту (це особливо актуально при великій кількості розробників). До них належать стандарти

проектування, оформлення проектної документації та інтерфейсу кінцевого користувача із системою. ***Стандарт проектування*** встановлює:

а) набір необхідних моделей (діаграм) на кожній стадії проектування і ступінь їх деталізації;

б) правила фіксації проектних рішень на діаграмах, у тому числі правила іменування об'єктів, набір атрибутів для всіх об'єктів і правила їх заповнення на кожній стадії, правила оформлення діаграм тощо;

в) вимоги до конфігурації робочих місць розробників, включаючи настроювання операційної системи та CASE-за-собів;

г) механізм забезпечення спільної роботи над проектом, у тому числі правила інтеграції підсистем проекту, правила підтримки проекту в однаковому для всіх розробників стані, правила аналізу проектних рішень на несуперечність.

***Стандарт оформлення проектної документації*** установлює:

а) комплектність, ***склад*** і структуру документації на всіх стадіях проектування;

б) вимоги до оформлення документації;

в) правила підготовки, розгляду, узгодження і затвердження документації із зазначенням граничних термінів для кожної стадії;

г) вимоги до засобів підготовки документації;

д) вимоги до настроювання CASE-засобів для забезпечення підготовки документації відповідно до встановлених правил.

***Стандарт інтерфейсу користувача із системою регламентує:***

а) правила оформлення екранних елементів і елементів управління;

б) правила використання клавіатури і миші;

в) правила оформлення текстів допомоги;

г) перелік стандартних повідомлень;

д) правила обробки реакції користувача.

**Структурний підхід до розроблення ПЗ**

Зараз у програмній інженерії е два основних підходи до розробки ПЗ ІС, принципова різниця між якими зумовлена різними способами декомпозиції систем: функціонально-модульний (структурний) підхід, в основу якого покладений принцип функціональної декомпозиції, при якій структура системи описується в термінах ієрархії її функцій і передачі інформації між окремими функціональними елементами, та ***об'єктно орієнтований підхід,*** що використовує об'єктну декомпозицію, описує структуру ІС у термінах об'єктів і зв'язків між ними, а поведінку системи - в термінах обміну повідомленнями між об'єктами.

Отже, сутність структурного підходу до розроблення ПЗ ІС полягає в його декомпозиції на автоматизовані функції: система розбивається на функціональні підсистеми, що у свою чергу поділяються на підфункції, вони - на задачі і так до конкретних процедур. При цьому ІС зберігає цілісність подання, де всі складові взаємопов'язані. При розробці системи "знизу нагору", від окремих задач до всієї системи, цілісність втрачається, виникають проблеми при описі інформаційної взаємодії окремих компонентів.

Базовими принципами структурного підходу є:

о принцип "***поділяй і пануй";***

о принцип ***ієрархічного упорядкування*** - принцип організації складових системи в ієрархічні деревоподібні структури з додаванням нових деталей на кожному рівні. Виділення двох базових принципів не означає, що інші принципи другорядні, оскільки ігнорування кожного з них може призвести до непередбачених наслідків.

Основними з цих принципів є:

***о абстрагування*** - виділення суттєвих аспектів системи;

***о несуперечності*** - обґрунтованість і погодженість елементів системи;

***о структурування даних*** - дані мають бути структуро-вані й ієрархічно організовані.

**Методичні основи технологій створення програмного забезпечення**

***Візуальне моделювання.*** Моделлю ПЗ у загальному випадку називають формалізований опис системи ПЗ на певному рівні абстракції. Кожна модель визначає конкретний аспект системи, використовує набір діаграм і документів заданого формату, а також відображає думки й є об'єктом діяльності різних людей з конкретними інтересами, ролями або завданнями.

Графічні (візуальні) моделі є засобами для візуалізації, опису, проектування і документування архітектури системи. Склад моделей, що використовуються в кожному конкретному проекті, і ступінь їх детальності в загальному випадку залежать від таких чинників:

о труднощів проектованої системи;

о необхідної повноти її опису;

о знань і навичок учасників проекту;

о часу, відведеного на проектування.

Візуальне моделювання дуже вплинуло на розвиток CASE-засобів зокрема. Поняття CASE (Computer Aided Software Engineering) використовується у широкому сенсі. Первинне значення цього поняття, обмежене тільки завданнями автоматизації розробки ПЗ, нині набуло нового значення, що охоплює більшість процесів життєвого циклу ПЗ.

CASE-технологія є сукупністю методів проектування ПЗ, а також набором інструментальних засобів, що дозволяють у наочній формі моделювати предметну область, аналізувати цю модель на всіх стадіях розробки і супроводу ПЗ і розробляти затосування відповідно до інформаційних потреб користувачів. Більшість наявних CASE-засобів базується на методах структурного або об'єктно орієнтованого аналізу і проектування, що використовують специфікації у вигляді діаграм або текстів для опису зовнішніх вимог, зв'язків між моделями системи, динаміки поведінки системи та архітектури програмних засобів.

**Методи структурного аналізу і проектування ПЗ**

У структурному аналізі і проектуванні використовуються різні моделі, що описують:

о функціональну структуру системи;

визначається як ієрархія діаграм потоків даних, що описують асинхронний процес перетворення інформації від її введення в систему до видачі споживачеві. Практично, будь-який клас систем успішно моделюється за допомогою DFD-орієнтованих методів. Вони із самого початку створювалися як засіб проектування інформаційних систем, тоді як SADT - як засіб моделювання систем взагалі, і мають багатший набір елементів, що адекватно відображають специфіку таких систем.

З іншого боку, ці різновиди засобів структурного аналізу приблизно однакові з погляду функціональних можливостей засобів моделювання. При цьому одним з основних критеріїв вибору того чи іншого методу є ступінь володіння ним з боку консультанта або аналітика.

Найбільш поширеним засобом моделювання даних є модель "сутність - зв'язок" (Entity-Relationship Model - ERM). Вона вперше була введена П. Ченом у 1976 р. Ця модель традиційно використовується у структурному аналізі і проектуванні, проте, по суті, це підмножина об'єктної моделі предметної області. Один з різновидів моделі "сутність - зв'язок" використовується в методі IDEF1-X, що належить

сімейству стандартів IDEF, і реалізується у низці поширених CASE-засобів (зокрема, AH Fusion ERwin Data Modeler).

**Методи об'єктно орієнтованого аналізу і проектування ПЗ. Мова UML**

Концептуальною основою об'єктно орієнтованого аналізу і проектування ПЗ (ООАП) є ***об'єктна модель.*** її основні принципи (абстрагування, інкапсуляція, модульність та ієрархія) і поняття (об'єкт, клас, атрибут, операція, інтерфейс тощо) найчіткіше сформульовані Г. Бучем у його фундаментальних працях.

Більшість сучасних методів ООАП базуються на використанні мови UML. Уніфікована мова моделювання UML (Unified Modeling Language) є мовою для визначення, подання, проектування і документування програмних систем, організаційно-економічних систем, технічних систем та інших систем різної природи. UML містить стандартний набір діаграм і нотацій найрізноманітніших видів.

UML - це наступник того покоління методів ООАП, які з'явилися в кінці 1980-х і на початку 1990-х років. Створення

UML фактично розпочалося в кінці 1994 р., коли Граді Вуч і Джеймс Рамбо почали роботу щодо об'єднання їх методів Booch і OMT (Object Modeling Technique) під егідою компанії Rational Software. До кінця 1995 р. вони створили першу специфікацію об'єднаного методу, названого ними Unified Method. Тоді ж у 1995 р . до них приєднався автор методу OOSE (Object-Oriented Software Engineering) Івар Якобсон. Таким чином, UML є прямим об'єднанням і уніфікацією методів Г. Буча, Д. Рамбо і Г. Якобсона, проте доповнює їх новими можливостями. Головними при розробці UML були такі цілі:

о надати користувачам готову до використання виразну мову візуального моделювання, що дозволяє їм розробляти осмислені моделі й обмінюватися ними;

о передбачити механізми розширюваності і спеціалізації для розширення базових концепцій;

о забезпечити незалежність від конкретних мов програмування і процесів розробки;

о забезпечити формальну основу для розуміння цієї мови моделювання (мова має бути одночасно точною і доступною для розуміння, без зайвого формалізму);

о стимулювати зростання ринку об'єктно орієнтованих інструментальних засобів;

о інтегрувати кращий практичний досвід.

UML прийнятий на озброєння практично всіма найбільшими компаніями - виробниками ПЗ (Microsoft, Oracle, IBM, Hewlett-Packard, Sybase тощо). Крім того, практично всі світові виробники CASE-засобів, крім IBM Rational Software, підтримують UML у своїх продуктах (Oracle Designer, Together Control Center (Borland), AllFusion Component Modeler (Computer Associates), Microsoft Visual Modeler). Стандарт UML версії 1.1, прийнятий OMG у 1997 р., містить такий набір діаграм: '

Структурні моделі (structural):

о діаграми класів (class diagrams) - для моделювання статичної структури класів системи і зв'язків між ними;

о діаграми компонентів (component diagrams) - для моделювання ієрархії компонентів (підсистем) системи;

о послідовність виконуваних дій;

о передачу інформації між функціональними процесами;

о відношення між даними. Поширеними моделями проектування ПЗ:

1) функціональна модель SADT (Structured Analysis and Design Technique);

2) модель IDEF3;

3) DFD (Data Flow Diagrams) - діаграми потоків даних.

***Метод SADT*** є сукупністю правил і процедур, призначених для побудови функціональної моделі об'єкта певної предметної області. Функціональна модель SADT відображає функціональну структуру об'єкта, тобто його дії і зв'язки між цими діями. Метод SADT розроблений Дугласом Россом у 1969 р. для моделювання штучних систем середньої складності. Цей метод успішно використовувався у військових, промислових і комерційних організаціях США для вирішення широкого кола завдань, таких як довгострокове і стратегічне планування, автоматизоване виробництво і проектування, розробка ПЗ для оборонних систем, управління фінансами і матеріально-технічним постачанням тощо. Метод SADT підтримується Міністерством оборони США, яке було ініціатором розробки сімейства стандартів IDEF (Icam DEFinition), які є основною частиною програми ICAM (інтегрована комп'ютеризація виробництва), що проводиться за ініціативою ВВС США.

***IDEF-0*** - це методологія функціонального моделювання. За допомогою наочної графічної мови система представляється у вигляді набору взаємопов'язаних функцій. ***IDEF-1*** - методологія моделювання інформаційних потоків, що дозволяє відображати та аналізувати їх структуру і взаємозв'язки. ***IDEF-їх*** - методологія побудови реляційних структур. ***IDEF-2*** - методологія динамічного моделювання розвитку систем. IDEF-3 - методологія документування процесів, що відбуваються в системі і використовуються, наприклад, при дослідженні технологічних процесів.

Метод SADT реалізовано саме в одному зі стандартів цього сімейства - IDEF-0, який був затверджений як федеральний стандарт США в 1993 р.

Моделі SADT (IDEF0) традиційно використовуються для моделювання організаційних систем (бізнес-процесів). Слід зазначити, що метод SADT успішно функціонує тільки при описі добре специфікованих і стандартизованих бізнес-процесів у зарубіжних корпораціях, тому він і прийнятий у США як типовий. Перевагами застосування моделей SADT для опису бізнес-процесів є:

о повнота опису бізнес-процесу (управління, інформаційні і матеріальні потоки, зворотні зв'язки);

о жорсткі вимоги методу, що забезпечують отримання моделей стандартного вигляду;

о відповідність підходу до опису процесів стандартам ISO 9000.

На вітчизняних підприємствах бізнес-процеси почали формуватися і розвиватися порівняно недавно. Вони слабо типізуються, тому розумніше орієнтуватися на менш жорсткі моделі.

Метод моделювання IDEF-3, що є частиною сімейства стандартів IDEF, розроблено у 1980 р. для закритого проекту Міноборони США. Цей метод призначений для таких моделей процесів, у яких важливо зрозуміти послідовність виконання дій і взаємозалежності між ними. Хоча IDEF-3 і не досяг статусу федерального стандарту США, він набув значного поширення серед системних аналітиків як доповнення до методу функціонального моделювання IDEF-0 (моделі IDEF-3 можуть використовуватися для деталізації функціональних блоків IDEF-0, що не мають діаграм декомпозиції). Основою моделі IDEF-3 слугує сценарій процесу, що виділяє послідовність дій і під-процесів аналізованої системи.

***Діаграми потоків даних*** (Data Flow Diagrams - DFD) є ієрархією функціональних процесів, пов'язаних потоками даних. Мета такого представлення - показати, як кожен процес перетворює свої вхідні дані у вихідні, а також виявити відношення між цими процесами.

Для побудови DFD традиційно використовуються дві різні нотації, відповідні методам Йордона - ДеМарко і Гейна - Сер-сона. Ці нотації відрізняються одна від одної графічним зображенням символів. Відповідно до цих методів модель системи

Ця технологія забезпечує створення на ранніх стадіях діючої інтерактивної моделі системи-прототипу, що дає змогу демонструвати користувачам майбутню

систему, уточнювати їх вимоги, оперативно модифікувати елементи інтерфейсів: форми введення повідомлень, меню, вихідні документи, склад функцій, структуру діалогу.

Підхід RAD-технології передбачає наявність трьох складових:

о невеликих груп розробників, що виконують роботи з проектування окремих підсистем ПЗ. Це зумовлено вимогою максимального управління колективом;

о короткого, але ретельно проробленого виробничого графіка;

о циклів повторення, при яких розробники в міру того, як програми починають працювати, вносять зауваження, отримані в результаті взаємодії із замовником.

***Команда*** - це група професіоналів, які мають досвід у проектуванні, програмуванні та тестуванні ПЗ, і здатні добре взаємодіяти з користувачами, трансформуючи їх пропозиції в робочі прототипи.

Життєвий цикл ПЗ за підходом RAD включає чотири стадії:

1) аналіз і планування вимог;

2) проектування;

3) реалізація;

4) введення в дію.

На стадії ***аналізу і планування вимог*** користувачі здійснюють такі дії:

а) визначають функції, що має виконувати система;

б) виділяють найважливіші функції, що вимагають про-робки в першу чергу;

в) описують інформаційні потреби, список вимог до системи складається на основі пояснень користувачів під керівництвом фахівців-розробників. Крім того, на цій стадії:

о обмежується масштаб проекту;

о встановлюються часові рамки для кожної з наступних стадій;

о визначається сама можливість реалізації проекту в заданих розмірах фінансування.

У результаті має бути складено список функцій, що задані пріоритетним шляхом, майбутнього ПЗ ІС; спроектовано моделі ПЗ.

На стадії ***проектування*** частина користувачів бере участь у технічному проектуванні системи під керівництвом фахівців-розробників. Для швидкого одержання прототипів застосувань використовуються відповідні інструментальні засоби (CASE-засоби). Користувачі, взаємодіючи з розробниками, уточнюють і доповнюють вимоги до ІС, що не були виявлені на попередніх стадіях. На цій стадії виконуються такі дії:

о детальніше розглядаються процеси ІС;

о за необхідності для кожного процесу створюється частковий прототип: екранна форма, діалог, звіт. При цьому усуваються неоднозначності;

о встановлюються вимоги розмежування доступу до даних;

о визначається склад необхідної документації.

Після детального визначення складу процесів оцінюється кількість функціональних точок (function point) проектної ІС і приймається рішення про поділ ІС на підсистеми, що має реалізовуватися однією командою розробників за певний час для RAD-проектів (до 3 місяців). Функціональною точкою може бути кожнен з таких елементів ІС:

о вхідний елемент застосування (вхідний документ або екранна форма);

о вихідний елемент застосування (звіт, документ, екранна форма);

о запит (пари "питання/відповідь");

о логічний файл (сукупність записів даних, що використовуються всередині застосування);

о інтерфейс застосування (сукупність записів даних, переданих іншому застосуванню).

Далі проект розподіляється між різними командами розробників. Досвід розроблення великих ІС показує, що для підвищення ефективності робіт необхідно розбити проект на окремі підсистеми. Реалізація підсистем має виконуватися окремими групами фахівців. При цьому необхідно забезпечити координацію ведення загального проекту і виключити дублювання

о діаграми розміщення (deployment diagrams) - для моделювання фізичної архітектури системи.

Моделі поведінки (behavioral):

о діаграми варіантів використання (use case diagrams) - для моделювання функціональних вимог до системи (у вигляді сценаріїв взаємодії користувачів з системою);

о діаграми взаємодії (interaction diagrams);

о діаграми послідовності (sequence diagrams) і кооперативні діаграми (collaboration diagrams) - для моделювання процесу обміну повідомленнями між об'єктами;

о діаграми станів (statechart diagrams) - для моделювання поведінки об'єктів системи при переході з одного стану в інший;

о діаграми діяльності (activity diagrams) - для моделювання поведінки системи в рамках різних варіантів використання або потоків управління.

UML має механізм розширення, призначений для того, щоб розробники могли адаптувати мову моделювання до своїх конкретних потреб, не змінюючи при цьому його метамодель. Наявність механізмів розширення принципово відрізняє UML від таких засобів моделювання, як IDEF-0, IDEF-IX, IDEF-3, DFD і ERM. Перераховані мови моделювання можна визначити як сильно типізовані (аналогічно з мовами програмування), оскільки вони не допускають довільної інтерпретації семантики елементів моделей. UML, допускаючи таку інтерпретацію (в основному за рахунок стереотипів), є мовою, що слабо типізується. До її механізмів розширення відносять: стереотипи; тегування (іменовані) значення; обмеження.

***Стереотип*** - це новий тип елементу моделі, який визначається на основі вже існуючого елементу. Стереотипи розширюють нотацію моделі і можуть застосовуватися до будь-яких елементів моделі. Стереотипи класів - це механізм, що дає змогу розділяти класи на категорії. Розробники ПЗ можуть створювати свої власні набори стереотипів, формуючи тим самим спеціалізовані підмножини UML. Такі підмножини (набори стереотипів) у стандарті мови UML мають назву профілів мови.

***Іменоване значення*** - це пара рядків "тег - значення", або "ім'я - вміст", у яких зберігається додаткова інформація про який-небудь елемент системи, наприклад час створення, статус розробки або тестування, час закінчення роботи над ним тощо.

***Обмеження*** - це семантичне обмеження, що має вид текстового виразу природною або формальною мовою (OCL - Object Constraint Language), який неможливо представити за допомогою нотації UML.

Основою взаємозв'язку між структурним і об'єктно орієнтованим підходами є спільність ряду категорій і понять обох підходів (процес і варіант використання, суть і клас тощо). Цей взаємозв'язок може проявлятися в різних формах. Так, одним з можливих варіантів є використання структурного аналізу як основи для об'єктно орієнтованого проектування. При цьому структурний аналіз слід припиняти, як тільки структурні моделі почнуть відображати не тільки діяльність організації, а і систему ПЗ. Після виконання структурного аналізу можна різними способами приступити до визначення класів та об'єктів. Іншою формою прояву взаємозв'язку можна вважати інтеграцію об'єктної і реляційної технологій. Реляційні СУБД є на сьогодні основним засобом реалізації великомасштабних баз даних і сховищ даних. Причини цього очевидні: реляційна технологія використовується досить довго, освоєна величезною

кількістю користувачів і розробників, стала промисловим стандартом, у неї вкладені значні засоби і створена множина корпоративних БД у найрізноманітніших галузях, реляційна модель проста і має суто математичне подання; є велика різноманітність промислових засобів проектування, реалізації та експлуатації реляційних БД. Внаслідок цього реляційні БД здебільшого використовуються для зберігання і пошуку об'єктів у так званих об'єктно реляційних системах.

RAD-технологія - кодування. Одночасність створення клієнтських і серверних місць ІС та активне залучення користувачів до процесу розроблення прикладного ПЗ зумовили поширення технологи швидкого розроблення застосувань RAD (Rapid Application Development) у рамках спіральної моделі ЖЦ.

результатів робіт кожної проектної групи, що може виникнути внаслідок наявності спільних даних і функцій. У разі використання CASE-засобів це означає поділ функціональної моделі системи (наприклад за допомогою діаграм потоків даних для структурного підходу або діаграм варіантів використання для об'єктно орієнтованого підходу). В результаті має бути створено:

о загальну інформаційну модель ІС;

о функціональні моделі системи в цілому і підсистеми, що реалізовані окремими командами розробників;

о інтерфейси між автономно працюючими підсистемами;

о прототипи екранних форм, звітів, діалогів.

Усі моделі і прототипи мають бути отримані із застосуванням саме тих CASE-засобів, які будуть використовуватися далі, при побудові системи. Ця вимога зумовлюється необхідністю уникнути неконтрольованого перекручування даних при передачі інформації про проект з однієї стадії на іншу.

У підході RAD кожен прототип розвивається таким чином, що наступна стадія наслідує попередню.

На ***стадії*** реалізації відбувається безпосереднє швидке розроблення застосування:

о розробники здійснюють ітеративну побудову реальної системи на основі отриманих на попередній стадії моделей, а також вимог нефункціонального характеру (вимог до надійності, продуктивності тощо);

о користувачі оцінюють результати і вносять коригування, якщо у процесі розроблення система перестає відповідати визначеним раніше вимогам. Тестування системи здійснюється у процесі розроблення.

Після закінчення робіт кожної окремої команди розробників здійснюється поступова інтеграція однієї частини системи з іншими, формується повний програмний код, проводиться тестування спільної роботи окремої частини застосування, а потім - тестування системи в цілому. Реалізація системи завершується такими процесами:

о аналізується використання даних і визначається необхідність їхнього розподілу;

о розробляється фізичне проектування бази даних;

о формулюються вимоги до апаратних ресурсів;

о установлюються способи підвищення продуктивності;

о завершується розроблення документації проекту.

На стадії ***упровадження*** здійснюється навчання користувачів та організаційні зміни, і паралельно із введенням нової системи продовжується експлуатація старої. Стадія реалізації займає небагато часу, тому планування і підготовка до впровадження мають починатися заздалегідь, ще на стадії проектування системи. Конкретна реалізація стадії залежить від умов, у яких починалось розроблення 1С, тобто потрібно:

о розробити нову систему "з нуля";

о створити модель діяльності підприємства, за якою можна розробити 1С;

о розробити 1С на основі старої.

Варто зазначити, що підхід RAD не претендує на універсальність. Він придатний тільки для невеликих проектів. Крім того, підхід RAD недоцільно застосовувати для побудови складних розрахункових програм, операційних систем чи програм управління складними об'єктами в реальному масштабі часу, тобто програм, що містять великий обсяг програмного коду.

Основна проблема процесу розробки 1С через підхід RAD полягає у визначенні моменту переходу на наступний етап, тому вводяться часові обмеження на кожен етап життєвого циклу. Після формування технічного завдання та декомпозиції системи здійснюється незалежне розроблення підсистем з наступним збиранням, тестуванням, упровадженням 1С.

Використовують два основних варіанти організації технологічного процесу проектування з використанням систем-про-тотипів. Перший з них використовують для специфікації вимог до 1С, після розроблення яких прототип стає непотрібним.

Основний недолік цього варіанта - неефективне використання системи-прототипу: після усування недоліків у проекті та вдосконалення постановки задачі прототипи не використовують.

Другий варіант передбачає ітераційниЙ розвиток системи-прототипу в готовий для експлуатації програмний продукт. Ітерації розроблення системи-прототипу включають створення та модифікацію системи-прототипу, її демонстрацію користувачу та узгодження, розроблення нових специфіка цій-вимог до системи, нову модифікацію доти, доки не буде створено готовий продукт. При цьому підході різко скорочуються час, ресурси на проектування, розроблення і впровадження ІС. Основні принципи технології RAD такі:

о розроблення застосувань ітераціями;

о необов'язковість повного завершення робіт на кожній стадії ЖЦПЗ;

о обов'язковість залучення користувачів у процес розроблення ІС;

о доцільність застосування CASE-засобів, що забезпечують цілісність проекту і генерацію коду застосувань;

о доцільність застосування засобів управління конфігурацією, що полегшують внесення змін у проект і супровід готової системи;

о використання прототипування, що дає змогу повніше з'ясувати і задовольнити потреби користувачів;

о тестування й розвиток проекту, що здійснюються одночасно з розробленням;

о ведення розробки завдяки нечисленній команді професіоналів;

о грамотне управління розробленням ІС, чітке планування і контроль виконання робіт.

**Класифікація CASE -засобів**

Зупинимося на двох найбільш відомих варіантах класифікації CASE -засобів: за типами і категоріями. Класифікація за типами відображає функціональну орієнтацію CASE-засобів на ті чи інші процеси ЖЦ і включає такі типи:

***о засоби аналізу і проектування,*** призначені для побудови й аналізу моделей діяльності підприємства, моделей проектної системи. До них належать BPwin, Silverrun, Oracle Designer, Rational Rose, Paradigm Plus, Power Designer, System Architect. Результатом таких засобів є специфікації компонентів системи та їхніх інтерфейсів, алгоритмів і структур даних;

***о засоби проектування БД,*** що забезпечують моделювання даних і генерацію схем баз даних для найбільш розповсюджених СУБД. До них відносять Silverrun, Oracle Designer, Paradigm Plus, Power Designer. Найбільш відомий - ERwin;

***о засоби управління вимогами,*** що забезпечують комплексну підтримку вимог до створюваної системи. Прикладами таких засобів є RequisitePro, DOORS - Dynamic Object Oriented Reqiurements System;

***о засоби управління конфігурацією ПЗ -*** PVCS (Merant), ClearCase (Rational Software);

***о засоби документування.*** Найбільш відомим із них є SoDA - Software Document Automation - для автоматизованого документування ПЗ (Rational Software);

***о засоби тестування.*** Найбільш відомим засобом є Rational Suite TestStudio (Rational Software) - набір продуктів для автоматичного тестування застосувань;

***о засоби керування проектом*** - Open Plan Professional (Welcom Software), Microsoft Project;

***о засоби реверсного інжинірингу,*** що призначені для перенесення ПЗ у нове середовище. Вони забезпечують аналіз програмних кодів і схем баз даних і формування на їх основі різних моделей та проектних специфікацій. Засоби аналізу схем БД і формування ERD входять до складу таких CASE-засобів, як Silverrun, Oracle Designer, Power Designer, Erwin. Аналізатори програмних кодів є у складі Rational Rose, Paradigm Plus.

Можна розглянути процеси, що виконуються як послідовно, так і паралельно окремими командами виконавців, це проектування:

о концептуальне;

о архітектурне;

о технічне;

о детальне.

***Концептуальне проектування*** полягає в уточненні розуміння й узгодженні деталей вимог; ***архітектурне проекту' вання*** - у визначенні головних структурних особливостей ІС; ***технічне проектування*** - у відображенні вимог середовища функціонування і розроблення ІС та у визначенні всіх конструкцій як композицій компонент; а ***детальне проектування*** - у визначенні подробиць функціонування та зв'язків для всіх компонент системи.

***Технічне проектування*** - це відображення вимог середовища функціонування і розроблення ІС та визначення всіх конструкцій як композицій компонентів. На цьому етапі відбувається прив'язка проекту до технічних особливостей платформи реалізації, СУБД, організації комунікацій, наявності фактора реального часу, виконавських вимог, таких як швидкість реагування системи на зовнішні стимули тощо.

Тестування системи провадиться, щоб переконатися у відповідності реалізації системи вимогам до неї. Але вимоги здебільшого обумовлюють, що має робити система, тоді як важливо також визначити, що вона не має права робити. Одним зі шляхів вирішення цього є явна фіксація виняткових ситуацій.

***Виняткові ситуації*** - це ситуації, що унеможливлюють злагоджену роботу системи. Причинами їх виникнення можуть бути:

о помилки користувача при зверненні до системи чи під час підготовки даних;

о непередбачені обставини, не виявлені під час тестування;

о випадкові збої обладнання.

Система може по-різному реагувати на виняткові ситуації, а саме: відмовитися виконувати певну послугу; виконати її з помилками; зруйнувати якісь дані.

Щоб поновити працездатність системи, слід виконати один із наведених нижче варіантів робіт:

о поновити стан системи, що передував винятковій ситуації, і спробувати застосувати іншу стратегію виконання послуги;

о поновити попередній стан системи, внести необхідні корективи і повторити виконання послуги за старою стратегією;

о поновити попередній стан системи, сформувати повідомлення про помилку й зупинити систему в очікуванні реакції користувача.

Щоб забезпечити надійність системи, слід передбачити виняткові ситуації для кожної послуги, проаналізувати їх причини та наслідки й побудувати механізми відтворення попереднього стану та виправлення ситуації.

Отже, ***проектування ІС*** - процес прийняття проектно-конструкторських рішень, що дають змогу одержати проект системи, яка задовольняє вимогам замовника. При цьому проектом називають конструкторську, технологічну, програмну документацію, в якій представлено опис усіх рішень зі створення й експлуатації системи в організаційному і програмно-апаратному середовищі. Методом проектування ІС називають сукупність процесів створення моделей, які описують різні аспекти розробленої системи з використанням чітко визначеної нотації.

Класифікація способів проектування ІС здійснюється за ступенем автоматизації робіт проектування - ручне й автоматизоване. За ступенем типізації розрізняють оригінальне проектування, при якому проектні рішення жорстко й однозначно прив'язані до вимог та особливостей конкретного об'єкта і типове проектування, що припускає створення проекту системи з типових елементів. За ступенем адитивності проектних рішень розрізняють метод реконструкції, коли проектоване рішення адаптується шляхом переробки відповідних компонентів та метод параметризації, коли проектоване рішення налагоджується відповідно до змінених параметрів, і метод реструктуризації, коли змінюється модель ПрО і на основі цього генерується нове проектне рішення.

***Технології проектування ІС:*** канонічна та індустріальна. Остання буває представлена автоматизованим проектуванням або через типове проектування.

***Типові способи обробки виняткових ситуацій:***

о подвійне обчислення й порівняння результатів або їх контрольних сум, утому числі виконаних на різних процесорах;

о таймери, що визначають часові інтервали фіксації поточного стану;

о додаткові перевірки коректності даних, які передають зовнішні системи або окремі компоненти однієї системи.

Усі ці дії призводять до додаткових витрат, які є ціною за надійність функціонування системи. їх доцільність визначається виключно специфікою ІС. Якщо наслідки помилок не-зворотні, як, приміром, у системах підвищеного ризику (космічні та ядерні системи, моніторинг хворих), доводиться йти на дублювання процесів і додаткові перевірки.

**Методологія створення ІС**

Стадії та етапи розроблення ІС визначають відповідні державні стандарти. У них наводиться повний перелік стадій та етапів створення автоматизованих систем для різних етапів життєвого циклу (в конкретних умовах стадії та етапи можуть поєднуватись одне з одним або не виконуватись взагалі залежно від особливостей ІС, які створюються, і від домовленості між розробником системи та її замовником).

Життєвий цикл інформаційної системи - це період, який починається з моменту прийняття рішення про необхідність створення ІС і закінчується у момент її повного вилучення з експлуатації.

Відомі такі стандарти життєвого циклу ІС:

о ГОСТ 34.601-90;

о ISO/IEC 12207:1995;

о Custom Development Method (методика Oracle);

о Rational Unif ied Process (RUP);

о Microsoft Solutions Framework (MSF) включає 4 фази: аналіз, проектування, розробка, стабілізація; припускає використання об'єктно орієнтованого моделювання;

о екстремальне програмування (Extrкme Programming, XP). В основі методології - командна робота, ефективна комунікація між замовником і виконавцем протягом усього проекту з розробки ІС. Розробка ведеться з використанням послідовних прототипів.

Стандарт ГОСТ 34.601-90 передбачає такі стадії й етапи створення автоматизованої системи (АС):

1. ***Формування вимог до АС:***

о обстеження об'єкта й обґрунтування необхідності створення АС;

о формування вимог користувача до АС;

о оформлення звіту про виконання робіт і заявки на розробку АС.

2. ***Розробка концепції АС:***

о вивчення об'єкта;

о проведення необхідних науково-дослідних робіт;

о розробка варіантів концепції АС і вибір варіанта концепції АС, що задовольняє вимоги користувачів;

о оформлення звіту про виконану роботу.

3. ***Технічне завдання;*** розробка і затвердження технічного завдання на створення АС:

4. ***Ескізний проект:***розробка попередніх проектних рішень щодо системи і її частин; розробка документації на АС і її частини.

5. ***Технічний проекті***

о розробка проектних рішень щодо системи і її частин;

о розробка документації на АС і її частини;

о розробка й оформлення документації на постачайня комплектуючих виробів;

о розробка завдань на проектування в суміжних частинах проекту.

6. ***Робоча документація:***

о розробка робочої документації на АС і її частини;

о розробка й адаптація програм.

7. ***Введення в дію:*** підготовка об'єкта автоматизації і персоналу.

8. ***Супровід АС:***

о виконання робіт відповідно до гарантійних зобов'язань;

о післягарантійне обслуговування.

Ескізний, технічний проекти і робоча документація - це послідовна побудова все більш точних проектних рішень всіх видів забезпечення інформаційної системи. Допускається виключати стадію Ескізний проект і окремі етапи робіт на всіх стадіях, об'єднувати стадії Технічний проект і Робоча документація в ***Технороботчий проект,*** паралельно виконувати різні етапи і роботи, включати додаткові.

Проте цей стандарт не зовсім підходить для проведення розробок у нинішніх умовах, оскільки багато процесів відображено у ньому недостатньо, а деякі положення застаріли.

***Стандарт ISO/IEC 12207:1995*** (Information Technology Software Life Cycle Processes) є основним нормативним документом, що регламентує склад процесів життєвого циклу ІС.

Він визначає структуру життєвого циклу, що містить дії, які мають бути виконані під час створення ІС.

Кожен процес поділяється на набір дій, кожна дія на набір завдань. Кожен процес, дія або завдання ініціюється і виконується іншим процесом в міру необхідності, причому немає наперед визначених послідовностей виконання. Зв'язки за вхідними даними при цьому зберігаються.

**Процеси життєвого циклу ІС**

***Основні:***

1. Придбання (дії і завдання замовника, що купує ІС).

2. Постачання (дії і завдання постачальника, який забезпечує замовника програмним продуктом або послугою).

3. Розробка (дії і завдання, що виконуються розробником: створення ПЗ, оформлення проектної та експлуатаційної документації, підготовка тестових і навчальних матеріалів).

4. Експлуатація (дії і завдання оператора організації, що експлуатує систему).

5. Супровід (дії і завдання, що виконуються супроводжуючою організацією, тобто службою супроводу). Супровід внесень змін до ПЗ для виправлення помилок, підвищення продуктивності або адаптації до умов, що змінилися, роботи або вимогам.

***Допоміжні:***

1. Документування (формалізований опис інформації, створеної протягом ЖЦ ІС)

2. Управління конфігурацією (застосування адміністративних і технічних процедур протягом ЖЦ ІС для визначення стану компонентів ІС, управління її модифікаціями).

3. Забезпечення якості (забезпечення гарантій того, що ІС і процеси її ЖЦ відповідають заданим вимогам і затвердженим планам).

4. Верифікація (визначення того, що програмні продукти, які є результатами певної дії, повністю відповідають вимогам або умовам, зумовленим попередніми діями).

5. Атестація (визначення повноти відповідності заданих вимог і створеної системи їх конкретному функціональному призначенню).

6. Загальна оцінка (оцінка стану робіт за проектом: контроль планування й управління ресурсами, персоналом, апаратурою, інструментальними засобами).

7. Аудит (визначення відповідності вимогам, планам і умовам договору).

8. Вирішення проблем (аналіз і вирішення проблем, незалежно від їх походження або джерела, які виявлені під час розробки, експлуатації, супроводу або інших процесів).

***Організаційні:***

1. Управління (дії і завдання, які можуть виконуватися будь-якою стороною, що управляє своїми процесами).

2. Створення інфраструктури (вибір і супровід технології, стандартів та інструментальних засобів, вибір та установка апаратних і програмних засобів, що використовуються для розробки, експлуатації або супроводу ПЗ).

3. Удосконалення (оцінка, вимірювання, контроль і удосконалення процесів ЖЦ).

4. Навчання (початкове навчання і подальше постійне підвищення кваліфікації персоналу).

Кожен процес включає низку дій. Наприклад, процес придбання охоплює такі дії:

- ініціація придбання;
- підготовка заявочних пропозицій;
- підготовка і коректування договору;
- нагляд за діяльністю постачальника;
- приймання і завершення робіт.

Кожна дія включає низку завдань. Наприклад, підготовка заявочних пропозицій має передбачати:

- формування вимог до системи;
- формування списку програмних продуктів;
- встановлення умов і угод;

о опис технічних обмежень;

о стадії життєвого циклу ІС, взаємозв'язок між процесами і стадіями.

***Модель*** життєвого циклу ІС - структура, що визначає послідовність виконання і взаємозв'язку процесів, дій і завдань впродовж життєвого циклу. Модель життєвого циклу залежить від специфіки, масштабу і складності проекту і специфіки умов, у яких система створюється і функціонує.

Модель ЖЦ ІС включає стадії, результати виконання робіт на кожній стадії, ключові події точки завершення робіт і прийняття рішень.

***Стадія*** - це частина процесу створення ІС, обмежена певними часовими рамками, що закінчується випуском конкретного продукту (моделей, програмних компонентів, документації) і визначається заданими для цієї стадії вимогами.

**Етапи створення ІС**

1. ***Формування вимог до ІС.*** На цьому етапі провадиться обстеження об'єкта та обґрунтовується необхідність створення ІС, формулюються вимоги користувача до ІС, оформляються звіти про виконану роботу.

Під час обстеження об'єкта перевіряються документообіг (у тому числі кількість документів та їх обсяг за певний період часу), форми початкових та вихідних документів, методики розрахунку окремих показників. Обстеження має виявити проблеми, які можна розв'язати засобами обчислювальної техніки, щоб оцінити доцільність створення ІС.

Обстеження провадиться за допомогою бесід та консультацій із працівниками установи, для якої буде створюватись інформаційна система. Спочатку із замовником погоджуються вимоги до ІС. Вимоги включають суми максимальних витрат та термін виконання розробки, умови функціонування системи, перелік функцій, які система має забезпечити, тощо.

Звіт про обстеження складається у довільній формі. На його підставі надалі розроблятиметься технічний проект, тому бажано в додатках до звіту навести форми використовуваних документів. У ньому також необхідно викласти погоджені із замовником методики розрахунку економічних показників.

Вимоги до системи можуть бути оформлені як окремий документ, а саме заявка на розроблення або технічне завдання.

2. ***Розроблення концепції ІС.*** Під час розроблення концепції ІС провадяться науково-дослідні роботи для пошуку шляхів та оцінки можливостей реалізації вимог користувача. На цьому етапі можна визначити методи, які будуть покладені в основу розрахунків, або принципові підходи до розв'язування конкретних задач. Наприклад, для ІС, що пов'язана з оптимальним плануванням виробництва, на цьому етапі можуть визначатися математичні моделі та методи (лінійне програмування, імітаційне моделювання тощо) для використання в розрахунках і стандартні пакети програмних засобів, які можна буде використати.

Цей етап закінчується складанням і затвердженням звіту про науково-дослідну роботу, який містить оцінку ресурсів, необхідних для реалізації розробки ІС, дає порівняльну характеристику різних варіантів розробки ІС, визначає порядок оцінювання якості системи.

3. ***Технічне завдання.*** Формується технічне завдання (ТЗ) на створення ІС - основний документ, що визначає вимоги та порядок створення ІС. На підставі ТЗ провадиться розроблення ІС, її приймання під час введення в дію. ТЗ розробляють на систему в цілому. Додатково можуть бути розроблені ТЗ на окремі частини ІС.

4. ***Ескізний проект. Розробляються*** попередні проектні рішення щодо всієї ІС або її частин. Може бути визначений перелік задач, які будуть розв'язуватися в системі, концепція інформаційної бази, що створюється (інфологічна модель), функції та параметри основних програмних засобів. Для кожної задачі в ескізному

проекті можуть бути наведені погоджені із замовником форми первинних та вихідних документів, структури інформаційних масивів або їх перелік, основні алгоритми обробки інформації.

5. ***Технічний проект.*** Розробляються проектні рішення щодо системи та її частин, документація на ІС та на постачання виробів для комплектації ІС. Проектні рішення за системою та її частинами визначають її організаційну структуру, функції персоналу в ІС, структуру технічних засобів, мови програмування або СУБД, наводять загальні характеристики ПЗ, систем класифікації та кодування (зокрема визначаються загальнодержавні або галузеві класифікатори, які необхідно використовувати), визначають варіанти ведення БД.

6. ***Робоча документація.*** Створюються проектні документи, які визначаються державними стандартами, постановка задачі, алгоритм її розв'язання, описується інформаційне забезпечення (організація інформаційної бази, системи класифікації та кодування, інформаційні масиви), організаційне, технічне та програмне забезпечення. Усі ці проектні документи можуть оформлюватися як окремі документи, а можуть входити у технічний проект як окремі розділи.

Документація на постачання виробів для комплектації ІС складається тоді, коли в установі не використовувалися засоби обчислювальної техніки або цих засобів недостатньо. У такій документації, яка складається у довільній формі, обґрунтовується закупівля тих чи інших засобів та наводяться їх можливі закупівельні ціни. Так, вибираються комплектуючі частини для ПЕОМ: обсяг оперативної пам'яті, ємність магнітного диска, характеристики принтера тощо.

Технічне завдання на розроблення технічних засобів необхідне лише тоді, коли для обробки інформації потрібне нестандартне обладнання, яке не випускається промисловістю. Наприклад, для створення автоматизованої системи для обліку роботи депутатів Верховної Ради були замовлені спеціальні пристрої для реєстрації депутатів та голосування, а також спеціальні табло, де відображуються результати голосування та інша інформація.

Розроблення завдань на проектування в суміжних частинах проекту виконується тоді, коли для впровадження інформаційної системи необхідно виконати ряд підготовчих робіт, приміром, пов'язаних із електротехнічними роботами.

Під час створення робочого проекту формуються документи, які визначають стандарт для цього етапу проектування, та розробляються або адаптуються програми обробки інформації. Серед документів робочого проекту можуть бути загальний опис системи, опис технологічного процесу обробки інформації, інструкції з виконання окремих операцій технологічного процесу, керівництво користувача, опис програм тощо.

7. ***Введення в експлуатацію.*** Найважливішою роботою під час створення робочого проекту є розроблення та налагодження програм, або їх адаптація. Адаптація відбувається тоді, коли для створення інформаційної системи використовуються вже готові програми: типові чи ті, які розроблялися для інших об'єктів. Для кожної програми розробляються її опис або паспорт. Якщо програми адаптовані, то можуть бути описані тільки зміни, які були внесені до програм. На етапі введення в експлуатацію необхідно виконати такий обсяг робіт: підготувати об'єкт до введення в експлуатацію; скомплектувати ІС, встановивши технічні та програмні засоби; виконати будівельно-монтажні роботи; провести попередні випробування системи; виконати дослідну експлуатацію системи та провести приймальні іспити. Підготовка об'єкта до автоматизації починається з видання наказу про зміни у структурі об'єкта, документообігу, розподілі обов'язків між персоналом, переході на нову технологію обробки інформації. Такий наказ видається у довільній формі, але в ньому обов'язково вказуються термін переходу до нової технології та особи, які відповідають за впровадження й експлуатацію інформаційної системи. Для

підготовки об'єкта можуть тиражуватися різноманітні посадові інструкції, бланки нових документів, готуватись класифікатори тощо.

На цьому етапі дуже важливо підготувати персонал до роботи в інформаційній системі. Підготовка персоналу може провадитися силами розробників системи (лекції, семінари, практичні заняття) або з допомогою спеціальних курсів чи факультетів підвищення кваліфікації. Під час такого навчання кожний працівник повинен не тільки опанувати зміни у своїх посадових обов'язках, а й навчитися роботі з обчислювальною технікою. Паралельно з підготовкою персоналу провадяться роботи з установлення технічних та програмних засобів. Визначаються місця встановлення ЕОМ, засоби їх охорони, особи, відповідальні за збереження та супроводження системного програмного забезпечення, встановлюються необхідні пакети програм. У разі потреби виконуються будівельно-монтажні роботи, пов'язані з прокладанням кабелів, встановленням унікального обладнання, зміною освітлення робочих місць.

Попередні випробування системи виконує розробник, щоб перевірити коректність роботи технічних і програмних засобів, можливість використання ПЗ. Під час дослідної експлуатації заповнюють інформаційну базу на машинних носіях. Це роблять спеціалісти, які експлуатуватимуть ІС. На основі контрольного прикладу або реальних даних за конкретний період рення до припинення функціонування. Ці етапи включають такі фази: передпроектну, логічне і технічне проектування - розробка відповідно до сформульованих вимог і виявлених інформаційних потреб системної і функціональної архітектури ІС, робоче проектування та саму експлуатацію, спочатку дослідну, а потім промислову. Базові напрями, що дають змогу описати бізнес-процеси підприємства: IDEF - структурний підхід та UML - об'єктно орієнтований підхід.

8. ***Супроводження ІС.*** На цьому етапі виконуються роботи згідно з гарантійними зобов'язаннями розробника. У цей період можуть усуватися недоліки, які виявляються під час експлуатації.

**Документація на розроблення ІС**

В Україні розроблення ІС здійснюється відповідно до таких нормативних документів (табл. 3.1).

***Таблиця*** 3.1. Перелік нормативних документів

| Стандарт | Назва |
|---|---|
| ДСТУ 2844-94 | Програмні засоби ЕОМ. Забезпечення якості. Терміни та визначення |
| ДСТУ 2850-94 | Програмні засоби ЕОМ. Показники і методи оцінювання якості |
| ДСТУ 2851-94 | Програмні засоби ЕОМ. Документування результатів випробувань |
| ДСТУ 2853-94 | Програмні засоби ЕОМ. Підготовлення і проведення випробувань |
| ДСТУ 2873-94 | Системи оброблення інформації. Програмування. Терміни та визначення |
| ДСТУ 2941-94 | Системи оброблення інформації. Розроблення систем. Терміни та визначення |
| ДСТУ 180 9000-98 | Стандарти з управління якістю та забезпечення якості |
| ДСТУ 3918-99 (І80/IEC 12207-1995) | Інформаційні технології. Процеси життєвого циклу програмного забезпечення |
| ДСТУ 3919-99 (ШО/IEC 14102:1995) | Інформаційні технології. ***Основні*** напрями оцінювання та відбору САвЕ-інс-трументів |

| | |
|---|---|
| ДСТУ ІвО 9001-2001 | Системи управління якістю. Вимоги |
| ДСТУ ІБО 9004-2001 | Системи управління якістю. Настанови щодо поліпшення діяльності |

В інформаційному суспільстві розроблення програмного забезпечення ІТ стало масовою діяльністю. За таких обставин світове суспільство прийшло до висновку, що технологія виробництва програм потребує свого оформлення у вигляді самостійного інженерного фаху, який мусить забезпечити у світі відповідний кадровий потенціал для постійно зростаючого обсягу програмних розробок. Розроблення ІС визначається як інженерна діяльність.

Виникнення програмної інженерії визначено кількома факторами: появою різноманітних складних методів аналізу та моделювання ПрО; великою кількістю помилок у ПЗ; потребою в організації роботи великих колективів розробників ПЗ; необхідністю використання високотехнологічних засобів керування розробкою ПЗ.

Життєвий цикл ІС - сукупність етапів, які проходить ІС у своєму розвитку від моменту прийняття рішення про її ствотичну, прогнозну тощо. Інформацію для економічного аналізу поділяють на кілька типів: факти, оцінки, прогнози, узагальнені зв'язки, конфіденційна інформація, чутки тощо.

Схему перетворення інформації в дані можна представити через процедури класифікації, кодування та моделювання елементів даних. Метою штрихового кодування є відображення основних інформаційних характеристик товару в штрихкодах, що забезпечує можливість простежити за рухом товару до споживача. При розв'язуванні економічних задач забезпечується їх порівнянність через Єдину систему класифікації та кодування техніко-економічної інформації, комплексу взаємопов'язаних класифікаторів техніко-економічної інформації, що пристосовані до безпосередньої обробки засобами ІКТ.

рення до припинення функціонування. Ці етапи включають такі фази: передпроектну, логічне і технічне проектування - розробка відповідно до сформульованих вимог і виявлених інформаційних потреб системної і функціональної архітектури ІС, робоче проектування та саму експлуатацію, спочатку дослідну, а потім промислову. Базові напрями, що дають змогу описати бізнес-процеси підприємства: IDEP - структурний підхід та UML - об'єктно орієнтований підхід.

## Рекомендована література

1. Хомоненко А.Д., Цыганков В.М., Мальцев М.Г. Базы данных/Учебник для высших учебных заведений. – СПб: Корона, 2004. – 736с.

2. Пасічник В.В., Резніченко В.А. Організація баз даних та знань/Підручник для ВНЗ. – К.: Видавнича група BHV, 2006. – 384 с.

3. Гайна Г.А. Основи проектування баз даних/Навчальний посібник для ВНЗ. – К: КНУБА, 2005. – 204 с.

4. Кирилов В., Громов Г. Введение в реляционные базы данных/Учебное пособие для высших учебных заведений. – СПб: BHV-Петербург, 2009. – 454с.

5. Гудов А.М., Шмакова Л.Е. Введение в язык структурированных запросов SQL/Учебное пособие. – Кемерово: Кемеровский госуниверситет, 2001.- 118с.

6. К.Дж. Дейт. Введение в системы баз данных. – М: Вильямс, 2005. - 1328с.

7. М. Грабер. Введение в SQL. – М: Лори, 1996. – 375с.

**Зміст**