## Зміст

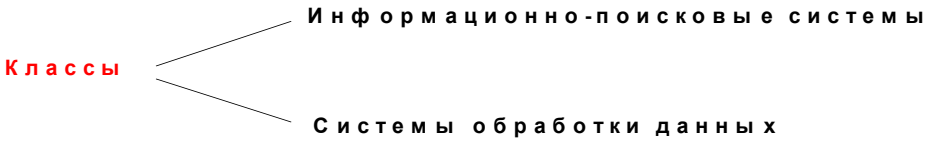
## Лекція 1. Введення в інформаційні системи та бази даних.

Області використання обчислювальної техніки:

- Виконання численних розрахунків
- Автоматичні чи автоматизовані інформаційні системи

**Інформаційна система (ІС) –** це сукупність структурованих даних і комплексу програмно-апаратних засобів для зберігання даних і маніпулювання ними.

**Класифікація інформаційних систем:**





**Клас 1. Інформаційно-пошукові системи**

- орієнтація на пошук даних із загальної множини по певному пошуковому критерію;
- користувача цікавить в більшій мірі інформація ,а не технологія обробки цих даних;

**Клас 2. Системи обробки даних**

- орієнтація на обробку даних;
- користувача цікавить результат обробки даних, а не самі дані;
- вивід інформації не обов'язковий;

**Вид 1. Фактографічні системи**

- реєстрація конкретних значень даних об`єктів реального світу;
- інформація має чітку структуру (формат);
- однозначні відповіді на поставлені запитання;

**Вид 2. Документальні системи**

- сукупність неструктурованих документів (текстових і графічних);
- нема однозначних відповідей на поставлені запитання. Результат – список документів чи об`єктів, в певній мірі задовольняючих сформованим в запиті умовам.

**Термінологія**

**Об`єкт** – це щось існуюче і існує спосіб відрізнити один подібний об`єкт від іншого.

**Дане** – це певний показник, який характеризує об`єкт і приймає для конкретного екземпляра цього об`єкта певне значення.

**Структура даних**

**Структурування інформації** – це введення яких-небудь погоджень щодо способів представлення даних (встановлення формату, тобто певного типу, довжини значень даних).

**Приклад неструктурованих даних:**

«Табельний номер 1234 Іванов Іван Іванович, дата народження 10 травня 1967 року.
Номер по табелю Петрова Сергія Олександровича №8191, д.н. 18.10.1972 г.
Табель №3451 Сидорова Алексія Петровича, народженого 5 липня 1964 року.»

**Пример структурованих даних:**

| Табельный номер | Фамилия | Имя | Отчество | Дата рождения |
|---|---|---|---|---|
| 1234 | Иванов | Иван | Иванович | 10.05.1967 |
| 8191 | Петров | Сергей | Александрович | 18.10.1972 |
| 3451 | Сидоров | Алексей | Петрович | 05.07.1964 |

**Традиційний підхід організації даних.**

**Файли та файлові системи**

**Файл** – це іменована область зовнішньої пам`яті, в котру можна записувати і із якої можна зчитувати дані.

З точки зору користувача файл містить лінійну послідовність записів.

**Стандартні операції над файлами:**

- створити файл (потрібного типу та розміру);
- відкрити раніше створений файл;

- прочитати із файлу певний запис (поточний, наступний, попередній, перший, останній);
- записати в файл на місце поточного запису новий, додати новий запис в кінець файлу.

**Типи файлів:**

**Файл послідовного доступу –** представляє собою послідовність записів даних у вигляді рядків довільної довжини, розділених комами або спеціальними символами, що позначають перехід на новий рядок.

Особливості:

– відсутність можливості впорядкувати збережені записи;

– Розміщення і вилучення записів в такий файл відбувається по рядкам в певній послідовності.

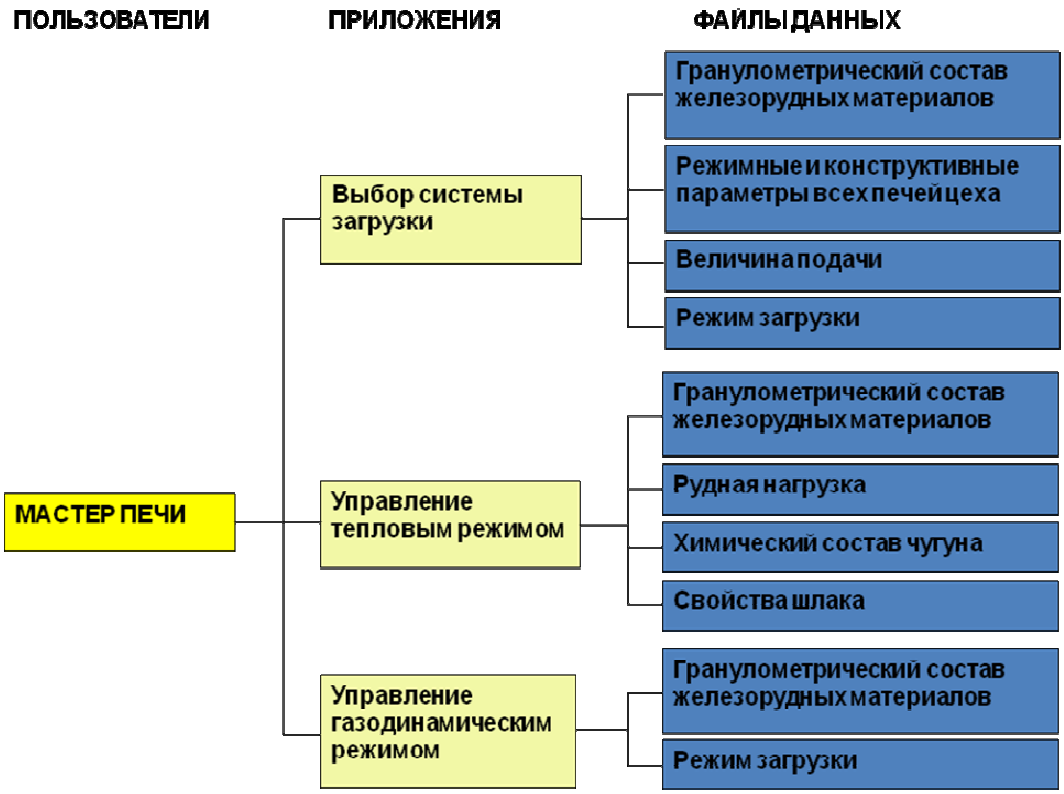
**Файл довільного доступу** – складається із записів фіксованої довжини, яка вказується при його створені.

Особливості:

– всі записи упорядковані, кожна має свій номер;

– можливість швидкого переміщення на будь-який запис, минаючи попередні.

**Приклад традиційного підходу до організації даних**




**Недоліки традиційного підходу**

- Збитковість даних
- Проблеми несуперечливості даних
- Обмежена доступність даних
- Складності в організації та керуванні
- Недостатня кількість засобів захисту даних, що зберігаються
- Низькопродуктивна робота в багатокористувацькому середовищі
- Відсутність процедур відновлення даних після виникнення відмов;
- Відсутність засобів маніпулювання даними;
- Висока вартість програмування та супроводження;
- Відсутність гнучкості до змін та ін.

**Організація даних з використання БД**

**Дані –** це будь-яка інформація про об`єкти оточуючого світу, представлена в формалізованому вигляді, придатному для її передачі, збереження і обробки за допомогою деякого процесу та з використанням засобів обчислювальної техніки.

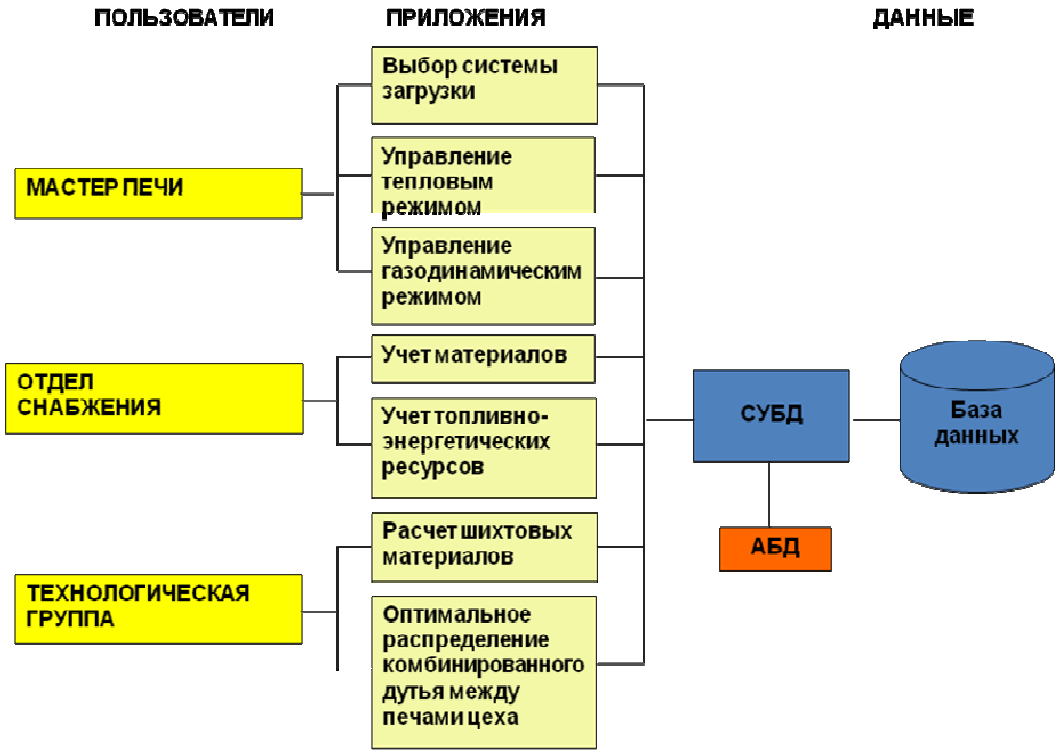
**База даних (БД) –** це іменована сукупність даних, організованих по певним правилам, що передбачає загальні принципи опису, зберігання і маніпулювання даними, не залежна від прикладних програм.

**Предметна область –** це частина реального світу, що підлягає вивченню для організації управління і, в кінцевому рахунку, автоматизації.

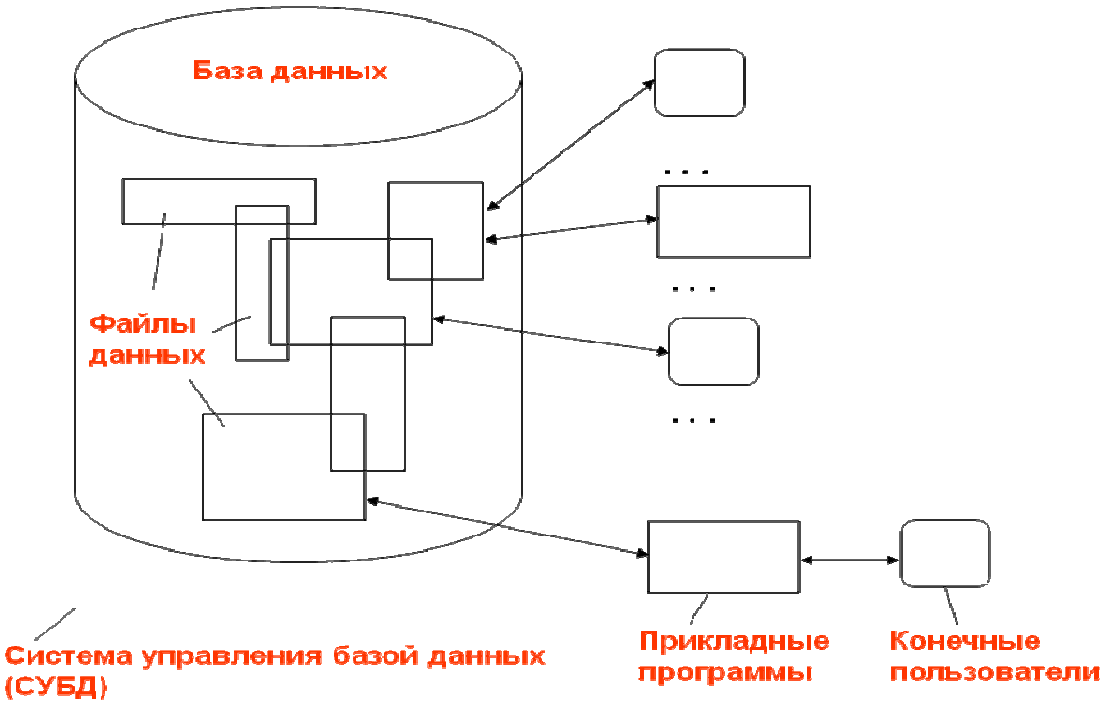
**Система керування базами даних (СКБД) –** це сукупність мовних та програмних засобів, призначених для створення, ведення і спільного використання БД багатьма користувачами.

**Банк даних (БнД) –** це система спеціальним чином організованих даних – баз даних, програмних, технічних, мовних, організаційно-методичних засобів, призначених для забезпечення централізованого накопичення і колективного багатоцільового використання даних.

**Пример організації баз даних**



**Компоненти систем баз даних**




**Основні функції СКБД**

- Безпосереднє керування даними в зовнішній пам`яті
- Керування буферами оперативної пам`яті
- Керування транзакціями
- Журналізація
- Підтримка мов БД

**Мовні засоби СКБД:**

- DDL – Data Definition Language, Мова Визначення Даних, МВД
- DML – Data Management Language, Мова Маніпулювання Даними, ММД
- Мова запитів (SQL, Structured Query Language)

**Програмні компоненти СКБД**

- Ядро
- Контролер словника
- Контролер БД
- Контролер файлів
- Компілятор
- Процесор запитів

- Препроцесор мови DML
- Компілятор мови DDL

**Категорії користувачів БД**

- Кінцеві користувачі
- Прикладні програмісти
- Адміністратори даних (АД), адміністратори бази даних (АБД),

**Основні функції адміністратора БД**

- Аналіз предметної області
- Проектування структури БД.
- Задання обмежень цілісності при описанні структури БД і процедур обробки БД.
- Первісне завантаження і ведення БД.
- Захист даних.
- Забезпечення відновлення БД.
- Аналіз звернень користувачів БД.
- Аналіз ефективності функціонування БД.
- Робота з кінцевими користувачами.
- Підготовка і підтримка системних засобів.

**Переваги і недоліки сучасного підходу до організації даних**

**Переваги:**

- Скорочення збитковості даних.
- Усунення суперечливих даних.
- Загальний доступ до даних.
- Дотримання стандартів.
- Введення обмежень для забезпечення безпеки.
- Забезпечення цілісності даних.

**Недоліки:**

- Втрата користувачами права одноосібного володіння даними.
- Підвищення ймовірності порушення захисту даних
- Підвищена загроза секретності даних, що зберігаються.

**Вимоги до сучасної СКБД**:

- Ефективне виконання різноманітних функції предметної області
- Мінімалізація і контроль збитковості даних, що зберігаються
- Надання для прийняття рішень несуперечливої (погодженої) інформації
- Забезпечення можливості одночасного доступу до бази даних декількох уповноважених користувачів
- Забезпечення керування безпекою
- Проста фізична реорганізація тобто. зміна структури даних в базі даних
- Можливість централізованого керування базою даних

**Класифікація систем баз даних**

**По характеру використання:**

**Однокористувацькі системи** – це системи, в яких в один і той же час до бази даних може отримати доступ не більше одного користувача, так звані базі даних з локальним доступом

**Багатокористувацькі системи** – це системи, в яких до бази даних можуть отримати доступ одночасно декілька користувачів, так звані бази даних з віддаленим (мережевим) доступом

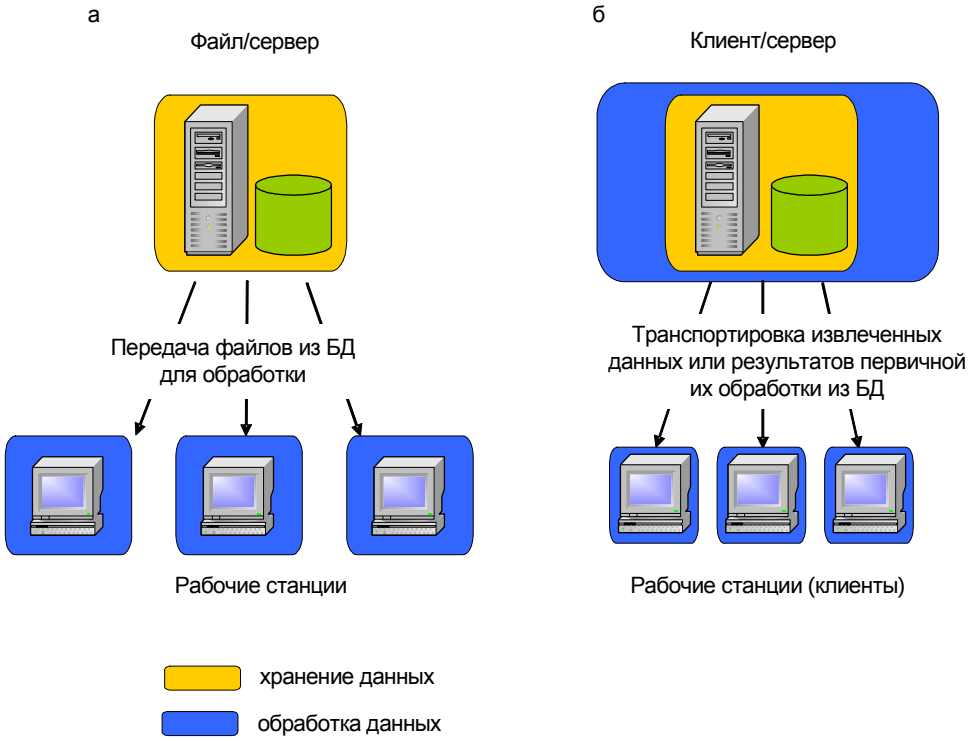
**По технології обробки даних:**

**Централізовані системи** – база даних фізично зберігається в пам`яті одного комп`ютера.

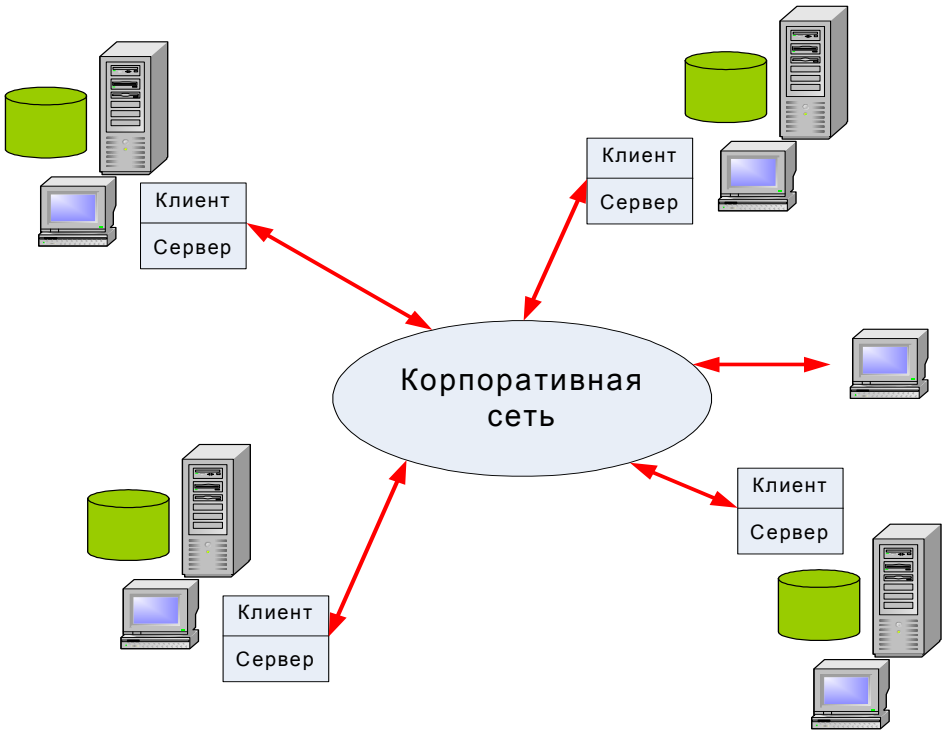
**Розподілена система баз даних** – складається з декількох, можливо, пересічних або навіть дублюючих одна одну частин БД, що зберігаються на різних комп`ютерах – серверах, які в загальному випадку можуть бути віддалені географічно одна від одної на значні відстані, тобто територіально розподілені.

## Архітектура побудови централізованих систем

Схема обробки інформації в БД по принципу файл/сервер (а) и клієнт/сервер (б)




## Фрагмент архітектури побудови розподілених систем



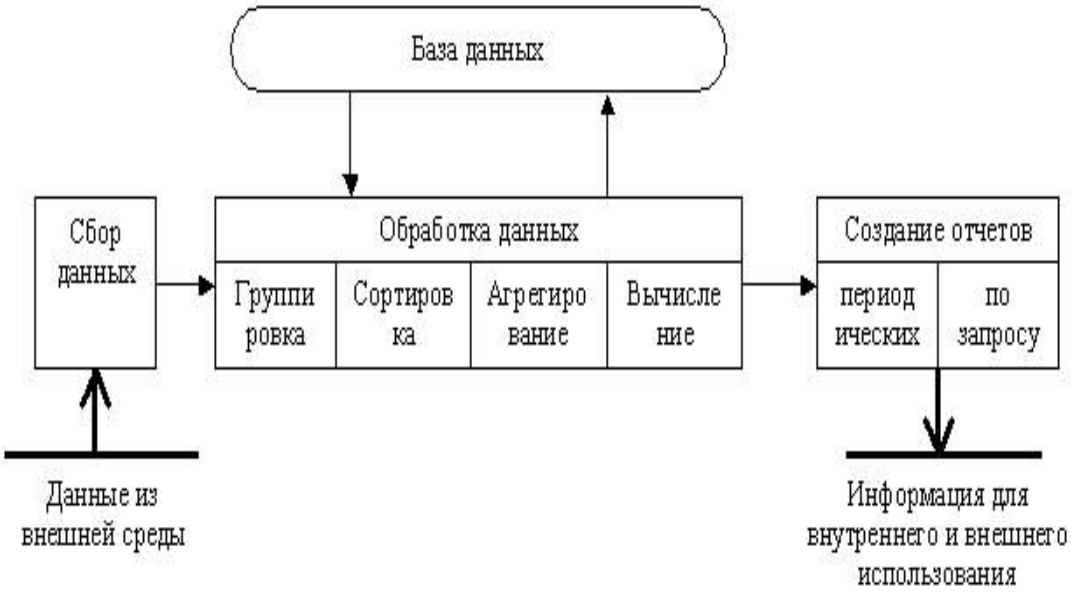
**Інформаційні системи, які використовують БД**

**Системи обробки даних**

- рішення тільки добре структурованих задач, для яких можливо розробити алгоритм;
- – виконання основного об`єму роботи в автоматичному режимі з мінімальною участю людини;
- – використання деталізованих даних. Записи щодо діяльності фірми мають детальний характер, що допускають проведення ревізій. В процесі ревізії діяльність фірми перевіряється хронологічно від початку періоду до його кінця і від кінця до початку;
- – акцент на хронологію подій;
- Вимога мінімальної допомоги у вирішенні проблем зі сторони спеціалістів других рівнів.

Функції вводу, вибірки, корекції інформації без використання методів оптимізації з використанням оперативної обробки транзакцій – OLTP (OnLine Transaction Processing)

Приклад: бухгалтерська програма 1С



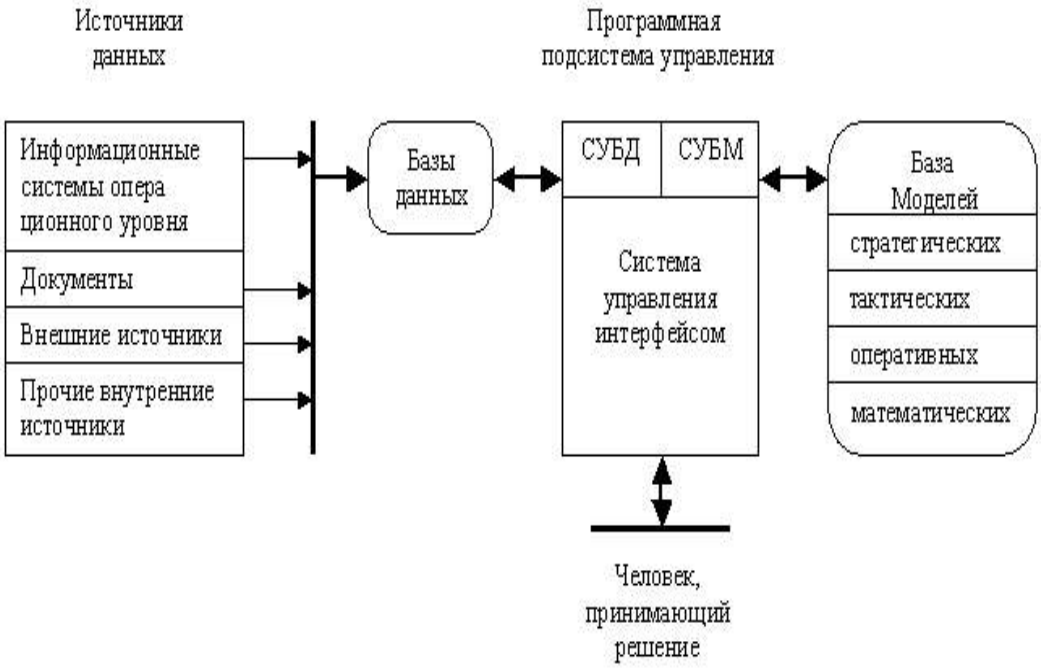

**Інформаційні системи керування**

- середньострокове планування,
- аналіз і організація робіт на протязі декількох тижнів (місяців).
- регламентованість (періодична повторюваність) формування результуючих документів і чітко визначений алгоритм вирішення задач. Вирішення подібних задач призначено для керівників

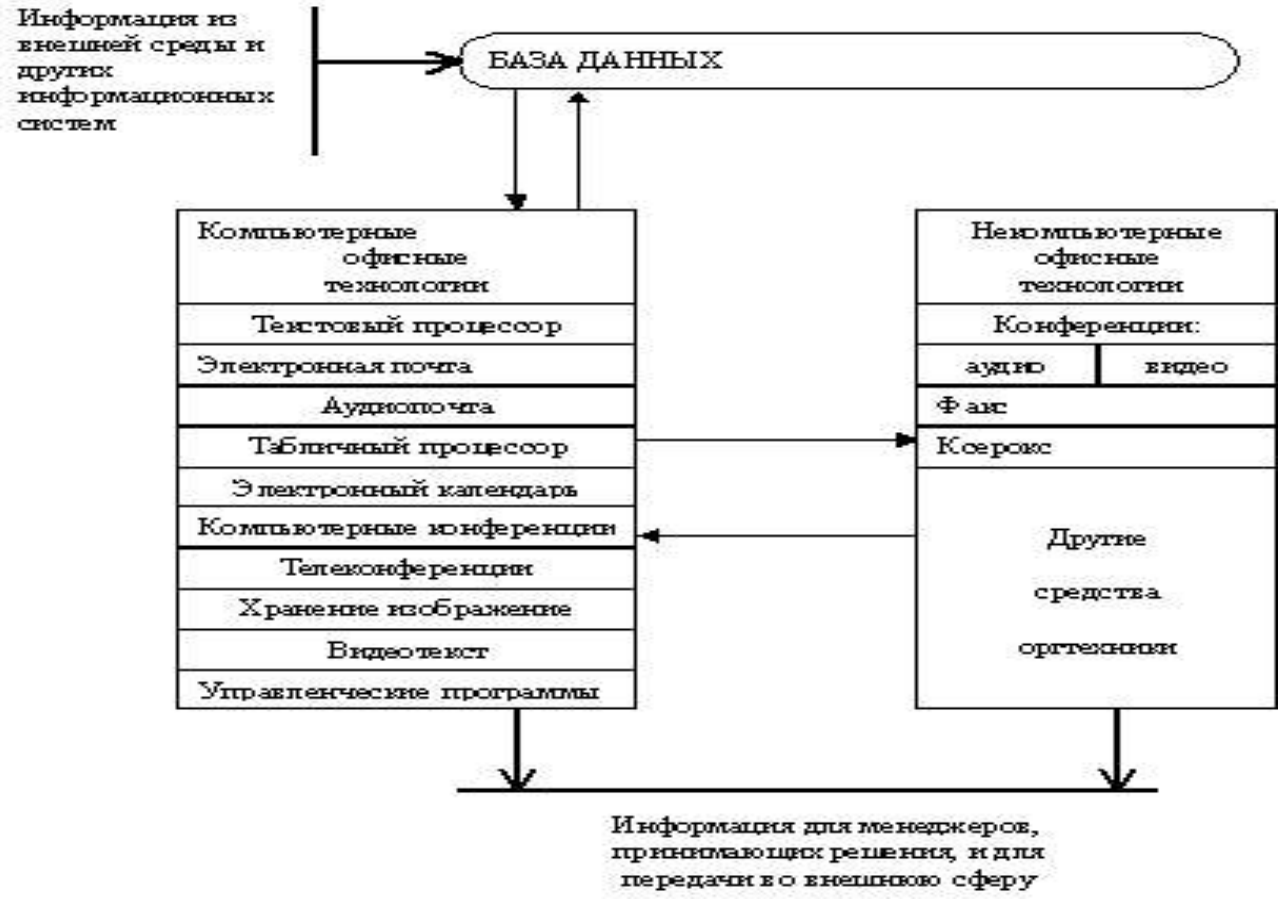
різноманітних служб підприємств (відділів матеріально-технічного забезпечення і збуту, цехів і т.д.). Задачі вирішуються на основі накопичених в базі оперативних даних


**Системи підтримки прийняття рішень**

- вилучення даних із різноманітних джерел, включаючи неструктуровану інформацію;
- багатовимірний аналіз даних;
- обробка статистики;
- моделювання правил і стратегій ділової діяльності;
- ділові графіки для представлення результатів аналізу;
- аналіз "що якщо";
- штучний інтелект.
- оперативно аналітична обробка – OLAP (OnLine Analysis Processing)
- сховища даних

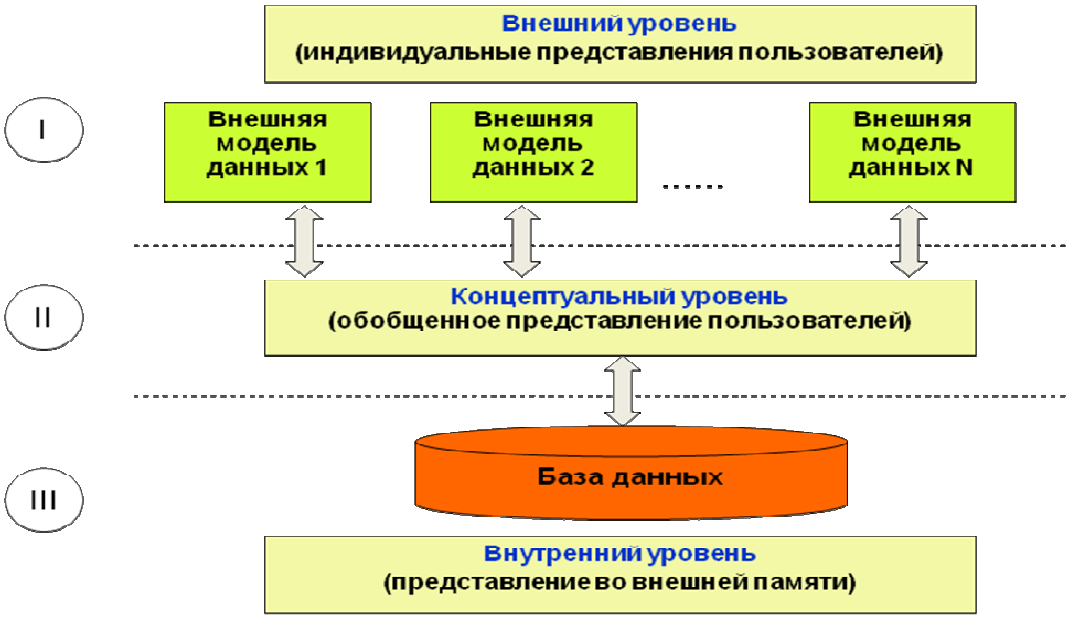



**Автоматизація офісу**



## Лекція 2. Моделі даних

### Архітектура бази даних, запропонована ANSI



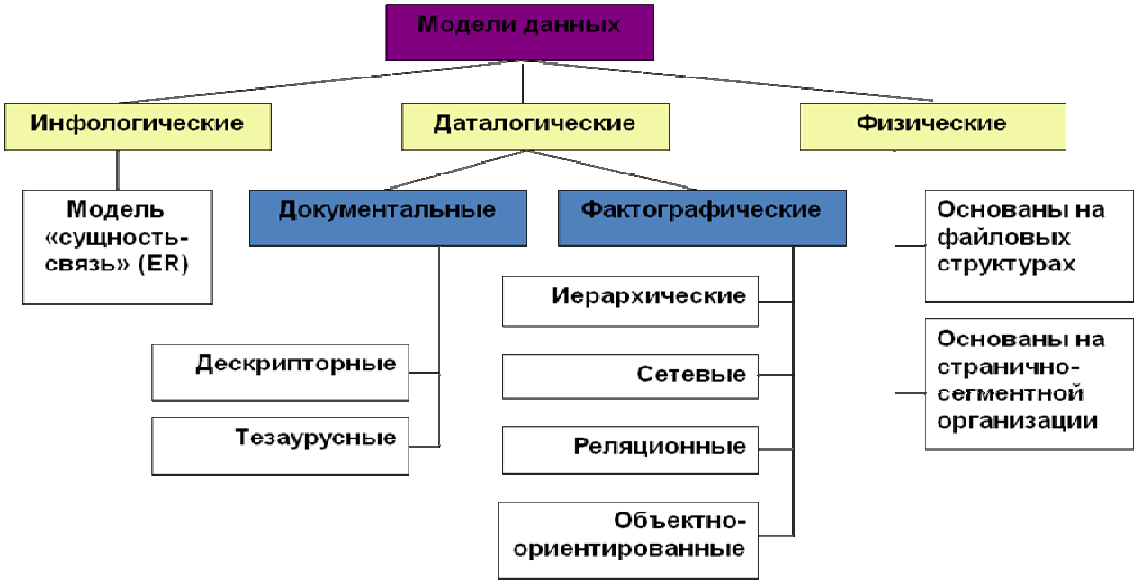

**Визначення:**

- **дані** — це набір конкретних значень, параметрів, що характеризують об`єкт, умови, ситуацію або будь-які інші фактори.
- **модель даних** це логічне представлення даних і сукупність операцій над ними.
- **аспект структури**: методи опису типів і логічних структур даних;
- **аспект маніпуляції**: методи маніпулювання даними;
- **аспект цілісності**: методи опису і підтримки цілісності даних.

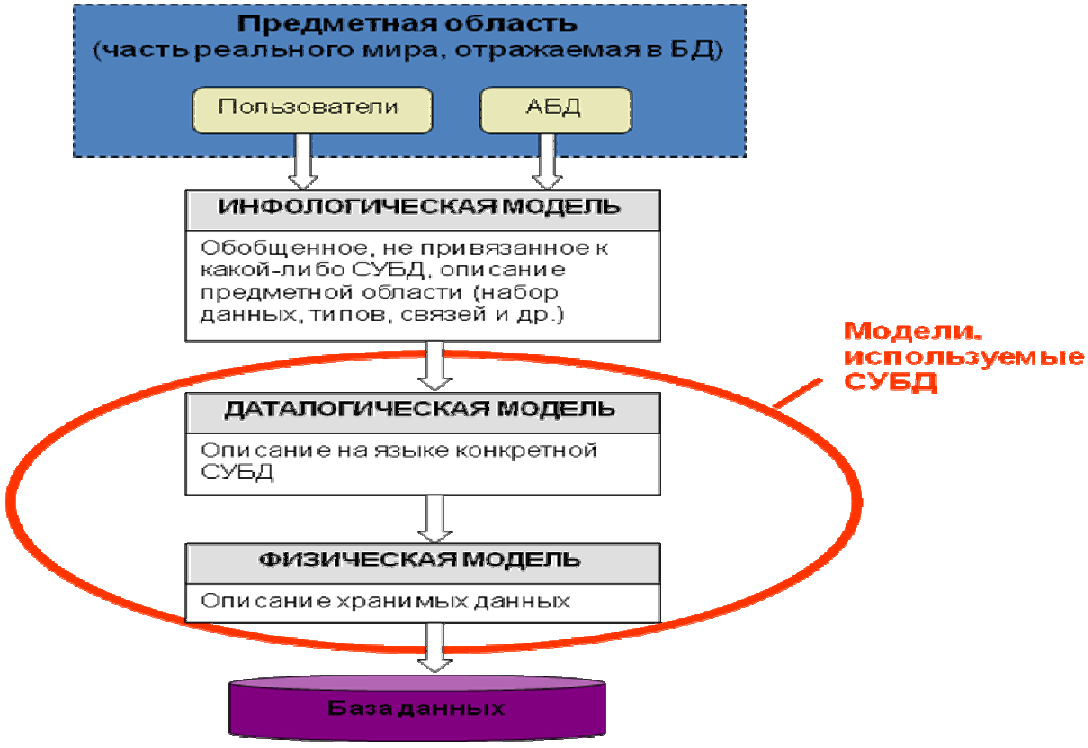
**Модель даних:**

- Набору типів структур даних.
- Набору операторів або правил виводу, які можуть бути застосовані до будь-яких вірних прикладів типів даних, перерахованих в (1), для того, щоб знаходити, виводити або перетворювати інформацію, що міститься в будь-яких частинах цих структур в будь-яких комбінаціях.
- Набору загальних правил цілісності, які прямо або побічно визначають множину несуперечливих станів БД і/або множину змін її станів.

**Класифікація моделей даних**




**Рівні моделей даних (послідовність розробки БД)**



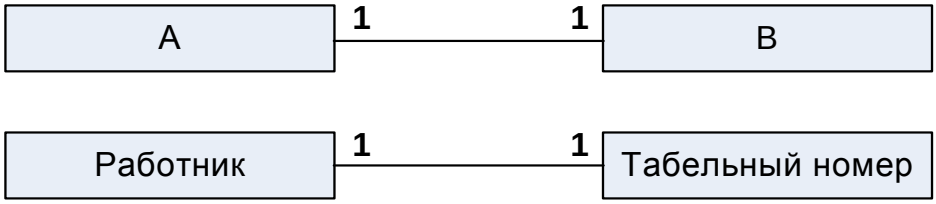

**Взаємозв`язок в моделях даних**

**Зв`язок** – це асоціювання двох або більше об`єктів.

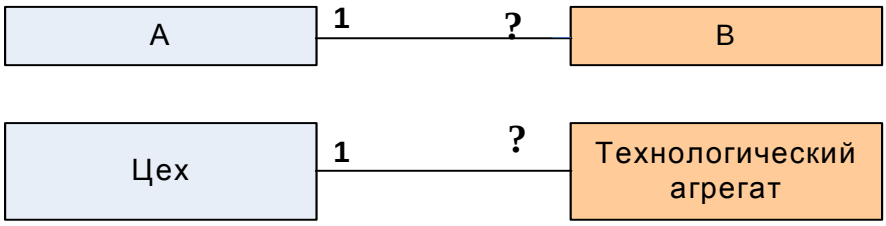
**Основне призначення зв`язків** – е можливість організації пошуку даних в базі даних

**Типи зв`язків**

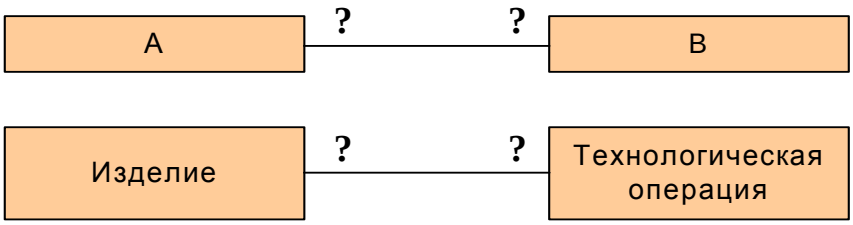
- один до одного;




- один до багатьох;




- багато до багатьох.∞

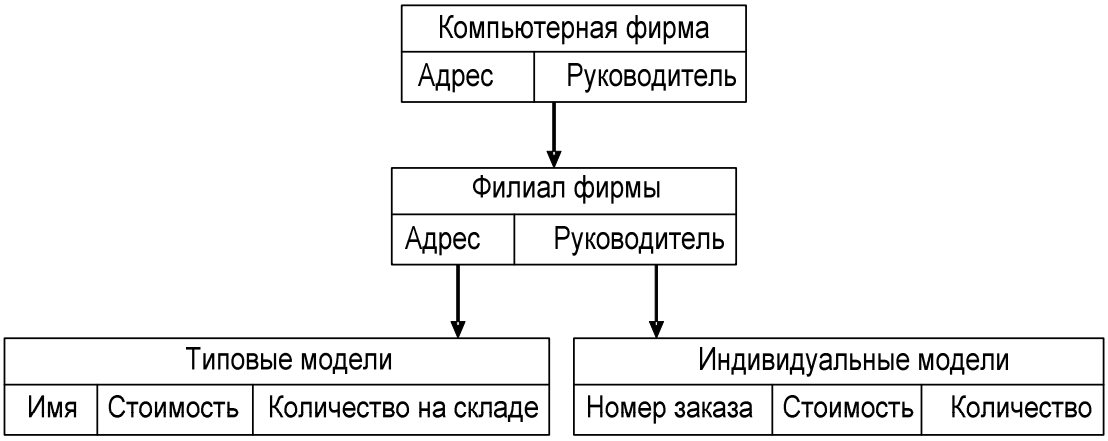



**Характеристики моделей даних**

| **Модель даних** | **Характер зв`язку між об`єктами** | **Формальне представлення** |
|---|---|---|
| Ієрархічна | Жорсткі зв`язки | Деревовидна структура |
| Мережева | Напівжорсткі зв`язки | Довільний граф |
| Реляційна | Перемінні зв`язки | Плоский файл |

**Ієрархічна модель даних**

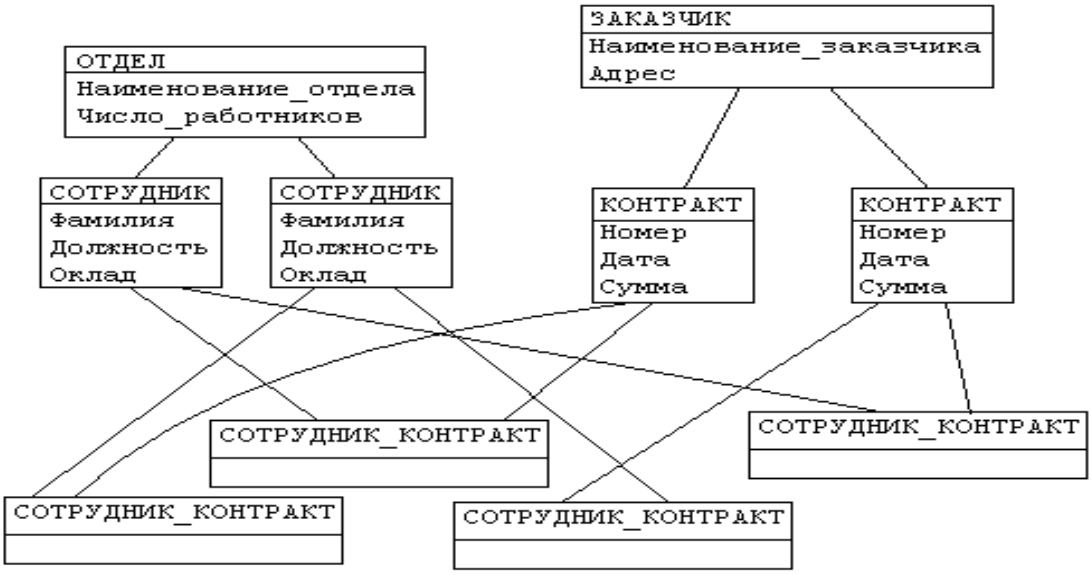
Приклад структури ієрархічної БД:




**Мережева модель даних**

Типи структур даних


Приклад структури мережевої БД:



**Переваги до реляційних моделей даних**

- Розвиток засобів керування даними в зовнішній пам`яті на низькому рівні
- Можливість побудови вручну ефективних прикладних систем;

**Недоліки до реляційних моделей даних**

- Занадто складно користуватись
- Фактично необхідні знання щодо фізичної організації
- Прикладні системи залежать від цієї організації (логічна і фізична залежності даних);
- Логіка побудови додатків перевантажена деталями організації доступу до БД

## Лекція 3. Реляційна модель даних

### Визначення та позначення

**D –** домен. Потенційна множина значень «простого» типу даних

$D_1$ x $D_2$ x … $D_n$ – повний декартовий добуток – множина виду

$d_1$ x $d_2$ x .. $d_n$

де $d_1 \subseteq D_1$, … $d_n \subseteq D_n$

Приклад

$D_1$ = {Іванов, Крилов, Степанов};

$D_2$ = (Теорія автоматів, Бази даних)

$D_3$ = {3, 4, 5}

**Приклад:** повний декартовий добуток множин $D_1$ x $D_2$ x $D_3$

| | | |
|---|---|---|
| Іванов | Теорія автоматів | **3** |
| Іванов | Теорія автоматів | 4 |
| Іванов | Теорія автоматів | 5 |
| Крилов | Теорія автоматів | 3 |
| Крилов | Теорія автоматів | 4 |
| Крилов | Теорія автоматів | 5 |
| Степанов | Теорія автоматів | 4 |
| Степанов | Теорія автоматів | 5 |
| Степанов | Теорія автоматів | 3 |
| Іванов | Бази даних | 3 |
| Іванов | Бази даних | 5 |
| Іванов | Бази даних | 4 |
| Крилов | Бази даних | 4 |
| Крилов | Бази даних | 3 |
| Степанов | Бази даних | 3 |
| Крилов | Бази даних | 5 |
| Степанов | Бази даних | 5 |

| | | |
|---|---|---|
| Степанов | Бази даних | 4 |

**Відношення (relation)**

N–им відношенням **R** називається **підмножина** декартового добутку

$$\mathbf{D_1 x\ D_2\ x\ ...\ xD_n}$$

множин $\mathbf{D_1, D_2, ..., D_n}$ **(n > 1),** необов'язково різних.

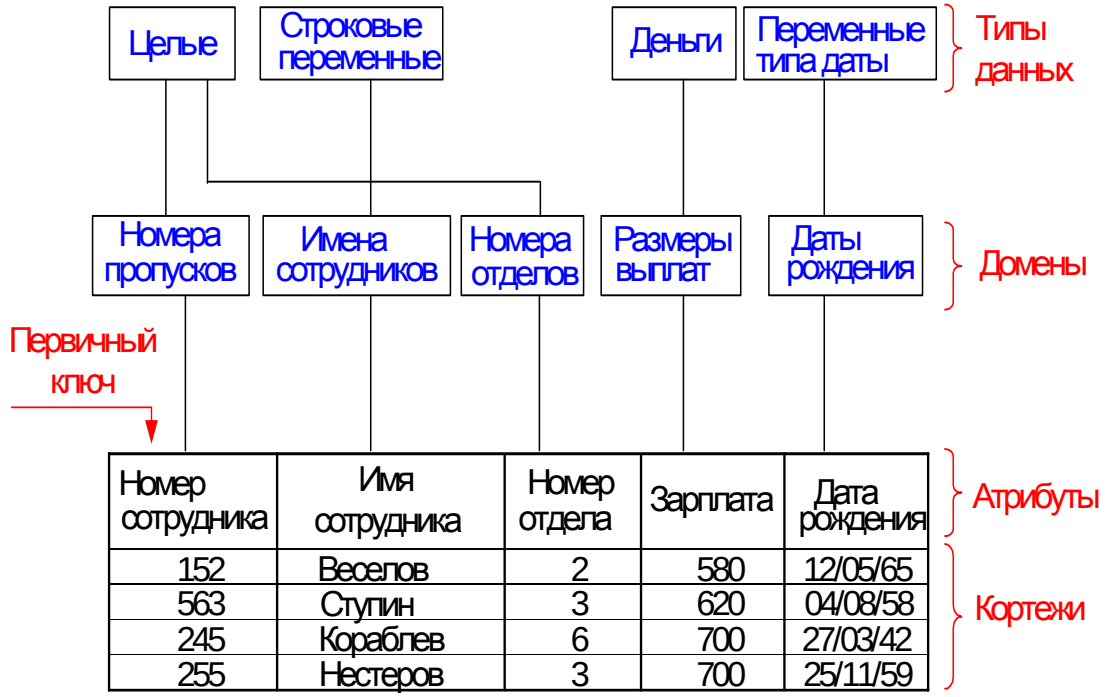
Вихідні множини $\mathbf{D_1, D_2, ..., D_n}$ називають в моделі доменами.

$\mathbf{R \subseteq D_1xD_2x...xD_m}$ де $\mathbf{D_1xD_2x\ ...xD_n}$— повний декартовий добуток.

**Приклад відношення**

| **Прізвище** | **Дисципліна** | **Оцінка** |
|---|---|---|
| Іванов | Теорія автоматів | 4 |
| Іванов | Бази даних | 3 |
| Крилов | Теорія автоматів | 5 |
| Степанов | Теорія автоматів | 5 |
| Степанов | Бази даних | 4 |

**Основні поняття реляційних баз даних**




**Неформальні визначення реляційних об`єктів даних**

| Формальний реляційний термін | Неформальний еквівалент |
|---|---|
| Відношення | Таблиця |
| Кортеж | Рядок або запис таблиці |
| Кардинальне число | Кількість рядків |
| Атрибут | Стовпчик або поле |
| Степінь (арність) | Кількість стовпців |
| Первинний ключ | Унікальний ідентифікатор |
| Домен | Загальна кількість допустимих значень даних в стовпці |

Схема відношення **R**

$\mathbf{S_R = (A_1, A_2,\ldots A_n),\quad A_i \subseteq D_i}$

Еквівалентні схеми

$\mathbf{S_{R1} = (A_1, A_2,\ldots A_n)}$

$\mathbf{S_{R2} = (B_1, B_2,\ldots B_m)}$

$\mathbf{S_{R1} \sim S_{R2},\ \longrightarrow\ n=m\ i\ A_i\ B_i \subseteq D_i}$

**Загальна характеристика реляційної моделі даних**


**Властивості відношень:**

- Цілісність сутностей

- Відсутність впорядкованості кортежів
- Відсутність впорядкованості атрибутів
- Атомарність значень атрибутів
- Null – значення и тризначна логіка (3VL)
- Цілісність посилань

**Тризначна логіка**

| AND | F | T | U |
|---|---|---|---|
| **F** | F | F | F |
| **T** | F | T | U |
| **U** | F | U | U |

| OR | F | T | U |
|---|---|---|---|
| **F** | F | T | U |
| **T** | T | T | T |
| **U** | U | T | U |

| NOT | |
|---|---|
| **F** | T |
| **T** | F |
| **U** | U |

**Зовнішній ключ**

Підмножина атрибутів відношення **R** називається *зовнішнім ключем* **FK,** якщо:

- Існує відношення **S** ( **S** і **R** не обов`язково різні) з потенційним ключем **K**.
- Кожне значення **FK у відношенні R** завжди співпадає зі значенням **K** для деякого кортежу із **S**, або являється null–значенням

**Цілісність по посиланням**

Для кожного значення зовнішнього ключа, що з’являється у відношенні, в таблиці, на котру веде посилання, повинен знайтись кортеж з таким же значенням первинного ключа або значення зовнішнього ключа повинно бути невизначеним, тобто ні на що не вказувати.

**Операції, що можуть порушити цілісність по посиланням**

- Вставка кортежу в родинному відношенні
- Оновлення кортежу в родинному відношенні
- Видалення кортежу в родинному відношенні
- Вставка кортежу в дочірньому відношенні
- Оновлення кортежу в дочірньому відношенні
- Видалення кортежу в дочірньому відношенні

**Стратегії підтримання цілісності по посиланням**

- RESTRICT (ОБМЕЖИТИ)
- CASCADE (КАСКАДУВАТИ)

- SET NULL (ВСТАНОВИТИ В NULL)
- SET DEFAULT (ВСТАНОВИТИ ЗА ЗАМОВЧУВАННЯМ)

## Лекція 4. Реляційна алгебра

теоретико–множинні операції реляційної алгебри:

- Об`єднання відношень;
- перетин відношень;
- різниця відношень;
- декартовий добуток відношень;

спеціальні операції РА:

- вибірка відношень;
- проекція відношень;
- з`єднання відношень;
- ділення відношень.

Відношення, що використовуватимуться в подальших прикладах

| $R_1$ | |
|---|---|
| **Шифр детали** | **Название детали** |
| 11073 | Гайка М1 |
| 11075 | Гайка М2 |
| 11076 | Гайка М3 |
| 11003 | Болт М1 |
| 11006 | Болт М3 |
| 13063 | Шайба М1 |
| 13066 | Шайба М3 |

| $R_2$ | |
|---|---|
| **Шифр детали** | **Название детали** |
| 11073 | Гайка М1 |
| 11076 | Гайка М3 |
| 11077 | Гайка М4 |
| 11004 | Болт М2 |
| 11006 | Болт М3 |

**Теоретико-множинні операції**

**Об`єднання відношень**

**$R_1$ = { $r_1$ } , $R_2$ = { $r_2$ }.**

де **$r_1$** і **$r_2$** — відповідно кортежі відношень **$R_1$** і **$R_2$**,

Об`єднання $R_1$ **U** $R_2$ = { r: r $\in R_1$ **U** r $\in R_2$ }.

**Приклад** $R_3 = R_1$ **U** $R_2$

| $R_3$ | |
|---|---|
| **Шифр детали** | **Название детали** |
| 11073 | Гайка М1 |
| 11075 | Гайка М2 |
| 11076 | Гайка М3 |
| 11003 | Болт М1 |
| 11006 | Болт М3 |
| 13063 | Шайба М1 |
| 13066 | Шайба М3 |
| 11077 | Гайка М4 |
| 11004 | Болт М2 |

**Перетин відношень**

$R_4 = R_1 \cap R_2 = \{ r : r \in R_1 \wedge r \in R_2 \}$,

| $R_4$ | |
|---|---|
| **Шифр детали** | **Название детали** |
| 11073 | Гайка М1 |
| 11076 | Гайка М3 |
| 11006 | Болт М3 |

**Різниця відношень**

$R_5 = R_1 \setminus R_2 = \{ r : r \in R_1 \cap r \notin R_2 \}$

| $R_5$ | |
|---|---|
| **Шифр детали** | **Название детали** |
| 11075 | Гайка М2 |
| 11003 | Болт М1 |
| 13063 | Шайба М1 |
| 13066 | Шайба М3 |

**Конкатенацією кортежів**

$c = \langle c_1, c_2, ..., c_n \rangle$ и $q = \langle q_1, q_2, ..., q_m \rangle$

називається кортеж, отриманий додаванням значень другого в кінець першого. З`єднання кортежів **c** і **q** позначається як **(c , q)**.

$(c, q) = \langle c_1\ c_2, ... , c_n, q_1, q_2, .... q_m \rangle$

**Розширеним декартовим добутком** відношення **R**, степені **n** зі схемою $S_{R1}=(A_1,A_2...,A_n)$ і відношення $R_2$ степені **m** зі схемою $S_{R2}=(B_1,B_2, ... , B_m)$ називається відношення $R_3$ степені **n+m** зі схемою

$S_{R3} = (A_1, A_2, ... , A_n, B_1, B_2, ..., B_m)$,

Що містять кортежі, отримані з`єднанням кожного кортежу **з відношення $R_1$** з кожним кортежем **q** відношення $R_2$.

**$R_1$ = { c }, $R_2$ = { q }**

**$R_1 \otimes R_2 = \{(c, q) : c \in R_1 \wedge q \in R_2\}$**

**Спеціальні операції реляційної алгебри**

**Вибірка**

Позначення

**R[C(r)] = {r | r ∈ R ^ C(r) = "Істина"}**

**Приклад**

**$R_{12} = R_{10}$ [ Шифр деталі = «11003» ]**

$R_{10}$

| Шифр детали | Название детали | Цех |
|---|---|---|
| 11073 | Гайка М1 | Цех 1 |
| 11075 | Гайка М2 | Цех 1 |
| 11076 | Гайка М3 | Цех 1 |
| 11003 | Болт М1 | Цех 1 |
| 11006 | Болт М3 | Цех 1 |
| 13063 | Шайба М1 | Цех 1 |
| 13066 | Шайба М3 | Цех 1 |
| 11077 | Гайка М4 | Цех 1 |
| 11004 | Болт М2 | Цех 1 |
| 11006 | Болт М3 | Цех 2 |
| 13063 | Шайба М1 | Цех 2 |
| 13066 | Шайба М3 | Цех 2 |
| 11077 | Гайка М4 | Цех 2 |
| 11004 | Болт М2 | Цех 2 |
| 11075 | Гайка М2 | Цех 3 |
| 11076 | Гайка М3 | Цех 3 |
| 11003 | Болт М1 | Цех 3 |
| 11006 | Болт М3 | Цех 3 |
| 13063 | Шайба М1 | Цех 3 |
| 13066 | Шайба М3 | Цех 3 |
| 11077 | Гайка М4 | Цех 3 |

$R_{12}$

| Шифр детали | Название детали | Цех |
|---|---|---|
| 11003 | Болт М1 | Цех 1 |
| 11003 | Болт М1 | Цех 3 |

**Проекція**

Позначення

**R[X,Y,Z]**

**Приклад**

**$R_{14}$** = $R_{13}$ [ Цех ]

$R_{13}$

| Шифр детали | Название детали | Цех |
|---|---|---|
| 11003 | Болт М1 | Цех 1 |
| 11003 | Болт М1 | Цех 3 |

$R_{14}$

| Цех |
|---|
| Цех 1 |
| Цех 3 |

**Умовне з`єднання**

Позначення

**R1[A1ΘA2]R2**

Визначення

{(r1||r2): r1∈R1 ∩ r2∈R2 ∩ (r1[A1]Θr2[A2]}

Окремі випадки з`єднань:

**Тета-з'єднання** Θ – один із операторів порівняння ( =, ≠, ≥, ≤, <>, і т.д.)

**Екві - з'єднання**: Θ – знак рівності (=)

**Природні з`єднання**

Вихідні відношення:

$R1(A_1,A_2,\ldots A_n, X_1,X_2,..X_m)$

$R2(B_1,B_2,\ldots B_r, X_1,X_2,\ldots X_m)$

де $X_1,X_2,..X_m$ – атрибути, що співпадають

Природні з`єднання

R3=R1 **JOIN** R2 = $(A_1,A_2,..A_n,X_1,X_2,\ldots X_m,B_1,B_2,\ldots B_r)$

Схема відношень

$\mathbf{SR_3}=(A_1,A_2,..A_n,X_1,X_2,\ldots X_m,B_1,B_2,\ldots B_r)$

Кортежі відношення

$(a_1,a_2,..a_n,x_1,x_2,\ldots x_m,b_1,b_2,\ldots b_r)$ де

$(a_1,a_2,..a_n,x_1,x_2,\ldots x_m)\subseteq R_1$

$(x_1,x_2,\ldots x_m,b_1,b_2,\ldots b_r) \subseteq R_2$

**Ділення відношень**

Відношення - ділене

$R_1(A_1,A_2,\ldots A_n, X_1,X_2,..X_m)$

Відношення - дільник

$R_2(X_1,X_2,\ldots X_m)$

де

$X_1,X_2,\ldots X_m$ – атрибути, що співпадають

Результат ділення - відношення

$R_3 = R_1[A_1 \div A_2 ]R_2$

Зі схемою

$S_{R3} = (A_1, A_2, \ldots A_n)$

**Приклад**

Визначити перелік цехів $\mathbf{R_{17}}$ в яких випускається номенклатура деталей

$\mathbf{R_7}$ – обов`язкова номенклатура випуску деталей для всіх цехів

$\mathbf{R_{10}}$ – реальний випуск деталей в кожному цеху

$\mathbf{R_{10}}$ [Шифр деталі, Назва деталі ÷ Шифр деталі, Назва деталі]$\mathbf{R_7}$

| **$R_7$** | |
|---|---|
| **Шифр детали** | **Название детали** |
| 11073 | Гайка М1 |
| 11075 | Гайка М2 |
| 11076 | Гайка М3 |
| 11003 | Болт М1 |
| 11006 | Болт М3 |
| 13063 | Шайба М1 |
| 13066 | Шайба М3 |
| 11077 | Гайка М4 |
| 11004 | Болт М2 |

| **$R_{10}$** | | |
|---|---|---|
| **Шифр детали** | **Название детали** | **Цех** |
| 11073 | Гайка М1 | Цех 1 |
| 11075 | Гайка М2 | Цех 1 |
| 11076 | Гайка М3 | Цех 1 |
| 11003 | Болт М1 | Цех 1 |
| 11006 | Болт М3 | Цех 1 |
| 13063 | Шайба М1 | Цех 1 |
| 13066 | Шайба М3 | Цех 1 |
| 11077 | Гайка М4 | Цех 1 |
| 11004 | Болт М2 | Цех 1 |
| 11006 | Болт М3 | Цех 2 |
| 13063 | Шайба М1 | Цех 2 |
| 13066 | Шайба М3 | Цех 2 |
| 11077 | Гайка М4 | Цех 2 |
| 11004 | Болт М2 | Цех 2 |
| 11075 | Гайка М2 | Цех 3 |
| 11076 | Гайка М3 | Цех 3 |
| 11003 | Болт М1 | Цех 3 |
| 11006 | Болт М3 | Цех 3 |
| 13063 | Шайба М1 | Цех 3 |
| 13066 | Шайба М3 | Цех 3 |
| 11077 | Гайка М4 | Цех 3 |

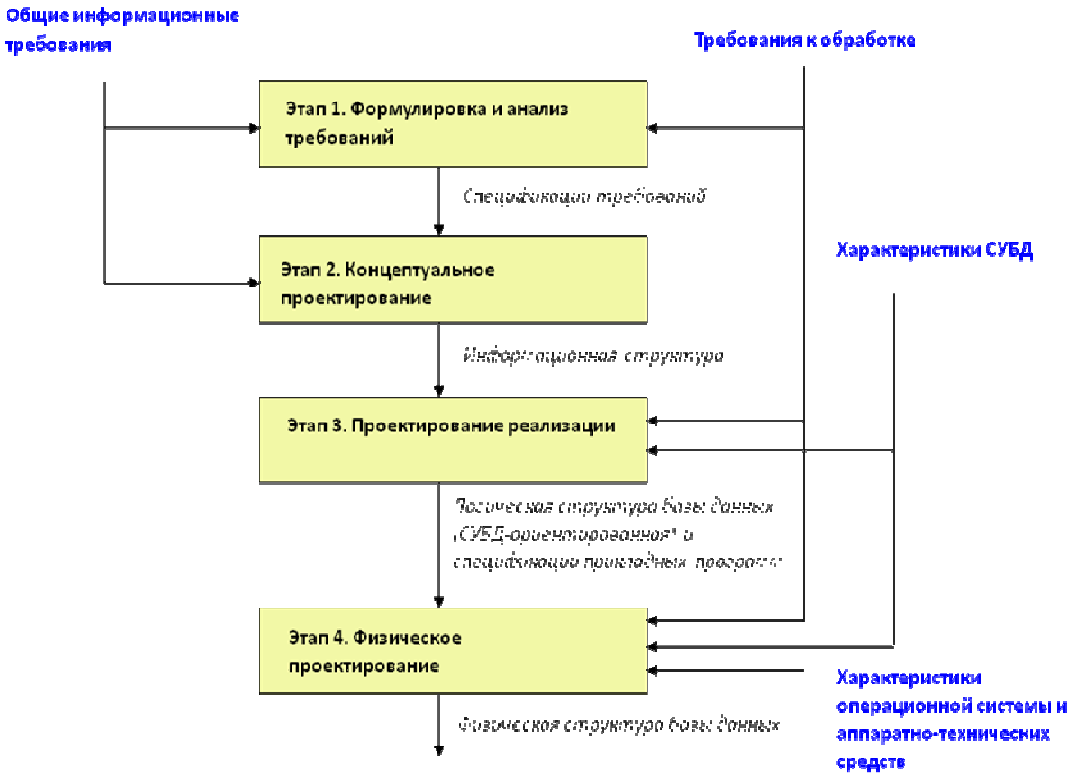
| **$R_{17}$** |
|---|
| **Цех** |
| Цех 1 |

**Пріорітет операцій реляційної алгебри**

| **Операція** | **Пріоритет** |
|---|---|
| Перейменування атрибутів | 4 |
| Вибірка | 3 |
| Проекція | 3 |
| Декартовий добуток | 2 |
| З`єднання | 2 |
| Перетин | 2 |
| Ділення | 2 |
| Об`єднання | 1 |
| Різниця | 1 |

## Лекція 5. Проектування реляційних БД на основі принципів нормалізації

### Основні етапи проектування баз даних




**Проектування реалізації**
**Компоненти:**

- проектування бази даних;
- проектування програм

**Ціль –** створення СКБД – орієнтованої схеми (логічної моделі) з використанням в якості вихідних даних результатів концептуального проектування і вимог обробки конкретної СКБД. Розробка схеми бази даних, тобто сукупності схем відношень (таблиць), які коректно моделюють об`єкти (предмети, явища та ін.) предметної області і семантичні зв`язки між цими об`єктами.

**Результати:**

- опис концептуальної схеми БД в термінах вибраної СКБД;
- опис зовнішніх моделей в термінах вибраної СКБД;
- опис правил підтримки цілісності бази даних;
- розробка процедур підтримки семантичної цілісності бази даних;

**Прийняті рішення:**

- Із яких відношень повинна складатись база даних?
- Яким чином повинні бути пов`язані ці відношення?
- Які атрибути повинні бути у цих відношень?

**Теорія нормалізації**

**Ціль –** скоротити надмірність даних, що зберігаються

**Переваги:**

- економія об`єму пам`яті, що використовується;
- зменшення витрат на багаторазові операції оновлення надмірності копій;
- усунення можливості виникнення протиріч внаслідок зберігання в різних місцях відомостей про один і той же об`єкт.

**Нормалізація схем відношень** – формальний апарат обмежень на формування відношень (таблиць), який дозволяє усунути дублювання, забезпечує несуперечливість даних, що зберігаються в базі, зменшує трудовитрати на супроводження (введення, корегування) бази даних.

**Послідовність нормальних форм**

- перша нормальна форма (1NF);
- друга нормальна форма (2NF);
- третя нормальна форма (3NF);
- посилена третя нормальна форма, чи нормальна форма Бойса–Кодда (*BCNF*);
- четверта нормальна форма (4*NF*);
- п`ята нормальна форма (5*NF*).

**Властивості нормальних форм**

- кожна наступна нормальна форма в деякому сенсі краще попередньої нормальної форми;
- при переході до наступної нормальної форми властивість попередніх нормальних форм зберігаються.

**Функціональна залежність:**

$$X \rightarrow У \qquad X, У \subseteq R$$

X, У – атрибути відношення R

X – детермінант відношення

Атрибут **У** функціонально залежить від атрибуту **Х**, якщо кожному значенню атрибута **Х** відповідає тільки одне значення атрибута **У**.

**Функціональна взаємозалежність**

Якщо одночасно існують функціональні залежності вигляду $A \rightarrow B$ і $B \rightarrow A$, тоді має місце функціональна взаємозалежність, яка зображається $A \leftrightarrow B$.

Приклад функціональної взаємозалежності

Табельний_номер ↔ Номер_паспорта

Приклад функціональних залежностей

**ГРАФІК_ПОЛЬОТІВ** (Пілот, Рейс, Дата_вильоту, Час_вильоту)

**Рейс → Час_вильоту**; (кожному рейсу відповідає певний час вильоту)

{**Пілот, Дата_вильоту, Час_вильоту** } → **Рейс**; (для кожного пілота, дати і часу вильоту можливий тільки один рейс)

{**Рейс, Дата_вильоту** } → **Пілот** (на певний день і рейс призначається певний пілот)

**Повна функціональна залежність:**

$$R.A \rightarrow R.B$$

$$\{ A1 \subset A \Rightarrow R.A \rightarrow R.B \cap R.A1 \text{ –/–> } R.B \}$$

Атрибут .**В** функціонально залежить від атрибута **А**, і не залежить від будь-якої підмножини атрибута **А**.

**Приклади:**

ЗАМОВЛЕННЯ (Номер_замовлення, Код_товару, КвоТоваруВЗамовленні, Найменування_товару)

**Повна** функціональна залежність:

{Номер_замовлення, Код_товару} → КвоТоваруВЗамовленні

**Неповна функціональна залежність:**

{ Номер_замовлення, Код_товару } → Найменування_товару

**Транзитивна** функціональна залежність

$$\mathbf{R.A \rightarrow R.B}$$

Якщо існує набір атрибутів **С** такий, що:

1. $\mathbf{C} \not\subset \mathbf{A}$
2. $\mathbf{B} \not\subset \mathbf{C}$ .

3. Існує функціональна залежність

**R.A → R.C**.

4. 4. Не існує функціональна залежність

**R.C → R.A**.

5. Існує функціональна залежність

**R.C → R.B**.

**Приклад**

**СПІВРОБІТНИКИ** (Номер_співробітника, Код_відділу, Найменування_відділу )

Транзитивна функціональна залежність:

Номер_співробітника → Найменування_відділу

**Аксіоми Армстронга**

- **Рефлексивність*:***

**B ⊆ A, ⇒ A→B**

- **Доповнення*:***

**A→B ⇒ AC→BC**

- **Транзитивність*:***

**A→B** и **B→C ⇒ A→C**.

**Перша нормальна форма (1НФ)**

Відношення знаходиться в 1NF якщо значення всіх його атрибутів атомарні

**Аномалії першої нормальної форми:**

**Аномалія включення – доки** не буде замовлення на товар, інформація про товар буде відсутня в базі даних (найменування товару, код типу товару, найменування типу товару);

**Аномалія оновлення –** при зміні найменування товару необхідний повний перегляд відношення з ціллю знайти всі замовлення, щоб зміна найменування товару було відображена у всіх замовленнях

**Аномалія видалення –** якщо який-небудь товар видалять із бази даних (знятий з продажу), то при цьому буде втрачена не тільки інформація про ті замовлення, в яких був присутній цей товар, але й про товар в цілому (код товару, його найменування і тип)

**Аномалія дублювання –** деякі значення атрибутів доводиться багаторазово повторювати (найменування товару і найменування типу товару).

**Друга Нормальна Форма (2НФ)**

Відношення знаходиться в другій нормальній формі тоді і тільки тоді, коли воно знаходиться в першій нормальній формі і не містить неповних функціональних залежностей не ключових атрибутів від атрибутів первинного ключа.

**Приклад. Здача сесії**

Stud = (ПІБ, Номер зал.кн., Група, Дисципліна, Оцінка)

**Аномалія другої нормальної форми :**

**Аномалія включення** Ми не можемо доповнити відношення **Stud** даними про студента, який в даний час ще не складав ні одного екзамену. (**Дисципліна** являється частиною первинного ключа і не може містити невизначених значень). Між тим студент вже зачислений до інституту.

**Аномалія оновлення** Щоб змінити номер групи студента, ми будемо змушені модифікувати всі кортежі з відповідними значеннями атрибуту **Номер.зал.кн.**.

**Приведення до 2NF**

Вихідне відношення:

$$R = (\underline{K_1,K_2},A_1,\ldots A_n,B_1,\ldots B_m) .$$

Ключ: $\{K_1,K_2\}$ – складний.

Функціональні залежності:

– залежність всіх атрибутів від ключа відношення

$$\{K_1,K_2\} \rightarrow \{A_1,\ldots A_n,B_1,\ldots B_m\}$$

- залежність деяких атрибутів від частини складного ключа

$$\{K_1\} \rightarrow \{A_1,\ldots A_n\}$$

Декомпоновані відношення:

- атрибути, винесені з вихідного відношення разом частиною складного ключа. Ключ $\{K_1\}$.

$$R_2 = (\underline{K_1},A_1,\ldots A_n)$$

– залишок від вихідного відношення. Ключ $\{K_1,K_2\}$ .

$$R_3 = (\underline{K_1,K_2},B_1,\ldots B_m)$$

**Приклад**

Stud = (ПІБ, Номер зал.кн., Група, Дисципліна, Оцінка)

⇓

$R_1$ = (ПІБ, Номер.зал.кн., Група)

$R_2$ = (Номер зал.кн., Дисципліна, Оцінка)

**Третя нормальна форма (3НФ)**

**Визначення**

Відношення знаходиться в **3НФ** *(3НФ, Third Normal Form, 3NF)* тоді і тільки тоді, коли воно знаходиться в другій нормальній формі і не містить транзитивних залежностей.
Іншими словами, додаткове обмеження полягає в тому, що всі не ключові атрибути повинні бути взаємно функціонально незалежними.

Приклад транзитивної залежності

R = (ПІБ, **Номер зал.кн.,** Група, Факультет, Спеціальність, Випускаюча кафедра)

**Аномалії третьої нормальної форми:**

**Аномалія включення –** неможливо зберегти дані про нову спеціальність, доки не з`явиться студент з новою спеціальністю. (Первинний ключ не може містити не визначенні значення.) ;

**Аномалія оновлення –** при зміні спеціальності в деякій групі, ми будемо змушені змінити значення атрибуту **Спеціальність** в кортежах всіх студентів, які відносяться до цієї групи.

**Аномалія видалення –** при випуску останнього студента з такої спеціальністю ми втратимо інформацію про наявність такої спеціальності.

Функціональні залежності відношення:

- Номер зал .кн. → ПІБ
- Номер зал.кн. → Група
- Номер зал.кн. → Факультет
- Номер зал.кн. → Спеціальність
- Номер зал.кн. → Випускаюча кафедра
- Група → Факультет
- Група → Спеціальність
- Група → Випускаюча кафедра
- Випускаюча кафедра → Факультет

**Приведення до ЗНФ**

Вихідне відношення:

$$R = (\underline{K},A_1,\ldots A_n,B_1,\ldots B_m),\ \text{ключ: } \{K\}$$

Функціональні залежності

- залежність всіх атрибутів від ключа відношення

$$\{K\} \rightarrow \{A_1,\ldots A_n,B_1,\ldots B_m\},$$

- залежність деяких не ключових атрибутів інших не ключових атрибутів
  - $\{A_1,\ldots A_n\} \rightarrow \{B_1,\ldots B_m\},$

Декомпоновані відношення:

- атрибути, винесені із вихідного відношення разом з детермінантом функціональної залежності.

$$R_2 = (\underline{A_1,\ldots A_n},B_1,\ldots B_m),\ \text{ключ: } \{A_1,\ldots A_n\}$$

- залишок від вихідного відношення.

$$R_3 = (\underline{K},A_1,\ldots A_n),\ \text{ключ: } \{K\}$$

R = (ПІБ, **Номер зал.кн.,** Група, Факультет, Спеціальність, Випускаюча кафедра)

⇓

$R_1$ =(Номер. зал. кн., ПІБ. Спеціальність. Група)

$R_2$ = (Група. Випускаюча кафедра)

$R_3$ = (Випускаюча кафедра, Факультет)

**НФБК**

Відношення знаходиться в нормальній формі Бойса—Кодда, якщо воно знаходиться в третій нормальній формі і кожний детермінант відношення являється можливим ключем відношення.

Приклад відношення

Співробітники_завдання {ТабНомер, ІНН, ПРОЕКТ, ЗАВДАННЯ}

**Аномалія оновлення**

У випадку якщо ключем являється ІНН, і змінився табельний номер службовця, вимагається оновити атрибут **ТАБНомер** у всіх кортежах відношення **Співробітники_завдання**, що відповідають даному службовцю.

Інакше буде порушена функціональна залежність **ТабНомер ІНН**, і база даних виявиться в непогодженому стані.

**Приклад**

**R** = (Номер зал.кн., Ідентифікатор_студента, Дисципліна, Дата, Оцінка)

**Можливі ключі (підкреслені)**:

– {Номер_зал.кн, Дисципліна, Дата}

– {Ідентифікатор_студента, Дисципліна, Дата}.

**Функціональні залежності**:

Номер_зал.кн, Дисципліна, Дата → Оцінка;

Ідентифікатор _студента, Дисципліна, Дата → Оцінка;

Номер зал.кн. → Ідентифікатор_студента;

Ідентифікатор_студента → Номер зал.кн.

**Аномалії оновлення**

У випадку якщо в ключ відношення входить ІНН і необхідно змінити номер залікової книжки, вимагається оновити атрибут **Номер зал.кн.** у всіх кортежах відношення **R**, що відповідають даному студенту. Інакше буде порушена функціональна залежність
**Ідентифікатор_студента → Номер зал.кн**,
і база даних виявиться в непогодженому стані.

**R** = (Номер зал.кн., Ідентифікатор_студента, Дисципліна, Дата, Оцінка)

⇓

(Ідентифікатор_студента, Дисципліна, Дата, Оцінка)

(Номер зал.кн., Ідентифікатор_студента)

чи навпаки:

– (Номер зал.кн., Дисципліна, Дата, Оцінка)

– (Номер зал.кн., Ідентифікатор_студента)

**Порівняння нормалізованих і ненормалізованих моделей**

| **Критерій** | **Відношення слабо нормалізовані (1НФ, 2НФ)** | **Відношенні сильно нормалізовані (3НФ)** |
|---|---|---|
| Адекватність бази даних | ГІРШЕ (–) | КРАЩЕ (+) |

| | | |
|---|---|---|
| предметної області | | |
| Легкість розробки і супроводження бази даних | СКЛАДНІШЕ (–) | ЛЕГШЕ (+) |
| Швидкість виконання вставки, оновлення, видалення | ПОВІЛЬНІШЕ (–) | ШВИДШЕ (+) |
| Швидкість виконання вибірки даних | ШВИДШЕ (+) | ПОВІЛЬНІШЕ (–) |

## Лекція 6. Мова SQL. Начальні відомості

### Форми мови SQL

**Інтерактивний SQL –** використовується для функціонування безпосередньо в базі даних, щоб виконувати вивід інформації для використання її користувачем.

**Вкладений SQL –** складається з команд SQL що містяться всередині програм, які звичайно написані на деякій іншій мові (типу C#).

### Функціональні категорії команд мови SQL

**DDL (Data Definition Language, Мова Визначення Даних)** – складається з команд які створюють об`єкти (таблиці, індекси, представлення і так далі ) в базі даних

**DML (Data Manipulation Language, Мова Маніпулювання Даними)** – це набір команд, які визначають які значення представлені в таблицях в будь-який момент часу, а також дозволяють змінювати і видаляти дані з таблиць БД;

**DCL (Data Control Language, Мова Керування Даними)** – – складається з засобів, які управляють виконанням транзакцій, а також визначають чи дозволити користувачу виконувати певні дії чи ні.

### Переваги мови SQL

- стандартизованість мови SQL
- можливість переносу з однієї обчислювальної системи на іншу
- реляційна основа мови
- можливість створення інтерактивних запитів
- можливість програмного доступу до БД
- забезпечення різноманітного представлення даних
- можливість динамічної зміни і розширення структури БД
- підтримка архітектури клієнт/сервер

### Керування базами даних з допомогою SQL

- будь-якому стовпцю таблиці присвоєно ім`я, яке повинно було бути унікальним для цієї таблиці;
- стовпці таблиці впорядковуються зліва направо, тобто стовпець 1, стовпець 2, ..., стовпець n.
- рядки таблиці не впорядковані (їх послідовність визначається лише послідовністю введення в таблицю);

- в полі на перетині рядка і стовпця будь-якої таблиці завжди є тільки одне значення даних і ніколи не повинно бути множини значень
- всім рядкам таблиці відповідає одна і та ж множина стовпців, хоча в певних стовпцях будь-який рядок може містити пусті значення (NULL–значення), тобто може не мати значень для цих стовпців;
- всі рядки таблиці обов`язково відрізняються один від одного хоча б єдиним значенням, що дозволяє однозначно ідентифікувати будь-який рядок такої таблиці;
- при виконанні операцій з таблицею її рядки і стовпці можна обробляти в любому порядку безвідносно до їх інформаційного змісту.

**Типи даних**

- **CHARACTER ( n )** чи **CHAR ( n )** — символьні рядки постійної довжини в **n** символів. При завданні даного типу під кожне значення завжди відводиться **n** символів, і якщо реальне значення займає менше, ніж **n** символів, то СКБД автоматично доповнює недостаючи символи пробілами.
- **NUMERIC [( n , m )]** — точні числа, тут **n** — загальна кількість цифр в числі, **m** — кількість цифр зліва від десяткової точки.
- **DECIMAL [( n , m )], DEC [( n , m )]** — точні числа, тут **n** — загальна кількість цифр в числі, **m** — кількість цифр зліва від десяткової точки
- **INTEGER** чи **INT** — цілі числа.
- **SMALLINT** — цілі числа меншого діапазону.

Незважаючи на те, що в стандарті SQL 1 не визначається точно, що мається на увазі під типом **INT** і **SMALLINT** (це віддано на конкретну реалізацію), вказано тільки відношення між цими типами даних, в більшості реалізацій тип даних **INTEGER** відповідає цілим числам, що зберігаються в чотирьох байтах, а **SMALLINT** — відповідає цілим числам, що зберігаються в двох байтах. Вибір одного із цих типів визначається розміром числа.

- **FLOAT [( n )]** — числа більшої точності, що зберігаються в формі з плаваючою точкою. Тут **n** — число байтів, зарезервоване під зберігання одного числа. Діапазон чисел визначається конкретною. реалізацією.
- **REAL** — дійсний тип чисел, який відповідає числам з плаваючою точкою, меншої точності, ніж **FLOAT**.
- **DOUBLE PRECISION** специфікує тип даних з визначеною в реалізації точністю, більшою, ніж визначена в реалізації точність для **REAL**.

- **VARCHAR ( n )** — рядки символів змінної довжини.
- **NCHAR ( N )** — рядки локалізованих символів постійної довжини.
- **NCHAR VARYING ( n )** — рядки локалізованих символів змінної довжини.
- **BIT ( n )** — рядок бітів довжини.
- **BIT VARYING ( n )** — рядки бітів змінної довжини.
- **DATE** — календарна дата.
- **TIMESTAMP(точність)** — дата і час.
- **INTERVAL** — часовий інтервал.

Більшість комерційних СКБД підтримують ще додаткові типи даних.

**Рекомендації по вибору типів даних**

- Тип VARCHAR при фізичному зберіганні займає на байт – два більше, ніж тип CHAR при одній і тій же об`явленій довжині.
- Для числових значень фіксованої довжини переважніше використовувати тип DEC. Він обробляється процесором швидше, ніж тип FLOAT.
- Використовуйте INT і SMALLINT виключно для лічильників.
- Намагайтесь уникати використання LONG VARCHAR без зайвої необхідності.
- Уникайте використовувати тип CHAR для представлення числових даних. По-перше, може знадобитись додаткова перевірка, а по-друге, можуть виникнути проблеми при сортуванні таких колонок.
- Використовуйте типи DATE і TIME тільки для зберігання хронологічних даних.
- Використовуйте тип DATETIME виключно для цілей керування даними.

**Оператори**

- додавання (+),
- віднімання (–),
- множення (*)
- ділення (/)
- конкатенація (+ ||)

**Команда створення таблиці**

```
CREATE TABLE < table name > (
     < column name >  < data type >  [ < size > ]
          [ < colconstr > ... ] } ,
               [ < tabconstr > ] , ...  ) ;
```

< table name > – ім`я таблиці, що створюється цією командою

< column name > – ім`я стовпця таблиці

< data type > – тип даних, який може міститись в стовпці.

< size > – розмір поля. Його значення залежить від <data type>

< colconstr > – обмеження стовпця. Може бути любим з наступних:

not null (обов`язковий для заповнення),

unique (унікальний, допускає null),

primary key (первинний ключ),

check(<predicate>) (перевірка предиката),

default = (за замовчуванням = <value expression>)

references <table name> [(<column name>, ... )]

< tabconstr > – обмеження таблиці. Може бути будь-яким з наступних:

UNIQUE (УНІКАЛЬНИЙ),

PRIMARY KEY (ПЕРВИННИЙ КЛЮЧ),

CHECK(<predicate>) (ПЕРЕВІРКА предиката),

FOREIGN KEY(<column name>) (ЗОВНІШНІЙ КЛЮЧ)

REFERENCES <table name> [(<column name>, ... )] (ПОСИЛАННЯ НА ім`я таблиці [(ім`я стовбцю) ] )

**Обмеження значень даних в таблицях**

- Обмеження стовпця – застосовується тільки до індивідуальних стовпців
- Обмеження таблиці – застосовується до груп з одного і більше стовпців

**Обмеження NULL**

1. Колонки, що являються частиною складеного первинного ключа, завжди повинні мати обмеження NOT NULL.

2. Зовнішні ключі повинні також визначатись як NOT NULL.

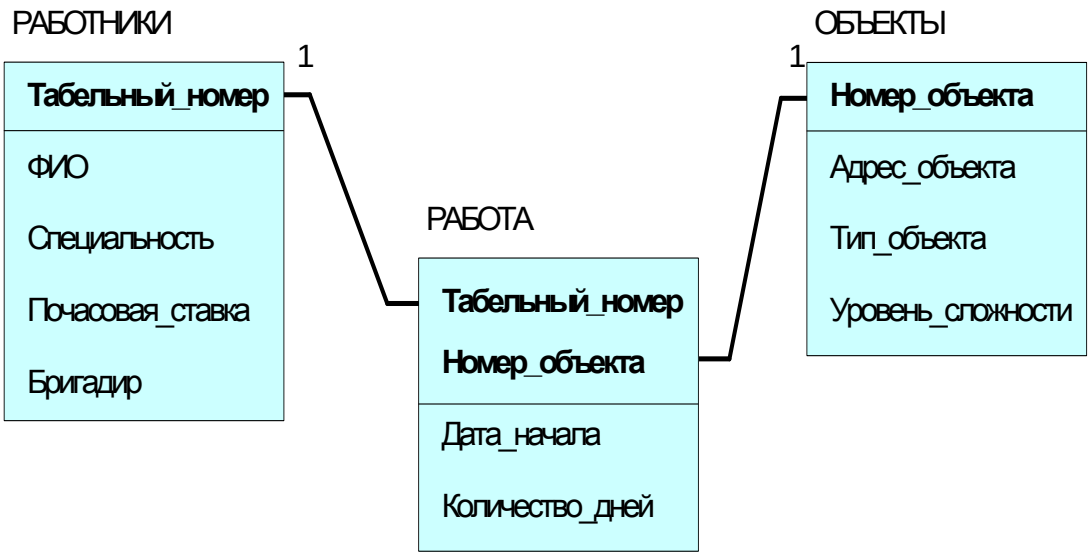
3. Зовнішні ключі з правилом видалення SET NULL повинні визначатись зі специфікацією NULL.

4. Використовуйте специфікацію NOT NULL з DEFAULT для колонок з типами даних DATE и TIME, щоб зберігати поточні дати і поточний час автоматично.

5. Дозволяйте використовувати NULL–значення тільки для тих колонок, які дійсно можуть мати невизначені значення.

6. Використовуйте NOT NULL з DEFAULT для всіх колонок, які не підпадають під перечисленні вище правила.

**Структура БД «Ремонтна бригада», що використовується далі в прикладах**




**Приклад визначення таблиць для БД «Рембригада»**

```
create  table   робітники  (
     табельний_номер     integer  primary key,
     піб            character (15) not null,
     спеціальність                    character (15),
     погодинна_ставка    decimal (6,1),
     бригадир       integer   )

create  table  об’єкти   (
     номер_об’єкта  integer  primary key,
     адрес_об’єкта       character (15) not null,
     тип_об’єкта         character (9)  default  ‘офіс’
            check (тип_об’єкта in (‘офіс’, ’склад’, ’магазин’,
’жилой дом’ )  ),
     рівень_складності   integer default 1
            check (рівень_складності > 0 and рівень_складності <
5) )
```

```
create  table  работа    (
     табельний_номер     integer not null,
     номер_об'єкта       integer not null,
     дата_початку        date,
     кількість_днів integer,
          primary key (табельний_номер, номер_об'єкта),
          foreign key табельний_номер  references  робітники
                         on delete cascade,
          foreign key номер_об'єкта  references  об'єкти
                         on delete cascade   )
```

**Опції ON DELETE**

- CASCADE (розповсюджувати)
- SET NULL (встановити пусте значення)
- SET DEFAULT (встановити значення за замовчуванням)

**Зміна таблиці після того як вона була створена**

**ALTER TABLE** < table name >

ADD < column name > < data type > < size > ;

- Стовпець буде доданий зі значенням NULL для всіх рядків таблиці.
- Новій стовпець стане останнім по порядку стовпцем таблиці.
- Є можливість додаси відразу декілька нових стовпців,

відділивши їх комами, в одній команді.

- Є можливість видалити чи змінити стовпці.
- Є можливість додавати, змінювати і видаляти обмеження

**Видалення таблиць**

**DROP TABLE** < table name >;

- Щоб мати можливість видаляти таблицю, користувач повинен мати права власника (тобто були творцем) цієї таблиці.
- Перш ніж видалити таблицю із бази даних SQL, може знадобитись очищення її від даних.

## Лекція 7. Мова SQL. Маніпуляція даними

**Простий запит –** запит, який звертається тільки до однієї таблиці БД

**Синтаксис команди SELECT**

```
SELECT  *  |  [ DISTINCT | ALL ] <список_полів>
     FROM  <список_таблиць > [ < псевдоніми > ]
     [ WHERE <предикат - умова вибірки чи об`єднання> ]
     [ GROUP  BY  <список_полей_групування> |< номери_колонок >]
     [ HAVING < предикат - умова для груп> ]
     [ ORDER BY  <список_полей_упорядкування>|<номери_колонок> ]
```

Визначення списку полів

- **ALL | DISTINCT**
- Повне ім'я поля: **Ім`я_таблиці.Ім`я_поля**
- Використання псевдонімів:

  **Ім`я_Поля Псевдонім**
- *** – всі поля**

Предикати порівняння в виразі **WHERE**

- **=, <>, >, <, >=, <=, !=**
- **Between A and B**
- **IN (***список***)**
- **LIKE** *шаблон*. Метасимволи шаблону

**_** (підкреслення) – заміщує рівно один невизначений символ,

**%** (відсоток) – заміщує довільне число символів, починаючи з нуля.

- **IS NULL**
- **EXIST**
- **NOT**

**Приклади простих запитів**

**Запит:** Хто працює малярами?

```
SELECT ПІБ
FROM РОБІТНИКИ
WHERE СПЕЦІАЛЬНІСТЬ = 'маляр'
```

**Запит:** Яка недільна заробітна плата кожного електрика (40–годинна робочий тиждень)?

```
SELECT ПІБ, ' Недільна заробітна плата = ',  40 *
Погодинна_ставка
```

```
     FROM РОБІТНИКИ
     WHERE СПЕЦІАЛЬНІСТЬ = електрик ’
     ORDER  BY  ПІБ
```

**Запит:** Перерахувати робітників, у яких назва спеціальності починається з фрази «елек»

```
SELECT *
     FROM РОБІТНИКИ
     WHERE Спеціальність LIKE ( ‘элек%’ )
```

### Оператор NULL в умовах запитів

IS NULL – оператор порівняння з невизначеним значенням (невідомим на даний момент часу). При порівнянні невизначених значень не діють стандартні правила порівняння: одне невизначене значення ніколи не вважається рівним іншому невизначеному значенню.

### Багатотабличні запити

Вибірка з декількох таблиць:

**From t1, t2** – декартовий добуток **t1**, **t2**

Використання псевдонімів:

**From t1 a, t2 b**

**Запит:** Робітники яких спеціальностей призначені на об`єкт № 435 ?

```
SELECT СПЕЦІАЛЬНІСТЬ
    FROM РОБІТНИКИ, РОБОТА
    WHERE РОБІТНИКИ.ТАБЕЛЬНИЙ_НОМЕР =   РОБОТА.ТАБЕЛЬНИЙ_НОМЕР
     AND НОМЕР_ОБ`ЄКТА = 435
```

**Запит:** Перерахувати робітників, призначених на об`єкти типу «офіс»**.**

```
SELECT DISTINCT ПІБ
FROM РОБІТНИКИ, РОБОТА, ОБ`ЄКТИ
WHERE РОБІТНИКИ. ТАБЕЛЬНИЙ_НОМЕР = РОБОТА. ТАБЕЛЬНИЙ_НОМЕР   AND
РАБОТА. НОМЕР_ОБ`ЄКТА =  ОБ`ЄКТИ.НОМЕР_ОБЪЕКТА   AND
ТИП_ОБ`ЄКТА = ‘офіс’
```

### Агрегатні функції

### Приклади запитів з використанням агрегатних функцій

- Яка максимальна і мінімальна погодинна ставки ?
- Яке середнє число днів роботи службовців на об`єкті 435 ?
- Яке загальне число днів, відведених на малярні роботи на об`єкті 312 ?
- Скільки всього різних спеціальностей у робітників?

Для реалізації запитів використовуються вбудовані в SQL агрегатні функції:

**COUNT [ (*) | (DISTINCT | ALL) (ім`я_стовпця)**

**SUM ([ DISTINCT ] ім`я _ стовпця)**

**AVG ([ DISTINCT ] ім`я _ стовпця)**

**MAX([ DISTINCT ] ім`я _ стовпця)**

**MIN([ DISTINCT ] ім`я _ стовпця**

Функції **MAX** і **MIN** оперують одним стовпцем таблиці. Вони вибирають максимальне або мінімальне значення відповідно із цього стовпця. Стовпець може містити числові чи рядкові значення, або значення дати/часу. Результат, що повертається цими функціями, має точно такий же тип даних, що й сам стовпець.

**Запит:.**. Які максимальні і мінімальні погодинні ставки?

```
SELECT MAX (ПОГОДИННА_СТАВКА),   MIN (ПОГОДИННА _СТАВКА)
     FROM   РОБІТНИКИ
```

Функція **SUM** вираховує суму всіх значень стовпця. Дані, що містяться в стовпці, повинні мати числовий тип (містити цілі числа, десяткові числа або числа з плаваючою точкою чи грошові величини)., Значення, що повертається функцією **SUM**, має той же тип даних, що й стовпець.

**Запит:** Яке загальне число днів, відведених на малярні роботи на об`єкті 312?

```
SELECT  SUM ( КІЛЬКІСТЬ_ДНІВ )
FROM   РОБОТА,  РОБІТНИКИ
WHERE  РОБІТНИКИ.ТАБЕЛЬНИЙ_НОМЕР =
РОБОТА.ТАБЕЛЬНИЙ_НОМЕР   AND
     СПЕЦІАЛЬНІСТЬ = ‘маляр’   AND
     НОМЕР_ОБ`ЄКТА = 312
```

Функція **COUNT** підраховує число значень в даному стовпці, що відрізняються від NULL або загальне число рядків в таблиці. Коли вона рахує значення стовпця, вона використовується з DISTINCT щоб виконувати підрахунок різних числових значень в даному полі. Тип даних в стовпці може бути будь-яким. Функція **COUNT** завжди повертає цілий тип даних незалежно від типу даних стовпців.

**Запит:** Скільки всього різних спеціальностей у робітників?

```
SELECT COUNT ( DISTINCT СПЕЦІАЛЬНІСТЬ )
     FROM  РОБІТНИКИ
```

**Запит:** Скільки всього робітників?

```
SELECT COUNT  (*)
        FROM  РОБІТНИКИ
```

Всі функції, крім **COUNT**, можна використовувати з виразами, що обраховуються.

**Запит:** Яка середня недільна заробітна плата робітників?

```
SELECT  AVG  ( 40 * ПОГОДИННА_СТАВКА )
     FROM   РОБІТНИКИ
```

Якщо в команді SELECT міститься вбудована функція і не використовується фраза GROUP BY, то більше в цій команді SELECT нічого крім вбудованих функцій не може бути!

Фраза **GROUP BY** значить, що рядки повинні бути розбиті на групи з загальними значеннями вказаного стовпця (стовпців).

Фраза **GROUP BY** використовується на практиці, коли необхідна статистична інформація не про окремий об`єкт, а по кожній групі.

- Імена вибраних стовпців повинні бути присутніми у фразі **GROUP BY** за виключенням тих, до яких застосовані вбудовані функції
- Замість імен стовпців можна використовувати їх номери

**Запит:** Для робітників кожної спеціальності розрахувати максимальну погодинну ставку.

```
SELECT СПЕЦІАЛЬНІСТЬ, MAX ( ПОГОДИННА_СТАВКА )
     FROM  РОБІТНИКИ
     GROUP  BY  СПЕЦІАЛЬНІСТЬ
```

Фраза **HAVING** накладає умови на групи.

**Запит:** Для кожного типу об`єктів, на яких працюють більше одного робітника, розраховують максимальну тривалість робіт. Розглядати тільки ті об`єкти, роботи на яких почались після 20.10.2001 г.

```
SELECT  НОМЕР_ОБ`ЄКТА,  MAX(КІЛЬКІСТЬ_ДНІВ)
     FROM   РОБОТА
     WHERE   ДАТА_ПОЧАТКУ >= 20/10/01
     GROUP  BY  НОМЕР_ ОБ`ЄКТА
     HAVING  COUNT ( DISTINCT ТАБЕЛЬНИЙ_НОМЕР ) >= 2
```

**З`єднання таблиць**

Синтаксис:

```
FROM MainTable
{INNER | {LEFT | RIGHT | FULL} OUTER | CROSS } JOIN
                JoinTable ON <condition>
```

де

- inner join (за замовчуванням) – внутрішнє з`єднання
- outer join: – зовнішнє з`єднання
- left outer join – ліве
- right outer join – праве

- full outer join – повне
- cross join – перехресне з`єднання (декартовий добуток)

**Приклад**

Вихідні таблиці

| Name | CityId |
|---|---|
| Андрей | 1 |
| Леонид | 2 |
| Сергей | 1 |
| Григорий | 4 |

| Id | Name |
|---|---|
| 1 | Киев |
| 2 | Москва |
| 3 | Одесса |

```
SELECT *
FROM Person
     INNER JOIN City ON Person.CityId = City.Id
```

| Person.Name | Person.CityId | City.Id | City.Name |
|---|---|---|---|
| Андрій | 1 | 1 | Київ |
| Леонід | 2 | 2 | Москва |
| Сергій | 1 | 1 | Київ |

```
SELECT *
FROM Person
LEFT OUTER JOIN City ON Person.CityId = City.Id
```

| Person.Name | Person.CityId | City.Id | City.Name |
|---|---|---|---|
| Андрій | 1 | 1 | Київ |
| Леонід | 2 | 2 | Москва |
| Сергій | 1 | 1 | Київ |
| Григорій | 4 | NULL | NULL |

```
SELECT *
FROM Person
RIGHT OUTER JOIN City ON Person.CityId = City.Id
```

| Person.Name | Person.CityId | City.Id | City.Name |
|---|---|---|---|

| | | | |
|---|---|---|---|
| Андрій | 1 | 1 | Київ |
| Леонід | 2 | 2 | Москва |
| Сергій | 1 | 1 | Київ |
| NULL | NULL | 3 | Одеса |

```
SELECT *
FROM Person
FULL OUTER JOIN City ON Person.CityId = City.Id
```

| **Person.Name** | **Person.CityId** | **City.Id** | **City.Name** |
|---|---|---|---|
| Андрій | 1 | 1 | Київ |
| Леонід | 2 | 2 | Москва |
| Сергій | 1 | 1 | Київ |
| Григорій | 4 | NULL | NULL |
| NULL | NULL | 3 | Одеса |

## Лекція 8. Мова SQL. Підзапити (вкладені запити)

**Вкладений підзапит -** це запит, що виділяється круглими дужками, вкладений в другу SQL команду. Підзапит можна використовувати в таких фразах:

– **Where**

– **Having**

– **From** (для команд Select, Delete)

**Синтаксис**:

```
Select  Список_вибору
From   Таблиці
Where вираз оператор порівняння
     (Select Список_ вибору
     From Таблиці
     …….)
```

**Підзапити**:

- **Однорядкові** – повертають завжди **тільки одне** значення. *Оператор:* **(>, =, >=, <, !=, <=)**
- **Багаторядкові** – повертають набір значень.
  *Оператор:* (**(NOT) IN, (NOT) Exist**)

**Правила оформлення:**

- Підзапит повинен бути виділений круглими дужками
- Підзапит повинен знаходитись справа від оператора порівняння
- В підзапиті не можна використовувати вираз ORDER BY

**Однорядкові підзапити**

**Запит:** Вивести список робітників, що мають таку ж спеціальність, що і робітник з таб. номером 145.

```
SELECT ФИО
FROM РОБІТНИКИ
WHERE Спеціальність =
      ( SELECT Спеціальність
        FROM РОБІТНИКИ
       WHERE Табельний_Номер = 145 )
```

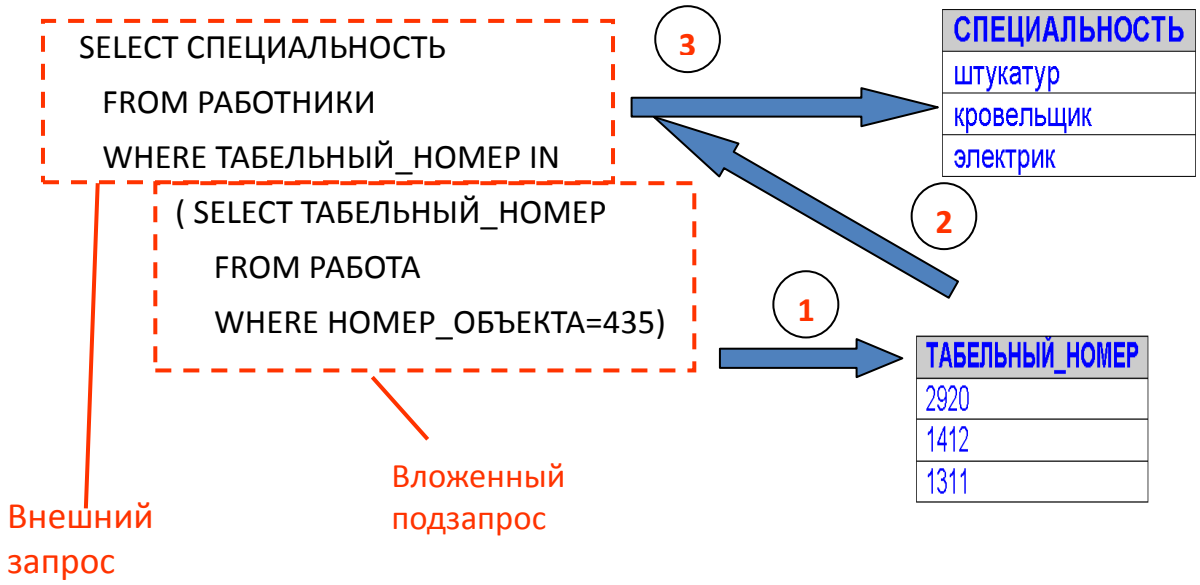
Для повернення одного значення можна використовувати агрегатні функції

**Запит:** Вивести табельні номери робітників, що мають кількість робочих днів більше середньої кількості по підприємству.

```
SELECT Табельний_Номер
FROM РОБОТА
WHERE Кількість_Днів  >
( SELECT AVG(Кількість_Днів  )
     FROM РОБОТА)
```

**Багаторядкові підзапити**

**Запит:** Робітники яких спеціальностей призначені на об`єкт № 435 ?




**Підзапити у фразі HAVING**

**Запит:** Вивести номери об`єктів, для яких середня кількість днів, відпрацьованих робітниками більше середньої кількості днів для об`єкта № 35

```
SELECT Номер_Об`єкта, AVG(Кількість_Днів)
FROM РОБОТА
GROUP BY Номер_Об`єкта
HAVING AVG(Кількість_Днів) >
( SELECT AVG(Кількість_Днів)
FROM РОБОТА
WHERE Номер_Об`єкта = 35 )
```

**Прості і корельовані підзапити**

**Простий (некорельований) підзапит** – підзапит, значення якого не залежить  ні від якого зовнішнього запиту.

**Корельований підзапит** – підзапит, результат якого залежить від рядка, що розглядається головним запитом.

**Запрос.** Перечислить работников, чьи почасовые ставки выше, чем почасовые ставки их бригадиров.


## Оператори EXISTS і NOT EXISTS

Оператор **EXISTS** повертає **Істина**, якщо підзапит повертає хоча б один запис і **Брехня**, якщо він не робить цього
**Запит.** Перерахувати робітників, не призначених на об`єкт 435.

Приклад неправильного запиту

```
SELECT   ТАБЕЛЬНИЙ_НОМЕР
FROM   РОБОТА
WHERE   НОМЕР_ОБ`ЄКТА <> 435
```

Правильний запит

```
SELECT ТАБЕЛЬНИЙ_НОМЕР
FROM РОБІТНИКИ A
WHERE NOT EXISTS
     ( SELECT *
     FROM РОБОТА Б
     WHERE  A.ТАБЕЛЬНИЙ_НОМЕР =    B.ТАБЕЛЬНИЙ_НОМЕР
     AND НОМЕР_ОБ`ЄКТА = 435 )
```

Оператор **NOT EXISTS** являється єдиним засобом вирішення запитів, що містять в умові слово «кожний».

**Запит.** Перерахувати робітників, призначених на кожний об`єкт.

**Друге формулювання цього ж запиту (подвійне заперечення):**

Перерахувати робітників, для яких **не** існує об`єкт, на яке вони **не** призначені.

```
SELECT  ТАБЕЛЬНИЙ_НОМЕР
FROM  РОБІТНИКИ A
     WHERE  NOT   EXISTS
```

```
        ( SELECT   НОМЕР_ОБ`ЄКТА
     FROM   ОБ`ЄКТИ B
     WHERE   NOT   EXISTS
        ( SELECT *
            FROM  РОБОТА  C
            WHERE  C. НОМЕР_ОБ`ЄКТА = B. НОМЕР_ОБ`ЄКТА
              AND   C.ТАБЕЛЬНИЙ_НОМЕР =
                  .ТАБЕЛЬНИЙ_НОМЕР )
)
```

**Агрегатні функції і підзапити**

Агрегатні функції можуть використовуватись тільки у виразі **SELECT** чи виразі **HAVING**. Але вираз SELECT, що містить вбудовану функцію, може бути частиною підзапиту.

**Запит:** У якого з робітників погодинна ставка вище середньої?

```
SELECT  ПІБ
        FROM  РОБІТНИКИ
        WHERE  ПОГОДИННА_СТАВКА >
          ( SELECT  AVG (ПОГОДИННА_СТАВКА)
                FROM РОБІТНИКИ)
```

В корельованих запитах також можуть використовуватись агрегатні функції .

**Запит**. **:** У якого з робітників погодинна ставка вище середньої погодинної ставки серед підпорядкованих одного і того ж бригадира?

```
SELECT  A. ПІБ
        FROM  РОБІТНИКИ A
        WHERE  A. ПОГОДИННА_СТАВКА >
          ( SELECT   AVG ( B. ПОГОДИННА_СТАВКА)
             FROM  РОБІТНИКИ B
             WHERE  B.БРИГАДИР = A.БРИГАДИР )
```

**Моделювання операцій реляційної алгебри**

**Операція Проекція**

```
SELECT DISTINCT R.a1, R.a2
       FROM R
```

**Операція Вибірка**

```
SELECT R.a1, R.a2
       FROM R
     WHERE  умова
```

**Операція Перетин**

```
SELECT R.a1, R.a2
       FROM R,S
       WHERE R.a1=S.b1 AND R.a2=S.b2
```

**Операція Різниця**

```
SELECT R.a1, R.a2
     FROM R
     WHERE NOT EXISTS
          (SELECT S.b1,S.b2
          FROM S
          WHERE S.b1=R.a2 AND S.b2=R.a1)
```

**Операція ділення відношень**

ВідношенняR визначено на множині атрибутів A, а відношення S – на множині атрибутів B, при чому **A⊇B** і **C=A – B**

Результат операції*ділення* **R:S** – набір кортежів відношення **R**, визначених на множині атрибутів **C**, які відповідають комбінації всіх кортежів відношення **S**.

$T1=\Pi_C( R );$

$T2=\Pi_C( (S \times T1) - R );$

$T=T1 - T2;$

$\Pi_C$ – Проекція по атрибутам C

**Операції зміни даних**

Значення даних можуть бути поміщені, скореговані і видалені з полів трьома командами мови SQL з категорії DML:

- INSERT (вставити),
- UPDATE (модифікувати),
- DELETE (видалити).

**Команда INSERT**

Дозволяє вносити в таблицю як окремі рядки шляхом вказівки значень коюного стовпця, так і множини рядків шляхом формування запиту, що визначає рядки, які вносяться.

```
INSERT  INTO < table name >
          VALUES ( < value >, < value > . . . ) ;
```

**Приклад**. Внести рядок в таблицю РОБОТА

```
INSERT  INTO  РОБОТА (
     ТАБЕЛЬНИЙ_НОМЕР, НОМЕР_ОБ'ЄКТА, ДАТА_ПОЧАТКУ)
          VALUES (  1284,  485,  13.11.2001 )
```

**Команда UPDATE**

Завжди застосовується по всім рядкам, що задовольняють умові виразу **WHERE**. Якщо вираз **WHERE** відсутній, то операція застосовується до кожного рядка таблиці.

**Приклад**. Підвищити на 5 відсотків погодинну ставку кожного маляра.

```
UPDATE   РОБІТНИКИ
SET  ПОГОДИННА_СТАВКА = 1,05 * ПОГОДИННА _СТАВКА
WHERE  СПЕЦІАЛЬНІСТЬ = `маляр`
```

**Команда DELETE**

Завжди застосовується до всіх рядків, що задовольняють умові виразу **WHERE**.

Якщо вираз **WHERE** відсутній, то виділяються всі рядки таблиці.

**Приклад**. Припустимо, що всі робітники, чий бригадир має табельний номер 1520, були звільнені, і необхідно видалити відповідні рядки з бази даних.

```
DELETE  FROM  РОБІТНИКИ
WHERE БРИГАДИР = 1520
```

**Визначення представлень даних**

**ПРЕДСТАВЛЕННЯ (VIEW)** – об'єкт даних, який не містить ніяких даних.

Це тип таблиці, зміст якої вибирається з інших таблиць з допомогою виконання запиту. Вони працюють в запитах і операторах DML точно так же як і основні таблиці, але не містять ніяких власних даних.

Оскільки значення в цих таблиць змінюється, то автоматично, їх значення можуть бути показані представленням

```
CREATE   VIEW   Ім'я_представлення [(Псевдоніми колонок)]  AS
         SELECT   - підзапит - критерій відбору
```

**Приклад**. Створити представлення даних, що показує всю інформацію про робітника, крім його погодинної ставки.

```
CREATE   VIEW   П_РОБІТНИКИ   AS
         SELECT   ТАБЕЛЬНИЙ_НОМЕР,
                  ФИО,  СПЕЦІАЛЬНІСТЬ,  БРИГАДИР
          FROM   РОБІТНИКИ
```

**Обмеження на оновлення представлень даних**

**УВАГА!** Представлення може змінюватись командами модифікації DML, але модифікація **не буде впливати на саме представлення**. Команди будуть насправді перенаправлені до базової таблиці.

Оновлювати представлення даних можна тільки тоді, коли його специфікація запиту така:

- В команді **FROM** є посилання тільки на одну таблицю. Таблиця, на яку посилається вираз **FROM**, повинна бути або базовою таблицею, або обновлюваним представленням бази даних
- Відсутні команди **GROUP BY, HAVING.**
- В команді **SELECT** перечисленні тільки імена стовпців, тобто нема розрахунків і вбудованих функцій), відсутнє ключове слово **DISTINCT**.
- В обновлюваній базовій таблиці відсутні стовпці **NOT NULL**, що не входять в представлення.

**Видалення представлень даних**

```
DROP  VIEW  < view name >
```

Нема необхідності спочатку видаляти весь зміст представлення даних як це робиться з базовою таблицею, тому що зміст представлення не являється створеним і зберігається на протязі певної команди.

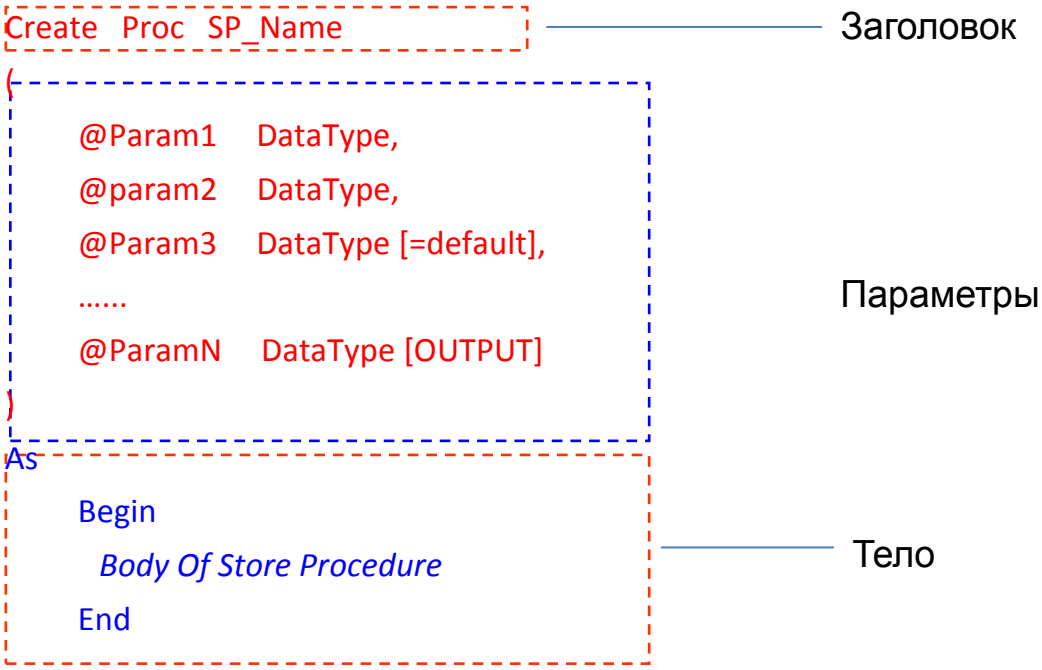
Користувач повинен бути  власником (тобто бути творцем) представлення, щоб мати змогу видалити його.

## Лекція 9. Процедури що зберігаються і тригери

### Властивості процедур що зберігаються (Stored Procedure – SP)

1. SP – об’єкт бази даних, що представляє собою набір SQL–інструкцій, який компілюється один раз і зберігається на сервері.
2. Клієнтський додаток може багаторазово виконувати SP, без пересилання його серверу і без повторної компіляції.
3. В порівнянні з динамічним SQL, SP підвищують продуктивність, зменшуючи мережевий трафік і завантаження процесора.
4. SP  дозволяють реалізовувати бізнес–логіку додатку.
5. С допомогою SP є можливість керувати доступом до таблиць БД.
6. Як і звичайна процедура SP складається з заголовка і тіла. Заголовок визначає ім’я процедури і її параметри. Тіло процедури - це SQL–інструкції.

**Створення процедури що зберігається**




**Оголошення змінних**

```
Declare @LastName varchar(50)
```

**Присвоєння значень змінним**

```
Select  @LastName = 'Smith',
            @FirstName = 'Bill'
```

— присвоєння значення декільком змінним

чи

```
Set  @LastName = 'Smith',
Set  @FirstName = 'Bill'
```

**Глобальні змінні**

```
@@error, @@rowcount
```

Передавання значень змінних відбувається процедурами – з допомогою параметрів.

**Керуючі конструкції**

**Коментарі:**

- Однорядкові (—) тире
- Багаторядкові **(/*……..*/)**

**Блоки програми**

**Begin**

*SQL команди і блоки*

**End**

**If** *логічний_вираз*

*SQL команди*

[**else**

*SQL команди*]

- Якщо SQL команд декілька, вони поміщаються в блок **Begin …. End**
- Конструкції **IF** можуть бути вкладеними

**If [NOT] EXIST** *запит*

*SQL команди*

[**else**

*SQL команди*]

### Організація циклів

**While** *логічний_вираз*

*SQL команди*

[Break] — вихід з циклу

[Continue] — новий цикл, без виконання подальших команд

### Приклад

```
Set @N = 10
While @N > 1
     Begin
     set @N = @N – 1
     set @F = @F + 1
     End
```

### Безумовний перехід

**GOTO** *мітка*

### Приклад:

```
IF @@Error <> 0 GOTO PROBLEM
…….
PROBLEM:
     Select ‘Unable to remove record’
     return 1
```

### Вибір одного з варіантів (Для MS SQL Server)

```
CASE input_expression
WHEN when_expression THEN Result
ELSE result_expression
END
```

**Пример:**

```
DECLARE @Type varchar(20)
SET @Type = 'Programming'

SELECT
CASE @Type
      WHEN  'Sql' THEN 'sqltutorials.blogspot.com'
      WHEN  'Programming' THEN
'programmingschools.blogspot.com'
      WHEN 'Travel' THEN 'travelyourself.blogspot.com'
      ELSE 'Not yet categorized'
      END
```

**Курсори**

1. Оголошення курсору
**DeclareCursor**

2. Відкриття курсору
**Open**

3. Позиціонування курсору на певному записі
**Fetch**

4. Витяг змісту поточного запису і його обробка

5. Повторення пп. 3, 4.

6. Закриття курсору і звільнення ресурсів
**Close**
**Deallocate**

**Пример**

```
Declare @Crsr Cursor                        --  об’явлення курсору
Set @Crsr = Cursor For                      --  ініціалізація
курсору
     Select Property, Value, Unit
     From Inventory
     Where Inventory.Type = @MyType
Open @Crsr                                  --  відкриття курсору
Fetch Next From @Crsr                       --  переміщення на
перший запис
Into @cPropery, @cValue, @cUnit
While (@@FETCH_STATUS =0)     -- перебір в циклі записів курсору
     Begin
          Set @cValue = @cValue * 1.5    --  обробка значень
          …….
          Fetch Next From @Crsr        -- переміщення на
```

```
          Into @cPropery, @cValue, @cUnit     -- наступний запис
     End
Close @Crsr                                   -- закриття курсору
Deallocate @Crsr                            -- звільнення ресурсів
```

**Приклад SP без параметрів**

```
-- ============================================
-- Author:          <Author,,Name>
-- Create date: <Create Date,,>
-- Description:<Description,,>
-- ============================================
Create   Proc   SP_Name
     As
     Begin
          Оператори процедури що зберігається
     End
```

**Виконання SP**

```
Execute SP_Name
```

```
Exec SP_Name
```

**Приклад SP з параметрами**

```
Create Proc FetchSpecificEmployeeInfo
(
     @EmpId  verchar(50)
)
As
Begin
     Select * from EmployeeInfo
     where EmployeeID= @EmpId
End
```

**Виконання SP**

```
Exec SP_FetchSpecificEmployeeInfo ‘1075’
```

**Приклад використання SP для вставки запису**

```
CREATE proc  InsertEmployeeInfo
(    @ID                varchar(20) ,
     @Name                   varchar (50) ,
     @Email                  varchar(50),
     @JoingDate              smalldatetime ,
     @Post                   varchar(50)
)
AS
Begin
    if exists(Select ID From  EmployeeInfo Where ID = @ID)
     Begin
     return 0
     End
    else
```

```
     Begin
        Insert into  EmployeeInfo
         (ID, Name, Email, JoingDate, Post)
          values  (@ID, @Name, @Email, @JoingDate, @Post)
      End
          Begin
      return 1
          End
End
```

**Приклад параметрів зі значеннями за замовчуванням**

```
CREATE PROC Update_Price
(
 @t VARCHAR(20)=Цукерки,
 @p FLOAT=0.1
)
 AS
     Begin
          UPDATE Product
          SET Price = Price / (1–@p)
          WHERE Type=@t
     End
```

**Звернення до процедури**

1. **Позиційне передавання параметрів**

```
Exec Update_Price ‘Фанта’, 0.32
```

2. **Передавання параметрів по імені**

```
Declare MyProduct VARCHAR(20)
Declare MyPrice float
Exec Update_Price @t = MyProduct, @p = MyPrice
```

**Приклад SP з вихідними параметрами**

```
CREATE PROC my_proc6
(
@m INT,
@s FLOAT OUTPUT
)
AS
     Begin
     SELECT @s=Sum(Товар.Ціна*Угода.Кількість)
     FROM Товар INNER JOIN Угода ON
Товар.КодТовару= Угода.КодТовару
     GROUP BY Month(Угода.Дата)
     HAVING Month(Угода.Дата)= @m
     End
```

Для *звернення до процедури* можна використовувати команди:

```
DECLARE @st FLOAT
EXEC my_proc6 1,@st OUTPUT
SELECT @st
```

**Використання вкладених процедур**

```
CREATE PROC my_proc8
(
@fam VARCHAR(20),
@kol INT OUTPUT
)
AS
     DECLARE @firm VARCHAR(20)
     EXEC my_proc7 @fam,@firm OUTPUT
     SELECT @kol=Sum(Угода.Кількість)
     FROM Клієнт INNER JOIN Угода ON
     Клієнт.КодКлієнта=Угода.КодКлієнта
     GROUP BY Клієнт.Фірма
     HAVING Клієнт.Фірма=@firm
```

## Тригери

**Засіб підтримання цілісності даних**:

– перевірка коректності введених даних і виконання складних обмежень цілісності даних, які важко, якщо взагалі можливо, підтримувати з допомогою обмежень цілісності, встановлених для таблиці;

– видача попереджень, які нагадують про необхідність виконання певних дій при оновленні таблиці, реалізованих певних чином;

– накопичення аудиторської інформації за допомогою фіксації відомостей про внесені зміни і тих особах, які їх виконували;

– підтримка реплікації.

**Події, що контролюються тригером**:

–Insert

– Update

– Delete

**Створення тригера:**

```
CREATE TRIGGER ім’я_тригера
BEFORE | AFTER <тригерна_подія>
ON < ім’я _таблиці>
AS
[
<тіло_тригера>
```

**Видалення тригера**

```
DROP TRIGGER ім’я_тригера
```

**Приклад тригера на подію Insert**

```
CREATE TRIGGER Тригер_ins
ON Угода
FOR INSERT
AS
IF @@ROWCOUNT=1
BEGIN
     IF NOT EXISTS(
     SELECT * FROM inserted
     WHERE
     inserted.кількість <=  (SELECT SUM(Склад.Остаток)
     FROM  Склад, Угода
     WHERE Склад.КодТовару= Угода.КодТовару))
     BEGIN
          ROLLBACK TRAN
          PRINT 'Відміна поставки: товару на складі немає
     END
END
```

**Рекомендації по програмуванню тригерів**

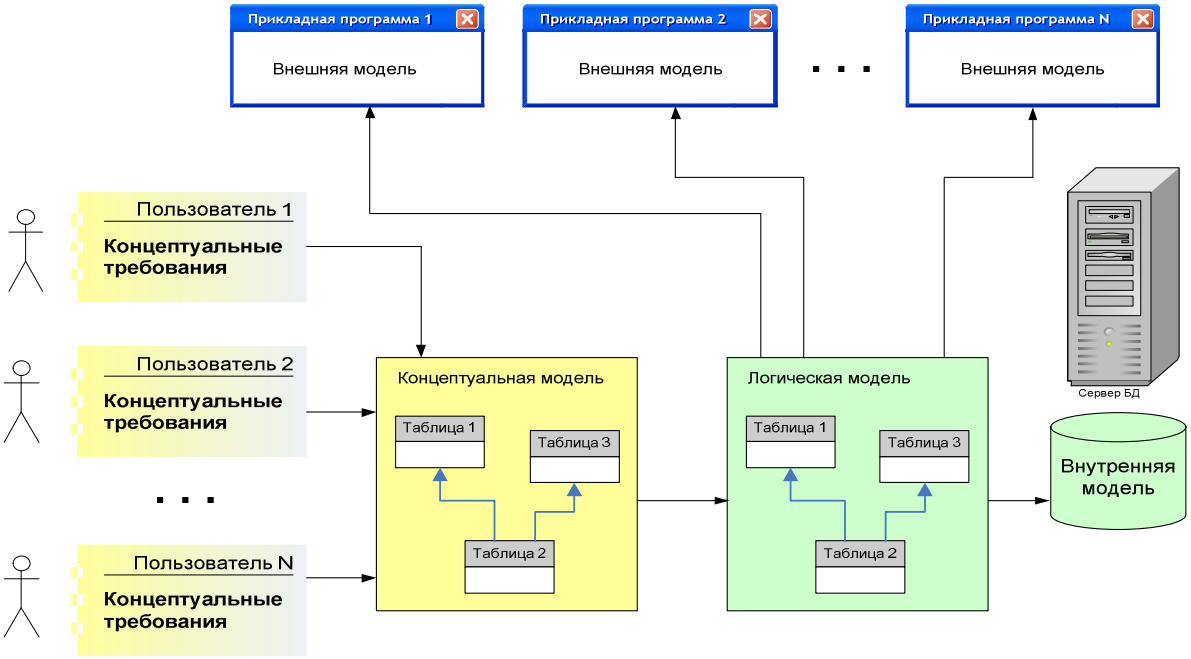
1. Використовувати тригери, тільки якщо вони дійсно необхідні.
2. Починати тригер з конструкції

   ```
   If @@rowcount = 0
        return
   ```

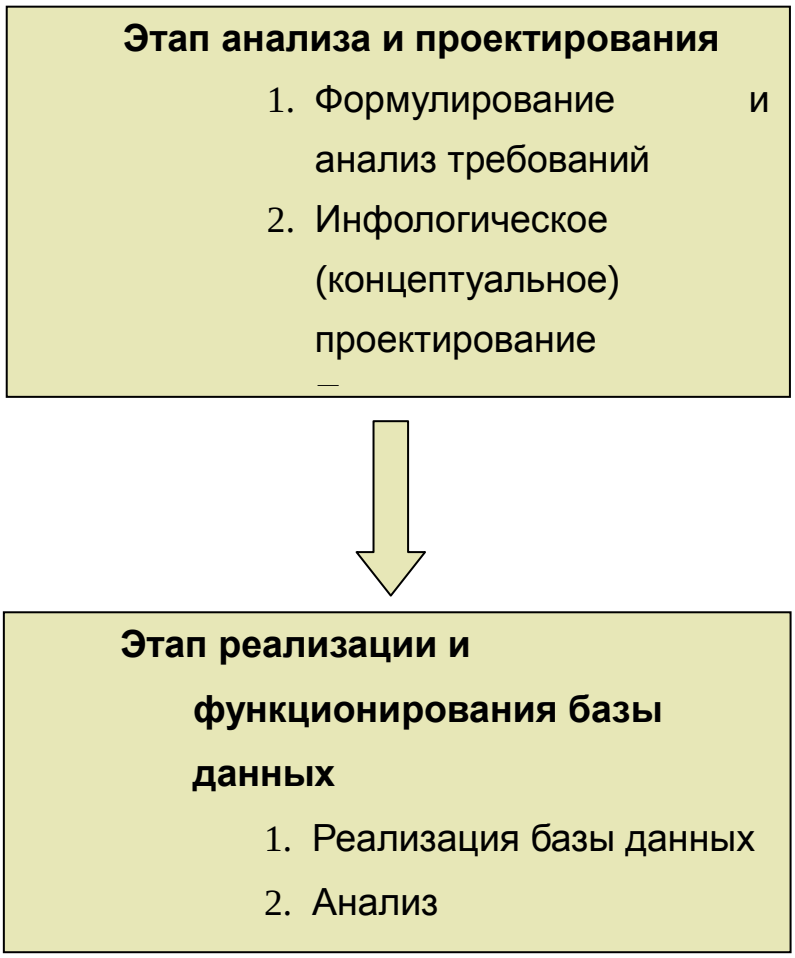
3. Не використовувати в тригері команд **Select** і **Print**

# Лекція 10. Концептуальне (інфологічне) проектування баз даних

## Класична методологія проектування БД



**Життєвий цикл системи баз даних**




**Етап 2 – інфологічне (концептуальне) проектування**

**Ціль** Побудова незалежної від СКБД інформаційної структури шляхом об`єднання інформаційних вимог користувачів.

**Результат –** Концептуальна модель – формалізований опис предметної області.

**Вимоги до моделі**

- адекватне відображення предметної області;
- несуперечливість;
- однозначне трактування моделі всіма її користувачами;
- легкість сприйняття різними категоріями користувачів;
- оптимальні розміри моделі;
- легкість модифікації;
- можливість композиції і декомпозиції моделі.
- бути обчислюваними, тобто сприйматись і оброблятись ЕОМ;

- використання «доброзичливих» для користувачів інтерфейсів, зокрема графічних;

**Метод виконання :**

1. Моделювання часткових представлень користувачів з використанням діаграм «сутність – зв`язок» (**Entity–Relationship Diagrams, ER–діаграм**):

   – визначення сутностей;

   – визначення атрибутів сутностей;

   – ідентифікація ключових атрибутів сутностей;

   – визначення зв`язків між сутностями.

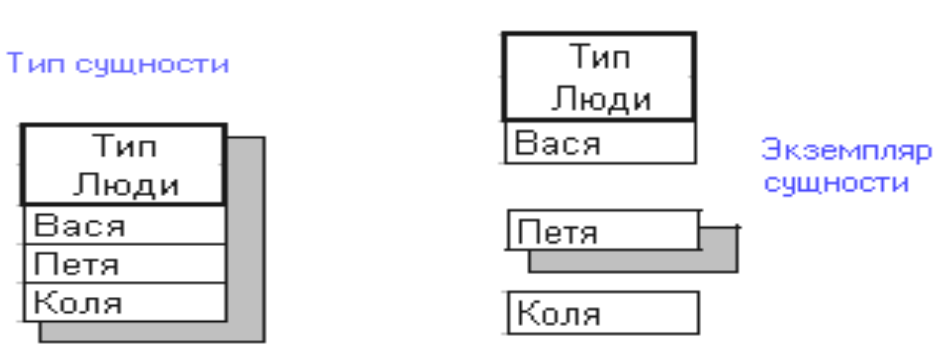
2. Інтеграція часткових представлень:

   Концептуальні вимоги окремих користувачів, визначені в стандартній формі (діаграми «сутність – зв`язок»), зливаються в єдине глобальне представлення. При цьому повинні бути виявлені і ліквідовані суперечливі і збиткові дані.

**Основні поняття інфологічного моделювання даних**

**Сутність (Entity) –** Реальний або уявляємий об`єкт (предмет або явище), що має суттєве значення для предметної області що розглядається і про який необхідно зберігати інформацію.

Тип і екземпляр сутності




**Атрибути**

Будь-яка характеристика сутності, значуща для предметної області і призначена, в загальному випадку, для класифікації, ідентифікації, кількісної характеристики і вираження стану сутності.

Атрибут виражає деяку окрему, що цікавить користувача властивість сутності, яке і характеризує її екземпляр.

Окремий атрибут визначається типом (числовий, текстовий, логічний, часовий та ін.), а також значенням, яке він приймає з деякої множини, що має назву *домен*.

Атрибут позначається іменем, унікальним в межах сутності.

Набір атрибутів повинен бути таким, щоб можна було розрізнити конкретні екземпляри сутності

Набір атрибутів, який однозначно ідентифікує конкретний екземпляр сутності, називають ключовим..

**Типи атрибутів:**

– описові; (Замовлення.Сума);

– що вказують; (Замовлення.НомерРахунку);

– допоміжні. (Замовлення.Покупець);

Атрибути, які успадковуються через зв`язок з родинною сутністю, являються зовнішніми ключами і служать для організації зв`язків між сутностями. Якщо зовнішній ключ сутності використовується в якості її первинного ключа або як частина складеного первинного ключа, то сутність являється ***залежною*** від родинної сутності, в противному випадку – ***незалежною*** від родинної сутності.

**Правила для атрибутів сутності:**

– Кожний атрибут повинен мати унікальне ім`я.

– Сутність може мати будь-яку кількість атрибутів.

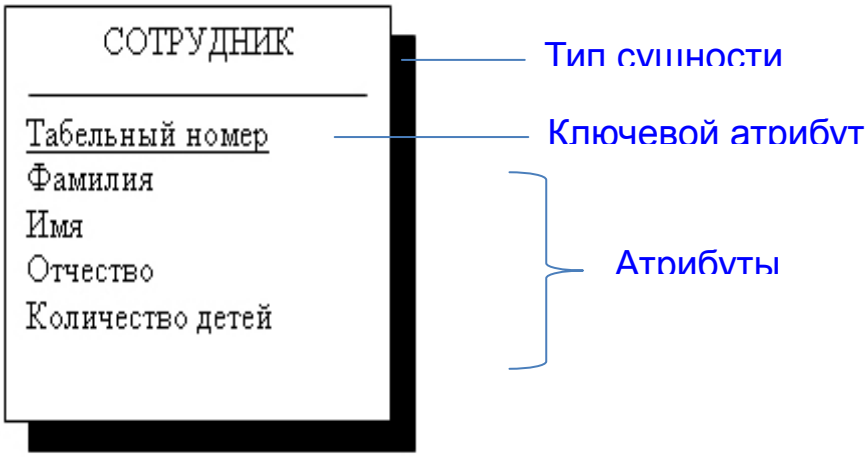
– Сутність може мати будь-яку кількість успадкованих атрибутів, але атрибут, що успадковується, повинен бути частиною первинного ключа сутності–предок.

– Для кожного екземпляра сутності повинно існувати значення кожного атрибута (правило не звернення в Null).

– Ні один з екземплярів сутності не може володіти більш ніж одним значенням для її атрибуту.

**Позначення**

- нотація Баркера;
- нотація IDEF1 (Erwin);
- нотація Yourdona .




**Сутність (Entity):**

- ***Незалежна*** кожний екземпляр сутності може бути однозначно ідентифікований без визначення його відношень з іншими сутностями.
- ***Залежна*** однозначна ідентифікація екземпляра сутності залежить від його відношення з іншою сутністю.

**Зв`язок (Relationship)**

- Іменована бінарна асоціація, функціональна залежність між двома сутностями, *значуща* для предметної області що розглядається.
  В даному випадку кожний екземпляр одної сутності асоційований з довільною (в тому числі і нульовою) кількістю екземплярів другої сутності і навпаки.
- Якщо сутності являється *залежною*, то зв`язок її з родинною сутністю називається ***ідентифікуючою***, в іншому випадку – ***не ідентифікуючою***.
- Зв`язок може існувати між двома різними сутностями або між сутністю і її ж самою ***(рекурсивний зв`язок)***.
- Зв`язку дається ім`я, яке виражається граматичною формою дієслова.. **Ім`я зв'язку завжди формується з точки зору предка**, так щоб могло бути утворено речення, якщо з`єднати ім`я сутності предка, ім`я зв`язку, вираз потужності і ім`я сутності–нащадка (наприклад "СТУДЕНТИ – здають – ЕКЗАМЕН").

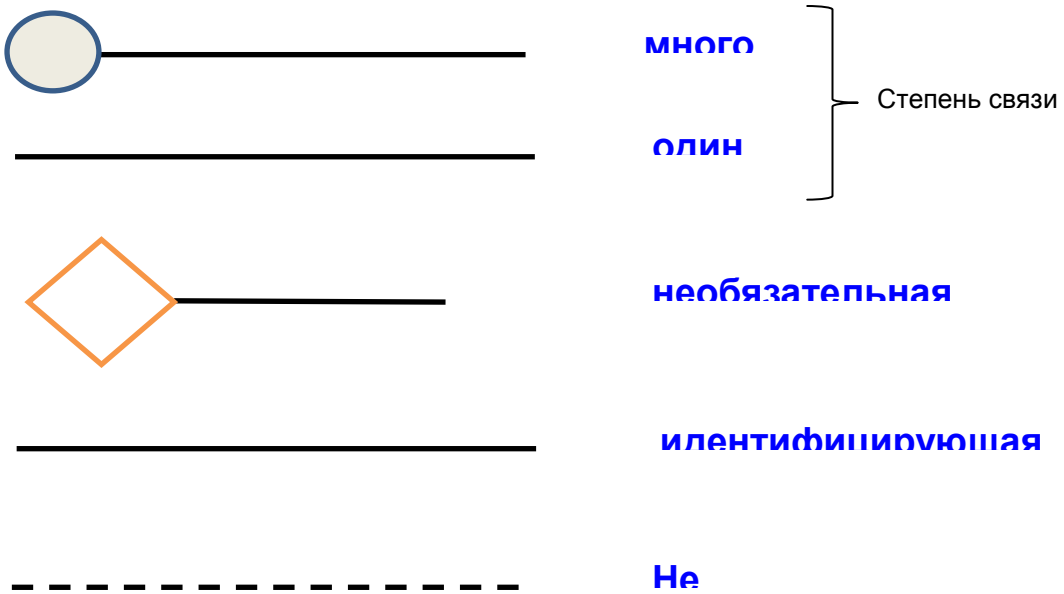
Зв`язок може бути **обов`язковим**, якщо в даному зв`язку повинен брати участь кожний екземпляр сутності, **не обов`язковим** — якщо не кожний екземпляр сутності повинен брати участь в ванному зв`язку. При цьому зв`язок може бути **обов`язковим з однієї сторони і не обов`язковим з іншої сторони**.

Бінарні зв`язки діляться на три типи по **ступеню**:

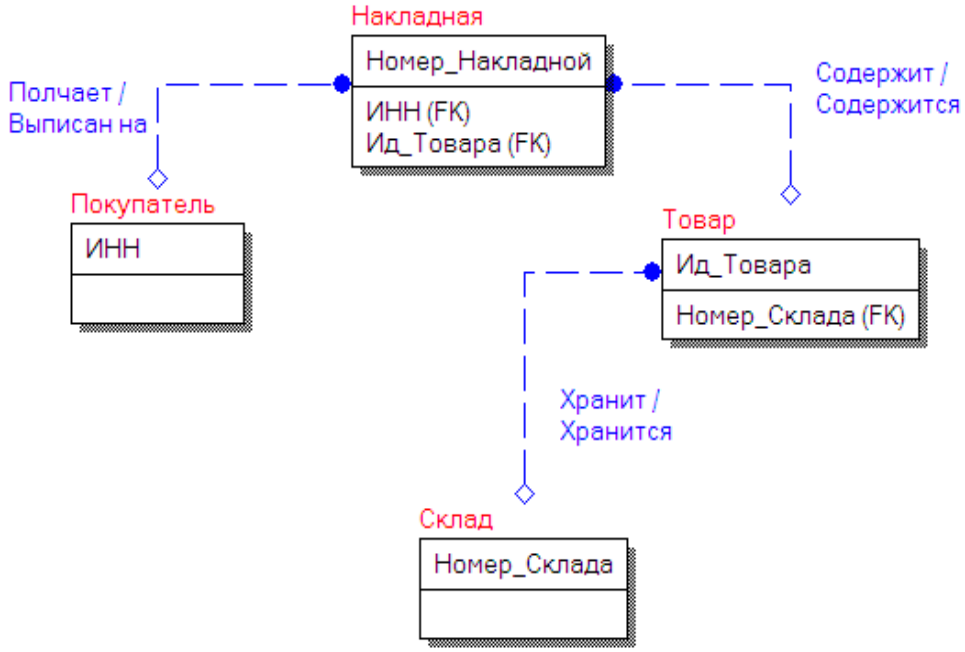
- **один – до – одного (1:1),**
- **один – до – багатьох (1:M),**
- **багато – до – багатьох (M:M).**

Між сутностями може бути встановлено декілька зв`язків, різних по змісту

**Позначення зв`язків в нотації IDEF 1x**

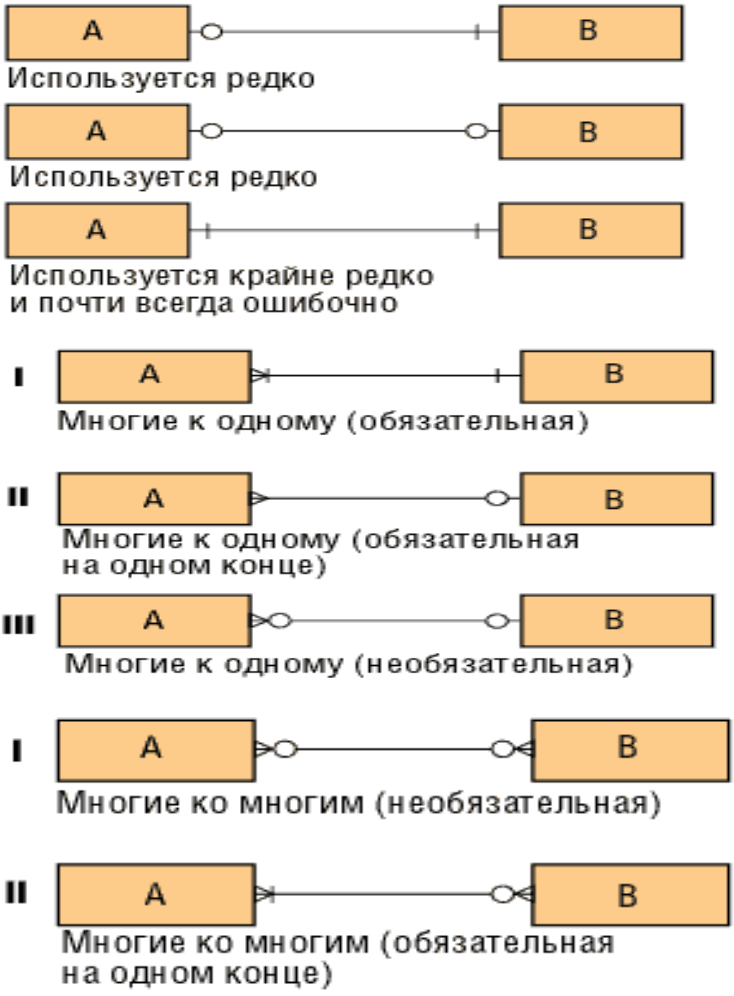



**Приклад зв`язків**

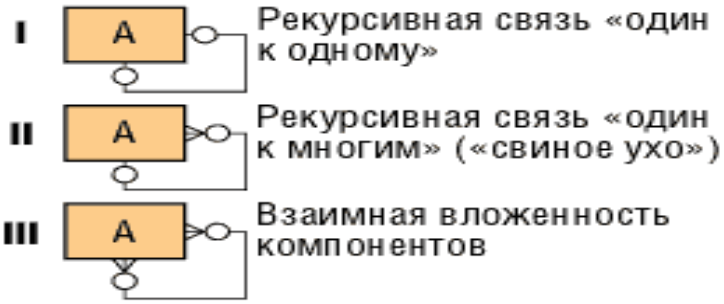


**Приклад рефлексивного зв`язку**

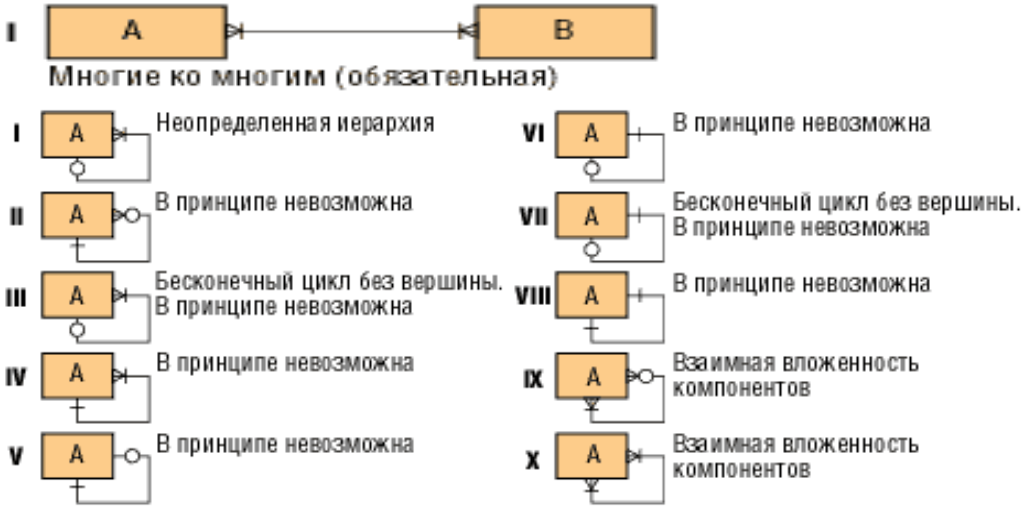

**Допустимі типи зв`язків**



**Допустимі типи рекурсивних зв`язків**




**Недопустимі типи зв`язків**




**Перевірка якості ER – моделі**

Перелік перевірочних запитань **для сутності**:

- Чи відображає ім’я сутності суть об`єкта?
- Чи нема перетинів з іншими сутностями?
- Чи є хоча б два атрибути?
- Всього атрибутів не більше восьми?
- Сутність визначена повністю?
- Чи є унікальний ідентифікатор?
- Чи є хоча б один зв`язок?
- Чи існує хоча б одна функція по створенню, пошуку, корегуванню, видаленню, архівуванню і використанню значення сутності?
- Чи ведеться історія змін?
- Чи має місце відповідність принципам нормалізації даних?
- Чи не має сутність занадто загальний зміст?

- Чи достатній рівень узагальнення, втілений в ній?

Перелік перевірочних запитань **для атрибутів**:

- Чи являється найменування атрибуту іменником однини, що відображає суть позначеної атрибутом властивості?
- Чи має атрибут тільки одне значення в кожний момент часу?
- Чи відсутні повторювані значення (або групи)?
- Чи описані формат, довжина, допустимі значення, алгоритм отримання і т.п.?
- Чи не може він бути пропущеним зв`язком?
- Чи нема де-небудь посилання на атрибут, як на "особливість проекту", яка при переході на прикладний рівень має зникнути?
- Чи є необхідність в історії змін?
- Чи залежать його значення тільки від даної сутності?
- Якщо значення атрибуту є обов`язковим, то чи завжди воно відомо?
- Чи залежить його значення тільки від якоїсь частини унікального ідентифікатора?
- Чи залежить його значення від значень деяких атрибутів, не включених в унікальний ідентифікатор?

Перелік перевірочних запитань **для зв'язків**:

- Чи є його опис для кожної сторони, яка бере участь, чи точно він відображає зміст зв`язку і чи вписується в прийнятий синтаксис?
- Чи беруть у ньому участь тільки дві сторони?
- Чи задана степінь зв`язку і обов`язковість для кожної сторони?
- Чи допустима конструкція зв`язку?
- Чи не відноситься конструкція зв`язку до рідко використовуваних?
- Чи не являється він збитковим?
- Чи не змінюється він з плином часу?
- Якщо зв`язок обов`язковий, чи завжди від відображає відношення до сутності, що представляє протилежну сторону?

**Перехід до реляційної моделі**

Сутність → Відношення

Атрибут сутності → Атрибут відношення

Первинний ключ сутності → PRIMARY KEY відношення

Зовнішній ключ сутності → FOREING KEY відношення

Обов`язковий зв`язок→ NOT NULL

## Лекція 11. Аналіз вимог до БД і їх формалізація

**Ціль –** збір, аналіз і формалізація вимог, що пред`являються до змісту і процесу обробки даних всіма відомими і потенційними користувачами бази даних.

**Результат –** технічне завдання (ТЗ), яке містить призначення, вимоги, обмеження, можливості, бізнес–процеси (функції), об`єм, кошторис витрат, терміни, показники економічної ефективності, виконавці.

**Підходи до проектування:**

- **Функціональний:**

Реалізує принцип руху «від завдань» і застосовується тоді, коли наперед відомі функції деякої групи осіб і комплексів завдань, для обслуговування інформаційних потреб яких створюється розглядувана БД. В даному випадку можна чітко виділити мінімально необхідний набір об`єктів предметної області, які повинні бути описані.

- **Предметний:**

інформаційні потреби майбутніх користувачів БД чітко не фіксуються. Вони можуть бути багатоаспектними і досить динамічними. Не можна точно виділити мінімальний набір об`єктів предметної області, які необхідно описати. В описі предметної області в даному випадку включаються також об`єкти і взаємозв`язки, які найбільш характерні і найбільш суттєві для неї. БД, що конструюється при цьому, називається предметною, тобто вона може бути використана при вирішенні багатьох різноманітних, раніше не визначених завдань.

Етап 1 –. **Формулювання і аналіз вимог**

**Ціль –** збір, аналіз і формалізація вимог, що пред`являються до змісту і процесу обробки даних всіма відомими і потенційними користувачами бази даних.

**Результат –** технічне завдання (ТЗ), яке містить призначення, вимоги, обмеження, можливості, бізнес–процеси (функції), об`єм, кошторис витрат, терміни, показними економічної ефективності, виконавці.

**Загальні вимоги при проектуванні бази даних:**

- Багатократне використання даних;
- Простота, легкість використання;
- Гнучкість при модифікації структури;
- Обробка незапланованих запитів;

- Невеликі витрати на експлуатацію;
- Мінімальна збитковість;
- Продуктивність, необхідна швидкість пошуку та доступу
- Секретність ;
- Контроль за цілісністю даних в базі;
- Захист від спотворень та збоїв;
- Стандартизація даних;
- Відновлення та реорганізація даних в базі.

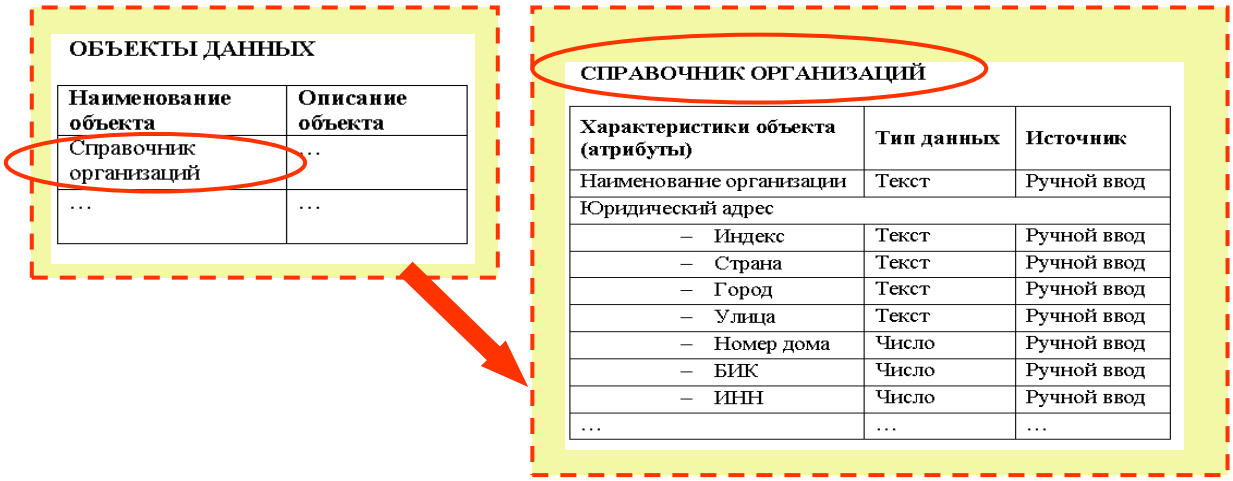
**Методи виконання:**

- вивчення існуючих форм документів, звітів, файлів, баз даних, програмного забезпечення;
- опитування персоналу різних рівнів, спеціалістів організації.

**Зміст вихідної інформації:**

- ім`я і опис об`єкта даних;
- елементи даних;
- тривалість зберігання і умови переведення в архів.

***Приклад змісту вихідної інформації***



ОБЪЕКТЫ ДАННЫХ

| Наименование объекта | Описание объекта |
|---|---|
| Справочник организаций | … |
| … | … |

СПРАВОЧНИК ОРГАНИЗАЦИЙ

| Характеристики объекта (атрибуты) | Тип данных | Источник |
|---|---|---|
| Наименование организации | Текст | Ручной ввод |
| Юридический адрес | | |
| – Индекс | Текст | Ручной ввод |
| – Страна | Текст | Ручной ввод |
| – Город | Текст | Ручной ввод |
| – Улица | Текст | Ручной ввод |
| – Номер дома | Число | Ручной ввод |
| – БИК | Число | Ручной ввод |
| – ИНН | Число | Ручной ввод |
| … | … | … |

**Складність формулювання вимог**

1. Аналітику складно отримати вичерпну інформацію для оцінки вимог до системи з точки зору замовника;

2. Замовник , в свою чергу, не має достатньої інформації про проблему обробки даних для того, щоб розуміти, що являється можливим, а що ні;

3. Аналітик стикається з надмірною кількістю детальних відомостей як про предметну область, так і про нову систему;

4. Специфікація системи через об`єм і технічні терміни часто незрозуміла для замовника;

5. У випадку зрозумілості специфікацій для замовника, вона буде недостатньою для проектувальників і програмістів, що створюють систему.

**Компоненти функціональної моделі:**

- функції, які система повинна виконувати;
- відношення між даними;
- залежна від часу поведінка системи (аспекти реального часу).

**Вимоги до функціональної моделі:**

1. Розподіл системи на окремі блоки:

- Кожний блок реалізує тільки одну функцію;
- Функції кожного блоку повинні бути легко зрозумілі незалежно від складності їх реалізації;
- Зв`язок між блоками повинен вводитись тільки при наявності зв`язку між відповідними функціями системи;

- Зв`язок між блоками повинен бути простим, наскільки це можливо, для забезпечення відносної незалежності блоків;

2. Ієрархічна організація блоків

3. Використання графічного представлення

**Діаграма потоків даних (DFD)**

**Основні особливості:**

1. Основний засіб моделювання функціональних вимог до системи;

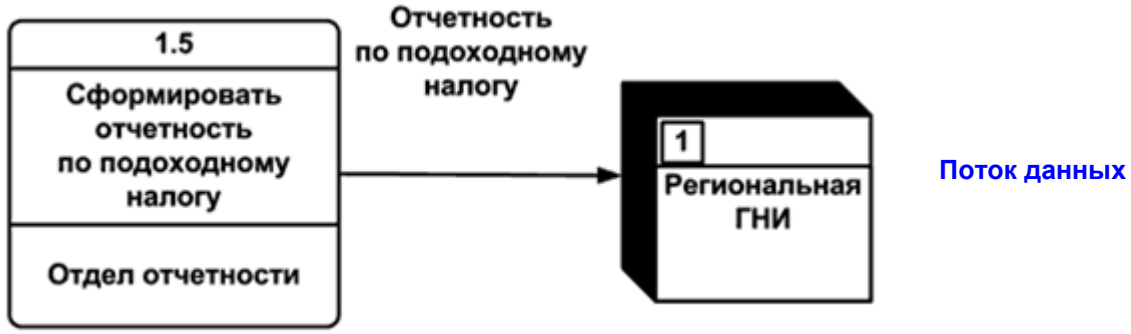
2. Головна ціль – продемонструвати, як кожний процес перетворює свої вхідні данні в вихідні, а також виявити відношення між цими процесами;

3. Модель системи визначається як ієрархія діаграм потоків даних, що описують асинхронний процес перетворення інформації від її вводу в систему до видачі споживачеві.

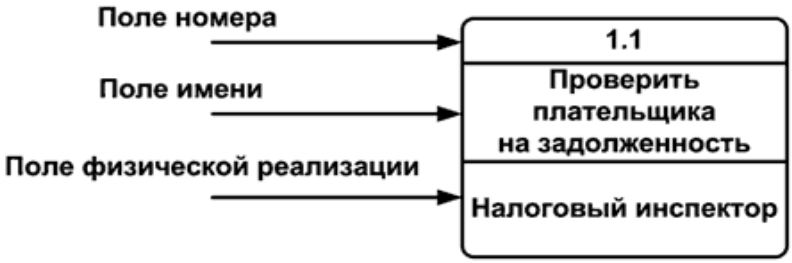
4. Джерела інформації (зовнішні сутності) породжують інформаційні потоки (потоки даних), що переносять інформацію до підсистем або процесів. Ті, в свою чергу, перетворюють інформацію і породжують нові потоки, які переносять інформацію до інших процесів або підсистем, накопичувачів даних або зовнішніх сутностей — споживачам інформації.

**Компоненти:**

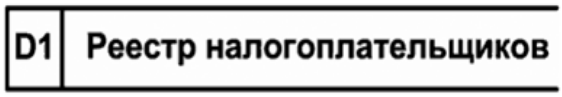
- Зовнішні сутності;
- Системи і підсистеми;
- Процеси;
- Накопичувачі даних;
- Потоки даних.

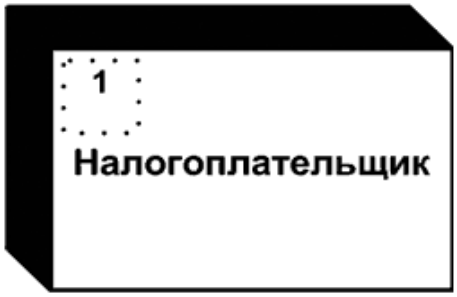


процесс




Хранилище




**Побудова ієрархії діаграм потоків даних:**

1. **Контекстна діаграма.**

Моделює систему найбільш загальним чином. Контекстна діаграма відображає інтерфейс системи з зовнішнім світом, а саме, інформаційні потоки між системою і зовнішніми сутностями,з якими вона повинна бути пов`язана. Вона ідентифікує ці зовнішні сутності в єдиний процес, відображаючи головну ціль системи (наскільки це можливо). Кожний проект повинен мати тільки одну контекстну діаграму, при цьому нема необхідності в нумерації її єдиного процесу.

2. Діаграма першого рівня

3. Діаграми другого рівня

**Ознаки закінчення процесу декомпозиції:**

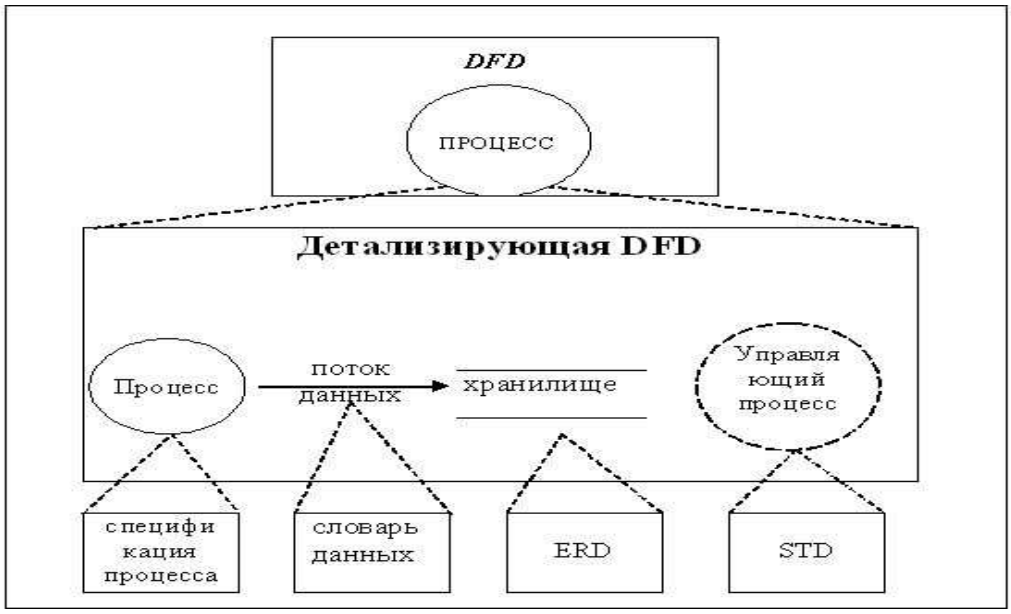
1. Наявність у процесу відносно невеликої кількості вхідних і вихідних потоків даних (2–3 потоки);

2. Можливість опису перетворення даних процесів у вигляді послідовного алгоритму;

3. Виконання процесом єдиної логічної функції перетворення вхідної інформації у вихідну;

4. Можливість опису логіки процесу з допомогою специфікацій невеликого об`єму (не більше 20–30 рядків).

**Зв`язок між функціональною моделлю і схемою БД**



## Лекція 12. Фізична організація бази даних

### Механізми середовища зберігання

Завдання механізму зберігання:

1**. При запам`ятовуванні нового об`єкту:**

– визначення місця розміщення нового об`єкту "фізичної" БД в просторі пам`яті;

– виділення необхідного ресурсу пам`яті;

– запам`ятовування цього об`єкту;

– формування зв`язків з іншими об`єктами.

**2. При пошуку об`єкта:**

– пошук місця розміщення об`єкта в просторі пам`яті по заданим атрибутам або «адресою»;

– вибірка об`єктів для обробки.

3. **При видаленні об`єкта**:

– видалення об`єкта зі звільненням пам`яті (фізичне видалення) або без звільнення (логічне видалення);

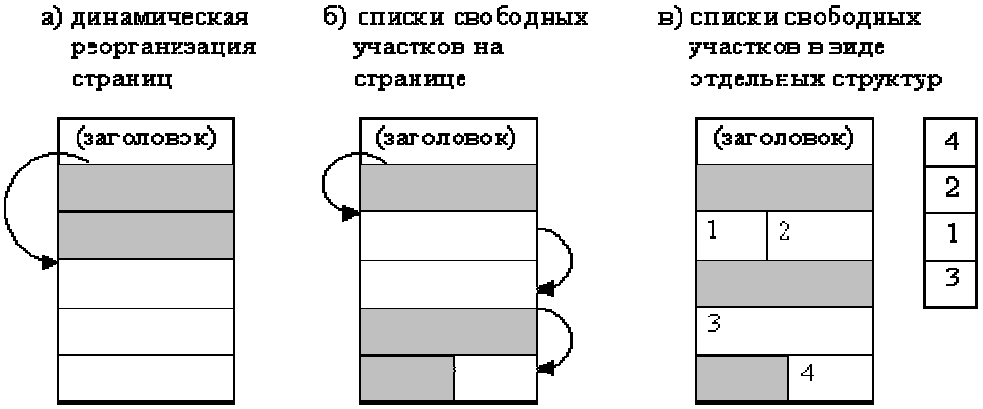
– руйнування зв`язків з іншими об`єктами.

### Структура простору пам`яті

Маніпуляція сторінками – ОС

Робота зі змістом сторінок – СУБД

Способи організації сторінок:




### Способи доступу до записів

- Послідовна обробка області БД.

- Доступ по ключу бази даних (КБД).
- Доступ по структурі.
- Доступ по первинному ключу.

**Індекс** – структура даних що встановлює відповідність значення атрибута що індексується і місцезнаходженням запису.

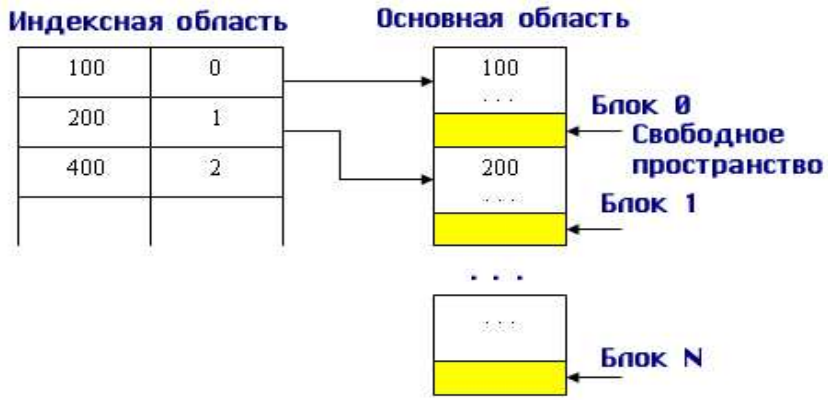
- Первинні і вторинні індекси.
- Одиночні і складені

**Способи організації індексів**

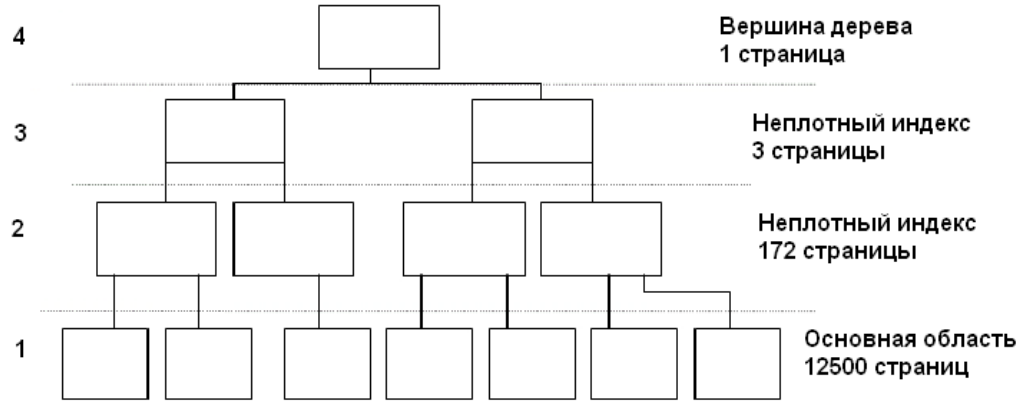
**Нещільний індекс –** один індексний запис для сторінки даних

Структура індексного запису

| **Значение ключа в первой записи блока** | **Номер блока с этой записью** |
|---|---|



**B – дерева**




Структура B–дерева має наступні **переваги**:

1. Всі блоки–листки в дереві однієї і тієї ж глибини, відповідно, пошук будь-якого запису в індексі займає приблизно один і той же час.

2. B–дерево автоматично підтримується в збалансованому вигляді.

3. B–дерева забезпечують хорошу продуктивність для широкого спектру запитів, включаючи пошук по конкретному значенню у заданому інтервалі.

4. Додавання, оновлення і видалення рядків виконується достатньо ефективно.

5. Продуктивність B–дерева однаково хороша для маленьких і великих таблиць, і не змінюється суттєво при рості таблиці

**Створення індексів**

```
CREATE INDEX index_name  ON table_name (column_name)
```

Використання індексу:

```
SELECT * FROM groups WHERE name = 'FI-21'
```

Кардинальність колонки (cardinality) таблиці – число дискретних різних значень колонки, які зустрічаються в рядках таблиці.

```
SELECT COUNT (DISTINCT колонка) FROM таблиця
```

Фактор селективності вибірки індексу – величина, обернена кардинальності індексної колонки:

**Фактори, що впливають на ефективність індексів**

**Хороші** кандидати для індексування:

1. колонки первинного ключа.

2. колонки зовнішнього ключа.

3. будь-які колонки, що містять унікальні значення;

4. колонки, запити або з`єднання по яким захоплюють від 5 до 10% рядків таблиці;

5. колонки, які часто входять як аргументи в функції агрегування;

6. колонки, які часто використовуються для перевірки правильності введення даних в програмах введення/редагування.

**Фактори, що впливають на низьку ефективність індексів**

1. Таблиці маленького розміру (менше п`яти фізичних сторінок).

2. Інтенсивні оновлення таблиць в пакетному режимі.

3. Асиметрія значень ключів (Skewness of keys).

**Погані кандидати на індексування:**

1. Колонки з низькою селективністю.

2. Колонка має багато невизначених значень (NULL–значення).

3. Колонки з часто змінюваними значеннями.

4. Значна довжина індексних колонок.

**Загальні правила для створення індексів:**

1. Повинно бути в наявності обмеження PRIMARY KEY.

2. Не варто створювати для таблиці декілька складених індексів, що містять одні і ті ж колонки, але в іншому порядку.

3. Ні в якому випадку не можна допускати, щоб у декількох індексів для однієї таблиці була однакова лідируюча таблиця.

Складений індекс бажаніше, якщо:

1. Декілька стовпців з низькою селективністю в комбінації один з одним можуть дати набагато більшу селективність.

2. Якщо в запитах часто використовуються тільки стовпці, що беруть участь в індексі, може взагалі не звертатись до таблиці для пошуку даних.

**Кластер с**труктура пам`яті, в якій зберігається набір таблиць (в одних і тих же сторінках даних). Ці таблиці повинні мати загальні стовпці,що використовуються для з’єднання .

| EMPLOYEE | | | | DEPARTMENT | | |
|---|---|---|---|---|---|---|
| EMPNO | ENAME | LNAME | DEPNO | DEPNO | DNAME | LOC |
| 996 | Козырев | Сергей | 10 | 10 | Торговля | Киев |
| 997 | Харченко | Алексей | 20 | 20 | Консалтинг | Бородянка |

| CLUSTER | | | |
|---|---|---|---|
| DEPNO | | | |
| 10 | DNAME | LOC | … |
| | Торговля | Москва | … |
| | EMPNO | ENAME | LNAME |
| | 996 | Козырев | Сергей |
| | … | … | … |
| | | | |
| 20 | DNAME | LOC | … |
| | Консалтинг | Черноголовка | … |
| | EMPNO | ENAME | LNAME |
| | 997 | Сапегин | Алексей |

**Кластерний ключ** – це стовпчик або набір стовпців (полів запису), загальних для таблиць що входять до кластеру.

**Переваги:**

1. Зменшується обмін з диском, покращується час доступу до таблиць з кластеру і їх з`єднання.

2. Значення кластерного ключа зберігається тільки один раз для кластеру не залежно від того, скільки рядків різних таблиць мають це значення кластерного ключа, за рахунок чого досягається економія пам`яті.

**Недоліки:**

1. Кластеризація уповільнює виконання операцій, в яких потрібно просканувати усю таблицю, так як вона може викликати розкидання рядків однієї таблиці по множині фізичних сторінок;

3. Кластеризація може уповільнити введення даних (Insert);

4. Кластеризація може уповільнити модифікацію даних в тих колонках, які поміщені в кластер.

## Лекція 13. Транзакції та їх обробка

### Визначення

Транзакція – це група інструкцій SQL, виконуваних як єдине ціле. Невиконання однієї інструкції призводить до відновлення стану даних на початок транзакції.

### Реалізація

- Транзакція складається з деякого набору (блоку) команд. Якщо хоча б одна інструкція була виконана з помилкою, то відбувається відновлення транзакції, тобто відміняється виконання всіх її операторів.
- Всі операції транзакції запам`ятовуються в деякому журналі – журналі транзакцій.
- Кожна окрема команда *Transact–SQL* виконується як самостійна транзакція.
- Завершення транзакції фіксується (оператор **commit**) в базі даних, яка переходить в новий узгоджений стан, а всі оновлення, виконані в базі даних за час транзакції, фіксуються, тобто стають постійними.
- Якщо оновлення даних в базі було перервано якою-небудь умовою помилки, то виконується оператор **rollback** і будь-які зміни відміняються.

### Властивості транзакцій

### Властивості ACID

- Атомарність (Atomicity)
- Узгодженість або несуперечливість (Consistency)
- Ізольованість (Isolation)
- Довговічність (Durability)

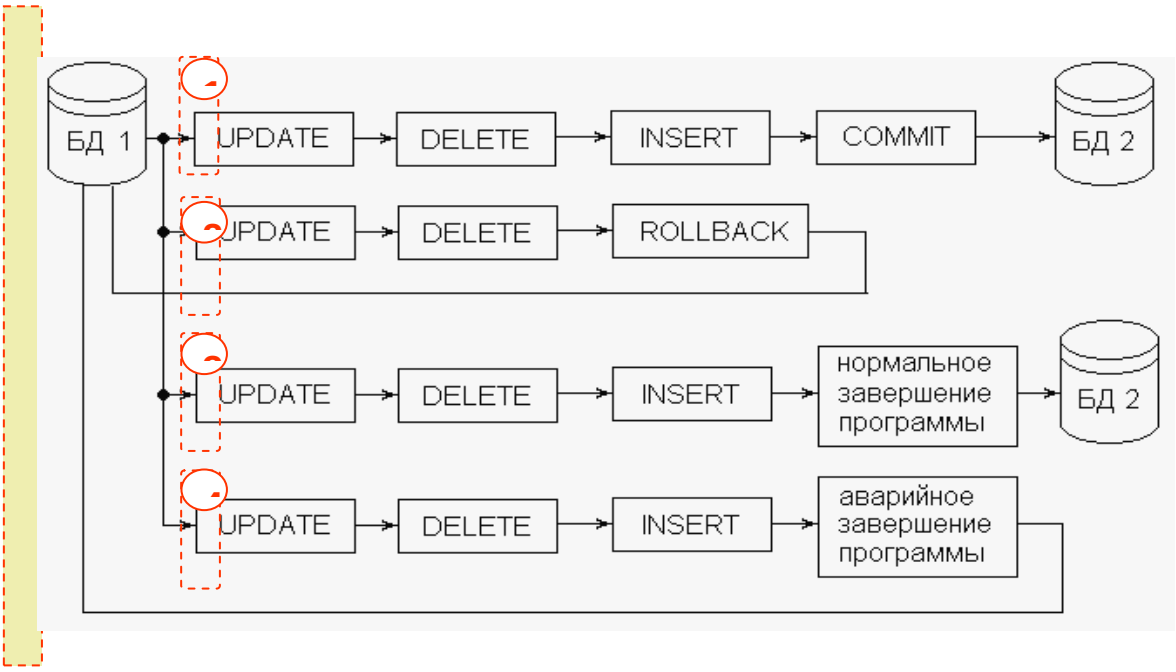
### Типи транзакцій

1. Явні транзакції
2. Автоматичні транзакції
3. Неявні транзакції
   - alter table — зміна таблиці;
   - create — створення будь-якого об`єкта бази даних;
   - delete — видалення рядків даних з таблиці;

- drop — видалення об`єктів бази даних;
- fetch — витяг стовпця з курсору;
- grant — дозвіл доступу до об`єктів бази даних;
- insert — вставка рядків в таблицю;
- open — відкриття курсору;
- revoke — неявне відхилення доступу до об`єктів бази даних;
- select — вибірка даних з однієї або декількох таблиць;
- truncate table — усікання таблиці;
- update — зміна змісту таблиці.
- Розподіл транзакції (Distributed transaction).

4. Вкладені транзакції (Nested transaction)

**Можливі схеми завершення транзакцій**




**Проблеми одночасного доступу до даних**

- Проблема втрати результатів оновлення
- Проблема незафіксованої залежності (читання "брудних" даних, неакуратне зчитування)
- Неповторювані зчитування
- Фантомні зчитування(фіктивні елементи)
- Власне несумісний аналіз

**Блокування транзакцій**

**Блокування (lock)** – тимчасово обмеження, що накладається, на виконання деяких операцій обробки даних

**Ідея –** у випадку, коли для виконання деякої транзакції необхідно, щоб деякий об`єкт не змінювався непередбачувано і без відома цієї транзакції, такий об`єкт блокується

**Результат –** ефект блокування полягає в тому, щоб заблокувати доступ до цього об`єкту зі сторони інших транзакцій, а значить, запобігти непередбачуваним змінам цього об`єкту.

**Рівні блокувань (*протокол розроблений ANSI*)**

**Рівень 0** – заборона «забруднення» даних (no trashing of data). Змінювати дані може тільки одна транзакція. Якщо іншій транзакції необхідно змінити ті ж дані, вона повинна дочекатись завершення першої транзакції.

**Рівень 1** – заборона на «брудне» читання (no dirty reads). Якщо транзакція почала зміну даних, то ніяка інша транзакція не зможе прочитати і дані до тих пір, поки перша транзакція не завершиться

**Рівень 2** – заборона неповторюваного читання (no nonrepeatable reads). Якщо транзакція зчитує дані, то ніяка інша транзакція не зможе їх змінити. Таким чином, при повторному читанні даних вони будуть знаходитись в вихідному стані.

**Рівень 3** – заборона фантомів (no phantom) Якщо транзакція звертається до даних, то ніяка інша транзакція не зможе додати/видалити рядки, які можуть бути зчитані при виконанні транзакції. Реалізація цього рівня досягається блокуванням діапазону ключів.

**Механізми фіксацій (відновлень) транзакцій**

**Принципи відновлення даних**

- Результати зафіксованих транзакцій повинні бути збережені в відновленому стані бази даних;
- Результати незафіксованих транзакцій повинні бути відсутніми в відновленому стані бази даних.

**Ситуації, що вимагають відновлення стану бази даних**

- Індивідуальне відновлення транзакцій
- Відновлення після раптової втрати змісту оперативної пам`яті (м’який збій).
- Відновлення після поломки основного зовнішнього носія бази даних (жорсткий збій).

**Журналізація та буферизація**

**Журнал транзакцій** – особлива частина бази даних, недоступна користувачам СКБД і підтримувана з особливою ретельністю, в яку поступають записи про всі зміни основної частини БД

Види буферів

- буфер журналу;
- буфер сторінок оперативної пам`яті

**Основний принцип погодженої політики виштовхування буфера журналу і буферів сторінок бази даних** . Запис про зміну об`єкта бази даних повинен потрапляти у зовнішню пам`ять журналу раніше, ніж змінений об`єкт потрапляє у зовнішню пам`ять бази даних.

**Протокол журналізації** Write Ahead Log (WAL) – «пиши спочатку в журнал». Якщо потрібно виштовхнути у зовнішню пам’ять змінений об’єкт бази даних, то перед цим потрібно гарантувати виштовхування у зовнішню пам’ять журналу запису про його зміни.

**Принципи відновлення бази даних**

**Індивідуальне відновлення транзакцій**

Можливе тільки для незавершених транзакцій.

Всі записи в журналі від даної транзакції пов’язуються у зворотній список.

**Технологія відновлення:**

1. Вибирається черговий запис зі списку даної транзакції.
2. Виконується протилежна за змістом операція:

- Замість операції INSERT виконується відповідна операція DELETE,
- Замість операції DELETE виконується INSERT,
- Замість прямої операції UPDATE обернена операція UPDATE, що відновлює попередній стан об’єкта бази даних.

**Відновлення після жорсткого збою**

Основа відновлення – журнал транзакцій і архівна копія бази даних

Технологія відновлення:

1. Зворотне копіювання бази даних із архівної копії.
2. Для всіх завершених транзакцій виконується redo,тобто операція повторно виконується в прямому сенсі. Тобто по журналу транзакцій у прямому напрямі виконуються всі операції

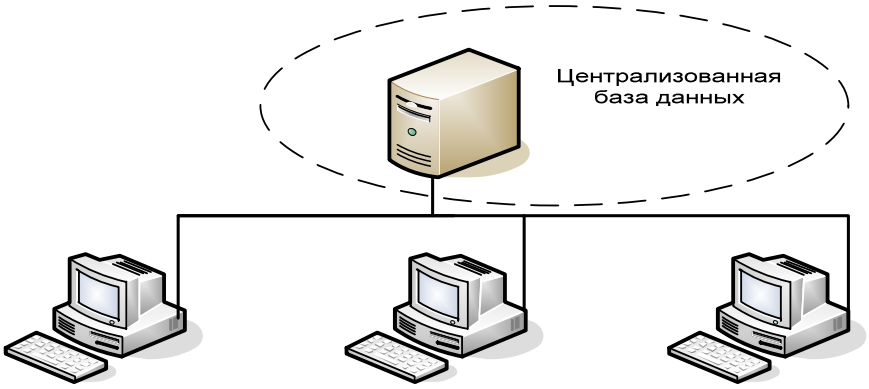
3. Для транзакцій, які не завершились до моменту збою, виконується відновлення.

## Лекція 14. Розподілена обробка даних

### Система розподіленої обробки даних

Режим розподіленого доступу до централізованої БД:

– паралельний доступ до однієї бази даних декількох користувачів;
– база даних фізично знаходиться на одній машині

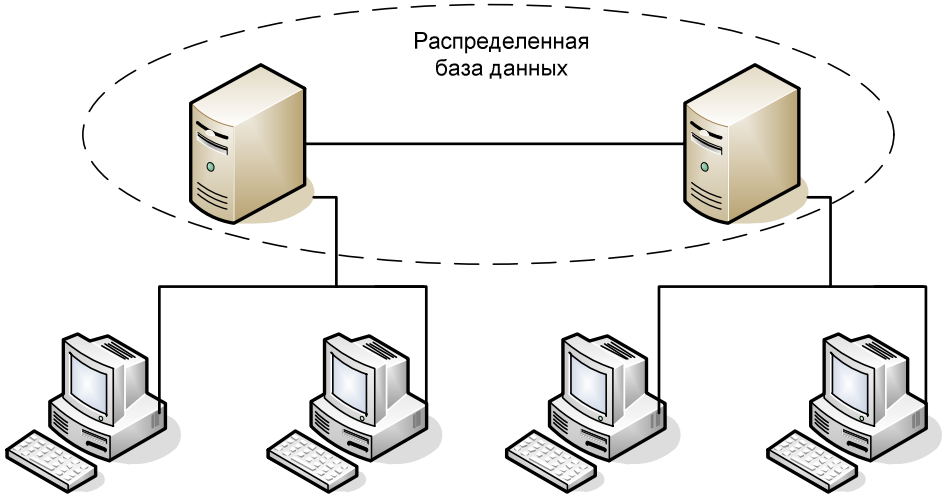



### Система розподілених баз даних

Режим паралельного доступу до розподіленої БД:

– база даних фізично розподілена по декільком комп'ютерам, розміщених в мережі;

– до бази даних можливий паралельний доступ декількох користувачів.




### Моделі «клієнт–сервер» в технології баз даних

Основні програмні процеси:

- «Клієнт» – сторона, запитуюча функції/обслуговування
- «Сервер» – сторона, що надає функції/обслуговування

**Особливість** – На рівні програмного забезпечення розподіл системи на клієнта і сервер являється логічним, а саме процеси клієнта і сервера можуть фізично розміщуватись як на одній, так і на різних машинах.

**Основний принцип** технології «клієнт–сервер» стосовно технології баз даних – розподіл функцій додатку на групи

**Основні функції стандартного інтерактивного додатку :**

1) функції вводу і відображення даних (Інтерфейс користувача, **Presentation Logic**, Логіка представлення);
2) прикладні функції, що визначають основні алгоритми вирішення завдань додатку (**Business Logic**, Бізнес–правила);
3) функції обробки даних всередині додатку(**Database Logic**);
4) функції керування інформаційними ресурсами (**Database Manager System**);
5) службові функції, що відіграють роль зв'язку між функціями перших чотирьох груп.

**Варіанти архітектури додатку**

**Централізована –** всі частини додатку розміщуються в єдиному середовищі і комбінуються всередині одної виконуваної програми.

**Децентралізована –** всі завдання можуть бути по різному розподілені між серверними і клієнтськими процесами.

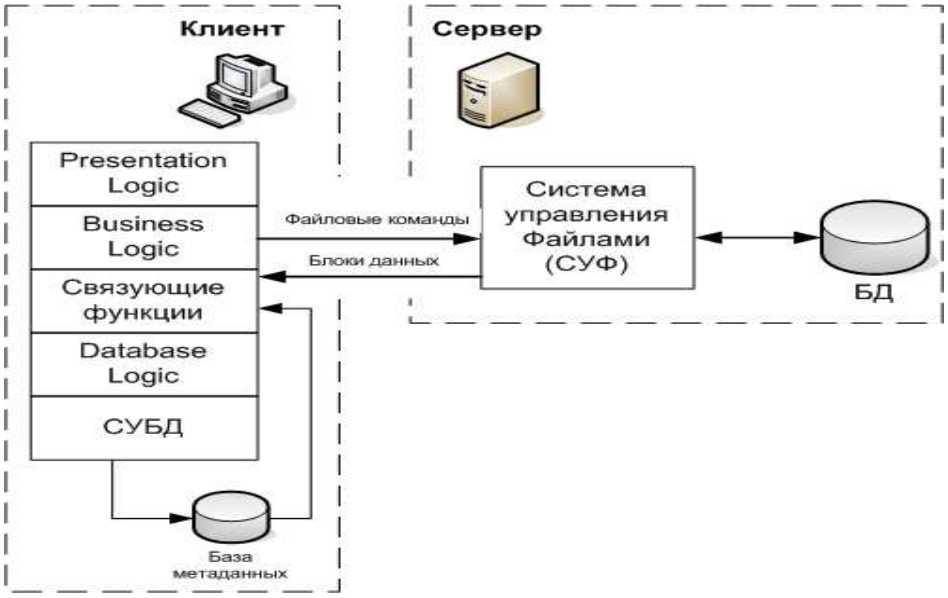
**Різниця архітектурних реалізації** визначається тим**:**

- як логічні компоненти бази даних розподілені в мережі;
- які механізми використовуються для зв'язку компонентів між собою.

**Варіанти (моделі) побудови архітектури додатку:**

- **дворівневі**
  - модель файлового сервера (модель віддаленого керування даними);
  - модель віддаленого доступу до даних;
  - модель сервера баз даних;
- **трирівнева** модель сервера додатків.

**Модель файлового сервера (модель віддаленого керування даними)**



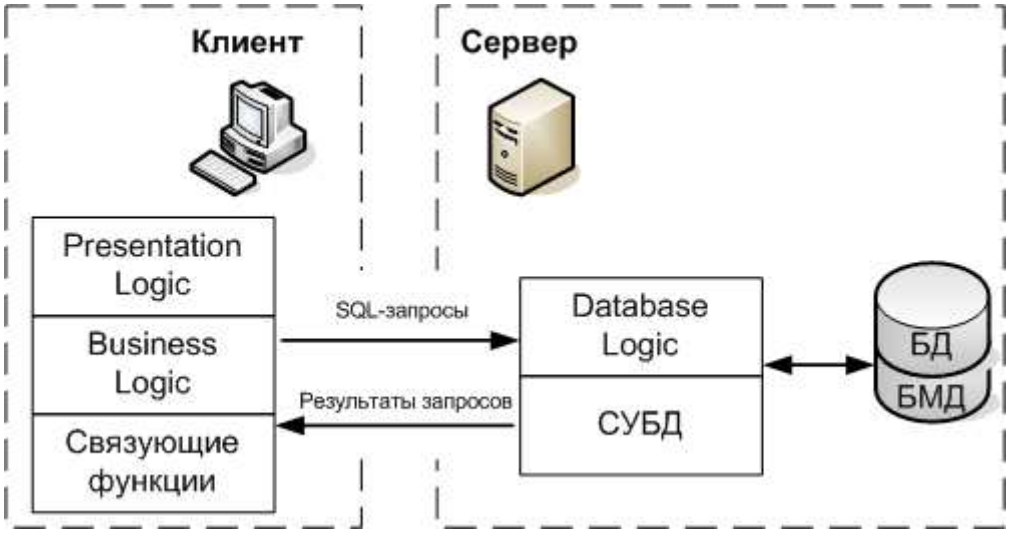

**Переваги**

- розділення монопольного додатку на два взаємодіючі процеси;
- сервер (серверний процес) може обслуговувати багатьох клієнтів

**Недоліки**

- високий мережевий трафік, який пов’язаний з передачею по мережі багатьох блоків і файлів, необхідних додатку;
- вузький спектр операцій маніпулювання з даними, який визначається тільки файловими командами. Драйвер «вміє» обробляти тільки прості запити;
- відсутні механізми оптимізації запитів і збереження результатів в кеш;
- відсутність адекватних засобів безпеки доступу до даних (захист тільки на рівні файлової системи).

**Модель віддаленого доступу до даних (Remote Data Access, RDA)**




**Переваги**

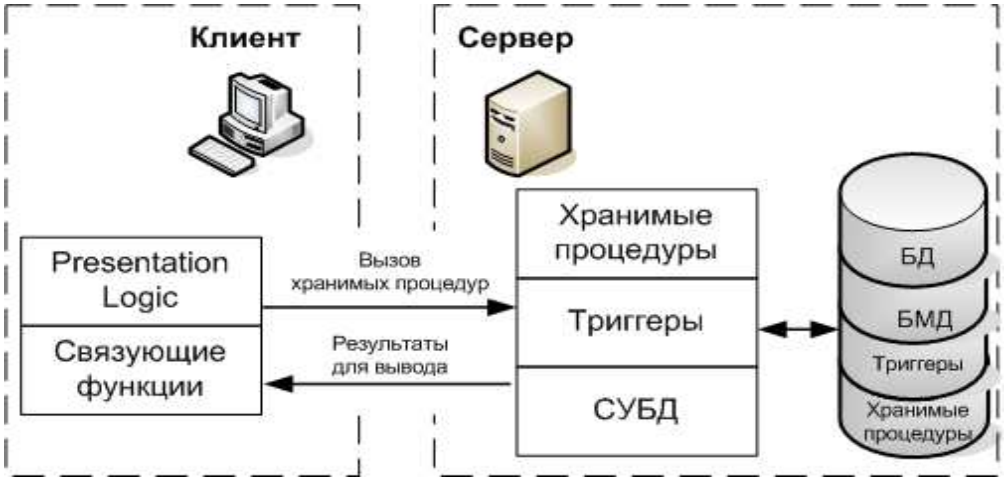
- різко зменшується завантаження мережі, так як по ній від клієнту до серверу передаються не запити на введення–виведення в файловій термінології, а запити на SQL, і їх об'єм суттєво менше. У відповідь на запити клієнт отримує тільки дані, релевантні (які відповідають за змістом) запиту, а не блоки файлів, як у FS–моделі,
- більш гнучкий розподіл доступу до даних (на рівні окремих записів);
- уніфікація (стандартизація) інтерфейсу «клієнт–сервер», стандартом при звертанням додатку–клієнта і серверу стає мова SQL.

**Недоліки**

- запити на мові SQL при інтенсивній роботі клієнтських додатків можуть суттєво завантажити мережу;
- складність адміністрування при зміні бізнес-правил, викликана зайвим дублюванням коду додатків;

сервер грає пасивну роль, тому функції керування інформаційними ресурсами повинен виконувати клієнт

**Модель сервера баз даних (Server DataBase)**



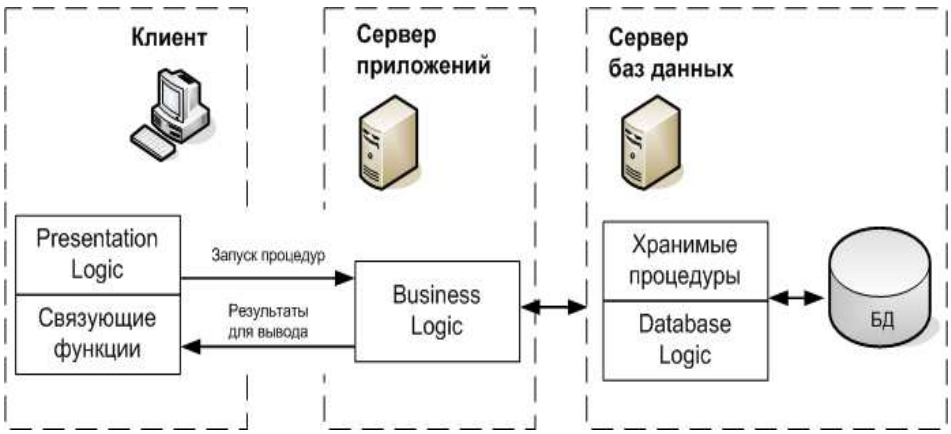

**Переваги**

- зниження мережевого трафіку;
- реалізація загальної для клієнтів бізнес-логіки засобами сервера, що суттєво зменшує дублювання алгоритмів обробки даних в різних клієнтських додатках;
- спрощення адміністрування серверу БД (наявність вбудованих механізмів вирішення адміністративних завдань по обслуговуванню сервера, наприклад реплікації даних між серверами, автоматичного запуску завдань, оповіщень та ін.).

**Недоліки**

- дуже велике завантаження сервера;
- в нинішній час комерційних СКБД нема зручного інструменту для написання і відлагодження процедур що зберігаються і тригерів;
- відсутність стандартів на процедури що зберігаються.

**Модель сервера додатків (Application Server, AS)**




**Переваги**

- подолання фундаментальних обмежень дворівневої архітектури по кількості одночасно підключених клієнтів;
- модель має більшу гнучкість, ніж дворівневі моделі. Найбільш помітні переваги в випадках виконання складних аналітичних розрахунків над базою даних (OLAP, On–line analytical processing);
- підвищення переносимості системи і її масштабованості, оскільки більша частина бізнес–логіки клієнта ізольована від можливостей вбудованого SQL, реалізованого в конкретній СКБД, і може бути виконана на стандартних мовах програмування.

**Недолік**

- складність програмування систем баз даних

## Лекція 15. Безпека баз даних (на прикладі SQL Server)

### Архітектура системи безпеки SQL Server

Система безпеки SQL Server базується на *користувачах* і *облікових записах*. Користувачі проходять наступні два етапи перевірки системою безпеки. На першому етапі користувач ідентифікується по імені облікового запису і паролю, тобто проходить аутентифікацію. Якщо дані введені правильно, користувач підключається до SQL Server. Підключення до SQL Server, або *реєстрація*, не дає автоматичного доступу до баз даних. Для кожної бази даних сервера *реєстраційне ім'я* (або обліковий запис — login) повинне відображатися *в ім'я користувача бази даних* (user). На другому етапі, на основі прав, виданих користувачеві як користувачеві бази даних (user), його реєстраційне ім'я (login) отримує доступ до відповідної бази даних. У різних базах даних *login* одного і того ж користувача може мати однакові або різні імена *user* з різними правами доступу.
Для доступу *програм* до баз даних їм також знадобляться права. Найчастіше програмам видаються ті ж права, які надані користувачам, що запускають ці програми. Проте для роботи деяких програм необхідно мати фіксований набір прав доступу, не залежних від прав доступу користувача. SQL Server дозволяє надати такі із застосуванням спеціальних *ролей програми*.
Отже, на рівні сервера система безпеки оперує наступними поняттями:

- аутентифікація (authentication);
- обліковий запис (login);
- вбудовані ролі сервера (fixed server roles).

На рівні бази даних використовуються наступні поняття:

- користувач бази даних (database user);
- фіксована роль бази даних (fixed database role);
- призначена для користувача роль бази даних (users database role);
- роль програми (application role).

### Компоненти структури безпеки

Фундаментом системи безпеки SQL Server є *облікові записи*(login), *користувачі* (user), *ролі* (role) і *групи* (group). Користувач, що підключається до SQL Server, повинен ідентифікувати себе, використовуючи *обліковий запис*. Після того, як клієнт успішно пройшов аутентифікацію, він дістає доступ до SQL Server. Для діставання доступу до будь-якої бази *даних обліковий запис користувача* (login) відображається в *користувача даної бази даних* (user). Об'єкт “користувач бази даних” застосовується для надання доступу до всіх об'єктів бази даних: таблицям, уявленням, процедурам, що зберігаються, і так далі. В користувача бази даних може відображатися:

- обліковий запис Windows;
- група Windows;
- обліковий запис SQL Server.

За ситуації, коли обліковий запис не відображається в користувача бази даних, клієнт все ж таки може дістати доступ до бази даних під гостьовим ім'ям guest, якщо воно, зрозуміло, є в базі даних. Зазвичай користувачеві guest надається мінімальний доступ тільки в режимі читання. Але в деяких ситуаціях і цьому доступу необхідно запобігти.

### Користувачі

Після того, як користувач пройшов аутентифікацію і *отримав ідентифікатор облікового запису* (login ID), він вважається зареєстрованим і йому надається доступ до сервера. Для кожної бази даних, до об'єктів якої користувачеві необхідно дістати доступ, *обліковий запис користувача* (login) асоціюється з *користувачем* (user) конкретної бази даних. *Користувачі* виступають як спеціальні *об'єкти* SQL Server, за допомогою яких визначаються всі дозволи доступу і володіння об'єктами в базі даних.
*Ім'я користувача* може використовуватися для надання доступу, як конкретній людині, так і цілій групі людей (залежно від типу облікового запису).
Якщо обліковий запис (login) не зв'язується явно з користувачем (user), останньому надається *неявний* доступ з використанням гостьового імені guest. Для забезпечення максимальної безпеки можна видалити користувача guest з будь-якої бази даних, окрім системних баз даних master і Tempdb.
*Власник* бази даних (DataBase Owner, DBO) — спеціальний користувач, що володіє максимальними правами в базі даних. Будь-який член ролі sysadmin автоматично відображається в користувача dbo. Якщо користувач, що є членом ролі sys admin, створює який-небудь об'єкт, то власником цього об'єкту призначається не даний користувач, а dbo.
Користувача dbo не можна видалити.
Для пов'язання облікового запису (login) з певним ім'ям користувача (user) можна скористатися наступною процедурою, що зберігається:

```
sp_adduser [@loginame =]  'login' [,[@name_in_db =] 'user']
[.[@grpname =] 'role']
```

Нижче дається пояснення використовуваних в ній параметрів:

- login — ім'я *облікового запису*, який необхідно пов'язати з ім'ям користувача бази даних;
- user — ім'я *користувача бази даних*, з яким асоціюється даний обліковий запис (у базі даних заздалегідь не повинно існувати користувача з вказаним ім'ям);

- role — цей параметр визначає *роль*, в яку даний користувач буде включений (докладніше про ролі буде розказано пізніше).

Користувач, який створює об'єкт в базі даних, наприклад таблицю, процедуру, що зберігається, або уявлення, стає *власником* об'єкту. Власник об'єкту (database object owner) має всі права доступу до створеного ним об'єкту. Щоб користувач міг створити об'єкт, власник бази даних (dbo) повинен надати користувачеві відповідні *права*. Повне ім'я створюваного об'єкту включає ім'я користувача, що створив його. Якщо користувач хоче звернутися до таблиці, використовуючи тільки її ім'я і не указуючи власника, SQL Server застосовує наступний алгоритм пошуку:

1. Шукається таблиця, створена *користувачем,* що виконує запит;
2. Якщо таблиця не знайдена, то шукається таблиця, створена *власником* бази даних (dbo).

SQL Server дозволяє передавати права володіння від одного користувача іншому. Щоб видалити власника об'єкту з бази даних, спочатку необхідно видалити всі об'єкти, які він створив, або передати права на їх володіння іншому користувачеві.

### Ролі сервера

*Роль* — це могутній інструмент, доданий в SQL Server, щоб замінити *групи,* які використовувалися в попередніх версіях. Роль дозволяє об'єднувати користувачів, що виконують однакові функції, для спрощення адміністрування системи безпеки SQL Server.
У SQL Server реалізовано два види стандартних ролей: на рівні *сервера* і *на рівні баз даних*. При установці SQL Server створюється 9 фіксованих ролей сервера і 9 фіксованих ролей бази даних. Ці ролі не можна видалити, крім того, не можна модифікувати права їх доступу. Не можна надати користувачеві права, які мають фіксовані ролі сервера, іншим способом, окрім як включенням його в потрібну роль.
У попередніх версіях SQL Server для адміністрування сервера можна було використовувати тільки обліковий запис sa або його аналог. Інакше кажучи, можна було дати або *всі* права, або *ніяких*. Тепер в SQL Server ця проблема вирішена шляхом додавання ролей сервера (server role), які дозволяють надати операторам сервера тільки ті права, які адміністратор порахує можливим надати. Ролі сервера не мають відношення до адміністрування баз даних. Можна включити будь-який обліковий запис SQL Server (login) або обліковий запис Windows в будь-яку роль сервера.

### Ролі баз даних

*Ролі бази даних* (database role) дозволяють об'єднувати користувачів в одну адміністративну одиницю і працювати з нею як із звичайним користувачем. Можна призначити права доступу до об'єктів бази даних для конкретної ролі,

при цьому всі члени цієї ролі автоматично наділяються однаковими правами. Замість того щоб надавати доступ кожному конкретному користувачеві, а згодом постійно стежити за змінами, можна просто включити користувача в потрібну роль. Якщо співробітник переходить в інший відділ, потрібно просто видалити його з однієї ролі і додати в іншу. Можна створити необхідну кількість ролей, які охоплювали б все різноманіття дій з базою даних. Пізніше, при зміні функцій членів однієї з ролей, досить змінити права доступу для цієї *ролі*, а не встановлювати нові права для кожного *користувача*.

У роль бази даних можна включати:

- користувачів SQL Server;
- ролі SQL Server;
- користувачів Windows;
- групи Windows, яким заздалегідь наданий доступ до потрібної бази даних.

При створенні бази даних для неї визначаються *стандартні ролі бази даних*, які, так само як і стандартні ролі сервера, не можуть бути змінені або видалені.

Окрім вказаних вище ролей існує ще одна — public. Ця роль має спеціальне призначення, оскільки її членами є всі користувачі, що мають доступ до бази даних. Не можна *явно* встановити членів цієї ролі, тому що всі користувачі і так автоматично є її членами. Цю роль слід використовувати для надання мінімального доступу користувачам, для яких права доступу до об'єктів не визначені явно. Якщо в базі даних дозволений користувач guest, то встановлений для public доступ матимуть всі користувачі, що дістали доступ до SQL Server. Роль public є у всіх базах даних, включаючи системні бази даних master, tempdb, msdb, model, і не може бути видалена.

### Ролі програми

Найчастіше для роботи з такими базами даних створюються спеціальні програми. Крім того, може знадобитися, щоб користувачі діставали доступ до бази даних тільки за допомогою певної програми, не даючи можливості безпосередньо звертатися до даних. Наприклад, мобільні користувачі можуть використовувати спеціальну клієнтську програму, за допомогою якої вони оперують даними, встановлюючи зв'язок з сервером через незахищені глобальні комунікації.

SQL Server вирішує перераховані проблеми шляхом *використання ролі програми*, створюваної на рівні бази даних. Відмінності між стандартними ролями і роллю застосування фундаментальні. Роль застосування не має *членів*. Користувачі SQL Server або Windows не можуть бути додані в цю роль. Роль активізується, коли програма встановлює з'єднання. Користувач,

що працює в цей час із програмою, не є членом ролі — тільки його програма використовує встановлене з'єднання.

## Захист даних

Як би добре не була спланована система безпеки SQL Server , залишається можливість копіювання файлів з даними і перегляду їх на іншому комп'ютері. Крім того, з використанням спеціальних пристроїв дані можуть бути перехоплені при передачі їх по мережі. Необхідно продумати засоби *фізичного* захисту даних. Дані у файлах бази даних зберігаються у відкритій формі, тобто їх можна проглянути в текстовому редакторові. Звичайно, вони не будуть структуровані, але все таки частину інформації можна буде прочитати.

### Шифрування даних

*Шифрування* — це метод, використовуваний SQL Server для зміни даних до нечитабельної форми. Шифрування гарантує, що цінна конфіденційна інформація не буде проглянута ким би то не було. Можна скопіювати дані, але не можна нічого з ними зробити. Для проглядання даних авторизованими користувачами використовується дешифровка. SQL Server дозволяє шифрувати наступні дані:

- будь-які дані, передавані між сервером і клієнтом по мережі;
- паролі облікових записів SQL Server або ролей програми;
- код, використаний для створення об'єктів бази даних (процедур,

що зберігаються, уявлень, тригерів і т. д.).

Паролі облікових записів і ролей програми *завжди* зберігаються в системних таблицях SQL Server в зашифрованій формі. Це оберігає їх від перегляду будь-яким користувачем, включаючи адміністратора. Крім того, пароль ролі програми може бути зашифрований перед відправкою його на сервер. Якщо необхідно шифрувати дані при передачі їх по мережі, необхідно використовувати мережеву бібліотеку Multiprotocol Net-Library.

### Обмеження доступу до файлів SQL Server

У своїй роботі SQL Server створює і використовує безліч файлів — бази даних, журнали помилок, резервні копії, файли для експорту і імпорту даних і багато що інше.
З метою обмеження можливостей неавторизованого доступу для файлів SQL Server необхідно встановити заборону на читання, видалення, модифікацію і виконання всім користувачам, окрім безпосередньо SQL Server.

## Права доступу

Коли користувачі підключаються до SQL Server, дії, які вони можуть виконувати, визначаються правами (дозволами), виданими їх обліковому запису, групі або ролі, в якій вони полягають.
Права в SQL Server можна розділити на три категорії:

- права на доступ до об'єктів баз даних;
- права на виконання команд TRANSACT-SQL;
- неявні права (дозволи).

Після створення користувач не має ніяких має рацію доступу, окрім тих, які дозволені для спеціальної ролі бази даних public. Права, надані цій ролі, доступні для *всіх* користувачів в базі даних.
Права користувачеві видаються адміністратором або власниками баз даних або конкретних об'єктів баз даних. Для надання користувачеві певного набору прав можна використовувати *ролі*.

## Права на доступ до об'єктів баз даних

Робота з даними і виконання процедур, що зберігаються, вимагає наявність класу доступу, званого *правами на доступ до об'єктів баз даних*. Під об'єктами маються на увазі таблиці, стовпці таблиць, уявлення, процедури, що зберігаються. Права на доступ до об'єктів баз даних контролюють можливість виконання користувачами, наприклад, команд SELECT, INSERT, UPDATE і DELETE для таблиць і уявлень. Таким чином, якщо користувачеві необхідно додати нові дані в таблицю, йому слід надати право INSERT (вставка записів в таблицю). Надання ж користувачеві права EXECUTE дозволяє йому виконання яких-небудь процедур, що зберігаються.
Для різних об'єктів застосовуються різні набори прав доступу до них:

- SELECT, INSERT, UPDATE, DELETE, REFERENCES – ці права можуть бути застосовані для *таблиці або уявлення;*
- SELECT і UPDATE — ці права можуть бути застосовані до конкретного *стовпця таблиці* або *уявлення;*
- EXECUTE— це право застосовується тільки до *процедур, що зберігаються*, і *функцій*. Нижче приводиться докладніший опис кожного з цих прав.
- INSERT. Це право дозволяє вставляти в таблицю або уявлення нові рядки. Як наслідок, право INSERT може бути видане тільки на рівні таблиці або уявлення і не може бути видано на рівні стовпця.
- UPDATE. Це право видається або на рівні таблиці, що дозволяє

змінювати всі дані в таблиці, або на рівні окремого стовпця, що дозволяє змінювати дані тільки в межах конкретного стовпця.

- DELETE. Це право дозволяє видаляти рядки з таблиці або уявлення. Як і право INSERT, право DELETE може бути видане тільки на рівні таблиці або уявлення і не може бути видано на рівні стовпця.
- SELECT. Вирішує вибірку даних. Може видаватися як на рівні таблиці, так і на рівні окремого стовпця.
- REFERENCES. Можливість посилатися на вказаний об'єкт. Стосовно таблиць дозволяє користувачеві створювати зовнішні ключі, що посилаються на первинний ключ або унікальний стовпець цієї таблиці. Стосовно уявлень право REFERENCES дозволяє пов'язувати уявлення з схемами таблиць, на основі яких будується уявлення. Це дозволяє відстежувати зміни структури початкових таблиць, які можуть вплинути на роботу уявлення. Право REFERENCES не існувало в попередніх версіях SQL Server.

### Заборона доступу

Система безпеки SQL Server має ієрархічну структуру. Це дозволяє ролям бази даних включати облікові записи і групи Windows, користувачів і ролі SQL Server. Користувач же, у свою чергу, може брати участь в декількох ролях. Наслідком ієрархічної структури системи безпеки є те, що користувач може одночасно мати різні права доступу для різних ролей. Якщо одна з ролей, в яких полягає користувач, має дозвіл на доступ до даним, то користувач автоматично має аналогічні права. Проте, може потрібно *заборонити* можливість доступу даним. Коли забороняється користувачеві доступ до даним або командам TRANSACT-SQL (deny access), тим самим анулюються *всі* дозволи на доступ, отримані користувачем на *будь-якому* рівні ієрархії. При цьому гарантується, що доступ залишиться забороненим незалежно від дозволів, наданих на більш високому рівні.

## Підвищення рівня захисту Microsoft SQL Server

*Захист компонентів бази даних*

Першим етапом настройки безпеки SQL Server як програми є вибір використовуваної моделі аутентифікації. Рекомендується застосовувати метод Windows викладеними вище проблемами методу SQL Server (пересилка пароля в закодованому вигляді).

Потім необхідно вирішити, чи будуть члени групи локальних адміністраторів мати доступ до сервера бази даних з привілеями системних адміністраторів. Автоматично створюваний серверний обліковий запис Bultin/Administrators рекомендується видалити. Ця рекомендація зв'язана, перш за все, з обов'язковою присутністю в даній групі облікового запису локального адміністратора, чий пароль може бути вмить змінений у разі наявності у зловмисника фізичного доступу або підібраний по мережі. Після того, як зловмисник дістане доступ до системи від імені цього облікового запису, він зможе обійти всі захисні механізми, включаючи шифрування баз даних за допомогою EFS.

**Аудит сервера SQL Server**

Існує декілька різних методів настройки аудиту в базах даних SQL Server. Кращим рішенням є використання комбінацій даних методів залежно від вирішуваних завдань. За умовчанням аудит на сервері не активізований.
Аудит аутентифікації сервера дозволяє відстежувати спроби аутентифікації на сервері SQL.
Події зберігаються в журналі Application операційної системи. Додаткові настройки дозволяють зберігати події в журналі помилок SQL Server.
Настройка даного типу аудиту припускає включення відстежування невдалих спроб аутентифікації (значення 2) з подальшим аналізом на предмет виявлення спроб несанкціонованого доступу або підбору паролів.
Інший тип аудиту відноситься до подій сервера баз даних. Настроювати і відстежувати їх можна за допомогою утиліти SQL Profiler.