

## Іспит БД

<b>Вступ .....</b>	<b>4</b>
1. Означення і властивості бази даних .....	4
2. Системи керування базами даних .....	4
3. Система баз даних .....	5
4. Логічний рівень архітектури баз даних .....	5
5. Концептуальний рівень архітектури баз даних .....	6
6. Фізичний рівень архітектури баз даних .....	6
7. Користувацькі мови та мови даних .....	7
8. ER-діаграми як спосіб моделювання даних .....	8
9. Принцип незалежності даних .....	8
<b>Реляційна алгебра .....</b>	<b>9</b>
10. Базові реляційні оператори .....	9
11. Розширені оператори реляційної алгебри .....	10
12. Способи забезпечення цілісності реляційних даних .....	12
13. Каталог (схема) в реляційних базах даних .....	13
14. Замкненість реляційної моделі .....	13
15. Представлення (VIEWS) .....	13
16. Домени і типи даних .....	13
17. Кортежі і їх властивості .....	15
18. Відношення і їх властивості .....	16
19. Реалізація реляційних операторів засобами SQL.....	16
<b>Функціональні залежності .....</b>	<b>17</b>
20. Означення функціональної залежності .....	17
21. Властивості ФЗ .....	17
22. Замикання множини функціональної залежності.....	17
23. Багатозначні залежності .....	17
24. Залежність сполучення .....	17
<b>Нормальні форми.....</b>	<b>18</b>
25. НФ2 .....	18
26. НФ3 .....	18
27. НФБК .....	18
28. НФ4 .....	18
29. НФ5 .....	18

30.	Теорема Хіта .....	19
31.	Теорема Фейгіна .....	19
32.	Незалежність проекцій відношення .....	19
<b>Обмеження цілісності .....</b>		<b>20</b>
33.	Потенційний ключ .....	20
34.	Суперключ .....	20
35.	Зовнішній ключ .....	21
36.	Тригер .....	22
37.	Курсор .....	23
38.	Обмеження цілісності рівня бази даних (ASSERTION) .....	23
39.	Збережені процедури .....	23
<b>Фізичне представлення даних .....</b>		<b>24</b>
40.	Типи памяті .....	24
41.	Сторінкове представлення бази даних .....	24
42.	Пришвидшення доступу до вторинних пристройів .....	24
43.	Фізичне представлення полів даних .....	24
44.	Фізичне представлення завписів .....	24
45.	Фізичне представлення відношень .....	24
<b>Одновимірні індекси .....</b>		<b>25</b>
46.	Індекси на впорядкованих даних .....	25
47.	Щільні індекси .....	25
48.	Розріджені індекси .....	25
49.	Вторинні індекси (на невпорядкованих даних) .....	26
50.	Індекси на основі геш-функцій .....	26
51.	Індекси на основі Бі-дерев .....	27
<b>Багатовимірні індекси .....</b>		<b>28</b>
52.	Багатовимірні сітки (Grid files) .....	28
53.	Розподілені хеш-функції (partitioned hash) .....	28
54.	KD Дерева .....	28
55.	Q дерево .....	29
56.	R дерево .....	29
57.	Bitmap індекси .....	29
<b>Транзакції .....</b>		<b>30</b>
58.	ACID властивості .....	30
59.	Журнал транзакцій і його використання .....	31

60.	<b>Блокування</b> .....	31
61.	<b>Аномалії при паралельній обробці</b> .....	31
62.	<b>Рівні ізоляції транзакцій</b> .....	32
63.	<b>Взаємне блокування (Deadlock)</b> .....	32
64.	<b>Способи забезпечення паралелізму (Песимістичне блокування і MVCC)</b> .....	33
	<b>Розподілені бази даних</b> .....	<b>33</b>
65.	<b>Головний принцип побудови розподілених баз даних</b> .....	33
66.	<b>CAP теорема</b> .....	34
67.	<b>BASE транзакції</b> .....	35
68.	<b>Розподілені запити</b> .....	35
69.	<b>Протокол двофазної фіксації транзакцій</b> .....	35
	<b>Навігаційні моделі</b> .....	<b>36</b>
70.	<b>Ієрархічна модель</b> .....	36
71.	<b>Сіткова (Мережева) модель</b> .....	36
	<b>XML</b> .....	<b>36</b>
72.	<b>Коректність (well formed)</b> .....	36
73.	<b>Валідність (valid)</b> .....	37
74.	<b>Парсери і їх особливості</b> .....	37
75.	<b>DTD</b> .....	38
76.	<b>XML Schema</b> .....	38
77.	<b>Трансформації. Дерево результату</b> .....	39
78.	<b>XSLT</b> .....	39
79.	<b>XPath</b> .....	40
80.	<b>XQuery</b> .....	40
	<b>Інші нереляційні моделі даних</b> .....	<b>40</b>
81.	<b>Об'єктно-орієнтована модель даних</b> .....	40
82.	<b>Об'єктно-реляційні бази даних</b> .....	41
83.	<b>Асоціативна модель даних</b> .....	42
84.	<b>EAW модель даних</b> .....	42
85.	<b>Основні моделі даних в NOSQL</b> .....	42
	<b>Інтеграція даних</b> .....	<b>43</b>
86.	<b>Об'єднана база даних (Federated database)</b> .....	43
87.	<b>Сховища даних (Data Warehouse)</b> .....	44

# Вступ

## 1. Означення і властивості бази даних

### Бази даних. Означення і властивості бази даних.

Термін "база даних" почав застосовуватися з 1963 р. і записувався англійською як *data base*. З розвитком обчислювальної техніки ці два слова були з'єднані (*database*). Один із розробників теорії баз даних, *Інгліс*, у 1972 р. дав таке визначення: **база даних** - сукупність збережених операційних даних, що використовуються прикладними системами деякого підприємства.

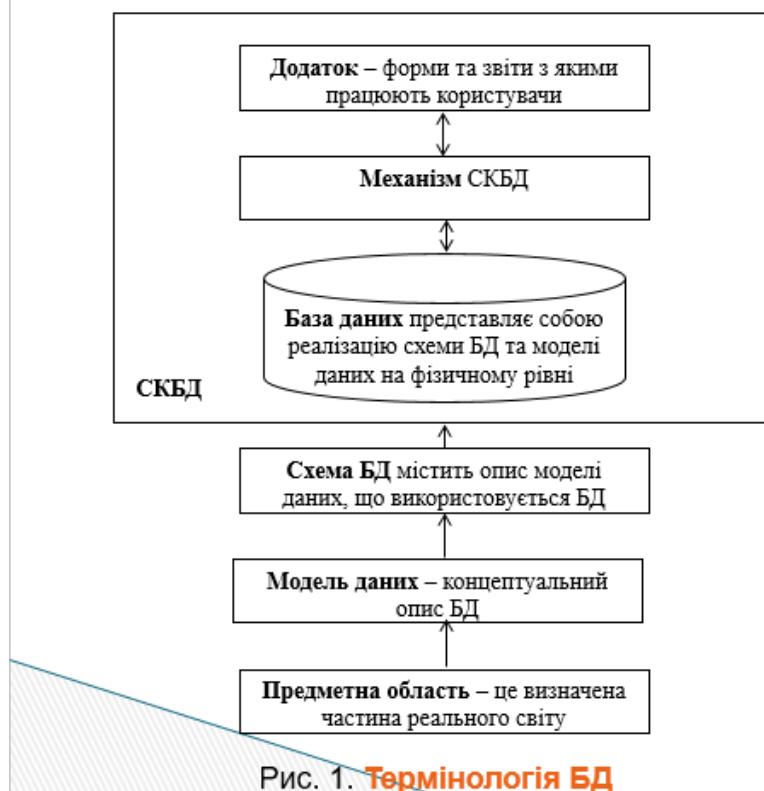
**База даних (БД)** - це пойменована сукупність даних, організованих за певними правилами, що передбачають загальні принципи опису, зберігання і маніпулювання даними, незалежно від прикладних програм.

Користувачі **БД** можуть:

- додавати нові порожні файли в базу даних;
- вставляти нові дані в існуючі файли;
- отримувати дані з існуючих файлів;
- видаляти дані з існуючих файлів;
- змінювати дані в існуючих файлах;
- видаляти існуючі файли з бази даних.

## 2. Системи керування базами даних

### Система керування базами даних



**СКБД** – програмне забезпечення за допомогою якого користувачі можуть визначати, створювати та підтримувати базу даних, а також здійснювати до неї контролюваний доступ.

Рис. 1. Термінологія БД

### 3. Система баз даних

#### Система баз даних

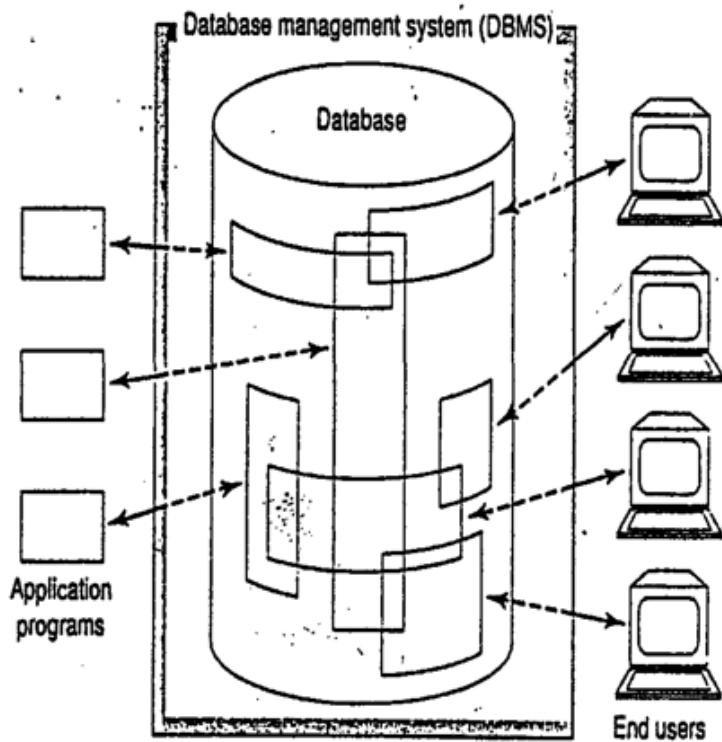


Fig. 1.4 Simplified picture of a database system

- це комп'ютеризована система зберігання записів. Вона містить самі **дані** (збережені в базі даних), **апаратне забезпечення**, **програмне забезпечення** (зокрема, систему керування базами даних, або **СКБД**), а також **користувачів** (що найважливіше): **прикладних програмістів**, **кінцевих користувачів** і **адміністраторів бази даних**. **АБД** відповідають за адміністрування бази даних і всієї системи баз даних відповідно до вимог, що встановлюються **адміністратором даних**.

### 4. Логічний рівень архітектури баз даних

Мета **логічного проектування** - розвинути концептуальне уявлення БД з урахуванням прийнятої моделі БД. Необхідно перевірити концептуальну модель за допомогою методів нормалізації та контролю виконання транзакцій

1. Видалення і перевірка елементів, що не відповідають прийнятій моделі даних.
  - 1.1. Видалення зв'язків N: M.
  - 1.2. Видалення зв'язків з атрибутами.
  - 1.3. Видалення складних зв'язків (зі ступенем участі більше 2).
  - 1.4. Видалення рекурсивних зв'язків (зі ступенем участі 1).
  - 1.5. Видалення багатозначних атрибутів (атрибутів мають кілька значень).
  - 1.6. Видалення надлишкових зв'язків.
  - 1.7. Повторна перевірка зв'язків 1:1.
2. Перевірка моделі за допомогою правил нормалізації.
3. Перевірка виконання транзакцій.
4. Визначення вимог підтримки цілісності даних.

## 5. Концептуальний рівень архітектури баз даних

Мета **концептуального проектування** - створення концептуальної моделі даних на основі уявлень про предметну область кожного окремого типу користувачів. Концептуальна модель представляє собою опис основних сутностей (таблиць) і зв'язків між ними без урахування прийнятої моделі БД та синтаксису цільової СКБД.

*Етапи:*

1. Виділення сутностей.
2. Визначення атрибутів.
3. Визначення зв'язків.
4. Визначення суперкласів і підкласів.

## 6. Фізичний рівень архітектури баз даних

Мета **фізичного проектування** - перетворення логічної моделі з урахуванням синтаксису, семантики і можливостей обраної цільової СКБД. Данна стадія включає в себе проектування таблиць і зв'язків між ними з урахуванням можливостей цільової СКБД.

1. Аналіз необхідності введення контрольованої надлишковості.
- 1.1. Використання похідних даних.
- 1.2. Дублювання атрибутів.
2. Перенесення логічної моделі даних в середовище цільової СКБД.
3. Реалізація бізнес-правил та аналіз транзакцій.
4. Розробка механізмів захисту.
5. Організація моніторингу та налаштування функціонування системи.

Для відображення *трирівневої архітектури* можна ідентифікувати наступні три зв'язані моделі даних:

1. **зовнішню модель** даних, що відображає представлення кожного типу користувачів (*користувацьким логічним рівнем*);
2. **концептуальну модель** даних, що відображає логічне (або узагальнене) уявлення про дані, незалежне від типу вибраної СКБД (*загальним логічним рівнем*);
3. **внутрішню модель даних**, що відображає концептуальну схему у термінах цільової СКБД (*фізичним рівнем*).

може існувати *кілька зовнішніх уявлень*, кожне з яких складається з більш-менш абстрактного уявлення певної частини бази даних, і тільки *одне концептуальне уяведення*, що складається з абстрактного уявлення бази даних в цілому. Крім того, і зовнішній, і концептуальний рівні *представляють* собою *рівні моделювання*, а *внутрішній* служить в якості *рівня реалізації*; іншими словами, перші два рівня визначені *в термінах таких користувальницьких інформаційних конструкцій, як записи і поля*, а останній - *в термінах машинно-орієнтованих конструкцій* на зразок бітів і байтів

## 7. Користувацькі мови та мови даних.

### *Користувацькі мови та мови даних.*

- *прикладного програміста* це, наприклад, *PL / I, C ++* або *Java*, або спеціальна мова даної системи ( мови *четвертого покоління*);
  - *кінцевого користувача* це або спеціальна *мова запитів*, або *мову спеціального призначення*, яка може бути заснована на використанні форм і меню, розроблена спеціально з урахуванням вимог користувача і може інтерактивно підтримуватися деякими оперативним додатком. *включають*
  - *підмова даних*, тобто підмножину операторів всієї мови, пов'язану тільки з об'єктами баз даних і операціями з ними.
- вбудовану в базову мову, яка додатково надає різні *не зв'язані* з БД засоби, такі як локальні (тимчасові) змінні, обчислювальні операції, логічні операції і т.д. Система може підтримувати будь-яку кількість базових мов і будь-яку кількість підмов даних. Одна мова, яка підтримується практично всіма сьогоднішніми системами: *SQL*.
- це комбінація принаймні двох мов - мови визначення даних (*Data Definition Language*) і мови маніпулювання даними (*Data Manipulation Language*).

## 8. ER-діаграми як спосіб моделювання даних

Модель «сущність-зв'язок» (ER-модель) (англ. *Entity-relationship model* або *entity-relationship diagram*) — модель даних, яка дозволяє описувати концептуальні схеми за допомогою узагальнених конструкцій блоків.

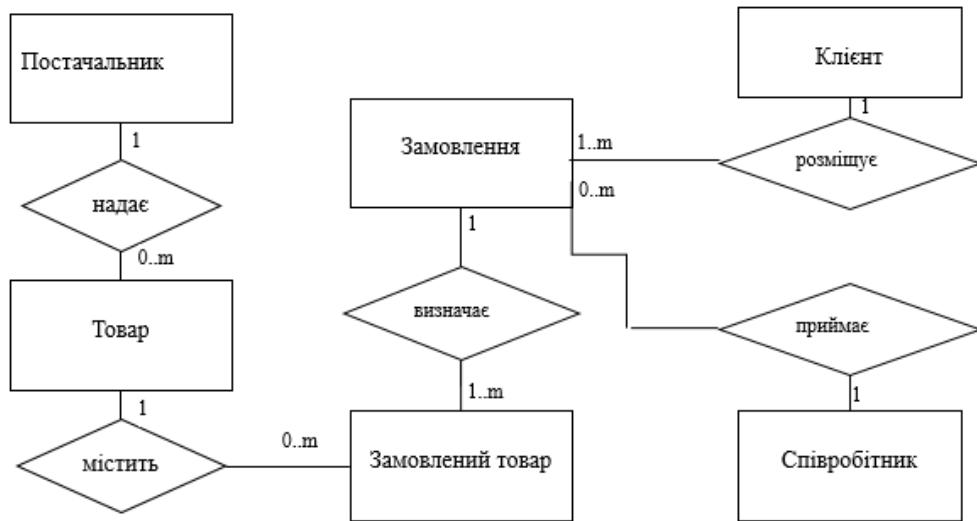


Рис. ER-діаграма в нотації Чена.

## 9. Принцип незалежності даних

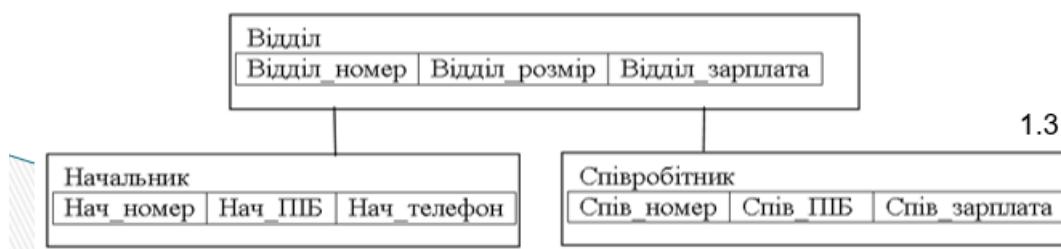
**Фізичні моделі даних** описують те, як дані зберігаються в комп'ютері, відображаючи інформацію про структуру записів, їх впорядкованість та існуючі шляхи доступу.

### даталогічною моделлю даних

1. На основі записів:

1. **реляційна модель даних** (*relational data model*). Дані у вигляді таблиць
2. **мережева модель даних** (*network data model*) кожен об'єкт може мати більше одного предка
3. **ієрархічна модель даних** (*hierarchical data model*). Між об'єктами існують зв'язки типу «предок-нащадок», можлива ситуація, коли об'єкт не має нащадків або має їх декілька, тоді як у об'єкта-нащадка обов'язково тільки один предок.

2. Об'єктно-орієнтована модель даних. Дані оформляють у вигляді моделей об'єктів



# Реляційна алгебра

## 10. Базові реляційні оператори

- **Реляційна алгебра** — це теоретична мова операцій, яка на основі одного або декількох відношень дозволяє створювати інше відношення без зміни самих початкових відношень.

існує мінімум **3 оператори** (**проекція, скорочення та сполучення**), які застосовуються до таблиць і результатом їх застосування є нові таблиці в концептуальному сенсі

**SELECT Fname, Lname  $\pi$  - project**  
FROM (EMPLOYEE JOIN WORKS\_ON ON Ssn=Essn)  
JOIN PROJECT ON Pno=Pnumber  
WHERE Dno=5  
AND Hours>=10  
AND Pname='ProductX'

$\bowtie$  - join

**$\sigma$  - select**

### ► Проекція.

Нехай відношення  $A\{X, Y, \dots, Z\}$  має атрибути  $X, Y, \dots, Z$ . Тоді проекцією  $\pi_{XY\dots}(A)$  буде називатися нове відношення, заголовок якого буде містити атрибути  $X, Y, \dots$  з  $A$ , а тіло буде містити **підмножини кортежів тіла  $A$** , які відповідають атрибутам заголовку.

**SELECT X, Y FROM A;**

### ► Скорочення.

Нехай **відношення  $A$**  має **атрибути  $X$  та  $Y$** . Нехай крім того існує оператор  $\Theta$ , такий що є застосовним до значень типу  $X, Y$ . І вираз  $X \Theta Y$  є правильно побудованим і має своїм результатом булівське значення (істинно або хибно). Тоді  **$\Theta$ -скороченням**  $\sigma_{X\Theta Y}(A)$  відношення  $A$  по атрибути  $X$  та  $Y$  буде відношення із заголовком еквівалентним  $A$  та тілом, яке буде містити усі кортежі з тіла  $A$ , для яких вираз  $X \Theta Y$  є істинним. Предикат  $X \Theta Y$  називають умовою скорочення.

**SELECT \* FROM A WHERE X  $\Theta$  Y;**

X	Y
---	---

### ► Перейменування.

Нехай є **відношення  $R$  з атрибутом  $A$** . Результатом операції **перейменування  $A$  в  $B$**  буде нове **відношення**, у якому будуть **наявні усі кортежі тіла** і еквівалентний **заголовок** у якому замість атрибути з іменем  $A$  буде атрибут з іменем  $B$ .

**Типи атрибутів і значення компонентів кортежів тіла** при цьому **не змінюються** ніяк.

**A**

**SELECT A AS B FROM R**

**B**

### ► Природне сполучення.

Нехай відношення **A** та **B** мають відповідно наступні атрибути:  
 $X_1, X_2, \dots, X_k, Y_1, Y_2, \dots, Y_n, Z_1, Z_2, \dots, Z_m$ .

Обидва відношення мають спільну множину атрибутів

$Y_1, Y_2, \dots, Y_n$ . Решта атрибутів **A** складають множину **X**, а решта атрибутів **B** складають множину **Z**. Тоді

Природне сполучення (*natural join*) **A JOIN B (A  $\bowtie$  B)** – це відношення з заголовком  $\{X, Y, Z\}$  та тілом що складається з усіх кортежів  $\{Xx, Yy, Zz\}$  кожен з яких міститься як  $\{Xx, Yy\}$  в **A** та як  $\{Yy, Zz\}$  в **B**.

**SELECT \* FROM A JOIN B ON A.X=B.Y**

## 11. Розширені оператори реляційної алгебри

► **Поділ.** Нехай відношення **A** має атрибути  $\{X_1, X_2, \dots, X_n\}$ , та **B** –  $\{Y_1, Y_2, \dots, Y_m\}$ . Імена атрибутів не співпадають. І нехай є **C**  $\{X_1, X_2, \dots, X_n, Y_1, Y_2, \dots, Y_m\}$ .

Тоді поділом **A** на **B** по **C** **A DIVIDE BY B PER C** буде відношення з заголовком  $\{X\}$  з **A** таких, що кортеж  $\{Xx, Yy\}$  присутній в **C** для всіх  $\{Yy\}$  присутніх в **B**.

**SELECT X FROM A, B, C WHERE A.X=C.X AND C.Y = B.Y**

A		C	
B	X	X	Y

## ► Напівсполучення.

### ► Ліве

Нехай відношення **A** має атрибути  $\{X_1, X_2, \dots, X_n, Y_1, Y_2, \dots, Y_m\}$ ,  
та **B** –  $\{Y_1, Y_2, \dots, Y_m, Z_1, Z_2, \dots, Z_k\}$  аналогічні до JOIN.

Тоді напівсполученням  $A \ltimes B$  (**A SEMIJOIN B**) називається  
операція  $(A \text{ JOIN } B) \{X, Y\}$ .

$$A \ltimes B = \pi_A (A \bowtie B)$$

Інакше – напівсполученням **A** до **B** називається результат  
сполучення відношень **A** та **B** до якого застосовано проекцію по  
атрибуатах **A**.

### ► Праве

$$A \rtimes B = \pi_B (A \bowtie B)$$

по атрибуатах **B**

## ► Напіврізниця.

Аналогічно до попереднього **A**  $\{X_1, Y_1\}$  та **B**  $\{Y_1, Z_1\}$ .

### Ліва

Тоді **A SEMIMINUS B** називається операція аналогічна до  $A \triangleleft B$  (**A SEMINJOIN B**).

$$A \triangleleft B = \pi_A (A \overset{\Theta}{\bowtie} B)$$

Інакше – тіло результуату складається з тих кортежів **A** для яких  
немає відповідників у **B**.

### ► SELECT A.\* FROM A, B WHERE A.Y <> B.Y

### ► Права

тіло результуату складається з тих кортежів **B** для яких немає  
відповідників у **A**.

$$A \triangleright B = \pi_B (A \overset{\Theta}{\bowtie} B)$$

## ► Розширення.

Операція розширення дозволяє створювати нове відношення з додатковим атрибутами, які є результатом деяких скалярних обчислень.

**EXTEND A ADD (expression) AS Z.**

Вираз не змінює вміст бази даних. Новий атрибут може бути параметром інших операцій.

**SELECT A.\* , A.X+1 AS X1 FROM A**

## ► Агрегація (узагальнення).

**SUMMARIZE A PER B ADD (expression) AS Z** визначається наступним чином:

- Відношення **B** повинно мати такий же вигляд як деяка проекція **A**
- Результатом буде відношення з атрибутами **{B<sub>1</sub>, B<sub>2</sub>, ..., B<sub>n</sub>, Z}** де **Z** є новим атрибутом
- Тіло результату містить всі кортежі **t**, де **t** є кортежем відношення **B**, розширеним новим атрибутом. Значення **Z** обчислюється за допомогою виразу по усіх кортежах відношення **A**, які мають однакові значення атрибутів **{B<sub>1</sub>, B<sub>2</sub>, ..., B<sub>n</sub>}**. Якщо в **A** немає таких кортежів, то результатом буде порожня множина кортежів. Тип атрибуту **Z** буде тип агрегуючого виразу.

**SELECT A.\* , SUM (A.X) FROM A GROUP BY A.\***

## 12. Способи забезпечення цілісності реляційних даних

### 3. Система підтримки цілісності – набір засобів для забезпечення несуперечливості даних у базі

Для **РМ** (реляційної моделі) ВАЖЛИВА **несуперечливість даних** відповідно до особливостей предметної області.

*Імперативний підхід – як зробити.*

*Декларативний – що хочемо отримати.*

*SQL- декларативна мова.* Тому підтримується набір **декларативних визначень**, які задають правила перевірки даних на предмет їх несуперечливості.

**Відношення**(таблиці) можна сприймати як предикат, який визначає модель, що описує частину реального світу. **Атрибути** (стовпці ) **пов'язані з типами та обмеженнями** даних , тому кожний рядок таблиці повинен задовольняти тим обмеженням, які ми вказали.

### **13. Каталог (схема) в реляційних базах даних**

**Каталог.** Або **схема** даних складається із системних відношень, які **описують структуру бази даних**, а також **набір обмежень цілісності** для бази. Описуються **базові змінні відношення** та **представлення**, або похідні змінні відношень. Визначення представлення міститься у каталозі, але **фізично** саме представлення **не існує** а задає результат для подальших операцій. Перевага **VIEW** : маємо скомпільовані дані для запиту, що дає **приріст продуктивності**. Є також **матеріалізовані представлення**, які мають своє відображення **на фізичних носіях**. У **РБД** система сама контролює оновлення даних у представлених.

### **14. Замкненість реляційної моделі**

Кожне відношення характеризується схемою (або заголовком) і набором кортежів (або тілом). Тому, якщо реалізовувати алгебру, операції якої замкнені відносно поняття відношень, кожна операція повинна породжувати відношення в повній мірі, тобто воно повинно володіти і тілом, і заголовком. Лише в цьому випадку буде можливість будувати вкладені вирази. Заголовок відношения є множиною пар імя-атрибута, імя-домена. Якщо зробити загальний огляд реляційних операцій, можна побачити, що домени атрибутів результуючого відношення однозначно визначаються доменами відношень-результатів.

### **15. Представлення (VIEWS)**

Представлення (VIEWS) – об'єкт бази даних, який являється результатом виконання запиту до бази даних, визначеного за допомогою оператора **SELECT**, в момент звернення до представлення.

Представлення іноді називають «віртуальними таблицями». Така назва пов'язана з тим, що представлення доступне для користувача як таблиця, але саме не зберігає даних, а вилучає їх із таблиців момент звернення до нього. Якщо дані змінені в базовій таблиці, то користувач отримає актуальні дані при зверненні до подання, котрі використовують цю таблицю; кешування результатів вибірки з таблиці при роботі уявлень не проводиться. При цьому, механізм кешування запитів (query cache) працює на рівні запитів користувача безвідносно до того, чи звертається користувач до таблиць або уявленням. Представлення можуть ґрунтуватися як на таблицях, так і на інших представлених.

### **16. Домени і типи даних**

Будь-який тип є або визначений системою або визначений користувачем. Перші - тісно пов'язані із особливостями фізичного відображення даних у

конкретній системі: як рахуються біти зліва направо чи навпаки. При передачі запакованих даних це важливо.

Тип даних може бути прив'язаний до мов програмування, платформ. **Визначені користувачем** типи переважно походять з предметної області і її особливостей у випадку конкретної БД. Для кожного типу даних **характерним є набір операторів**, застосовних до цього типу.

Формально, **тип даних – це множина допустимих значень для одиниці даних**. Еквівалентом типів даних у **РМ** є **DOMAIN Домени** з відмінностями, які накладаються моделлю. Можна оголошувати свої домени **на базі системних доменів**, які вбудовані у **СКБД**.

З іншого боку **можна звузити вимоги**, наприклад, вік людини, що є в структурі Університет: int year > 0 , >1900 , <2018 . < 2010. Щоб кожен раз не викликати функцію, то, оскільки **РМ** має декларативний характер, для цього є **домени**.

Множина типів даних необмежена.

**Типи – це об'єкти**, які є предметом даних, відношення – це факти, які є предметом даних. Кожне відношення має **заголовок і тіло**.

**Заголовок** є предикатом, а **тіло** утворене набором фактів при яких предикат є **істинним**. **Типи та відношення** є необхідними і достатніми для представлення будь-яких даних на логічному рівні.

CREATE DOMAIN та CREATE TYPE у PostgreSQL.

**Домени відрізняються від типів** можливістю звуження множини допустимих значень та накладання інших додаткових обмежень.

**Нескалярним** є тип, для якого **явно оголошена множина значень видимих користувачеві допустимих компонентів (інкапсульованим)**

Тип який **не є нескалярним** називають **скалярним (атомарним)**

## 17. Кортежі і їх властивості

### Структури реляційної моделі. Кортежі

Означення. Якщо дана множина типів  $T_i (i=1,2,\dots,n)$ , необов'язково різних, то **значенням кортежа** (або кортежем)  $t$  буде множина трійок виду  $\{ A_i, T_i, v_i \}$ , де  $A_i$  – імя атрибута,  $T_i$  – імя типу атрибута,  $v_i$  – значення типу  $T_i$  для даного атрибута.

Крім того **кортеж відповідає** наступним **вимогам**:

- Значення  $n$  називають **степенем або арністю** кортежа
- Впорядкована трійка є **компонентом**  $t$
- Впорядкована двійка  $\langle A_i, T_i \rangle$  є **атрибутом**  $t$  і однозначно визначається іменем атрибуту  $A_i$
- Повна множина атрибутів  $\{\langle A_i, T_i \rangle\}$ , які відрізняються тільки значеннями  $v_i$ , називається **заголовком** кортежа. У  $\{\langle A_i, T_i \rangle\}$ ,  $A_i$  не можуть співпадати в межах одного кортежа
- Тип кортежа  $t$  визначається його заголовком

### Властивості кортежів:

- **Кожен кортеж завжди містить точно одне значення** для кожного свого атрибута
- Для компонентів кортежа **некоректне відношення порядку**
- Кожна підмножина кортежа теж є **кортежем (навіть порожня множина)**
- Кортежі  $t_1$  і  $t_2$  є **рівними** тоді і лише тоді, коли вони мають **однакові атрибути** і для всіх  $i (i=1,\dots,n)$  значення  $v_i$  атрибута  $A_i$  в кортежі  $t_1$  рівне значенню  $v_i$  атрибута  $A_i$  в кортежі  $t_2$ . Такі кортежі називають **дублікатами**. Усі **нуль-арні** кортежі є дублікатами.

## 18. Відношення і їх властивості

Об'єднання кортежів — це **відношення**. Складається із **заголовку та тіла**.

Заголовком відношення є **заголовок деякого кортежа**.

Відповідно атрибути кортежа будуть **атрибутами відношення**.

Тіло відношення — це **множина кортежів**, які мають той **самий заголовок** і їх заголовок **співпадає із заголовком відношення**. **Характерним** для тіла відношення є **відсутність компонент без значення і більше ніж з одним значенням**.

**Скільки дублікатів може бути в тілі відношення?**

Тоді **потужністю** відношення буде **потужність множини тіла відношення**.

**Відношення має властивості:**

- Кожен кортеж тіла містить **точно одне значення відповідного типу для кожного свого атрибута** (1 НФ)
- Для атрибутів **немає** встановленого **порядку**
- Для кортежів тіла немає **встановленого порядку**
- **Відношення не містить** дублікатів кортежів

## 19. Реалізація реляційних операторів засобами SQL

SELECT \* FROM A UNION SELECT \* FROM B

SELECT \* FROM A INTERSECT SELECT \* FROM B

SELECT \* FROM A EXCEPT SELECT \* FROM B

SELECT \* FROM A,B

SELECT A AS B FROM R

SELECT \* FROM A JOIN B ON A.X=B.Y

SELECT X FROM A, B, C WHERE A.X=C.X AND C.Y = B.Y

# Функціональні залежності

## 20. Означення функціональної залежності

**Функціональна залежність** (далі часто ФЗ) — концепція, що лежить в основі багатьох питань, пов'язаних з реляційними базами даних, включаючи, зокрема, їхнє проектування. Математично являє собою бінарне відношення між множинами атрибутів даного відношення і є, по суті, зв'язком типу «один-до-багатьох». ФЗ забезпечує основу для наукового підходу до розв'язання деяких проблем, оскільки володіє багатим набором цікавих формальних властивостей.

## 21. Властивості ФЗ

- Замикання множини залежностей
- Замикання множини атрибутів
- Незвідні множини залежностей

## 22. Замикання множини функціональної залежності

Одні функціональні залежності можуть припускати інші функціональні залежності. Наприклад,

$$(A \rightarrow B) \wedge (B \rightarrow C) \Rightarrow (A \rightarrow C).$$

Множина  $S +$  всіх ФЗ, які припускаються даною множиною ФЗ  $S$  називається **замкненням** множини  $S$ .

## 23. Багатозначні залежності

В теорії баз даних, **багатозначна залежність** — повне обмеження між двома множинами атрибутів у відношенні.

На відміну від функціональної залежності, багатозначна залежність вимагає наявність певних кортежів у відношенні. Отже, багатозначна залежність це особливий випадок кортеж-твірної залежності. Поняття багатозначної залежності використовується при визначенні четвертої нормальної форми.

## 24. Залежність сполучення

???

# Нормальні форми

## 25. НФ2

**Друга нормальна форма** [ред. | ред. код]

Друга нормальна форма (2НФ, 2NF) вимагає, аби дані, що зберігаються в таблицях із композитним ключем, не залежали лише від частини ключа:

- Схема бази даних повинна відповідати вимогам першої нормальної форми.
- Дані, що повторно з'являються в декількох рядках, виносяться в окремі таблиці.

## 26. НФ3

**Третя нормальна форма** [ред. | ред. код]

Третя нормальна форма (3НФ, 3NF) вимагає, аби дані в таблиці залежали винятково від основного ключа:

- Схема бази даних повинна відповідати всім вимогам другої нормальної форми.
- Будь-яке поле, що залежить від основного ключа та від будь-якого іншого поля, має виноситись в окрему таблицю.

## 27. НФБК

**Нормальна форма Бойса — Кодда** [ред. | ред. код]

Відношення знаходитьться в НФБК тоді і лише тоді, коли детермінант кожної функціональної залежності є потенційним ключем. Якщо це правило не виконується, то, щоб привести вказане відношення до НФБК, його слід розділити на два відношення шляхом двох операцій проекції на кожну функціональну залежність, детермінант якої не є потенційним ключем:

1. Проекція без атрибутів залежності частини такої функціональної залежності;
2. Проекція на всі атрибути цієї функціональної залежності.

Визначення НФБК не потребує жодних умов попередніх нормальних форм. Якщо проводити нормалізацію послідовно, то в переважній більшості випадків при досягненні ЗНФ автоматично будуть задовольнятися вимоги НФБК. ЗНФ не збігається з НФБК лише тоді, коли одночасно виконуються такі 3 умови: [джерело?]

1. Відношення має 2 або більше потенційних ключів.
2. Ці потенційні ключі складені (містять більш ніж один атрибут)
3. Ці потенційні ключі перекриваються, тобто мають щонайменше один спільний атрибут.

## 28. НФ4

**Четверта нормальна форма** [ред. | ред. код]

Четверта нормальна форма (4НФ, 4NF) потребує, аби в схемі баз даних не було нетривіальних багатозначних залежностей множин атрибутів від будь чого, окрім надмножини ключа-кандидата. Вважається, що таблиця знаходитьться у 4НФ тоді і лише тоді, коли вона знаходитьться в НФБК та багатозначні залежності є функціональними залежностями. Четверта нормальна форма усуває небажані структури даних — багатозначні залежності.

## 29. НФ5

**П'ята нормальна форма** [ред. | ред. код]

П'ята нормальна форма (5НФ, 5NF, PJ/NF) вимагає, аби не було нетривіальних залежностей об'єднання, котрі б не витікали із обмежень ключів. Вважається, що таблиця в п'ятій нормальній формі тоді і лише тоді, коли вона знаходитьться в 4НФ та кожна залежність об'єднання зумовлена її ключами-кандидатами.

## 30. Теорема Хіта

Якщо є відношення  $R$  у якого є атрибути  $A, B, C$ , пов'язані залежністю  $A \rightarrow B$ , то декомпозиція на  $r1(A, B)$  і  $r2(A, C)$  буде обертою:  $R(A, B, C) = r1(A, B) \text{ join } r2(A, C)$ .

Наприклад, ви с друзями посидели в пабе. Имеет место отношение Pub (man, beer, payment).

man	beer	payment
Darth	Leffe Blonde	5\$
Yoda	Heineken	10\$
Jabba the Hutt	Amstel Light	8\$

Каждый из вас пьет один определенный сорт пива. Это зависимость  $\text{man} \rightarrow \text{beer}$ .

Значит, по теореме Хита мы можем разделить данные на 2 отношения:

- $\text{man} \rightarrow \text{beer} == \text{beer\_man}(\text{man}, \text{beer})$

man	beer
Darth	Leffe Blonde
Yoda	Heineken
Jabba the Hutt	Amstel Light

- $\text{man} \rightarrow \text{payment} == \text{pay\_man}(\text{man}, \text{payment})$

man	payment
Darth	5\$
Yoda	10\$
Jabba the Hutt	8\$

Этих данных достаточно, чтобы однозначно и правильно воссоздать оригинальное отношение Pub (man, beer, payment), используя natural join:

- $\text{beer\_man} \text{ JOIN } \text{pay\_man} = \text{pub}$

P.S. На всякий случай напоминаю, что при natural join, о котором идет речь в теореме, все пары атрибутов из двух отношений, имеющих общее имя, приравниваются, а одна из пар равных атрибутов удаляется путем проекции.

## 31. Теорема Фейгіна

Нехай  $A, B, C$  є множина атрибутів  $R(A, B, C)$ . Відношення  $R$  дорівнюватиме з'єднанню його проекцій  $R1(A, B)$  і  $R2(A, C)$  тоді і тільки тоді, коли для відносини  $R$  виконується багатозначна залежність  $A \rightarrow\!\!> B \mid C$ .

## Теорема Фейгіна

Пусть  $A, B$  и  $C$  поля таблицы. Если  $A \gg B$  и  $A \gg C$ , то таблицу можно представить как результат соединения ее проекций  $(A, B)$  и  $(A, C)$ .

Фамилия - Должность; Фамилия - Телефон.

## 32. Незалежність проекцій відношення

???

## Обмеження цілісності

### 33. Потенційний ключ

Потенційний ключ для  $K$  для відношення  $R$  — це підмножина множини атрибутів  $R$ , що характеризується такими двома властивостями:

1. Властивість унікальності.

Немає двох різних кортежів в  $R$  з однаковим значенням  $K$ .

2. Властивість мінімальності (ненадмірності).

Ніяка з підмножин  $K$  не має унікальності.

Будь-яке відношення має, щонайменше, один потенційний ключ через те, що не містить двох одинакових кортежів; тобто, комбінація всіх атрибутів має властивість унікальності. Тому можливі два варіанти:

- або ця комбінація водночас має властивість мінімальності, тобто і є потенційним ключем (єдиним).
- або існує щонайменше одна підмножина цієї комбінації, що теж має властивість унікальності, а також мінімальності.

У відношенні може бути декілька потенційних ключів. Один з них може бути вибраний як первинний ключ відношення, тоді інші називають альтернативними ключами.

Потенційні ключі забезпечують основний механізм адресації на рівні кортежів. Тобто, єдиний гарантований спосіб точно вказати який-небудь кортеж — це вказати значення якогось потенційного ключа. В інструкції SQL це, зазвичай, робиться в частині WHERE.

Таким чином потенційні ключі мають таке саме фундаментальне значення для успішної роботи реляційної системи, як адресація основної пам'яті для успішної роботи машини, на якій ця система встановлена.

### 34. Суперключ

В реляційній моделі баз даних **суперключ** — підмножина атрибутів змінної відношення для якої виконується, що для будь-яких значень цієї змінної не існує двох кортежів (рядків, записів), які містять однакові значення для атрибутів з цієї підмножини. Тотожним визначенням суперключа буде наступне, — множина атрибутів змінної відношення від яких функціонально залежні всі атрибути змінної відношення.

Зауважимо, що якщо підмножина атрибутів  $K$  є суперключем змінної відношення  $R$ , тоді завжди вірно, що проекція  $R$  по атрибутам з  $K$  має однакову потужність з  $R$ .

Неформально, суперключ це підмножина атрибутів в таблиці чиї значення можуть бути використані для унікальної ідентифікації кортежу. Потенційний

ключ — найменша підмножина атрибутів необхідних для ідентифікації кортежу, його також називають найменшим (мінімальним) суперключем. Наприклад, дана схема відношення Працівникі з атрибутами працівникІд, ім'я, посада і відділІд, ми можемо використати працівникІд в сполученні з будь-яким або всіма іншими атрибутами цієї таблиці для унікальної ідентифікації кортежу в таблиці. Прикладами суперключів в цій таблиці будуть {працівникІд, ім'я}, {працівникІд, ім'я, посада}, і {працівникІд, ім'я, посада, відділІд}.

В дійсності, ми не потребуємо всі ці значення для ідентифікування конкретного кортежу. Ми потребуємо лише, в нашему прикладі, підмножину {працівникІд}. Це найменший суперключ – тобто, найменша підмножина атрибутів, які можна використати для ідентифікації конкретного кортежу. Тож, працівникІд це потенційний ключ.

Англійські монархи

Ім'я	Номер	Династія
Едвард	II	Плантагенет
Едвард	III	Плантагенет
Річард	III	Плантагенет
Генрі	IV	Ланкастер

В цьому прикладі, можливими суперключами є:

- {Ім'я, Номер} (потенційний ключ)
- {Ім'я, Номер, Династія}

### 35. Зовнішній ключ

**Зовнішній ключ** — атрибут (набір атрибутів) в деякому відношенні R, який відповідає первинному ключу іншого відношення або того ж таки відношення R.

В реляційних базах даних зовнішній ключ задається обмеженням FOREIGN KEY.

```
CREATE TABLE fools (
    id      INTEGER PRIMARY KEY AUTO_INCREMENT,
    name   CHAR(20),
    folly_id  INTEGER,
    FOREIGN KEY(folly_id)
        REFERENCES follies(id) ON DELETE CASCADE);
```

## 36. Тригер

Тригер (англ. *trigger*) — це збережена процедура особливого типу, яку користувач не викликає явно, а використання якої обумовлено настанням визначені події (дії) у реляційній базі даних:

- додаванням INSERT,
- вилученням рядка в заданій таблиці DELETE,
- або зміною даних у певному стовпці заданої таблиці UPDATE.

Тригери застосовуються для забезпечення цілісності даних і реалізації складної бізнес-логіки. Тригер запускається сервером автоматично при спробі зміни даних у таблиці, з якою він пов'язаний. Всі здійснені ним модифікації даних розглядаються як виконані в транзакції, в якій виконано дію, що викликало спрацьування тригера. Відповідно, у разі виявлення помилки або порушення цілісності даних може відбутися відкат цієї транзакції.

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

Крім того, тригери можуть бути прив'язані не до таблиці, а до розрізу (VIEW). В цьому випадку за їхньої допомоги реалізується механізм «оновлюваного виду». В цьому випадку ключові слова BEFORE і AFTER впливають лише на послідовність виклику тригерів, бо власне подія (вилучення, вставка або оновлення) не відбувається.

В деяких серверах тригери можуть викликатися не дляожної модифікації запису, а один раз на зміну таблиці. Такі тригери називаються *табличними*.

```
/ * Тригер на рівні таблиці * /
CREATE OR REPLACE TRIGGER DistrictUpdatedTrigger
AFTER UPDATE ON district
BEGIN
    INSERT INTO info VALUES ('table "district" has changed');
END;

/ * Тригер на рівні рядка * /
CREATE OR REPLACE TRIGGER DistrictUpdatedTrigger
AFTER UPDATE ON district FOR EACH ROW
BEGIN
    INSERT INTO info VALUES ('one string in table "district" has changed');
END;
```

## 37. Курсор

**Курсор** - це об'єкт реляційної бази даних, який дозволяє виконувати запит (маніпулювання даними) і зберігає результат в БД.

**Курсор в SQL** - це область в пам'яті бази даних, яка призначена для зберігання останнього оператора SQL.

Якщо **поточний оператор** - запит до бази даних, в пам'яті зберігається і **рядок даних запиту**, званий **поточним значенням**, або поточним рядком курсору. Вказана область в пам'яті **пойменована і доступна** для прикладних програм.

Зазвичай **курсори використовуються** для вибору з бази даних **деякої підмножини інформації**, що зберігається в ній.

Сучасні СКБД підтримують **3 оператори SQL** для роботи з курсорами.

1. Для **заповнення курсору** використовується оператор **OPEN**.
2. Для **отримання окремих кортежів** з результируючого набору є оператор **FETCH**. Як правило, оператор **FETCH** виконується багато разів (для кожного кортежу результируючого набору).
3. Для **закриття курсору** застосовується оператор **CLOSE**. Це завершує **активну операцію з курсором і звільнення деяких ресурсів**. Після **закриття курсор залишається оголошеним**, тому його можна повторно відкрити оператором **OPEN**.

## 38. Обмеження цілісності рівня бази даних (ASSERTION)

???

## 39. Збережені процедури

**Процедура** (англ. Stored procedure) - це іменований програмний об'єкт БД. У SQL Server є процедури, що зберігаються декількох типів.

**Системні збережені процедури** (англ. System stored procedure) поставляються розробниками СУБД і використовуються для виконання дій з системним каталогом або отримання системної інформації. Їх назви зазвичай починаються з префікса "sp\_". Запускаються збережені процедури всіх типів за допомогою команди EXECUTE, яку можна скоротити до EXEC. Наприклад, збережена процедура sp\_helplogins, запущена без параметрів, формує два звіти про імена облікових записів (англ. Logins) і відповідних їм в кожній БД користувачів (англ. Users).

EXEC sp\_helplogins;

## Приклади системних збережених процедур SQL Server

Назва	Дія
spchangedbowner	Дозволяє змінити власника БД
spdatabases	Повертає список БД і їх розмір
sphelp	Приймає в якості вхідного параметра ім'я об'єкта і повертає про нього відомості. Наприклад: exec sp_help @ objname = 'dbo.Book1'
spstoredprocedures	Повертає список збережених процедур в поточній БД
sp_tables	Повертає список таблиць і уявлень в поточній БД, які можна запитувати

## Фізичне представлення даних

### 40. Типи памяті

???

### 41. Сторінкове представлення бази даних

???

### 42. Пришвидшення доступу до вторинних пристройв

???

### 43. Фізичне представлення полів даних

???

### 44. Фізичне представлення завписів

???

### 45. Фізичне представлення відношень

???

# Одновимірні індекси

## 46. Індекси на впорядкованих даних

???

## 47. Щільні індекси

**Щільний індекс** (Dense index) - індекс в базах даних, файл з послідовністю пар ключів і покажчиків на запис у файлі даних. Кожен ключ в щільному індексі, на відміну від розрідженоого індексу, має відповідний йому дороговказом на запис в сортованому файлі даних. Ідея використання індексів прийшла від того, що сучасні бази даних дуже масивні і не поміщаються в основну пам'ять. Ми зазвичай ділимо дані на блоки і розміщуємо дані в пам'яті по блоках. Однак пошук запису в БД може зайняти багато часу. З іншого боку, файл індексів або блок індексів набагато менше блоку даних і може поміститися в буфері основний пам'яті що збільшує швидкість пошуку записи. Оскільки, ключі відсортовані можна скористатися бінарним пошуком. У кластерних індексах з дубльованими ключами щільний індекс вказує на перший запис з вказаним ключем.

## 48. Розріджениі індекси

**Розріджений індекс** (Англ. Sparse index) в базах даних - це файл з послідовністю пар ключів і покажчиків. Кожен ключ в розрідженному індексі, на відміну від щільного індексу, має відповідний йому дороговказом на блок в сортованому файлі даних. Ідея використання індексів прийшла від того, що сучасні бази даних дуже масивні і не поміщаються в основну пам'ять. Ми зазвичай ділимо дані на блоки і розміщуємо дані в пам'яті по блоках. Однак пошук запису в БД може зайняти багато часу. З іншого боку, файл індексів або блок індексів набагато менше блоку даних і може поміститися в буфері основний пам'яті що збільшує швидкість пошуку записи. Оскільки ключі відсортовані, можна скористатися бінарним пошуком. У кластерних індексах з дубльованими ключами розріджений індекс вказує на найменший ключ в кожному блоці.

Щоб полегшити процедуру пошуку, можна створити другий файл під назвою розрідженим індексом, який складається з пар  $(x, b)$ , де  $x$  - значення ключа, а  $b$  - фізичну адресу блоку, в якому значення ключа першого запису дорівнює  $x$ . Розріджений індекс повинен бути відсортований за значеннями ключів.

## 49. Вторинні індекси (на невпорядкованих даних)

### (UNIQUE KEY / INDEX).

Під вторинним (альтернативним) ключем мають на увазі так зване обмеження UNIQUE, яке забезпечує унікальність даних. По суті це обмеження майже повністю відповідає обмеженню первинного ключа, оскільки вимагає наявності унікальних значень у всьому вказаному у ньому стовпці (чи комбінації стовпців) таблиці. При додаванні стрічки з дублюючим значенням для стовпця, для якого встановлено обмеження UNIQUE, SQL Server автоматично згенерує помилку та відхилить спробу введення такої стрічки.

**Примітка:** при створенні вторинного ключа, аналогічно як і для первинного ключа, SQL Server автоматично створює унікальний некластерний індекс.

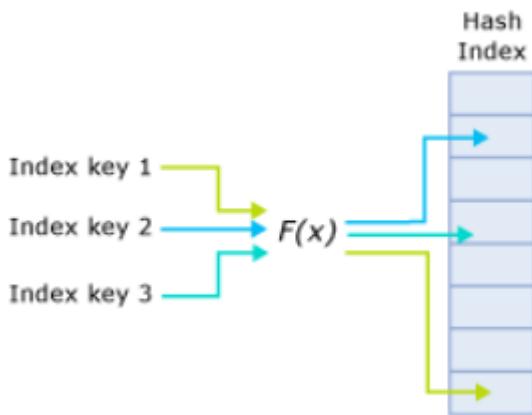
З іншого боку, створення вручну окремого унікального індексу теж забезпечує унікальність даних для визначеного стовпця (стовпців) таблиці. Фактично, для забезпечення унікальності даних немає різниці між створенням обмеження UNIQUE та створенням унікального індексу. Ця відмінність полягає лише в їхній семантиці: унікальний ключ призначений для забезпечення обмежень на дані, а індекси призначенні для прискорення доступу до даних.

Основною суттєвою відмінністю унікального ключа від унікального індексу є те, що він усе таки ключ, і на нього можуть посыпатися зовнішні ключі (FOREIGN KEY) для забезпечення цілісності даних. Особливістю ж індексів є те, що вони можуть бути як унікальними, так і не унікальними.

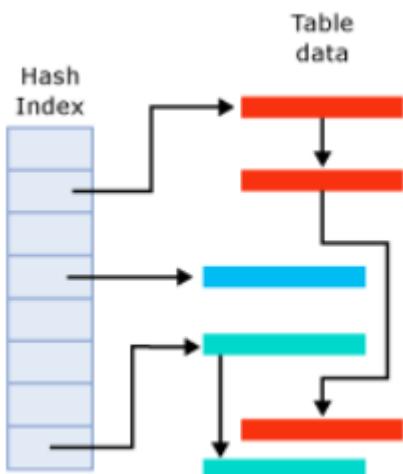


## 50. Індекси на основі хеш-функцій

Індекси використовуються в якості точок входу для таблиць, оптимізованих для пам'яті. Для зчитування рядків з таблиці потрібно індекс, який визначає місце розташування даних в пам'яті. Хеш-індекс складається з колекції контейнерів, організованих в масив. Хеш-функція зіставляє ключі індексу з відповідними контейнерами в хеш-індексі. На наступному малюнку показані три ключа індексу, зіставлені з трьома різними контейнерами в хеш-індексі. Для наочності хеш-функція називається  $f(x)$ .



Якщо два ключі індексу зіставляються з одним хеш-контейнером, відбувається конфлікт хеша. Велике число конфліктів хеша може чинити негативний вплив на операції читання. Структура хеш-індексу в пам'яті складається з масиву покажчиків пам'яті. Кожен контейнер пов'язаний з певним зміщенням в цьому масиві. Кожен контейнер в масиві вказує на перший рядок у цьому хеш-контейнері. Кожен рядок в контейнері вказує на наступний рядок, утворюючи таким чином ланцюжок рядків для кожного хеш-контейнера, як показано на малюнку нижче.



На малюнку зображено три контейнери з рядками. Другий контейнер зверху містить три нові рядки. Четвертий контейнер - одну блакитну рядок. Нижній контейнер містить дві зелені рядки. Це можуть бути різні версії одного рядка.

## 51. Індекси на основі Бі-дерев

Індекс на основі збалансованої ієрархічної структури, або індекс B-Tree (Balanced Tree structured object), використовується як індекс за замовчуванням в СКБД Oracle. Ця структура нагадує дерево (якщо дивитися знизу вгору), в якому спочатку читається самий верхній блок - кореневий

вузол (root), потім блок на наступному рівні - блок-гілка (branch) і так до тих пір, поки не буде витягнутий блок-лист (leaf) з ідентифікатором рядки.

Значення ключа зберігаються в індексі (рис. 11.1). Така структура дозволяє скоротити до мінімуму число операцій введення / виводу. Для отримання ідентифікатора рядка зазвичай потрібно одне відвідування блок-листа, тобто фізичної сторінки бази даних, відведененої під індекс.

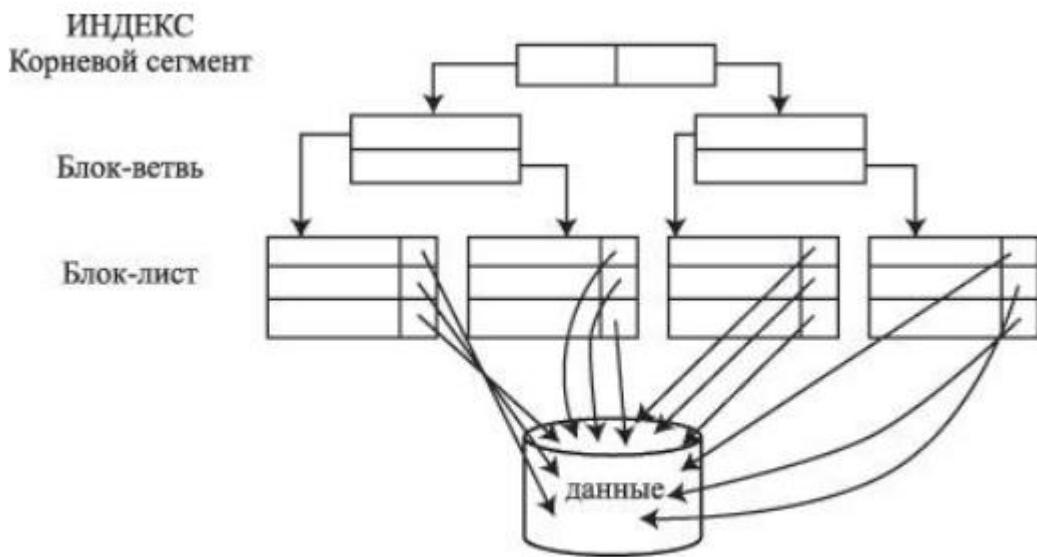


Рис. 11.1. Концептуальная организация B-Tree индекса

## Багатовимірні індекси

### 52. Багатовимірні сітки (Grid files)

???

### 53. Розподілені хеш-функції (partitioned hash)

???

### 54. KD Дерева

В інформації **k-d дерево** (англ. *k-d tree*, скорочення від *k-мірне дерево*) — це структурі даних з поділом простору для упорядкування точок в k-мірному просторі. *K-d* дерева використовуються для деяких застосувань, таких як пошук у багатовимірному просторі ключів (пошук діапазонів<sup>[len]</sup> і пошук найближчого сусіда). *K-d* дерева — особливий вид дерев двійкового поділу простору.

## 55. Q дерева

**Q-дерево** (*quadtree*) описує просторові відносини між елементом тами в межах будь-якої обмеженій площині. Наприклад, область може бути картою, а елементи будуть позначати розташування будинків або підприємств на ній. Кожен вузол в Q-дереві є частиною загальної області, представленої даним деревом. Кожен вузол, який не є листом, має чотири дочірніх, вузла, які відповідають північно-західному, північно-східному, південно-східному і південно-західному квадранту області вузла. Лист може зберігати елементи в свійському списку.

## 56. R дерева

**R-дерево** (англ. *R-trees*) — деревоподібна структура даних, яка використовується для організації доступу до просторових даних, тобто для індексації багатовимірної інформації, такої, наприклад, як географічні координати, прямокутники або многокутники.

### Структура R-дерева

Кожна вершина R-дерева має змінну кількість елементів (не більше деякого заздалегідь заданого максимуму). Кожен елемент не листової вершини зберігає два поля даних: спосіб ідентифікації дочірньої вершини і обмежує прямокутник (кубоїд), що охоплює всі елементи цієї дочірньої вершини. Всі збережені кортежі зберігаються на одному рівні глибини, таким чином, дерево ідеально збалансовано. При проектуванні R-дерева потрібно задати деякі константи:

- `maxNumOfEntries` — максимальне число дітей у вершини
- `minNumOfEntries` — мінімальне число дітей у вершини, за винятком кореня.

Для коректної роботи алгоритмів необхідно, щоб `minNumOfEntries <= maxNumOfEntries / 2`. У кореневій вершині може бути від 2 до `maxNumOfEntries` нащадків. Часто вибирають `minNumOfEntries = 2`, тоді для кореня виконуються ті ж умови, що і для інших вершин. Також іноді розумно виділяти окремі константи для кількості точок в листових вершинах, так як їх часто можна робити більше.

## 57. Bitmap індекси

Bitmap index - метод бітових індексів полягає в створенні окремих бітових карт (послідовність 0 і 1) для кожного можливого значення стовпця, де

кожному біту відповідає рядок з індексовані значенням, а його значення рівне 1 означає, що запис, відповідна позиції біта містить индексуємо значення для даного шпальти чи властивості.

Основна перевага бітових індексів в тому, що на великих множинах з низькою потужністю і хорошою кластеризацією за їх значенням індекс буде менше ніж B \* -tree.

## Транзакції

### 58. ACID властивості

#### ACID властивості:

**Atomicity** — атомарність, гарантує, що ніяка транзакція не буде зафікована в системі частково **або так або попередній стан**.

**Consistency** — узгодженість кожна успішна транзакція по визначенню **фіксує тільки допустимі результати**.

Якщо є обмеження **DEFERRED** : перевірка відбувається всередині транзакції

**Isolation** – ізольованість. Під час виконання транзакції паралельні транзакції не повинні впливати на її результат. Ця вимога дорога, тому в реальних БД існують режими, які не повністю ізоляють транзакцію (рівні ізольованості **Repeatable Read** і нижче).

**Durability** – довговічність. Незалежно від проблем на нижніх рівнях (зеструмлення системи або збої в обладнанні) зміни, зроблені успішно завершеною транзакцією, повинні залишитися збереженими після повернення системи в роботу).

## 59. Журнал транзакцій і його використання

Системний компонент СКБД для підтримки транзакцій називається **менеджером транзакцій**.

Ключовими операторами є **BEGIN, COMMIT, ROLLBACK**. Менеджер веде **журнал транзакцій**. Записується наступна інформація:

- Порядковий номер, тип і час зміни;
- Ідентифікатор транзакції;
- Об'єкт, що піддався зміні
- Попередній стан об'єкта і новий стан об'єкта.
- позначки початку і завершення транзакції, і позначки прийняття Контрольних точок.

## 60. Блокування

### Механізми блокування

Блокування, що накладаються СКБД називаються **неявним** блокуванням (**Implicit locks**), а блокування за командою користувача, - **явними** блокуванням (**Explicit locks**). **Блокування на окремі рядки, сторінки, таблиці, всю БД цілком**.

**Розмір блок. ресурсу** називається **глибиною деталізації блокування** (**Lock granularity**).

**Більша** глибина деталізації – СКБД краще адмініструє блокування, але **конфлікти**. **Зменшенні розміру** блокованого фрагмента – ймовірність конфлікту **знижується**, але **складніше управляти**. **Блокування означає**, що будь-яка операція над даними призводить до **встановлення стану блокування на відповідні записи**.

### 2 БАЗОВІ ТИПИ БЛОКУВАННЯ:

1. **Розділене (SHARED)** може виконуватись лише читання блокованого елемента даних, але не його оновлення
2. **Виняткове (EXCLUSIVE)** може виконуватись і читання цього елемента даних, і його оновлення

## 61. Аномалії при паралельній обробці

???

## 62. Рівні ізоляції транзакцій

### Рівні ізоляції і вирішувані ними проблеми

проблеми	рівні ізоляції			
	Read uncommitted	Read committed	Repeatable read	Serializable
втрачене поновлення	неможливо	неможливо	неможливо	неможливо
"Брудне читання"	можливо	неможливо	неможливо	неможливо
неповторюване читання	можливо	можливо	неможливо	неможливо
читання фантомів	можливо	можливо	можливо	неможливо

## 63. Взаємне блокування (Deadlock)

### ДЕДЛОК

Взаємне блокування виникає за одночасного виконання 4 умов:

1. Взаємне виключення. Кожен ресурс або зайнятий або вільний. Використання ресурсу є ексклюзивним.
2. Утримання та очікування. Захоплений ресурс є блокованим до завершення транзакції, яка може чекати на інші потрібні їй ресурси.
3. Відсутність примусового звільнення ресурсів. Транзакція сама вирішує, коли звільнити ресурс.
4. Циклічне очікування. Можливе утворення циклу очікувань за ресурсами.

### Способи вирішення:

- Нехтування проблемою
- Виявлення і відновлення
- Уникнення шляхом правильного розподілу ресурсів
- Невиконання принаймі однієї умови виникнення

## 64. Способи забезпечення паралелізму (Песимістичне блокування і MVCC)

**Песимістична:** блокування накладається перед передбачуваною модифікацією даних на всі модифіковані рядки. Виключена модифікація даних зі сторонніх сесій, дані з блокованих рядків доступні відповідно до рівня ізоляції транзакцій. Після закінчення модифікації гарантовано узгоджений запис результатів.

**(MVCC - MultiVersion Concurrency Control)** - один з механізмів забезпечення паралельного доступу до БД. Надання кожному користувачеві «знімка» БД, тобто, зміни в БД, які вносяться користувачем, невидимі для інших користувачів до моменту фіксації ТА-ції. Дозволяє обробку запису в будь-який час таким чином, що кожна ТА бачить свою версію даних, не заважаючи сусіднім. Таким чином, ТА-ції, що читають, не блокують ТА-ції, що пишуть і навпаки. Okрім того, це дає можливість відмовитись від логу ТА-цій і таким чином зменшити ймовірність пошкодження службової інформації бази даних.

## Розподілені бази даних

### 65. Головний принцип побудови розподілених баз даних

#### Загальні принципи

**Розподілена база** складається із **мережі зв'язаних між собою вузлів**, кожен з яких є повноцінною **СКБД** сам по собі, але разом з тим вузли взаємодіють між собою так, що **користувач має доступ до даних які містяться на будь-якому з вузлів**.

**РБД** можна розглядати як деяку **віртуальну базу даних**, компоненти якої містяться у різних фізичних базах даних на різних вузлах розподіленої системи.

**Два типи РСКБД:** однорідні (гомогенні); на всіх вузлах використовуються однотипні СКБД.

Максимум – всі вузли використовують **однакові технічні засоби** (однакові типи комп’ютерів і програмного забезпечення:

- операційні системи,
- програмного забезпечення СКБД ,
- моделей даних, що підтримуються.

## 66. CAP теорема

**Теорема CAP** (Fowler, 2000). У системі яка може підтримувати **узгодженість** (Consistency), **доступність** (Availability) та **розподіленість** (Partition Tolerance).

**Consistency** читання даних з будь-якого вузла приводить до тих самих даних, що знаходяться на декількох вузлах.

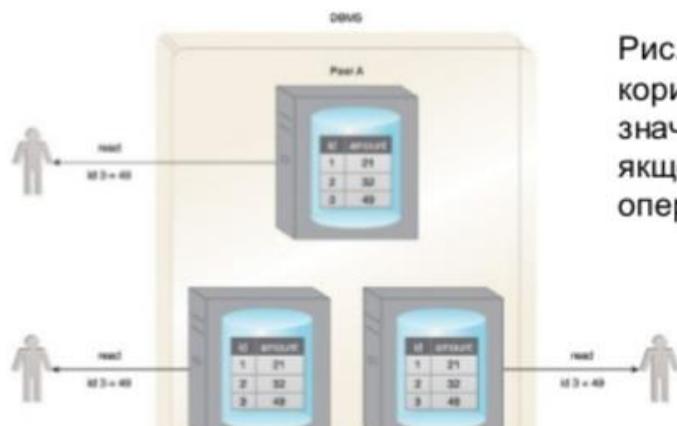
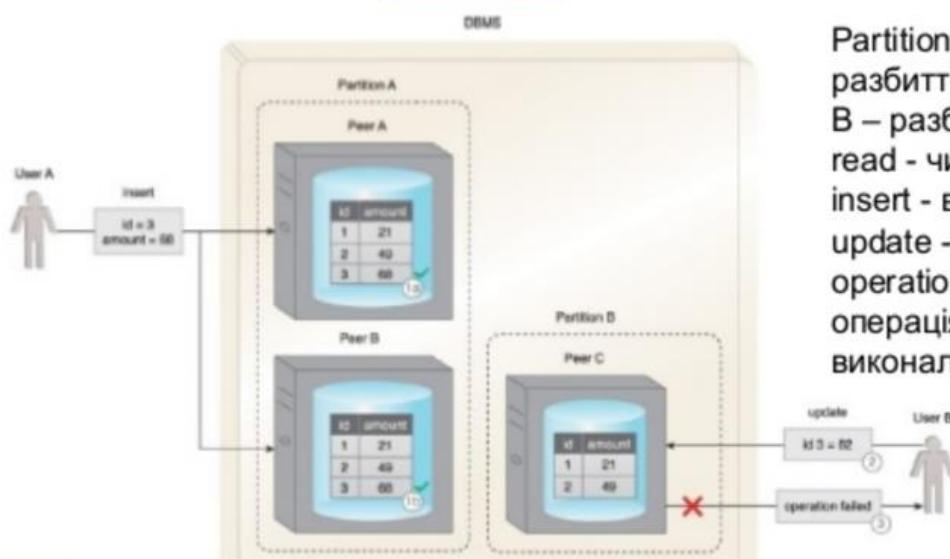


Рис. **узгодженість** усі три користувачі одержують однакові значення для суми стовпця, навіть якщо три різні вузли обслуговують операцію запису.

- **Доступність**. Запит на читання/запис завжди буде підтверджений у вигляді успіху або відмови [ ] .



Partition A –  
розділ A; Partition  
B – розділ B  
read - читання;  
insert - вставка;  
update - оновлення;  
operation failed -  
операція не  
виконалася;

- Рис. [ ] Доступність і стійкість при розбитті: у випадку порушення обміну даними запити від обох користувачів усе ще обслуговуються (1, 2). Однак у користувача В запис не виконується, тому що запис із id = 3 не була скопійована в одноранговий вузол С. К Сторінка в лінії 440 був спрощений (3) про те, що оновлення не вдалося.

## 67. BASE транзакції

### BASE

BASE є принципом проектування баз даних на основі теореми CAP і максимального використання концепцій систем баз даних, які використовують розподілену технологію. BASE розшифровується як:

- зовсім доступні; • basically available
- м'який стан ; • soft state
- випадкова погодженість . • eventual consistency

Коли база даних підтримує BASE, те це сприяє доступності завдяки погодженості . Інакше кажучи, база даних А+Р з погляду CAP. По суті, BASE використовує оптимістичні елементи керування паралелізмом, послаблюючи сильні обмеження погодженості , обумовлені властивостями ACID.

## 68. Розподілені запити

**Обробка розподілених запитів** - спеціальні модулі СКБД на вузлах для декомпозиції запитів та агрегації результатів.

## 69. Протокол двофазної фіксації транзакцій

Для гарантії дотримання **атомарності** розподіленої транзакції координатор може використовувати **протокол двофазної фіксації**:

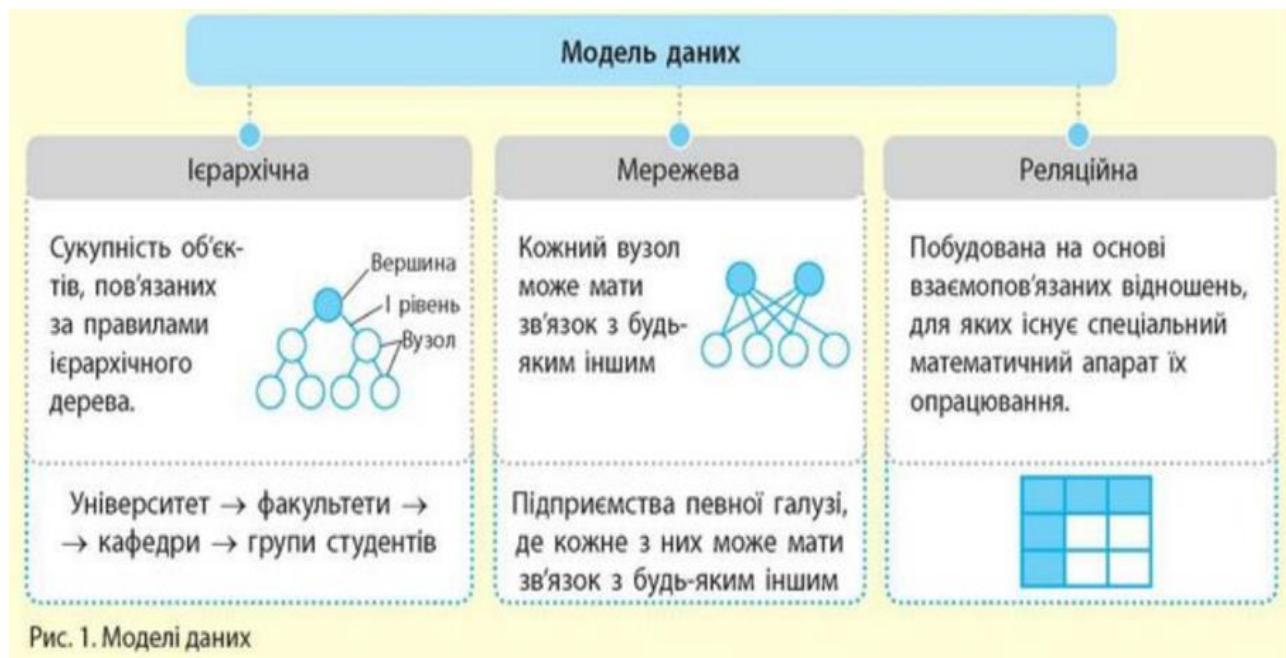
- **кожному вузлу** , якого торкається транзакція **надсилається повідомлення** про те, що він повинен зробити;
- **кожному вузлу надсилається запит про готовність** завершити дану розподілену транзакцію;
- кожен вузол відповідає "так" або "ні";
- коли **координатор отримав від всіх вузлів** повідомлення "так", то посилає **друге повідомлення** про готовність завершення транзакції можна; якщо **хоча б один вузол** не підтвердив свою готовність завершити транзакцію, то всім надсилається **повідомлення про відкат**.

Протокол двофазної фіксації **не гарантує** обов'язкового **виконання транзакції**.

Конкретні реалізації даного протоколу включають **вказівку тимчасового інтервалу**, протягом якого транзакція повинна бути виконана.

## Навігаційні моделі

70. Ієрархічна модель  
71. Сіткова (Мережева) модель



## XML

72. Коректність (well formed)

• **Коректність (well formed)** - Коректний документ ([англ. well-formed document](#)) відповідає всім [синтаксичним правилам XML](#).

Документ- *некоректний*, не може називатись XML-документом.

*Сумісний синтаксичний аналізатор* ([англ. Conforming parser](#)) не повинен обробляти *некоректні* документи.

## Коректний XML документ має:

- Лише **один елемент** у корені.
- **Непорожні** елементи розмічено **початковим та кінцевим тегами** (*наприклад*, <пункт>Пункт 1</пункт>).

**Порожні** елементи можуть помічатись «закритим» тегом,

- *наприклад* <IAmEmpty />.
- Що **еквівалентно** <IAmEmpty></IAmEmpty>.

- **Один елемент не може мати декілька атрибутів з одинаковим іменем.**

**Значення атрибутів** знаходяться або **в одинарних ('')**,  
**або у подвійних ("") лапках**.

- **Теги** можуть бути **вкладені**, але **не можуть перекриватись**.

Кожен **некореневий елемент** мусить **повністю** знаходитись **в іншому елементі**.

- **Документ** має складатися тільки з **правильно закодованих дозволених символів** множини **Юнікоду**.

XML-процесор **обов'язково** має розуміти **UTF-16** та **UTF-8**.

**Фактичне та задеклароване** кодування ([англ. character encoding](#)) документа мають збігатись.

## 73. Валідність (valid)

### • Валідність (valid)

– Документ називається **валідним**, якщо він є **коректним**, містить **посилання на граматичні правила та повністю відповідає обмеженням**, вказаним у цих правилах (**DTD** або **XML Schema** або іншому подібному документі).

## 74. Парсери і їх особливості

### • Синтаксичний аналізатор (parser)

– **СА** називається **програма або компонент**, що читає XML-документ, проводить **синтаксичний аналіз, та відтворює його структуру**. Якщо СА перевіряє документ на валідність, то такий аналізатор називають **валідатором**.

## 75. DTD

Найдавнішим форматом схем для XML є успадкований від SGML формат визначення типу документа (Document Type Definition, DTD). У той час, як через включення до стандарту XML 1.0 DTD став поширеним форматом схем, він має такі обмеження:

- Відсутність нових можливостей XML, із найважливішою з посеред них простори назв.
- Брак виразності. Деякі формальні аспекти XML-документів неможливо відобразити в DTD.
- Використовується спеціалізований, заснований не на XML синтаксис для опису схем.

DTD все ще використовується в багатьох програмах, оскільки він вважається найпростішим форматом для аналізу та збереження.

## 76. XML Schema

### Schema

мова опису схем **XML** опублікована **2001** року консорціумом **W3C** в якості рекомендацій.

Використовується **для визначення валідності XML** документа.

**Примірник схеми** називається **визначенням схеми** (**XML Schema Definition - XSD**)

**XSD** містить:

- **оголошення елементів**
- визначає **властивості** елементів
- **оголошення** атрибутів
- визначає **властивості** атрибутів
- **описи** простих і складених типів даних:
  - **модель груп** або **опис груп атрибутів** - іменовані групи елементів і атрибутів для повторного використання у схемі
  - **використання атрибутів (attribute use)** - описує **відношення складних типів** і описів **атрибутів** і вказує **обмеження** накладені на **значення атрибутів**
  - **використання елемента** - описує **відношення складних типів** та **елементів**

## 77. Трансформації. Дерево результату

### XSL

Дерево трансформацій дозволяє *структурі результатуючого дерева* суттєво відрізнятися від структури дерева початкового документа.

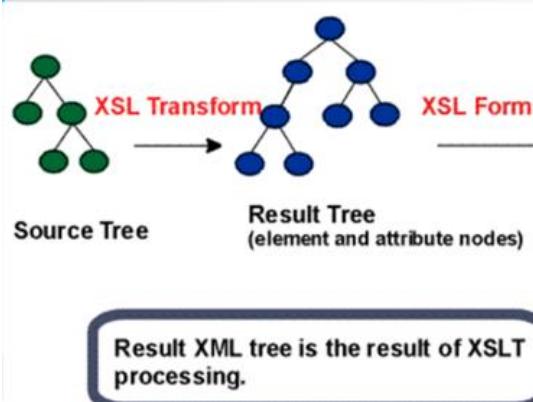
Форматування забезпечується включенням *форматувальної семантики у результатуюче дерево*.

Семантика форматування виражається в термінах *кatalogу класів об'єктів форматування* (FO). Вузли результатуючого дерева є *об'єктами форматування*. Класи об'єктів форматування описують, наприклад, типографічні абстракції (*сторінка, абзац, таблиця...*) В XSL класи об'єктів форматування і властивостей форматування формують **словник виразів для відображення**.

## 78. XSLT

### XSLT - XSL Transformation

- Трансформація формує **результатуюче дерево**, яке в XSL називається *деревом елементів та атрибутів*, з об'єктами з області імен "об'єктів форматування". У цьому дереві об'єкти є представлені як *XML-елементи з властивостями* описаними за допомогою *атрибути*. Вміст об'єкту форматування є вмістом *елементу результатуючого дерева*.
- Трансформація XSLT описується в *правильно-зформованому XML-документі*, який відповідає *Namespace Recommendation*, який може містити як **елементи визначені в XSLT** так і невизначені ним.



Визначені елементи **належать до** описаних областей імен (*namespace*). **Трансформація** описує **правила перетворення** початкового дерева у результатуюче. Досягається за допомогою **асоціації шаблонів із зразками**(масками/patterns) Зразок **порівнюється** з **елементами у початковому дереві**. Утворюються **примірники шаблона** для **генерації частини** результатуючого дерева.

## 79. XPath

**XPath** - мова виразів для адресації частин XML документу, або для обчислення величин (рядкових, числових або булевих) на основі вмісту XML документа.

У **XPath** використовується **компактний синтаксис**, відмінний від прийнятого в **XML**.

У 2007 році завершилася розробка **версії 2.0**, яка тепер є складовою частиною мови **XQuery**.

Мова **XPath** основана на представленні XML документа у вигляді **дерева**, і надає можливість навігації всередині дерева, вибирати **вузли за різними критеріями**.

## 80. XQuery

**XQuery** - мова запитів, розроблена для **обробки даних у форматі XML**.

**XQuery** використовує XML як свою **модель даних**.

**XQuery** надає **спосіб витягати і маніпулювати** даними з XML документів або ж з будь- якого джерела даних, що можуть розглядатися як XML, **наприклад**, з реляційних баз даних або з документів офісного пакету.

**XQuery** для адресації потрібних ділянок XML документа використовує синтаксис виразів **XPath**.

Це **доповнюється SQL-подібними «FLWOR-виразами»** для виконання **об'єднань**.

**FLWOR** вирази з конструктовані з **п'яти речень**, за чиєю абревіатурою вони й названі: **FOR, LET, WHERE, ORDER BY, RETURN**.

Мова базується на **три-структурній моделі інформаційного вмісту XML-документа**, що складається з **семи видів вузлів**:

1. вузли документа,
2. елементи,
3. атрибути,
4. текстові вузли,
5. коментарі,
6. інструкції обробки, і
7. простори імен.

## Інші нереляційні моделі даних

## 81. Об'єктно-орієнтована модель даних

**Об'єктно-орієнтована база даних** (ООБД) — база даних, в якій дані моделюються у вигляді класів і об'єктів<sup>[2]</sup>, їх атрибутів і методів.

Результатом суміщення можливостей (особливостей) баз даних і можливостей об'єктно-орієнтованих мов програмування є об'єктно-орієнтовані системи керування базами даних (ООСКБД). ООСКБД дозволяє працювати з об'єктами баз даних так само, як з об'єктами у програмуванні в ООМП. ООСКБД розширює мови програмування, прозоро додаючи довгочасні дані, керування паралелізмом, поновлення даних, асоційовані запити та інші можливості.

Деякі об'єктно-орієнтовані бази даних розроблені для щільної взаємодії з такими об'єктно-орієнтованими мовами програмування як Python, Java, C#, Visual Basic .NET, C++, Objective-C і Smalltalk; інші мають свої власні мови програмування. ООСКБД використовують точно таку ж модель, що й об'єктно-орієнтовані мови програмування.

СУБД повинна забезпечувати:

- довготермінове зберігання;
- використання зовнішньої пам'яті;
- паралелізм;
- відновлення;
- нерегламентовані запити.

## 82. Об'єктно-реляційні бази даних

**Об'єктно-реляційна база даних** (англ. *object-relational database*, *ORD*), чи **Об'єктно-реляційна СКБД**, це система керування базами даних подібна до реляційної, але з об'єктно-орієнтованою моделлю бази: об'єкти, класи та наслідування підтримуються в схемі даних та мові запитів. На додачу, just як і в реляційних БД, вона підтримує розширення моделі даних новими типами даних та методами.

Об'єктно-реляційна база даних, можна сказати, забезпечує проміжне положення між реляційними базами даних і об'єктно-орієнтованими базами даних (об'єктними базами даних). В об'єктно-реляційних базах даних, підхід, по суті, як у реляційних базах даних: дані зберігаються в базі даних і маніпулюють всі разом із запитами мовою запитів; на іншому полюсі знаходиться OODBMSes, в якій база даних є по суті стійким об'єктом сховищем для програмного забезпечення, написаного об'єктно-орієнтованою мовою програмування, з програмуванням API для зберігання та вилучення об'єктів, і мало або взагалі не має конкретної підтримки запитів.

### **83. Асоціативна модель даних**

**Асоціативна модель даних** (англ. *Associative model of data*) - це запропонована Саймоном Вільямсом модель представлення даних, в якій база даних складається з двох типів структур даних - елементів і посилань, що зберігаються в єдиній однорідної загальній структурі в якості альтернативи реляційної та об'єктної моделей даних. Близька до моделі даних сущность-зв'язок.

### **84. ЕАВ модель даних**

**Модель Сутність-Атрибут-Значення (ЕАВ)** - це модель даних, призначена для опису сутностей, в яких кількість атрибутів (властивостей, параметрів), що характеризують їх, потенційно величезна, але то кількість, яке реально буде використовуватися в конкретної суті, відносно мало.

### **85. Основні моделі даних в NOSQL**

**NoSQL** (зазвичай розшифровується як англ. non SQL або англ. non relational<sup>11</sup>, іноді англ. not only SQL) — база даних, яка забезпечує механізм зберігання та видобування даних відмінний від підходу таблиць-відношень в реляційних базах даних. Подібні бази даних існували вже в другій половині 1960-х років, але тоді вони ще не здобули гучне ім'я «NoSQL», одержане після сплеску популярності на початку 21-ого століття, що був спричинений потребами Web 2.0 компаній, такими як Facebook, Google, та Amazon.com. NoSQL бази даних все більше і більше використовуються в задачах із застосуванням великих даних та real-time web-застосунках.<sup>1</sup> NoSQL системи також називають «Not only SQL» (англ. not only SQL — не тільки SQL) для підкреслення того, що вони можуть підтримувати SQL-подібну структуру та мову запитів.

Оригінальна назва	Тип	Приклад
Key-Value Cache	Ключ-значення: кеш	Coherence, eXtreme Scale, GigaSpaces, GemFire, Hazelcast, Infinispan, JBoss Cache, Memcached, Replicated, Terracotta, Velocity
Key-Value Store	Ключ-значення: ховище	Flare, Keyspace, RAMCloud, SchemaFree, Hyperdex, Aerospike
Key-Value Store (Eventually-Consistent)	Ключ-значення: ховище (випадково-узгоджене)	DovetailDB, Oracle NoSQL Database, Dynamo, Riak, Dynamite, MotionDb, Voldemort, SubRecord
Key-Value Store (Ordered)	Ключ-значення: ховище (впорядковане)	Actord, FoundationDB, Lightcloud, LMDB, Luxio, MemcacheDB, NMDB, Scalaris, TokyoTyrant
Data-Structures Server	Сервер структурованих даних	Redis
Tuple Store	Кортеж: ховище	Apache River, Coord, GigaSpaces
Object Database	Об'єктна база даних	DB4O, Objectivity/DB, Perst, Shoal, ZopeDB
Document Store	Документ: ховище	Clusterpoint, Couchbase, CouchDB, DocumentDB, IBM Domino, MarkLogic, MongoDB, Qizx, RethinkDB, XML-databases
Wide Column Store	Широко-колонкове ховище	BigTable, Cassandra, Druid, HBase, Hypertable, KAI, KDI, OpenNeptune, Qbase

## Інтеграція даних

### 86. Об'єднана база даних (Federated database)

**Система об'єднаної бази даних** - це тип системи управління мета-бази даних (СУБД), яка прозоро відображає кілька автономних систем баз даних в єдину об'єднану базу даних. Складові бази даних з'єднані між собою через комп'ютерну мережу і можуть бути географічно децентралізовані. Оскільки складові бази даних систем залишаються автономними, федераційна система баз даних є контрастною альтернативою (іноді складному) завдання об'єднання декількох розрізнених баз даних. Об'єднана база даних або віртуальна база даних є складовою всіх складових баз даних у федераційній системі баз даних. В результаті об'єднання даних не існує фактичної інтеграції даних у складові розрізнені бази даних.

Через абстракцію даних системи федераційних баз даних можуть надавати єдиний користувальницький інтерфейс, що дозволяє користувачам і клієнтам зберігати і отримувати дані з декількох безперервних баз даних з одним запитом, навіть якщо складові бази є неоднорідними. З цією метою система об'єднаної бази даних повинна мати можливість розкладати запит на підзапити для подання до відповідних складових СУБД, після чого система повинна складати набори результатів підзапитів. Оскільки різні системи управління базами даних використовують різні мови запитів, системи об'єднаних баз даних можуть застосовувати обгортки до підзапитів, щоб перевести їх у відповідні мови запитів.

## 87. Сховища даних (Data Warehouse)

Сховище даних (англ. *data warehouse*) — предметно орієнтований, інтегрований, незмінний набір даних, що підтримує хронологію і здатний бути комплексним джерелом достовірної інформації для оперативного аналізу та прийняття рішень. В основі концепції сховища даних (СД) лежить розподіл інформації, що використовують в системах оперативної обробки даних (OLTP) і в системах підтримки прийняття рішень (СППР). Такий розподіл дозволяє оптимізувати як структури даних оперативного зберігання для виконання операцій введення, модифікації, знищення та пошуку, так і структури даних, що використовуються для аналізу. В СППР ці два типи даних називаються відповідно оперативними джерелами даних (ОДД) та сховищем даних.

