Анатолій Пилипчук

ПМі-31

|  |
| --- |
| ЛНУ ім. І. Франка |
| Бази даних та інформаційні системи |
| Конспект лекцій, семестр 5 |

|  |
| --- |
|  |

# Вступ

Де використовуються бази даних:

* На виробництві;
* Паспорти, свідоцтва про народження тощо;
* Навчальні заклади;
* Комп’ютери, телефони, інші пристрої;
* Майже всюди.

Для роботи з базами даних використовуються мови програмування баз даних. В основному вони не є повні за Тюрінгом. Але спричиняють стислі вирази, у яких ми пишемо, що ми хочемо отримати, а не як. Одразу ж виникають проблеми: як швидко виконати такий вираз, чи є оптимальніший метод виконання тощо.

Файлова система – найпростіша база даних. Є ієрархія. В кінці цієї ієрархії зберігаються дані у файлах.

Якщо виконуються ці правила для операції (ACID), то швидше за все проблем не буде:

* Atomicity;
* Consistency;
* Isolation;
* Durability;

Означення База даних (БД) – впорядкований набір логічно взаємопов’язаних даних, що використовується спільно, та призначений для задоволення інформаційних потреб користувачів.

Часто під БД розуміють систему керування базою даних (СКБД).

Головним завданням БД є гарантоване збереження значних обсягів інформації (так звані записи даних) та надання доступу до неї користувачу або прикладній програмі.

База даних складається з 2 частин: дані та системи керування ними.

## Історія розвитку баз даних

1960-ті роки: Розробка перших баз даних, зокрема CODASYL (розробка North American Rockwell), де використовувалась мережева модель даних, і яка пізніше була взята за основу для розробки СКБД IMS від IBM.

1970-ті роки: Наукове обґрунтування Едґаром Коддом, яке зацікавило лише наукові кола, і використовувалось тільки в Ingres та IBM.

1980-ті роки: Поява перших комерційних реляційних баз даних від Oracle та DB2. Реляційні бази даних починають витісняти ієрархічні. В теорії починаються дослідження розподілених баз даних.

1990-ті роки: Увага спрямовується до об’єктно-орієнтованих баз даних.

2000-ті роки: Підтримка та застосування XML у базах даних. Розробники комерційних баз даних отримують конкуренцію зі сторони безкоштовних баз даних. Реакцією на це стає випуск безкоштовних версій комерційних баз даних.

2010-ті роки: Одним з напрямків розвитку є No-SQL-бази даних.

## Основні реляційні бази даних

Комерційні:

* DB2;
* Oracle;
* MS SQL Server

Відкриті:

* MySQL;
* Firebird;
* PostgreSQL.

## Моделі даних

Моделі даних складаються з:

* Математичне представлення даних:
  + У реляційній моделі дані зображуються реляціями (відношеннями) і таблицями;
  + У ієрархічній моделі дані зображуються графами і деревами
* Операції над даними;
* Обмеження.

Схема відношення – це назва відношення та список атрибутів. Також можна додавати тип атрибутів.

База даних – це колекція відношень. Схема бази даних – це множина всіх схем відношень у цій базі даних.

Таблиці використовуються, тому що вони є найпростішими для розуміння. На основі відношень дуже просто можна побудувати мову інтерпретації даних SQL.

Напівструктуровані дані – модель, яка базується на графах і деревах. Дозволяє гнучко представляти дані. Графи у напівструктурованих даних мають вузли (об’єкти), написи на ребрах (назви атрибутів), значення у листках.

XML (Extensible Markup Language) – розширювана мова розмітки.

# Entity Relationship Model

## ER-діаграми

ER-діаграма дозволяє намалювати схему бази даних. Включає в себе певні обмеження, але не операції.

Основними завданнями ER-діаграми є:

* Візуальне зображення структури бази даних;
* Засіб обговорення бази даних між професіоналами і непрофесіоналами.

Entity – це сутність. Entity set – множина сутностей. Аналогами в ООП є об’єкт і клас.

Атрибут – це властивість кожної з сутностей в множині сутностей. Атрибути є простими значеннями (integer, string тощо). Атрибути не є структурами, множинами та іншими складеними об’єктами.

В ER-діаграмі entity set позначається прямокутником, атрибут – овалом з лінією до entity set.

Ключі в ER-діаграмах підкреслюються.

## Відношення зв’язку

Відношення зв’язку (relationship) поєднує одну, дві та більше сутностей. Малюється ромбом з лініями до множин сутностей, які він пов’язує. Відношення зв’язку теж можуть мати свої атрибути.

Відношення «багато до багатьох» – бінарне відношення зв’язку, яке поєднує багато сутностей з багатьма сутностями. Кожна сутність першої множини може бути з’єднана з довільною кількістю сутностей другої множини, а кожна сутність другої множини – з довільною кількість сутностей з першої множини.

Відношення «багато до одного» – бінарне відношення зв’язку, яке поєднує багато сутностей з однією. Кожна сутність першої множини з’єднана не більше, ніж з однією сутністю другої множини. Сутності другої множини можуть бути з’єднані з довільною кількістю сутностей з першої множини.

Відношення «один до одного» – бінарне відношення зв’язку, яке поєднує одну сутність з однією сутністю. Кожна сутність першої множини може бути з’єднана з однією сутністю другої множини, а кожна сутність другої множини – з однією сутністю першої множини.

## Позначення кількості на ER-діаграмах

У відношенні «багато до одного» ставиться стрілка біля одного. У відношеннях «один до одного» стрілки ставляться з обох боків.

Позначення «замальована стрілка» означає «точно один».

Атрибутів у відношеннях зв’язку можна позбуватися, створюючи додаткову таблицю, і це відношення зв’язку буде поєднувати вже не дві, а три таблиці.

## Ролі і підкласи

Якщо множина сутностей з’являється у відношенні зв’язку більше одного разу, можна писати роль.

Підклас – частина сутностей, які мають певну властивість. У ER-діаграмах підкласи формують дерево. Трикутник «is-a» використовується для вказання підкласу.

## Перетворення з ER-діаграм у відношення

Множина сутностей перетворюється у відношення, атрибути переходять в атрибути, відношення зв’язку теж перетворюються у відношення, атрибутами якого є ключі поєднуваних відношень і атрибути відношення зв’язку.

Перетворення відношень зв’язку «багато до багатьох» утворює три таблиці. Відношення «багато до одного» утворює дві таблиці. Саме відношення об’єднується зі стороною «багато».

Є три способи перетворення відношення «is-a» у таблиці:

1. Об’єктно-орієнтований: одна таблиця на кожен підклас зі всіма атрибутами, що є в ньому.
2. З використанням NULL: одна таблиця для всієї ієрархії класів, значення NULL вписується у тих колонках, яких сутність не має.
3. ER-підхід: по одному відношенню на кожен підклас, спільні атрибути виносяться в таблицю для верхнього класу.

# Реляційна алгебра

## Основні операції

Алгебра – це математична система, яка складається з:

* операндів – змінні або значення, з яких конструюються інші значення;
* операторів – символи, які позначають дії над операндами, які створюють нове значення.

Реляційна алгебра – це алгебра, де операндами є відношення, таблиці, операторами є дії, які найчастіше виконуються над таблицями.

Класи базових операцій:

1. Перетин, об’єднання та різниця – звичайні множинні операції, лише схема операндів має збігатись.
2. Вибір певних рядків за умовою.
3. Проекція певних колонок за заданим списком.
4. Добутки та об’єднання.
5. Перейменування відношення та атрибутів.

Вибір: . – умова.

Проекція: . – список атрибутів. Якщо ми оперуємо множинами сутностей, тоді після проекції можуть виникнути однакові сутності. Відповідно, проекція витирає дублікати.

Розширена проекція: у списку крім атрибутів можуть бути прості дії над ними.

Добуток: . До кожного з рядків дописується рядок .

Тета-об’єднання: . – умова. Слово «тета» є застарілим.

Природне з’єднання: . Природне з’єднання – це тета-з’єднання з умовою рівності однаково названих колонок.

Перейменування: . Вказується, як перейменувати , щоб з’явилось . Перейменування є корисним зокрема для об’єднання, перетину і різниці.

Вирази у реляційній алгебрі будуються за допомогою запису операторів з круглими дужками та пріоритетом. Можна будувати вирази послідовністю присвоєнь. Можна будувати графічно, зображуючи дерево виразу.

Пріоритети:

1. , ;
2. Добуток та з’єднання;
3. Перетин;
4. Об’єднання та різниця.

В дереві виразів листки є операндами, внутрішні вузли є операторами, які застосовуються до дочірніх елементів.

## Реляційна алгебра над мультимножинами

Мультимножина – це множина, де один елемент може зустрічатись більше одного разу.

Деякі операції, наприклад проекція, ефективніше працюють над мультимножинами. Деякі, наприклад об’єднання, ефективніше працюють на множинах.

Об’єднання мультимножин:

Перетин мультимножин:

Різниця мультимножин:

Правила дій над множинами і мультимножинами є різними. Тому тотожності, які виконуються на множинах, переважно не виконуються на мультимножинах.

Загалом SQL працює над мультимножинами.

# SQL

В SQL записуємо, що ми хочемо зробити, не описуючи, як це зробити.

Алгоритм виконання вибирається самою СКБД, яка враховує кількість даних в таблиці, розмір цих даних, наявність індексів, завантаження цих таблиць тощо.

## Вираз SELECT

Найпростіший запит SQL – це SELECT. Найпростіший варіант – SELECT 2. Ускладнення SELECT – SELECT, FROM, WHERE. У частині SELECT пишеться аналог розширеної проекції. Перейменування колонок теж відбувається у цій частині. У частині FROM виконується з’єднання та добутки. Щоб виконати добуток двох таблиць, вони записуються через кому. У частині WHERE пишеться аналог вибору.

\* у частині SELECT означає всі колонки. Перейменування відбувається за допомогою AS.

Аналогічно з розширеною проекцією, у SELECT можна писати прості вирази.

Булівські вирази записуються за допомогою AND, OR, NOT. Порівняння значень на рівність – =, на нерівність – <>. Порівняння – <, >, >=, <=. Деякі СКБД можуть використовувати != для порівняння на нерівність.

Для порівняння рядка з шаблоном використовується вираз row [NOT] LIKE template. % використовується в значенні «будь-що». \_ використовується як будь-який символ.

NULL означає, що значення цієї колонки не застосовне до цього рядка або значення відсутнє або невизначене. Порівняння з NULL ніколи не дає TRUE, але й не дає FALSE. Булева логіка у SQL є триарною. Результатом будь-якого виразу є TRUE, або FALSE, або UNKNOWN. Порівняння NULL з будь-чим (навіть з NULL) дає UNKNOWN.

Рядок після фільтрування в частині WHERE іде в результат іде тільки тоді, коли там TRUE. Він не іде в результат, якщо там FALSE або UNKNOWN.

Для порівняння з NULL використовуються оператори IS та IS NOT – row IS [NOT] NULL. TRUE ≡ 1. FALSE ≡ 0. UNKNOWN ≡ 0.5.

## Підзапити

Якщо записати запит SELECT у дужках, то залежно від його типу його можна буде використати у багатьох місцях.

Якщо запит повертає одне значення, його можна використовувати у частині SELECT чи у частині WHERE у порівнянні цього значення з іншим. Якщо він повертає багато рядків/колонок, його можна використовувати у частині FROM.

Якщо від підзапиту очікують одного значення, а він поверне більше або менше, виникне помилка виконання.

Оператори IN та NOT IN використовуються для визначення того, чи значення є в певній таблиці або підзапиті.

Оператор EXISTS(subquery) повертає TRUE, якщо subquery повертає хоч один рядок.

Оператор ANY(subquery) використовується для визначення хоча б одного елемента в subquery.

Оператор ALL(subquery) використовується для визначення всіх елементів в subquery.

Оператори UNION, INTERSECT та EXCEPT використовуються для об’єднання, перетину і різниці підзапитів – subquery UNION/INTERSECT/EXCEPT subquery.

За замовчуванням у SELECT використовуються мультимножини, а в UNION, INTERSECT та EXCEPT використовуються множини, а дублікати видаляються. Для того, щоб SELECT видавав множини, потрібно написати SELECT DISTINCT. Для того, щоб UNION, INTERSECT та EXCEPT видавали мультимножини, потрібно дописати слово ALL.

## Добутки та JOIN

* R NATURAL JOIN S – природне з’єднання
* R [CROSS] JOIN S – звичайний добуток.
* R JOIN S ON умова – тета-з’єднання.

# Реляційна алгебра (продовження)

## Розширення реляційної алгебри

* – стирання дублікатів: ;
* – сортування: , – атрибути , за якими відбувається сортування;
* – групування та агрегація.

Функції агрегації:

* SUM – сума;
* AVG – середнє значення;
* MAX – максимальне значення;
* MIN – мінімальне значення;
* COUNT – кількість елементів.

Оператор групування: , – окремий атрибут групування або функція агрегації. Алгоритм:

1. З таблиці вибираються значення з однаковим атрибутом групування.
2. Обчислюється функція агрегації.
3. З кожної групи представником іде один рядок зі значенням функції агрегації.

## Зовнішні з’єднання

Різновиди зовнішнього з’єднання:

* FULL OUTER JOIN – природне з’єднання, в яких всі рядки, які не мали би вписуватись в результат, вписуються зі значенням NULL для атрибутів, які вони не мають.
* LEFT OUTER JOIN – те саме, що і FULL OUTER JOIN, але вписуються всі елементи тільки з лівої таблиці.
* RIGHT OUTER JOIN – те саме, що і FULL OUTER JOIN, але вписуються всі елементи тільки з правої таблиці.

## Групування та агрегація

NULL ігнорується агрегацією, крім COUNT (\*).

Оператор групування – GROUP BY.

Для того, щоб створювати групи, треба:

1. Записати атрибути групування у частині SELECT (не обов’язково).
2. Записати атрибути групування в частині GROUP BY (обов’язково).

В частині SELECT не може бути нічого, крім атрибутів групування (якщо вони є).

Частина HAVING може іти після частини GROUP BY для фільтрування груп.

У HAVING можуть бути:

1. Атрибути групування.
2. Функції агрегації.
3. Підзапити

# Модифікація бази даних

CRUD – Create, Read, Update, Delete. Read – SELECT. Create – INSERT. Update – UPDATE. Delete – DELETE.

## Вираз INSERT

INSERT INTO relation [(list of attributes)]  
VALUES (list of values);

INSERT INTO relation [(list of attributes)]  
підзапит;

## Вираз DELETE

DELETE FROM relation  
WHERE condition;

Рядки видаляються тільки після того, як виконається підзапит.

## Вираз UPDATE

UPDATE relation  
SET col1=’value’, …, coln=’value’  
WHERE condition;

# Обмеження даних

Обмеження (англ. constraint) – це відношення між елементами даних, які примусово підтримує СКБД. Наприклад, обмеження унікальності на ключ.

Тригер – це дія, яка відбувається під час настання певних умов. Тригери дуже часто є причиною неясної (неявної) поведінки бази даних.

Типи обмежень:

1. Ключ – в певній колонці чи наборі колонок може бути тільки унікальне значення.
2. Зовнішній ключ – посилання на іншу таблицю.
3. Обмеження на значення колонки – задається на певне значення певного типу.
4. Обмеження на значення рядка.
5. Загальні обмеження.

## Ключі

Колонки, позначені як ключі, мають унікальні значення, і визначаються за допомогою UNIQUE або PRIMARY KEY.

Для зовнішнього ключа СКБД не дозволяє вводити значення, яких немає в певній колонці таблиці, на які посилається ця таблиця.

Якщо є зовнішній ключ з таблиці A до таблиці B, може відбутись наступне:

1. Додавання або оновлення таблиці A створює значення в таблиці B, якого немає в цій таблиці.
2. Оновлення або витирання рядків з таблиці B призводить до того, що деякі рядки у таблиці A перестають мати відповідник у таблиці B.

Способи вирішення проблеми 2:

1. Заборонити модифікацію основного ключа, на який посилаються зовнішні ключі.
2. Каскадне оновлення – витерти стерті, оновити змінені.
3. Встановлення NULL.

При визначенні зовнішнього ключа для задання способу вирішення проблеми оновлення/видалення треба написати ON [UPDATE|DELETE] [SET NULL|CASCADE]. Для задання першого способу не потрібно писати нічого, тому що він є за замовчуванням.

## Перевірка значень

Деякі СКБД дозволяють дописати CHECK при визначенні колонки або в схемі для задання обмеження на значення колонки. В CHECK записується умова, можна записувати підзапит.

## Тригери

Мотивація для тригерів:

1. ASSERTION – дуже потужний механізм, але неясно, коли він має перевірятись.
2. Перевірки на атрибут та рядок досить просто перевіряти, але вони не є потужним механізмом.
3. Тригери дозволяють специфікувати, коли щось має відбуватись, і дозволяють вказувати складнішу дію.

Тригери також називають event, condition, action:

1. Event – подія, яка спричиняє виконання тригера.
2. Condition – умова виконання тригера.
3. Action – будь-який SQL-запит.

Всі тригери виконуються до реальної зміни бази даних, відповідно, якщо у тригері виникне помилка, то зміни не відбудуться.

# Транзакції, VIEW, індекси

## Контроль порядку виконання запитів

Транзакція – процес, який поєднує запити до бази даних та її модифікацію з чіткими правилами послідовного виконання. Транзакція формується або неявно з кожного окремого SQL-запиту, або явно, використовуючи BEGIN TRANSACTION і далі або ROLLBACK, або COMMIT.

Властивості транзакцій (ACID):

1. Atomicity – атомарність;
2. Consistency – цілісність;
3. Isolation – ізоляція;
4. Durability – стійкість – після закінчення транзакції гарантується, що дані будуть збережені.

СКБД підтримують і слабші рівні транзакцій у порівнянні з вимогами ACID.

BEGIN TRANSACTION починає транзакцію. COMMIT завершує транзакцію і гарантує, що дані будуть збережені. ROLLBACK скасовує всі зміни так, ніби транзакція і не починалась. Деколи ROLLBACK виконується неявно, наприклад, при виникненні помилки виконання.

Транзакція випливає на те, як відображаються дані.

Є 4 базових рівні ізоляції даних у транзакції (5 для Microsoft):

1. SERIALIZABLE – єдиний, який відповідає вимогам ACID;

## REPEATABLE READ;

1. READ COMMITED;
2. READ UNCOMMITED.
3. SNAPSHOT AT – тільки для Microsoft.

Рівень ізоляції встановлюється через SET ISOLATION LEVEL.

## Віртуальні та матеріалізовані VIEW

VIEW – це відношення, яке визначається через інші відношення. VIEW бувають віртуальними (реально не зберігаються в БД) та матеріалізованими (реально зберігаються в БД). Створюються через CREATE [MATERIALIZED] VIEW name AS query;. За замовчуванням VIEW є віртуальними.

Загалом неможливо модифікувати віртуальний VIEW, оскільки його як такого не існує, але можна додавати INSTEAD OF-тригери і реалізовувати якусь логіку додавання рядків.

## Пришвидшення доступу до даних

Індекс – це структура даних, яка дозволяє пришвидшити певні запити за певними критеріями.

Є кластеризований індекс – один на таблицю, та звичайні індекси з доданими колонками або без таких – багато на таблицю.

Спосіб реалізації індексів невизначений. Найчастіше використовуються так звані B-дерева.

Якщо дані в таблиці непосортовані, то пошук має складність , а додавання – .

Кластеризований індекс – спосіб зберігання даних при якому дані в таблиці посортовані. Звичайний індекс – додаткова таблиця, в якій дані оригінальної таблиці посортовані за певним атрибутом.

Пошук за атрибутом, за яким посортовано, має складність , а додавання – .

Індекси необов’язково пришвидшують роботу БД, деколи є надлишковими, і призначені для вузького класу виразів у частині WHERE.

# XML

## Визначення

XML (Extensible Markup Language) – розширювана мова розмітки.

Well-formed XML – XML сформовано відповідно до правил написання XML. Зокрема, для XML є такі правила:

* <?xml version="1.0" encoding="utf-6" standalone="yes"?>;
* Один кореневий тег;
* Внутрішні теги не перетинаються.

Valid XML – це XML, який відповідає певним вимогами. Ці вимоги оформлюють у DTD (Document type definition) або в XML Schema.

## Document Type Definition

Структура DTD така:

<!DOCTYPE root\_tag[  
 <!ELEMENT name (components)>  
]>

Підтеги мають зустрічатись в XML у заданій послідовності. Альтернативні послідовності позначаються через |.

Щоб використати DTD до XML, потрібно записати standalone="no" та записати сам DTD прямо перед XML, причому можна завантажити з іншого файлу:

<!DOCTYPE root\_tag SYSTEM 'file.dtd'>

Список атрибутів визначається через ATTLIST:

<!ATTLIST element attribute type #REQUIRED|#IMPLIED>

На кожен атрибут потрібен окремий ATTLIST.

Якщо type – CDATA, то атрибут може мати довільне значення.

Якщо додати до певного елемента атрибут з типом ID, тоді значення цього атрибута мусить бути унікальним по всьому XML-документі. Для посилання на цей унікальний атрибут використовуються атрибути типу IDREF або IDREFS.

## XML Schema

XML Schema – це спосіб задання правил XML, аналогічний до DTD і є потужнішим за нього. На відміну від DTD, XML Schema є XML-документом.

Структура XML Schema:

<?xml version="1.0" encoding="utf-8" standalone="yes"?>  
  
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">  
 <xs:element name type />  
</xs:schema>

name – назва тега, type – тип тега, який може бути стандартним (наприклад, xs:string) або визначеним користувачем.

Для визначення складного типу використовується xs:complexType, який має атрибут name, в основному також піделемент xs:sequence, у якому зустрічаються xs:element. Атрибути minOccurs та maxOccurs вказують, скільки разів цей елемент зустрічається. Значенням цих атрибутів є число або unbounded.

Атрибути визначаються через xs:attribute. xs:attribute може бути визначений у xs:complexType і має атрибути use, який може мати значення required або optional, name, який визначає назву, і type, який визначає тип.

xs:simpleType – перерахування та обмеження на значення для простих типів. Має атрибут name та піделементи xs:restriction.

xs:restriction має атрибути base, який визначає базовий тип. Також може мати атрибут xs:{min|max}{Inclusive|Exclusive}, а також піделементи xs:enumeration.

xs:element може мати піделемент xs:key, який в свою чергу має піделементи xs:selector та xs:field. Пов’язаним з цим елементом є елемент xs:keyref, який є посиланням на ключ.

## XPath, XQuery та XSLT

XPath – проста мова для знаходження елементів XML та їх значень.

XQuery – мова виразів над XML та XPath.

XSLT – мова перетворення XML-даних.

Аналогом до відношення у реляційній моделі тут є послідовність сутностей. Сутністю є або примітивне значення, або вузол. Є три типи вузлів:

* Документ – це весь документ;
* Елемент – починається з відкриваючого тега і закінчується закриваючим тегом;
* Атрибут – пара ім’я-значення.

Вузол-документ формується явно (doc або document) або неявно. Кожен запит XPath або XQuery явно або неявно включає в себе вузол-документ.

XPath – мова визначення шляху в XML-документі. Результатом виразу XPath є послідовність сутностей.

Найпростіші вирази починаються зі /, наприклад, /root/element/subelement, результатом якої буде послідовність вузлів subelement.

Для того, щоб вказати атрибут, перед його ім’ям потрібно написати @. Для того, щоб вказати будь-який вузол, потрібно записати \*. Для того, щоб вказати будь-який рівень ієрархії, потрібно написати //. Можна записувати умову у квадратних дужках. Відповідно у результат підуть тільки ті елементи, які відповідають цим умовам. Значення елемента записується за допомогою крапки.

На кожному кроці виконання XPath іде по якомусь напрямку. За замовчуванням, це напрямок child∷. При записі @ це буде attribute∷. parent∷ – це батьківський елемент. descendent‑or‑self∷ - це нащадок будь-якого рівня або сам елемент (скорочення - //). ancestor‑or‑self∷ - батьківський елемент будь-якого рівня або сам елемент. self∷ – сам елемент (скорочення – крапка).

XQuery розширює XPath до аналогічної потужності SQL. XQuery є мовою виразів, тобто будь-який XQuery-вираз може бути аргументом іншого XQuery-виразу.

XQuery деколи продукує послідовність послідовностей, які вона спрощує в звичайну послідовність.

FLWR-вирази – базова конструкція XQuery:

* for;
* let;
* where;
* return.

Кожен for створює цикл. let створює локально визначену константу. На кожній ітерації вкладених циклів перевіряється умова where. Якщо where повертає true, тоді return доповнює результат, не перериваючи виконання. Локальні змінні позначаються через $. order by expression впорядковує результат за виразом expression.

Щоб вимагати від елементів, які порівнюються, бути послідовністю з одного елементу, потрібно використати eq (equal), ne (not equal), lt (less than), le (less or equal), gt (greater than), ge (greater or equal). Якщо елемент порівнюється з простим значенням, тоді замість елемента вставляється його значення. Якщо порівнюються на рівність елементи між собою, то це означає порівняти, чи це один і той самий елемент. Якщо треба порівнювати значення, а не самі елементи, потрібно використати функцію data, яка повертає вміст елемента.

Функція distinct‑values бере значення сутностей, бере з них кожне значення лише один раз, і повертає їх.

Діюче булівське значення виразу є:

1. Значення виразу, якщо вираз є типу boolean;
2. false, якщо значенням виразу є 0, порожній рядок або порожня послідовність;
3. true в інших випадках.

Оператори and, or, not працюють зі значеннями типу boolean.

Оператор if: if (expr1) then expr2 else expr3. Якщо expr1 має значення true, то виконується expr2, якщо false – то expr3. Частина else обов’язкова.

XSLT (Extensible Stylesheet Language Transformations) – мова, яка може перетворювати XML-документи в інші XML-документи або в інші формати.

# OLAP. Склад даних, куби даних

Традиційні бази даних оптимізовані під виконання багатьох маленьких запитів. Коли потрібно робити запит, який підсумовує (агрегує) багато різних даних, така реляційна структура виконує їх повільно.

Процедура копіювання всіх даних з різних місць в одне місце забезпечує можливість аналізу даних за одним розташуванням називається складом даних.

OLTP (Online Transactional Processing) передбачає виконання нескладних, невеликих змін у базу даних, які виконуються у транзакціях. OLAP (Online Analytical Processing) – коли потрібно опрацьовувати небагато запитів, але об’єм даних в цих запитах є великий. Переважно великі компанії мають OLTP-системи на місцях та OLAP-системи в центрі.

У зірковій схемі складу даних є два типи таблиць:

* Таблиця фактів – дуже велике зібрання фактів.
* Таблиця розмірностей – маленькі, переважно статичні таблиці, які певним чином характеризують факти.

Склад даних будується на базі ROLAP (Relational OLAP – склад даних) або на базі MOLAP (Multidimensional OLAP – куб даних).

Аналіз даних (data mining) – спочатку об’єднання даних шляхом агрегації (roll up), а потім розбирання даних на складові частини (drill down).

# Нормальні форми

Перша нормальна форма:

1. Немає сортування між рядками.
2. Відсутнє сортування колонок.
3. Відсутні однакові рядки.
4. На перетині рядка та стовпця є чітко одне значення з предметної області.
5. Відсутня прихована інформація.

Друга нормальна форма:

1. Таблиця є в першій нормальній формі.
2. Кожен неключовий атрибут повністю залежить від ключа.

Третя нормальна форма:

1. Таблиця є в другій нормальній формі.
2. Кожен неключовий атрибут нетранзитивно залежить від ключа.

# Систематичний підхід до розробки баз даних