Міністерство освіти і науки України

Відокремлений структурний підрозділ
«Ковельський промислово-економічний фаховий коледж
Луцького національного технічного університету»

МЕТОДИЧНИЙ ПОСІБНИК

з дисципліни

«*Організація баз даних та знань*»

для здобувачів освітньо-кваліфікаційного рівня фаховий молодший бакалавр спеціальності 122 «Комп'ютерні науки» денної форми навчання

Ковель 2023

УДК 004.01

Затверджено до видання методичною радою ВСП «КПЕФК ЛНТУ»
протокол №_____ від « ___» _____ 2023 року
Голова методичної ради ВСП «КПЕФК ЛНТУ» _____ Ігор ІЛЮШИК

Розглянуто і схвалено на засіданні циклової комісії 122 «Компютерні науки», протокол № 5 від « 25» січня 2023 року.
Голова циклової методичної комісії _____ Олександр ПРИСАДА

Електронна копія друкованого видання передана до книжкового фонду бібліотеки ВСП «КПЕФК ЛНТУ»
Завідувачка бібліотеки_____Неля ТЕЛЮЧИК

Укладач:_____ Олександр ПРИСАДА, викладач ВСП «КПЕФК ЛНТУ»

Рецензент _____Степан БАБАРИКА, к.т.н., викладач ВСП «КПЕФК ЛНТУ»

Відповідальний за випуск: _____Леся ПРОКОПЧУК, методист ВСП «КПЕФК ЛНТУ»

Методичний посібник з дисципліни ”Організація баз даних і баз знань”, для здобувачів освітньо-кваліфікаційного рівня фаховий молодший бакалавр спеціальності 122 «Комп’ютерні науки» денної форми навчання /уклад. О.В. Присада.. – Ковель: КПЕК Луцького НТУ, 2023. – 103 с.

Відповідно до нормативної програми дисципліни «Організація баз даних та знань» приведено навчальний матеріал для вивчення проектування та створення баз даних з використанням структурованої мови запитів SQL.

Методичний посібник з дисципліни ”Організація баз даних і баз знань” призначений для здобувачів передвищої освіти вищих навчальних закладів. Можуть бути широко використані під час самостійного освоєння дисципліни «Організація баз даних та знань» як для початківців, так і для спеціалістів.

ЗМІСТ

РОЗДІЛ 1. СИСТЕМИ БАЗ ДАНИХ. МОДЕЛІ ДАНИХ.....5
1.1 Поняття баз даних.....5
1.1.1 Сучасні інформаційні технології.....5
1.1.2 Поняття баз даних.....6
1.2 Моделі даних.....7
1.2.1 Характеристика моделей даних.....7
1.2.2 Реляційна модель даних.....9
1.3 Система керування базами даних.....11
1.3.1 Архітектура СКБД.....14
1.3.2 Функції СКБД.....15
1.3.3 Архітектура багатокористувацьких СКБД.....16
РОЗДІЛ 2. АНАЛІЗ І ПРОЕКТУВАННЯ БАЗ ДАНИХ.....20
2.1 Планування бази даних.....20
2.1.1 Планування розробки бази даних.....22
2.1.2 Модель "сутність-зв'язок".....23
2.2 Проектування баз даних.....30
2.2.1 Концептуальне проектування.....33
2.2.2 Логічне проектування.....34
2.2.3 Нормалізація даних.....35
2.2.4 Фізичне проектування.....37
РОЗДІЛ 3. МОВА СТРУКТУРОВАНИХ ЗАПИТІВ TRANSACT SQL.....41
3.1 Основи мови структурованих запитів SQL.....43
3.1.1 Архітектура баз даних СКБД SQL Server.....45
3.1.2 Елементи мови Transact-SQL.....48
3.2 Мова визначення даних.....51
3.2.1 Створення бази даних.....52
3.2.2 Створення таблиці.....53
3.2.3 Створення первинних та вторинних ключів.....54
3.2.4 Зміна структури таблиці.....55
3.2,5 Видалення таблиці 56
3.3 Маніпулювання даними.....57
3.3.1 Робота з даними.....57
3.3.2 Вибірка даних.....59
3.3.3 Вибірка даних з різних таблиць.....60
3.3.4 Підлеглі запити.....65
3.3.5 Розділ GROUP BY.....70
3.3.6 Агрегатні функції.....76
3.3.7 Керуючі конструкції мови.....80
3.3.8 Створення індексів і представлень.....82
3.3.9. Розробка процедур і функцій.....84
3.3.10 Розробка тригерів 87
РОЗДІЛ 4. ЗАХИСТ БАЗ ДАНИХ.....90
4.1 Адміністрування системи безпеки.....90
4.1.1 Основи забезпечення безпеки.....92
4.1.2 Компоненти структури безпеки.....94
4.2 Захист даних.....95
4.2.1 Шифрування даних.....96
4.2.2 Права доступу.....98
4.3 Приклад.....99
РОЗДІЛ 5. БАЗИ ДАНИХ В ІНТЕРНЕТІ..... 102

„цивілизація – це насамперед інформація, інформація робить народи людством."
С.Залигін
„жити плідно, досягати життєвих цілей – це означає жити, володіючи інформацією"
Н.Вінер

РОЗДІЛ 1. СИСТЕМИ БАЗ ДАНИХ. МОДЕЛІ ДАНИХ

1) Поняття баз даних
2) Моделі даних
3) Системи керування базами даних

1.1 Поняття баз даних

1) Сучасні інформаційні технології
2) Поняття баз даних

1.1.1 Сучасні інформаційні технології

Інформаційна технологія — це комплекс взаємозалежних, наукових, технологічних, інженерних дисциплін, що вивчають методи ефективної організації праці людей, зайнятих обробкою і збереженням інформації; обчислювальну техніку і методи організації і взаємодії з людьми і виробничим устаткуванням, їхні практичні додатки, а також зв'язані з усім цим соціальні, економічні і культурні проблеми.

Самі інформаційні технології вимагають складної підготовки, великих первісних витрат і наукомісткої техніки. Їхнє введення повинне починатися зі створення математичного забезпечення, формування інформаційних потоків у системах підготовки фахівців.

З появою персонального комп'ютера почався новий етап розвитку інформаційної технології. Основною метою стає задоволення персональних інформаційних потреб людини як для професійної сфери, так і для побутової.

Вимоги до інформаційної технології

Інформаційна технологія, повинна відповідати наступним вимогам:

- забезпечувати високий ступінь розчленовування всього процесу обробки інформації на етапи (фази), операції, дії;
- включати весь набір елементів, необхідних для досягнення поставленої мети;
- мати регулярний характер. Етапи, дії, операції технологічного процесу можуть бути стандартизовані й уніфіковані, що дозволить більш ефективно здійснювати цілеспрямоване керування інформаційними процесами.

Технічними засобами виробництва інформації є апаратне, програмне і математичне забезпечення цього процесу. З їхньою допомогою виробляється переробка первинної інформації в інформацію нової якості. Виділимо окремо з цих засобів програмні продукти і назвемо їх інструментарієм.

Інструментарій інформаційної технології — один або кілька взаємозалежних програмних продуктів для визначеного типу комп'ютера, технологія роботи в якому дозволяє досягти поставлену користувачем мета.

Як інструментарій можна використовувати наступні розповсюджені види програмних продуктів для персонального комп'ютера: текстовий процесор (редактор), настільні видавничі системи, електронні таблиці, системи керування базами даних,

електронні записні книжки, електронні календарі, інформаційні системи функціонального призначення (фінансові, бухгалтерські, для маркетингу й ін.). експертні системи і т.д.

Розвиток способів обробки даних

Складність сучасної обробки даних є результатом розвитку протягом декількох десятиліть способів обробки даних і керування інформацією.

Обробка даних розвивалася від примітивних методів 50-того років до складних інтегрованих систем сьогодення.

Перші системи обробки даних виконували лише канцелярську роботу, зменшуючи кількість документів. Оскільки ці системи виконували звичайні функції роботи з документами, вони були названі *системами обробки даних.* Розроблювачі цих систем втілювали в програми ті операції, що раніш виконувалися вручну. Дані зберігалися на магнітних носіях у файлах. Файл уміщав ту інформацію, що раніш могла лежати в одній папці. У той час у якості магнітних носіїв виступали магнітні стрічки, що допускали лише послідовний доступ. Обробка таких файлів вимагала багато часу. Використання прямого доступу в 60-і роки до файлу скоротило час обробки даних, але використання таких файлів вирішило проблему лише частково.

Системи обробки даних виконували конкретну роботу, тобто така система давала конкретний кінцевий результат: нарахована заробітна плата, дані про виконану роботу, витрату на закупівлі, і т.п.

А якщо доповнити систему таким чином, щоб вона могла відповідати на деяке питання тоді, коли це необхідно? Наприклад, який поточний баланс компанії?

Таке питання є питаннм до інформаційної системи. Тобто, наприкінці 60, на початку 70 років відбулася зміна: перехід від обробки даних до обробки інформації. Це привело до створення інформаційно-керуючих систем. Такі системи використовують дані, що уже є в комп'ютері, відповідаючи на широке коло питань.

Необхідно відрізняти *дані* й *інформацію*.

У загальному розумінні:

Дані — це опис явищ і процесів на яких-небудь носіях (книга, диск,…)...

Інформація — це миттєва форма існування даних (етап їхньої інтерпретації) після чого породжуються нові дані.

В інформаційній системі:

Дані — це деякі факти. Ці факти розміщаються у файлах.

Інформація — це оброблені дані. Інформація може бути отримана в результаті обробки великої кількості фактів. Таким чином, інформація – це ресурс, що володіє визначеною цінністю і вимагає упорядкування і керування.

1.1.2 Поняття баз даних

Технологія баз даних

Технологія організації даних, що дозволяє упорядковувати дані і керувати ними є ***технологією баз даних.***

У широкому змісті слова база даних — це сукупність відомостей про конкретні об'єкти реального світу в якій-небудь предметній області. Під ***предметною областю*** прийнято розуміти частину реального світу, що підлягає вивченню для організації керування і автоматизації. Прикладом може служити підприємство, вуз і т.д. Предметна область розглядається як сукупність реальних об'єктів, що володіють визначеним

набором властивостей і взаємозв’яних один з одним. Значення властивостей і зв'язки можуть змінюватися в часі. Тому з кожним моментом часу зіставляється деякий *стан предметної області.* Стани предметної області мають сукупність властивостей, що характеризують *семантику предметної області.* Ці властивості можуть бути задані за допомогою *обмежень цілісності,* що визначають припустимі стани предметної області. Об'єднання деякого представлення структури предметної області і відповідних їй обмежень цілісності являють собою і*нфологічну модель* предметної області.

Один із принципів технології баз даних полягає в тім , що на основі інфологічної моделі предметної області повинна бути побудована інша модель, що складається вже із сукупності зв'язаних даних яка і являє собою базу даних.

Створюючи базу даних, користувач прагне упорядкувати інформацію з різних ознак і швидко витягати потрібні зведення з довільним сполученням ознак. Зробити це можна, тільки якщо дані структуровані.

Основна задача бази даних — зберігати і при необхідності представляти на першу вимогу користувачів усі необхідні дані в одному місці, крім їхнього повторення і надмірності.

База даних - це спільно використовуваний набір логічно зв'язаних даних і описів цих даних, призначений для задоволення інформаційних потреб предметної області і які зберігаються з регульованою надмірністю.

База даних — це єдине, велике сховище даних, що однократно визначається, а потім використовується одночасно багатьма користувачами— представниками різних підрозділів. Замість розрізнених файлів з надлишковими даними тут усі дані зібрані разом з мінімальною часткою надмірності.

Типи баз даних

У залежності від розташування бази даних розрізняють *централізовані* і *розподілені* бази даних.

Розподілена база даних складається з декількох частин, збережених у різних комп'ютерах обчислювальної мережі. Цей спосіб обробки має на увазі наявність декількох серверів, на яких може зберігатися пересічна або навіть дублюється інформація.

Централізована база даних зберігається в пам'яті однієї обчислювальної системи, тобто база даних розташовується на одному комп'ютері. Якщо для цього комп'ютера встановлена підтримка мережі, то безліч користувачів із клієнтських комп'ютерів можуть одночасно звертатися до інформації, що зберігається в центральній базі даних, У локальних мережах найчастіше використовується саме такий спосіб обробки даних. Системи централізованих баз даних можуть істотно розрізнятися в залежності від їхньої архітектури.

1.2 Моделі даних

1) Характеристика моделей даних
2) Реляційна модель даних

1.2.1 Характеристика моделей даних

Для підтримки даних на різних рівнях використовуються моделі даних.

Модель даних - це інтегрований набір понять для опису й обробки даних, зв'язків між ними й обмежень, що накладаються на дані в деякій предметній області.

Модель є представленням об'єктів і подій "реального світу", а також існуючих між

ними зв'язків. Це деяка абстракція, у якій акцент робиться на найважливіших і невід'ємних аспектах, а всі другорядні властивості ігноруються. Модель повинна відбивати основні концепції, представлені в такому виді, що дозволить проектувальникам і користувачам бази даних обмінюватися думками про ролі тих або інших даних у предметній області. Модель даних можна розглядати як сполучення трьох компонентів:

- Структурна частина, тобто набір правил, по яких може бути побудована база даних.
- Керуюча частина, що визначає типи припустимих операцій з даними (відновлення, витяг даних, зміна структури бази даних).
- Набір обмежень підтримки цілісності даних, що гарантують коректність використовуваних даних.

Представлення даних у повному виді. Для цього можна визначити наступні три зв'язані моделі даних:

- зовнішня модель даних, що відображає представлення кожного існуючого в предметній області типу користувачів;
- концептуальна модель даних, що відображає логічне (або узагальнене) представлення про дані, незалежне від типу обраної СУБД;
- внутрішня модель даних, що відображає концептуальну схему певним чином, що підходить для обраної СУБД.

Типи моделей

Усі моделі даних можна підрозділити на три категорії:

- об'єктні (object-based) моделі даних,
- моделі даних на основі записів (record-based) і
- фізичні моделі даних.

Перші дві використовуються для опису даних на концептуальному і зовнішньому рівнях, а остання — на внутрішньому рівні,

Об'єктні моделі даних. При створенні об'єктних моделей даних використовуються такі поняття, як сутності, атрибути і зв'язки. Сутність — це окремий елемент діяльності організації (співробітник або клієнт, місце або річ, поняття або подія), що повинен бути представлений в базі даних. Атрибут — це властивість, що описує деякий аспект об'єкта і значення якого варто зафіксувати, а зв'язок є асоціативним відношенням між сутностями. Найбільш загальними типами об'єктних моделей є:

- Модель типу "сутність-зв'язок", або ER-модель (Entity-Relationship model).
- Семантична модель.
- Функціональна модель.
- Объ’єктно-орієнтована модель.

В даний час ER-модель стала одним з основних методів концептуального проектування баз даних. Объектно-орієнтована модель розширює визначення сутності, включаючи в нього не тільки атрибути, що описують стан об'єкта, але і дії, що з ним зв'язані, тобто його поводження.

Моделі даних на основі записів. У моделі на основі записів база даних складається з безлічі записів фіксованого формату, що можуть мати різні типи. Кожен тип запису визначає фіксована кількість полів, кожне з яких має фіксовану довжину. Існують три основних типи логічних моделей даних на основі записів:

- реляційна модель даних (relational data model),

- мережна модель даних (network data model) і
- ієрархічна модель даних (hierarchical data model).

Ієрархічна і мережна моделі даних були створені майже на десять років раніш реляционной моделі даних, тому їхній зв'язок з концепціями традиційної обробки файлів більш очевидна.

В основі всіх моделей лежить деякий математичний апарат.

Реляційцнна модель даних заснована на понятті математичних відносин, що є предметом реляционной алгебри. У реляціоній моделі дані і зв'язки представлені у виді таблиць, кожна з яких має кілька стовпців з унікальними іменами.

Мережна модель даних заснована на використанні теорії графів. У мережній моделі дані представлені у виді колекцій записів, а зв'язки – у виді наборів. Записи є вузлами графа, а набори - його ребрами.

Ієрархічна модель даних є обмеженим підтипом мережної моделі. У ній дані також представлені у виді дерева: тобто дані представляються як колекції записів, а зв'язки — як набори.

Засновані на записах (логічні) моделі даних використовуються для визначення загальної структури бази даних і високорівневого опису її реалізації.

Більшість сучасних комерційних систем засновано на реляционной моделі, тоді як найперші системи баз даних створювалися на основі мережної або ієрархічної моделі.

Фізичні моделі даних описують те, як дані зберігаються в комп'ютері, представляючи інформацію про структуру записів, їхньої упорядкованості, забезпеченню цілісності й існуючих шляхів доступу. Вона ще називається внутрішньою моделлю системи і форма її представлення залежить від обраної СКБД.

1.2.2 Реляційна модель даних

Реляційна модель даних — логічна модель даних. Вперше була запропонована британським ученим співробітником компанії IBM Едгаром Франком Коддом (E. F. Codd) і опубліковані в 1970 р. Вона визначає спосіб представлення даних (структуру даних), методи захисту даних (цілісність даних), і операції, які можна виконувати з даними (маніпулювання даними). В даний час ця модель є фактичним стандартом, на який орієнтуються практично всі сучасні комерційні системи керування базами даних (СКБД).

У реляційній моделі досягається більш високий рівень абстракції даних, ніж в ієрархічній або мережевій. У згаданій статті Е. Ф. Кодда стверджується, що «реляційна модель надає засоби опису даних на основі тільки їх природної структури, тобто без потреби введення якоїсь додаткової структури для цілей машинного представлення». Іншими словами, подання даних не залежить від способу їх фізичної організації. Це забезпечується за рахунок використання математичного поняття відношення (сама назва «реляційна» походить від англійського relation — «відношення»).

Принципи реляційної моделі

· – Всі дані на концептуальному рівні представляються у вигляді впорядкованої організації, визначеної в вигляді рядків і стовпців і званої ставленням (relation). Більш поширений синонім слова "відношення" - таблиця (або "набір записів", або набір результатів - result set. Саме від цього і походить термін "реляційні бази даних", а зовсім не від відносин між таблицями;

· – Всі значення є скалярами. Це означає, що для будь-якого рядка і стовпця будь-якого відношення існує одне і тільки одне значення;

· – Всі операції виконуються над цілим ставленням і результатом цих операцій також є ціле ставлення. Цей принцип називається замиканням. Тому результати однієї операції (наприклад, запиту), можна використовувати в якості вихідних даних для виконання іншої операції (підзапиту).

Термінологія:

· ***Відношення*** (relation) - це вся структура цілком, набір записів (в звичайному розумінні - таблиця).

· ***Кортеж*** - це кожен рядок, що містить дані. Більш поширений, але менш формальний термін - запис.

· ***Потужність*** - число кортежів щодо (простіше кажучи, число записів);

· ***Домен*** (***Атрибут)*** - це стовпець у відношенні;

· ***Розмірність*** - це число атрибутів щодо (в даному випадку - 3);

· Кожне відношення можна розділити на дві частини - заголовок і тіло. Простою мовою заголовок відносини - це список стовпців, а тіло - це самі записи (кортежі).

· Тіло відносини складається з невпорядкованого набору кортежів (його кількість може бути будь-яким - від 0 до нескінченно великого).

Сукупність усіх відносин визначає базу даних. Кожне відношення зберігає свою логічну частину інформації. Щоб отримати певні відомості може знадобитися зіставлення інформації з різних відносин. Кодд описав вісім основних операцій реляційної алгебри, що дозволяють маніпулювати з кортежами:

об'єднання;
перетин;
віднімання;
Декартово твір;
вибірка;
проекція;
з'єднання;
Розподіл.

Чудова властивість реляційної алгебри - це її замкнутість, тобто операції над відносинами задаються таким чином, щоб результат сам був ставленням. Тобто, маючи кілька таблиць і виробляючи відповідні операції над ними, ми отримаємо результатом теж таблицю.

Дванадцять правил Кодда, яким повинна відповідати реляційна СКБД

Кодд у 1985 році сформулював 12 правил, яким повинна задовольняти будь-як база даних, що претендує на тип реляційної. З того часу дванадцять правил Кодда вважаються визначенням реляційної СУБД.

1. Правило інформації.

Вся інформація в базі даних повинна бути надана винятково на логічному рівні і тільки одним способом - у виді значень, що містяться в таблицях.

2. Правило гарантованого доступу.

Логічний доступ до всіх і кожного елементу даних (атомарному значенню) у реляційній базі даних повинний забезпечуватися шляхом використання комбінації імені таблиці, первинного ключа та імені стовпця.

3. Правило підтримки недійсних значень.

У реляційній базі даних повинна бути реалізована підтримка недійсних значень, що відрізняються від рядка символів нульової довжини, рядка символів пробілів чи нуля будь-якого іншого числа і використовуватися для представлення відсутніх даних незалежно від типу цих даних.

4. Правило динамічного каталогу, заснованого на реляційній моделі.

Опис бази даних на логічному рівні необхідно представити в тому ж виді, що й основні дані, щоб користувачі, які володіють відповідними правами, могли працювати з ним за допомогою тієї ж реляційної мови, яку вони застосовують для роботи з основними даними.

5.Правило вичерпної підмови даних.

Реляційна система може підтримувати різні мови і режими взаємодії з користувачем (наприклад, режим питань і відповідей). Однак повинна існувати принаймні одна мова, оператори якої підтримують наступні елементи: визначення даних; визначення представлень; обробку даних (інтерактивну і програмну); умови цілісності; ідентифікацію прав доступу; границі транзакцій (початок, завершення і скасування).

6. Правило відновлення представлень.

Усі представлення, які теоретично можна обновити, повинні бути доступні для відновлення.

7. Правило додавання, відновлення і вилучення.

Можливість працювати з відношенням як з одним операндом повинна існувати не тільки при читанні даних, але і при додаванні, відновленні і вилученні даних.

8. Правило незалежності фізи данихчних.

Прикладні програми й утиліти для роботи з даними повинні на логічному рівні залишатися недоторканими при будь-яких змінах методів збереження даних чи методів доступу до них.

9. Правило незалежності логічних даних.

Прикладні програми й утиліти для роботи з даними повинні на логічному рівні залишатися недоторканими при внесенні в базові таблиці будь-яких змін, які теоретично дозволяють зберегти недоторканими дані, що містяться в цих таблицях.

10. Правило незалежності умов цілісності.

Повинна існувати можливість визначати умови цілісності, специфічні для конкретної реляційної бази даних, на підмові реляційної бази даних і зберігати їх у каталозі, а не в прикладній програмі.

11. Правило незалежності поширення.

Реляційна СКБД не повинна залежати від потреб конкретного клієнта.

12. Правило одиничності.

Складові реляційної моделі

До складу реляційної моделі даних зазвичай включають теорію нормалізації. Крістофер Дейт визначив три складові частини реляційної моделі даних:

- структурна
- маніпуляційна
- цілісна

Структурна частина моделі визначає, що єдиною структурою даних є нормалізоване n-арне відношення. Відношення зручно представляти у формі таблиць, де кожен рядок є кортеж, а кожен стовпець — атрибут, визначений на деякому домені. Даний неформальний підхід до поняття відношення дає більш звичну для розробників і користувачів форму представлення, де реляційна база даних являє собою кінцевий набір таблиць.

Маніпуляційна частина моделі визначає два фундаментальних механізми маніпулювання даними — реляційну алгебру і реляційне числення. Основною функцією маніпуляційної частини реляційної моделі є забезпечення заходів реляційності будь-якої конкретної мови реляційних БД: мова називається реляційною, якщо вона має не меншу виразність і потужність, ніж реляційна алгебра або реляційне числення.

Цілісна частина моделі визначає вимоги цілісності сутностей і цілісності посилань. Перша вимога полягає в тому, що будь-який кортеж будь-якого відношення відмінний від будь-якого іншого кортежу цього відношення, тобто іншими словами, будь-яке відношення має володіти первинним ключем. Вимога цілісності щодо посилань, або вимога зовнішнього ключа полягає в тому, що для кожного значення зовнішнього ключа, що з'являється у відношенні, на яке веде посилання, повинен знайтися кортеж з таким же значенням первинного ключа, або значення зовнішнього ключа повинно бути невизначеним (тобто ні на що не вказувати).

Переваги реляційної моделі:

- простота і доступність для розуміння користувачем. Єдиною використовуваною інформаційною конструкцією є «таблиця»;
- суворі правила проектування, які базуються на математичному апараті;
- повна незалежність даних. Зміни в прикладній програмі при зміні реляційної БД мінімальні;
- для організації запитів і написання прикладного ПЗ немає необхідності знати конкретну організацію БД у зовнішній пам'яті.

Недоліки реляційної моделі:

- далеко не завжди предметна область може бути представлена у вигляді «таблиць»;
- в результаті логічного проектування з'являється множина «таблиць». Це призводить до труднощів розуміння структури даних;
- БД займає відносно багато зовнішньої пам'яті;
- відносно низька швидкість доступу до даних.

1.3 Система керування базами даних — СУБД

1) Історія розвитку баз даних
2) Компоненти середовища СУБД
3) Архітектура СКБД
4) Функції СКБД
5) Архітектура багатокористувацьких СКБД

СКБД – це програмне забезпечення, за допомогою якого користувачі можуть визначати, створювати і підтримувати базу даних, а також здійснювати до неї контрольований доступ.

СКБД — це програмне забезпечення, призначене для створення бази даних для безлічі додатків, підтримки її в активному стані і забезпечення ефективного доступу користувачів до даних, що утримуються в ній.

Історія розвитку баз даних

Перша система, що використовувала базу даних, з'явилась в середині 60-х років і була основана на ієрархічній моделі.

В кінці 60, на початку 70 з'явились мережеві СУБД.

І в ієрархічній і в мереженій системах БД для зв'язку файлів використовувались фізичні вказники. Ці вказники дозволяли вилучати дані, зв'язані відповідними відношеннями. Але недоліком таких систем було те, що ці відношення повинні бути визначені до запуску системи. Вилучити дані на основі інших відношень було неможливо.

В 1970 році Е.Ф.Кодд опублікував статтю в якій висунув ідею, що дані потрібно зв'язувати в відповідності з їх внутрішніми логічними взаємовідношеннями, а не фізичними вказниками. Таким чином користувачі зможуть комбінувати дані з різних джерел, якщо логічна інформація, що необхідна для такого комбінування, присутня в висхідних даних. Така модель даних дістала назву *реляційної* моделі. Для роботи з даними використовується реляційна алгебра і реляційне числення. Це забезпечує роботу з даними на основі логічних характеристик. В реляційних системах однією командою може оброблятись цілий файл, а в ієрархічних і мережених – тільки один запис.

Логічний підхід до даних зробив можливим створення мов запитів, що є більш доступними для користувачів, що не є спеціалістами.

В другій половині 70 років з'явились реляційні системи, які підтримували мову структурованих запитів (SQL), мову запитів (Quel), запити по зразку (QBE).

Сьогодні реляційні системи розглядаються як стандарт систем роботи з даними.

Реляційні бази даних продовжують вдосконалюватись, їх внутрішня природа змінюється. Це є використання об'єктно-орієнтованих баз даних (використання концептуальної моделі даних) і використання технології клієнт-серевер.

Компоненти середовища СУБД

У середовищі СУБД можна виділити наступних п'ять основних компонентів: апаратне і програмне забезпечення, дані, процедури і користувачів.

*Апаратне забезпечення***.** Це може бути персональний комп'ютер, мэйнфрейм або мережа з багатьох комп'ютерів. Використовуване апаратне забезпечення залежить від вимог даної організації і типу СУБД.

*Програмне забезпечення***.** Цей компонент охоплює програмне забезпечення самої СУБД і прикладних програм, операційну систему, мережне програмне забезпечення, якщо СУБД використовується в мережі. Звичайно додатки створюються на мовах третього покоління, таких як З, С++, Java, Visual Basic, COBOL, Fortran, Ada або Pascal, або на мовах четвертого покоління, таких як SQL. СУБД може мати свої власні інструменти четвертого покоління, призначені для швидкої розробки додатків з використанням убудованих непроцедурних мов запитів, генераторів звітів, форм, графічних зображень.

*Дані***.** Є самим важливим компонентом середовища СУБД (з погляду кінцевих користувачів). База даних містять як робітники дані, так і метадані, тобто "дані про дані".

Процедури. До процедур відносяться інструкції і правила, що повинні враховуватися при проектуванні і використанні бази даних. Користувачам і обслуговуючому персоналові бази даних необхідно надати документацію, що містить докладний опис процедур використання і супроводи даної системи:

- Реєстрація в СУБД.
- Використання окремого інструмента СУБД або додатка.
- Запуск і останов СУБД.
- Створення резервних копій СУБД.
- Обробка збоїв апаратного і програмного забезпечення, а також відновлення бази даних після усунення несправності.
- Зміна структури таблиці, реорганізація бази даних, розміщеної на декількох

дисках, способи поліпшення продуктивності і методи архівування даних на вторинних пристроях збереження.

Користувачі. Серед них можна виділити чотири різні групи: адміністратори даних і адміністратори баз даних, прикладні програмісти і кінцеві користувачі баз даних.

– *Адміністратори даних і адміністратори баз даних*

Адміністратор даних, або ПЕКЛО (Data Administrator — DA), відповідає за керування даними, включаючи планування бази даних, розробку і супровід стандартів, прикладних алгоритмів і ділових процедур, а також за концептуальне і логічне проектування бази даних.

Адміністратор бази даних, або АБД (Database Administrator — DBA), відповідає за фізичну реалізацію бази даних, включаючи фізичне проектування і втілення проекту, за забезпечення безпеки і цілісності даних, за супровід операційної системи, а також за забезпечення максимальної продуктивності додатків і користувачів. У порівнянні з ПЕКЛО обов'язку АБД носять більш технічний характер, і для нього необхідне знання конкретної СУБД і системного оточення.

– *Розроблювачі баз даних*

У проектуванні великих баз даних беруть участь розроблювачі двох різних типів: розроблювачі логічної бази даних і розроблювачі фізичної бази даних. Розроблювач логічної бази даних займається ідентифікацією даних (тобто сутностей і їхніх атрибутів), зв'язків між даними, і встановлює обмеження, що накладаються на збережені дані. Робота розроблювача логічної бази даних поділяється на два етапи: Концептуальне проектування бази даних і Логічне проектування бази даних.

Розроблювач фізичної бази даних одержує готову логічну модель даних і займається її фізичною реалізацією, у тому числі:

- перетворенням логічної моделі даних у набір таблиць і обмежень цілісності даних;
- вибором конкретних структур збереження і методів доступу до даних, що забезпечують необхідний рівень продуктивності при роботі з базою даних;
- проектуванням будь-яких необхідних мір захисту даних.

– *Прикладні програмісти*

Після створення бази даних розробляють додатка, що надають користувачам необхідні їм функціональні можливості. Прикладні програмісти працюють на основі специфікацій, створених системними аналітиками.

– *Користувачі*

Користувачі є клієнтами бази даних — вона проектується, створюється і підтримується для того, щоб обслуговувати їхні інформаційні потреби. Користувачів можна класифікувати по способі використання ними системи.

– Рядові користувачі. Ці користувачі звичайно навіть і не підозрюють про наявність СУБД. Вони звертаються до бази даних за допомогою спеціальних додатків, що дозволяють у максимальному ступені спростити виконувані ними операції. Такі користувачі ініціюють виконання операцій бази даних, уводячи найпростіші команди або вибираючи команди меню

– Досвідчені користувачі. Це користувачі, що знайомі зі структурою бази даних і можливостями СУБД. Для виконання необхідних операцій вони можуть використовувати таку мову запитів високого рівня, як SQL. А деякі досвідчені користувачі можуть навіть створювати власні прикладні програми.

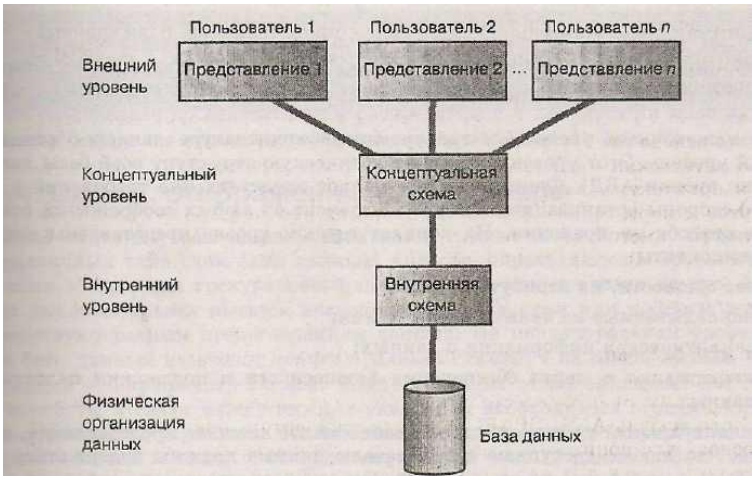
1.3.1 Архітектура СКБД

Кожна СУБД має свою архітектуру. Архітектура – це сукупність її основних функціональних компонентів і засобів забезпечення їхньої взаємодії один з одним, з користувачами і системним персоналом.

Архітектурні рівні СКБД

Перша спроба створення стандартної термінології і загальної архітектури СУБД була почата в 1971. Вона описувала використання дворівневого підходу, побудованого на основі використання системного представлення бази даних, тобто схеми (schema), і користувальницьких представлень, тобто подсхем (subschema). Пізніше була визнана необхідність використання трирівневої архітектури. Вона охоплює зовнішній, концептуалъный і внутрішній рівні, як показано на малюнку.

Трехуровневая архітектура ANSI-SPARC



Ціль трирівневої архітектури полягає у відділенні користувальницького представлення бази даних від її фізичного представлення. З кожним архітектурним рівнем зв'язана деяка модель даних.

Зовнішній рівень призначений для опису представлення бази даних з погляду користувачів. Цей рівень описує ту частину бази даних, що відноситься до кожного користувача. Зовнішній рівень складається з декількох різних зовнішніх представлень бази даних. Зовнішнє представлення містить тільки ті сутності, атрибути і зв'язки "реального світу", що цікаві користувачеві. Представлення бази даних на зовнішньому рівні називається *зовнішніми базами даних*, а опису таких представлень називаються *зовнішніми схемами*.

Концептуальний рівень призначений для підтримки єдиного погляду на базу даних, загального для всіх додатків. Цей рівень описує те, які дані зберігаються в базі даних, а також зв'язки, що існують між ними. Він містить логічну структуру всієї бази даних. Представлення бази даних на концептуальному рівні називається *концептуальною базою даних*, а опис такого представлення називається *концептуальною схемою*.

Внутрішній рівень служить для підтримки представлення БД у середовищі збереження. Він містить опис структур даних і організації окремих файлів, використовуваних для збереження даних на запам'ятовуючих пристроях. Представлення бази даних на внутрішньому рівні називається *внутрішньою або збереженою базою даних*, а опис такого представлення називається *внутрішньою схемою*.

СУБД відповідає за установлення відповідності між трьома типами схем, а також

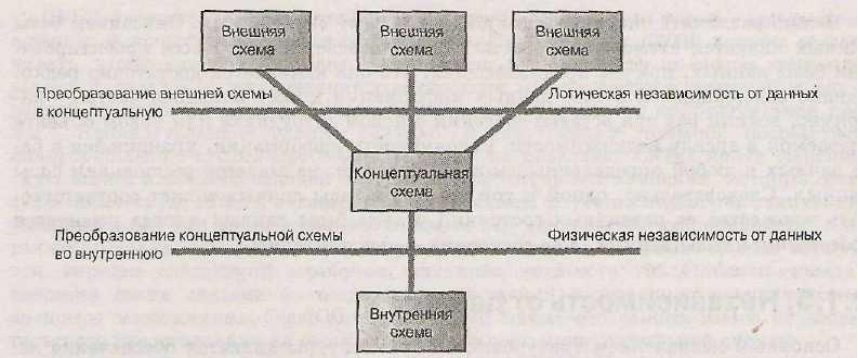
за перевірку їхньої несуперечності.

Сукупність інформації, що зберігається в базі даних у будь-який визначений момент часу, називається *станом бази даних*.

Забезпечення незалежності даних

Основним призначенням трирівневої архітектури є забезпечення незалежності від даних, що означає, що зміни на нижніх рівнях не впливають на верхні рівні. Розрізняють два типи незалежності від даних: логічну і фізичну.

Реалізація незалежності від даних у трьоехурівневій архітектурі ANSI-SPARC



Мови СУБД

Внутрішня мова СУБД для роботи з даними складається з двох частин: мови визначення даних (Data Definition Language — DDL) і мови маніпулювання даними (Data Manipulation Language— DML). Мова DDL використовується для визначення схеми бази даних, а мова DML — для читання і відновлення даних, збережених у базі. Ці мови називаються підмовами даних, оскільки в них відсутні конструкції для виконання всіх обчислювальних операцій, звичайно використовуваних у мовах програмування високого рівня, таких як умовні оператори або оператори циклу. Крім того, на сучасний момент часу стандартом роботи з БД є мова *структурованих запитів* (Structured Query Language, SQL), що забезпечує користувача простим і ефективним інструментом доступу до даних і підтримувана всіма сучасними СУБД.

1.3.2 Функції СКБД

- *Визначення структури створюваної БД.* Для створення БД розроблювач описує її логічну структуру, організацію в середовищі збереження. Для цього використовуються язикові засоби й інтерфейс системи.
- *Керування ресурсами*. Основний ресурс – структури даних. СУБД повинна надавати користувачам можливість зберігати, витягати, видаляти, додавати й обновляти дані в базі даних. Для цього використовуються мовні засоби й інтерфейс системи.
- *Забезпечення логічної і фізичної незалежності даних*.

Логічна незалежність даних. означає повну захищеність зовнішніх схем від змін, внесених у концептуальну схему. Такі зміни концептуальної схеми, як додавання або видалення нових сутностей, атрибутів або зв'язків, повинні здійснюватися без необхідності внесення змін у вже існуючі зовнішні схеми або коректування прикладних програм.

Фізична незалежність даних. Означає захищеність концептуальної схеми від змін, внесених у внутрішню схему. Такі зміни внутрішньої схеми, як використання різних файлових систем або структур збереження, різних пристроїв збереження, модифікація індексів або хешування, повинні здійснюватися без необхідності внесення зобов'язань у концептуальну або зовнішню схему.

- *Захист логічної і фізичної цілісності даних.* Цілісність бази даних означає

коректність і несуперечність збережених даних. СУБД повинна мати інструменти контролю за тим, щоб дані і їхньої зміни відповідали заданим правилам.

Логічна цілісність звичайно виражається у виді обмежень або правил збереження несуперечності даних, що не повинні порушуватися в базі. Наприклад, можна вказати межі зміни значень. Ці значення вказуються при описі структури БД. Порушення цілісності можуть бути також наслідком несвоєчасного переривання процедур запитів зміни даних, виданих іншим користувачем. Для виключення таких ситуацій використовується механізм транзакцій. *Транзакція* являє собою набір дій, виконуваних окремим користувачем або прикладною програмою з метою доступу або зміни вмісту бази даних. СУБД повинна забезпечити рівнобіжний доступ безлічі користувачів до спільно оброблюваних даних.

Фізична цілісність даних виражається в захисті даних від фізичного руйнування в результаті збоїв і відмовлень устаткування. СУБД повинна надавати засоби відновлення бази даних від якого-небудь її ушкодження або руйнування. Для цього використовуються контрольні копії даних, тобто запам'ятовування стану бази даних у контрольних крапках, ведення системного журналу й ін.

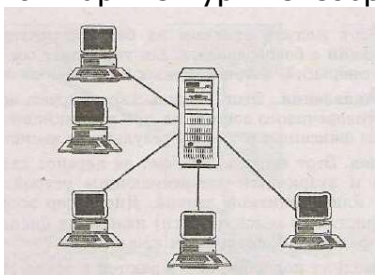
- *Доступ користувачів до даних і керування їхніми повноваженнями*. У різних груп користувачів права доступу до даних різні. Найвищі права в адміністратора БД, у системних програмістів менше, а в кінцевих користувачів ще менше. СУБД повинна мати механізм, що гарантує можливість доступу до бази даних тільки санкціонованих користувачів.

1.3.3 Архітектура багатокористувацьких СКБД

Телеобробка

Традиційною архітектурою багатокористувацьких систем раніш вважалася схема, що одержала назву телеобробки, при якій один комп'ютер з єдиним процесором був з'єднаний з декількома терміналами, як показано на малюнку.

Топологія архітектури телеобробки



При цьому вся обробка виконувалася за допомогою єдиного комп'ютера, а приєднані до нього користувальницькі термінали були типовими "неінтелектуальними" пристроями, не здатними функціонувати самостійно. З центральним процесором термінали були зв'язані за допомогою кабелів, по яких вони посилали повідомлення користувальницьким додаткам, а користувальницькі додатки зверталися до необхідних служб СУБД.

В останні роки був досягнутий істотний прогрес у розробці високопродуктивних персональних комп'ютерів і складених з них мереж. При цьому спостерігається тенденція до децентралізації (downsizing), тобто заміні дорогих мэйнфреймов більш ефективними мережами персональних комп'ютерів, що дозволяють одержати такі ж, якщо не кращі, результати. Ця тенденція привела до появи наступних двох типів архітектури СУБД: технології файлового сервера і технології "клієнт/сервер".

Файловий сервер

У середовищі файлового сервера обробка даних розподілена в мережі, що звичайно представляє собою локальну обчислювальну мережу (ЛВС). Файловий сервер містить файли, необхідні для роботи додатків і самої СУБД. Однак користувальницькі додатки і СУБД розміщені і функціонують на окремих робочих станціях, і звертаються до файлового сервера тільки в міру необхідності одержання доступу до потрібного їм файлами, як показано на малюнку.

Архітектура з використанням файлового сервера

Таким чином, файловий сервер функціонує просто як спільно використовуваний твердий диск. СУБД на кожній робочій станції посилає запити файловому серверові по всім необхідним їй даним, що зберігаються на диску файлового сервера. Такий підхід характеризується значним мережним трафіком, що може привести до зниження продуктивності всієї системи в цілому. Прикладами СУБД, призначеними безпосередньо для розробки локальних користувальницьких додатків БД, тобто додатків, що працюють на одному локальному комп'ютері або в комп'ютерній мережі, є: Microsoft Visual FoxPro, Microsoft Access, Paradox for Windows, dBase for Windows і ін.

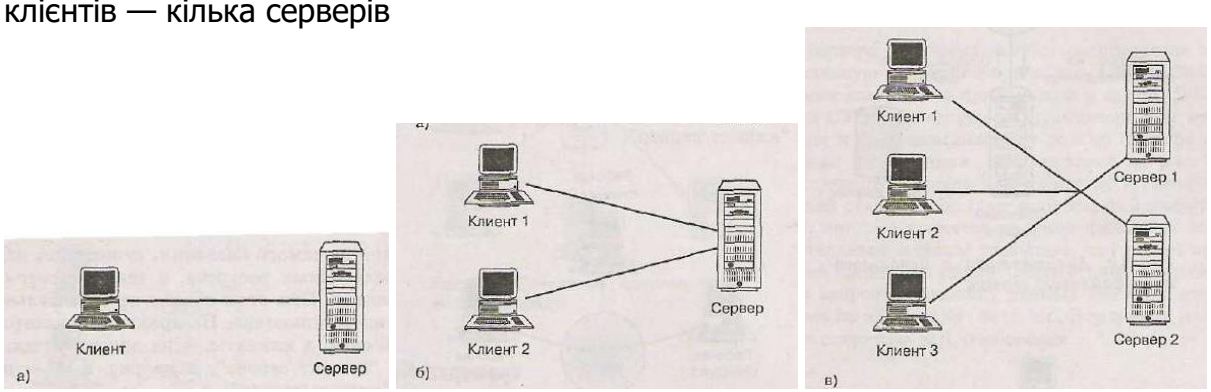
Технологія "клієнт/сервер"

Технологія "клієнт/сервер" була розроблена з метою усунення недоліків, що маються в перших двох підходах. У цій технології використовується спосіб взаємодії програмних компонентів, при якому вони утворять єдину систему. Існує якийсь *клієнтський* процес, що вимагає визначених ресурсів, а також *серверний* процес, що ці ресурси надає. При цьому зовсім не обов'язково, щоб вони знаходилися на тому самому комп'ютері. На практиці прийнято розміщати сервер на одному вузлі локальної мережі, а клієнти — на інших вузлах. На малюнку показана архітектура типу "клієнт/сервер".

Загальна схема побудови систем з архітектурою "клієнт/сервер"

На наступному малюнку показані альтернативні топології систем з архітектурою "клієнт/сервер": а) один клієнт — один сервер; б) трохи клієнтів — один сервер; в)

трохи клієнтів — кілька серверів



.

У контексті бази даних клієнт керує користувальницьким інтерфейсом і логікою додатка, діючи як складна робоча станція, на якій виконуються додатки баз даних. Клієнт приймає від користувача запит, перевіряє синтаксис і генерує запит до бази даних мовою SQL або іншій мові бази даних, що відповідає логіці додатка. Потім він передає повідомлення серверові, очікує надходження відповіді і форматирует отримані дані для представлення їх користувачеві. Сервер приймає й обробляє запити до бази даних, а потім передає отримані результати назад клієнтові. Така обробка включає перевірку повноважень клієнта, забезпечення вимог цілісності, підтримку системного каталогу, а також виконання запиту і відновлення даних. Крім цього, підтримується керування рівнобіжною роботою і відновленням.

Цей тип архітектури має наступні переваги,

- Забезпечується більш широкий доступ до існуючих баз даних.
- Підвищується загальна продуктивність системи. Оскільки клієнти і сервер знаходяться на різних комп'ютерах, їхні процесори здатні виконувати додатки паралельно. При цьому настроювання продуктивності комп'ютера із сервером спрощується, якщо на ньому виконується тільки робота з базою даних.
- Вартість апаратного забезпечення знижується. Досить могутній комп'ютер з великим пристроєм збереження потрібний тільки серверові — для збереження і керування базою даних.
- Скорочуються комунікаційні витрати. Додатки виконують частину операцій на клієнтських комп'ютерах і посилають через мережу тільки запити до бази даних, що дозволяє істотно скоротити обсяг даних, що пересилаються по мережі.
- Підвищується рівень несуперечності даних. Сервер може самостійно керувати перевіркою цілісності даних, оскільки всі обмеження визначаються і перевіряються тільки в одному місці.
- Ця архітектура досить природно відображається на архітектуру відкритих систем.

Прикладами СУБД технології клієнт-сервер є Microsoft SQL Server, Oracle, IBM DB2, Sybase і ін. Специфікою архітектури клієнт-сервер є використання спеціальної *мови структурованих запитів* (Structured Query Language, SQL), що забезпечує користувача простим і ефективним інструментом доступу до даних.

РОЗДІЛ 2. АНАЛІЗ І ПРОЕКТУВАННЯ БАЗ ДАНИХ

1) Планування бази даних
2) Проектування баз даних

База даних є фундаментальним елементом більш широкого поняття - інформаційної системи.

Інформаці́йна систе́ма (англ. Information system) — сукупність організаційних і технічних засобів для збереження, обробки та передачі інформації з метою забезпечення інформаційних потреб користувачів. Інформаці́йна сісте́ма є середовищем, складовими елементами якої є комп'ютери, комп'ютерні мережі, програмні продукти, БД, люди, різного роду технічні й програмні засоби зв'язку і т.д.

2.1 Планування бази даних

1) Планування розробки бази даних
2) Модель сутність-зв‘язок

2.1.1 Планування розробки бази даних

1) Життєвий цикл додатку баз даних
2) Планування розробки бази даних
3) Підходи до проектування БД
4) Моделювання даних

Життєвий цикл додатку баз даних

Життєвий цикл додатків баз даних нерозривно пов'язаний з життєвим циклом інформаційної системи. Етапи життєвого циклу додатку бази даних показані на рис. . Ці етапи не є строго послідовними, а передбачають в деяких випадках повернення до попередніх етапів за допомогою зворотних зв'язків (feedback loops).

Етапи:

1) Планування - це підготовчі дії, що дозволяють з максимально можливою ефективністю реалізувати етапи ЖЦ. Планування має бути нерозривно пов’язане з загальною будовою ІС. При цьому необхідно вирішити наступні питання:

   - Визначення бізнес-планів та цілей організації;
   - Оцінка показників уже існуючих ІС;
   - Оцінка можливостей використання ІТ для дослідження переваг перед конкурентами.
2) Формування вимог
3) Проектування БД
4) Проектування додатків
5) Реалізація
6) Тестування
7) супроводження

Планування розробки бази даних

Це підготовчі дії, що дозволяють з максимально можливою ефективністю реалізувати етапи життєвого циклу програми бази даних.

Планування розробки бази даних має бути нерозривно пов'язане з загальною стратегією побудови інформаційної системи організації. При виробленні такої стратегії необхідно вирішити такі основні завдання:

- визначення бізнес-планів та цілей організації з наступним виділенням її потреб в інформаційних технологіях;
- оцінка показників уже існуючих інформаційних систем з метою виявлення їх сильних і слабких сторін;
- оцінка можливостей використання інформаційних технологій для досягнення переваг перед конкурентами.

Підходи до проектування БД

Основними підходами є

- Висхідний
- Спадний

При використання висхідного підходу робота починається з самого нижнього рівня – атрибутів, які групуються у відношення. Цей підхід може використовуватись для проектування простих БД, т.я. для складних БД важко з самого початку чітко визначити всі атрибути.

Спадне проектування більш прийнятна стратегія. В цьому випадку робота розпочинається з розробки моделей даних, які містять сутності і зв`язки і надалі виконується уточнення.

Крім цих підходів для проектування баз даних можуть застосовуватися інші підходи, наприклад, підхід "від загального до конкретного" або "змішана стратегія проектування ". Підхід " від загального до конкретного " нагадує висхідний підхід, але відрізняється від нього тим, що спочатку виявляється набір основвих сутностей з подальшим розширенням кола аналізованих сутностей, зв'язків і атрибутів, які взаємодіють з первісно визначеними сутностями.

У змішаній стратегії спочатку використовуються висхідний і спадний підходи для створення різних частин моделі, після чого всі підготовлені фрагменти збираються в єдине ціле.

ER-моделювання використовує спадне проектування.

Моделювання даних

Основні цілі моделювання даних полягають у вивченні значення (семантики) даних і спрощення процедур опису вимог до даних. При створенні моделі даних необхідно отримати відповіді на певні питання про окремі сутності, зв'язки і атрибути.

Моделювання даних спрощує розуміння сенсу елементів даних, тому створення моделі необхідно для того, щоб гарантувати розуміння наступних аспектів даних:

- вимоги до даних окремих користувачів;
- характер самих даних незалежно від їх фізичного представлення;
- використання даних в межах області застосування додатка.

Моделі даних можуть використовуватися для демонстрації розуміння розробником тих вимог до даних, які існують на підприємстві. Якщо обидві сторони знайомі з системою позначень, використовуваної для створення моделі, то наявність моделі даних буде сприяти більш плідному спілкуванню користувачів і розробників. На

підприємствах все ширше застосовуються засоби стандартизації для моделювання даних шляхом вибору певного методу моделювання і використання його в усіх проектах розробки бази даних. Найпопулярніша технологія високорівневого моделювання даних, найчастіше використовувана при розробці реальних баз даних, побудована на концепції моделі "сутність-зв'язок" (Entity-Relationship model - ER-модель).

Оптимальна модель даних повинна задовольняти критеріям, перерахованим в табл.

Структурна достовірність	Відповідність способу визначення та організації інформації на даному підприємстві
Простота	Зручність вивчення моделі як професіоналами в області розробки інформаційних систем, так і звичайними користувачами
Виразність	Здатність представляти відмінності між даними, зв'язки між даними і обмеження
Відсутність надмірності	Виняток зайвої інформації, тобто будь-яка частина даних повинна бути представлена тільки один раз
Здатність до спільного використання	Відсутність приналежності до якомусь особливому додатку або технології і, отже, можливість використання моделі під багатьох додатках і технологіях
Розширюваність	Здатність розвиватися і включати нові вимоги з мінімальним впливом на роботу вже існуючих додатків
Цілістність	Узгодженість зі способом використання та управління інформацією всередині підприємства
Схематичне представлення	Можливість подання моделі за допомогою наочних схематичних позначень

2.1.2 Модель ”сутність-зв`язок”

1) Типи сутностей
2) Зв`язки
3) Атрибути

Типи сутностей

Тип сутності. *Група об'єктів з однаковими властивостями, яка розглядається в конкретній предметній області як має незалежне існування.*

Основною концепцією ER-моделі є тип сутності (entity type), який представляє групу об'єктів реального світу, що володіють однаковими властивостями. Тип сутності характеризується незалежним існуванням і може бути об'єктом з фізичним (або реальним) існуванням або об'єктом з концептуальним (або абстрактним) існуванням.

Приклад сутностей:

Сутність	**Опис**	**Знапходження екземпляра сутності**
працівник	Загадьне позначення всіх працівниеків підпритємства	Кожен працівник працює в конкретному відділі
відділ	Загальне визначення для всіх відділів підприємства	Відділ є структурною одиницею підприємства

Екземпляр сутності. *Однозначно ідентифікований об'єкт, який відноситься до сутності певного типу.*

Кожен однозначно ідентифікований об'єкт типу сутності, який відноситься до сутності певного типу, називається просто екземпляром сутності (entity occurrence).

Способи схематичного зображення типів сутностей

Кожен тип сутності зображується у вигляді прямокутника з ім'ям сутності всередині нього; в якостві імені зазвичай застосовується іменник в однині. В UML прийнято використовувати великі літери на початку кожного слова, складового ім'я сутності.

Сутності сильного і слабкого типів (Головні та залежні сутності)

Типи сутностей можуть також підрозділятися на сильні і слабкі.

Сутність сильного типу. Тип сутності, існування якого не залежить від якогось іншого типу сутності.

Тип сутності називається сильним, якщо його існування не залежить від наявності сутностей іншого типу. Характерною особливістю сильного типу є те, що кожен екземпляр сутності може бути однозначно ідентифікований за допомогою атрибуту (атрибутів) первинного ключа сутності цього типу. Наприклад, кожен співробітник компанії може бути однозначно ідентифікований за допомогою атрибуту STAFF, який є первинним ключем сутності типу Персонал.

Сутність слабкого типу. Тип сутності / існування якого залежить від якогось іншого типу сутності.

Сутність слабкого типу залежить від наявності сутності іншого типу. Приклад суті слабкого типу Перевагу (Побажання). Характерною особливістю слабкою суті є те, що кожен екземпляр сутності можна однозначно ідентифікувати за допомогою тільки тих атрибутів, які відносяться до сутності цього типу. Наприклад, для сутності Перевагу немає первинного ключа. Це означає, що кожен екземпляр сутності типу Перевагу неможливо ідентифікувати тільки за допомогою атрибутів цієї сутності. Однозначно ідентифікувати кожне побажання клієнта можна тільки, враховуючи зв'язок конкретного побажання з деяким клієнтом, який однозначно ідентифікується з використанням первинного ключа для сутності типу клієнта, а саме: clientNo. У даному прикладі сутність Перевагу розглядається як залежна у своєму існуванні від сутності Клієнт яка описує майбутнього орендаря, охочого орендувати об'єкти нерухомості конкретного типу і на певних умовах.

Сутності слабкого типу іноді називають дочірніми, залежними або підлеглими, а сутності сильного типу - батьківськими, сутностями власниками або домінантними.

Зв`язки

Зв`язок - поіменована асоціація двох або більшої кількості сутностей.

Тип зв'язку. *Набір осмислених асоціацій між сутностями різних типів.*

Тип зв'язку (relationship type) є набором асоціацій між одним (або кількома) типами сутностей, що беруть участь у цьому зв'язку. Кожному типу зв'язку присвоюється ім'я, яке повинне описувати його призначення.

Як і при використанні понять сутності і типу сутності, необхідно розрізняти поняття "екземпляр зв'язку" і "тип зв'язку".

Екземпляр зв'язку

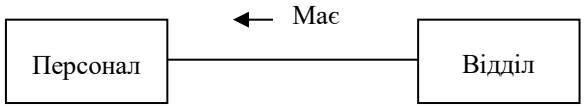
Асоціація, що однозначно ідентифікується, яка включає по одному екземпляру сутностей з кожного типу сутності,що бере участь у зв'язку.

Екземпляр зв'язку позначає всі конкретні екземпляри сутності, що беруть участь у цьому зв'язку.

Схематичне зображення типів зв'язків

Кожен тип зв'язку зображається у вигляді лінії, що з'єднує відповідні типи сутностей, що позначені ім'ям цьому зв'язку. Зазвичай для позначення імені зв'язку прийнято використовувати дієслово або коротку фразу, що містить дієслово. В цьому випадку перша буква кожного слова в імені зв'язку є прописною. У кожній конкретній ER-моделі всі імена зв'язків повинні бути унікальними.

На схемі повинно бути показано напрямок дії кожного зв'язку, оскільки зазвичай має сенс тільки один напрямок цього зв'язку. Тому поряд з ім'ям зв`язку на схемі розміщується стрілка, яка показує напрямок її дії.



Приклад зв’язку: *відділ має персонал*

Ступінь типу зв'язку

Кількість типів сутностей, які охоплені даною зв'язком.

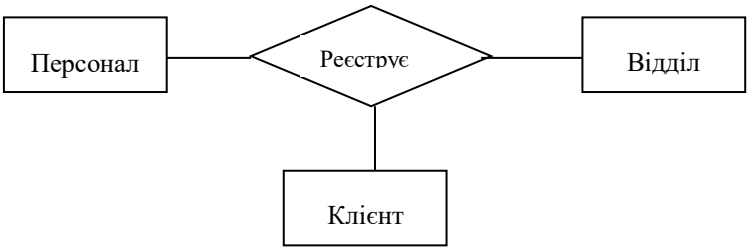
Сутності, охоплені деяким зв'язком, називаються *учасниками цього зв'язку*.

Кількість учасників зв'язку певного типу називається *ступенем (degree) цього зв'язку*. Отже, ступінь зв'язку вказує кількість типів сутностей, охоплених даним зв'язком.

Зв'язок зі ступенем два називається двостороннім (binary, бінарним). Прикладом двостороннього зв'язку є зв'язок *Має*, в якому беруть участь сутності двох типів, а саме Персонал і Відділ

Зв'язок зі ступенем три називається тристороннім (ternary). Прикладом тристороннього зв'язку є зв'язок *Реєструє* в якому беруть участь три типи сутностей, а саме *Персонал, Відділ і Клієнт*. Цей зв'язок являє процес реєстрації клієнта

представником персоналу відділення. Для опису зв'язків зі ступенем більше двох прийнято застосовувати термін *складний зв'язок*.



Співробітник реєструє клієнта у відділенні

На практиці найчастіше зустрічаються зв'язки зі ступенем два, тобто двосторонні зв'язки. При логічному проектуванні всі багатосторонні зв'язки повинні бути приведені до бінарних, так як багатосторонні зв'язки не можуть бути описані в таблицях бази даних. Тому будемо розглядати бінарні зв'язки.

Рекурсивний зв'язок

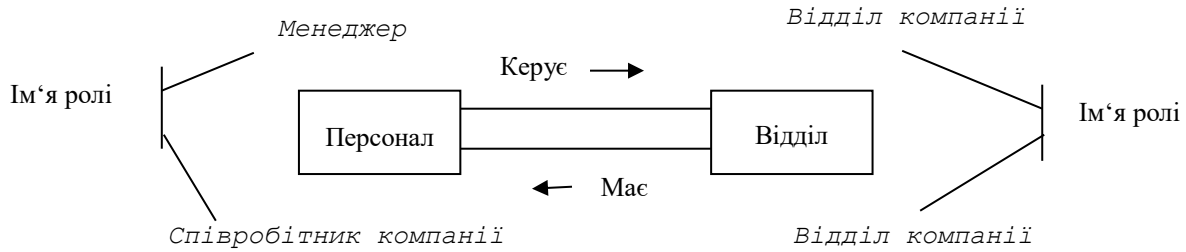
Зв'язок, в якому одні й ті ж сутності беруть участь кілька, разів у різних ролях.

Наприклад, рекурсивним буде зв'язок *Курирує*, який представляє взаємозв'язок персоналу з інспектором, а також входить до складу персоналу. Інакше кажучи, сутність *Персонал* бере участь у зв'язку *Курирує* двічі: перший раз - в якості інспектора, а другий - в якості співробітника, яким управляють.

Рекурсивні зв'язки іноді називаються односторонніми (unary).

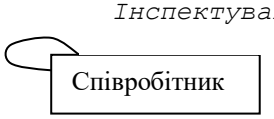
Зв'язкам можуть присвоюватися рольові імена для вказівки призначення кожної сутності, що бере участь у даному зв'язку. Рольові імена мають велике значення в рекурсивних зв'язках, оскільки дозволяють визначити функції кожного учасника.

Рольові імена можуть також використовуватися, коли дві сутності пов'язані декількома зв'язками. Наприклад, сутності *Персонал* та *Відділ* пов'язані двома різними зв'язками - *Керує* і *Має*. У зв'язку *Керує* один із представників персоналу (сутності *Персонал*) з рольовим ім'ям *Менеджер* керує відділенням компанії (сутність *Відділ*). А в разі зв'язку *Має* відділення з рольовим ім'ям *Відділ* має персонал, представникам якого присвоєно рольове ім'я *Співробітник компанії*.



Рольові імена зазвичай не потрібні, якщо функції сутностей, що беруть участь у зв'язку, визначені однозначно.

Зв'язок сутності з самим собою



Інспектор інспектує інших співробітників, але він також є співробітником.

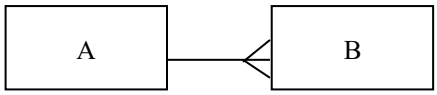
Характеристики бінарних зв'язків

З кожного боку бінарний зв'язок має такі характеристики:

- ім.я,
- множинність,
- обов'язковість.

Множинність може бути *один* і *багато*. Якщо зв'язок між сутностями А і В з боку сутності А має множинність *один*, то це означає, що кожен екземпляр В асоціюється не більше ніж з одним екземпляром А. Якщо зв'язок з боку В є множинним, це означає, що екземпляр А асоціюється з довільною кількістю екземплярів В.

На діаграмах множинність *один* зображується одинарним кінцем лінії, *багато* – потрійним кінцем (пташина лапка) .



Зв'язок може бути *обов'язковим* або *факультативним*.

Зв'язок називається *обов'язковим* з боку сутності А, якщо кожен її екземпляр повинен асоціюватись хоча б з одним екземпляром сутності В.

Якщо участь у зв'язку екземплярів А не є обов'язковою, то зв'язок називається *факультативним*.

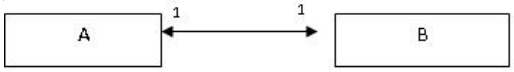
На діаграмах обов'язковий зв'язок зображується неперервною лінією, факультативний – пунктирною.

Типи зв'язків

Зв'язок дозволяє моделювати відносини між об'єктами предметної області. Найменування зв'язку повинно бути унікальним у всій моделі.

Існує 4 типи зв'язків:

а) *«Один-до-одного»* - будь екземпляру сутності А відповідає тільки один екземпляр сутності В, і навпаки.



У будь-якого конкретного учня може бути тільки одна характеристика, і ця характеристика відноситься до єдиного учня.

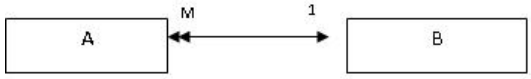
б) *«Один-до-багатьох»* - будь екземпляру сутності А відповідає кілька екземплярів сутності В, але будь якому екземпляру сутності В відповідає тільки один екземпляр сутності А.

Учневі ставлять багато оцінок; поставлена оцінка належить тільки одному учневі.

У відділі працює кілька співробітників, але кожний співробітник працює в одному відділі.

В групі навчаються кілька студентів, але кожен студент навчається тільки в одній групі.

в) *«Багато-до-одного»* - будь якому екземпляру сутності А відповідає тільки один екземпляр сутності В, але будь якому екземпляру сутності В відповідає кілька екземплярів сутності А.



Викладач працює тільки в одному кабінеті, однак робочий кабінет може бути закріплений за кількома викладачами.

Яка ж різниця між зв'язками «один-до-багатьох» та «багато-до-одного»? Така ж, як між фразами «портфель учня» і «учень портфеля». Тобто важливо, хто у взаємовідносинах двох об'єктів головний - учень або портфель. Суть відносин двох об'єктів відображається в імені зв'язку.

г) *«Багато-до-багатьох»* - будь екземпляру сутності А відповідає 0, 1 або кілька екземплярів сутності В, і будь екземпляру сутності В відповідає 0, 1 або кілька екземплярів сутності А.

Учень Іванов вчиться у декількох викладачів. І кожен викладач працює з багатьма учнями.

Один викладач може читати кілька дисциплін; і кожну дисципліну може читати кілька викладачів.

Опис зв‘язків здійснюється в таблиці

Сутність 1	**Сутність 2**	**Назва зв‘язку**	**Тип зв‘язку**
студент	група	входить	m : 1

Атрибути

Атрибут. Властивість типу сутності чи типу зв'язку.

Окремі властивості сутности називаються атрибутами. Атрибут містять значення, Які описують кожен екземпляр сутності и складають основнову частину інформації, что зберігається в базі даних.

Атрибути підрозділяються на прості і складені, однозначні і багатозначні, а також похідні.

Прості і складені атрибути

Простий атрибут. Атрибут, що складається з одного компонента з незалежним існуванням.

Прості атрибути не можуть бути розділені на більш дрібні компоненти. Прикладом простих атрибутів є атрибут позиція (Посада) або зарплата (Зарплата) сутності персонал. Прості атрибути іноді називають елементарними,

Складений атрибут: Атрибут, що складається з кількох компонентів, кожен з яких характеризується незалежним існуванням.

Деякі атрибути можуть бути розділені на більш дрібні компоненти, які характеризуються незалежним існуванням. Наприклад, атрибут адресу (Адреса) може бути розбитий на окремі атрибути вулиця ('163 Main St'), місто ('Glasgow1) і поштовий індекс (' Gi 2 9QX1) -

Рішення про моделювання атрибуту адреса у вигляді простого атрибута або розбивці його на атрибути вулиця, місто та поштовий індекс залежить від того, як розглядається атрибут адреса в користувальницькому уявленні - як єдине ціле або як набір окремих компонентів.

Однозначний і багатозначний атрибути

Однозначний атрибут. Атрибут, який містить одне значення для кожного екземпляра сутності певного типу.

Більшість атрибутів є однозначними. Наприклад, для кожного окремого екземпляра сутності Галузь завжди є єдине значення в атрибуті номера відділення компанії (branchNo), скажімо, "BOO3. Тому атрибут branchNo є однозначним.

Багатозначний атрибут. Атрибут, який містить кілька значень для кожного екземпляра сутності певного типу.

Деякі атрибути можуть мати кілька значень для кожного екземпляра сутності. Наприклад, сутність Branch може мати кілька значень для атрибута telNo (Номер телефону відділення компанії). Отже, атрибут telNo в цьому випадку буде багатозначним.

Похідні атрибути

Похідний атрибут. Атрибут, який представляє значення, похідне від значення пов'язаного з ним атрибута або деякого безлічі атрибутів, належать деякому (не обов'язково даним) типом сутності.

Деякі атрибути можуть бути пов'язані з певною сутністю. Наприклад, значення атрибута duration (Термін дії) сутності Lease (Договір оренди) обчислюється на основі атрибутів rentstart (Початок строку оренди) і rentFinish (Кінець терміну оренди), які також відносяться до типу сутності Lease.

Опис атрибутів здійснюється в таблиці

Сутність	Атрибут	Опис	Тип	Обмеження
Студент	Прізвище		Рядок	
	Ім`я		Рядок	
	Дата Народження		Дата	1900-сьогодні

Ключі

Потенційний ключ. Атрибут або мінімальний набір атрибутів, який однозначно ідентифікує кожен екземпляр типу сутності.

Потенційний ключ - це один або декілька атрибутів, значення яких однозначно ідентифікують кожен екземпляр сутності даного типу. Наприклад, номер відділення компанії (branchNo) є потенційним ключем для сутності відділення оскільки він містить різні значення для кожного окремого екземпляра сутності Галузь. Потенційний ключ повинен містити значення, які унікальні для кожного окремого примірника сутності даного типу. Це означає, що потенційний ключ не може містити значення NULL

Первинний ключ. Потенційний ключ, який обраний для однозначної ідентифікації кожного екземпляра сутності певного типу.

Тип суті може мати кілька потенційних ключів. Наприклад, кожен співробітник може мати унікальний табельний номер STAFF_KO, а також унікальний номер картки державного соціального страхування.

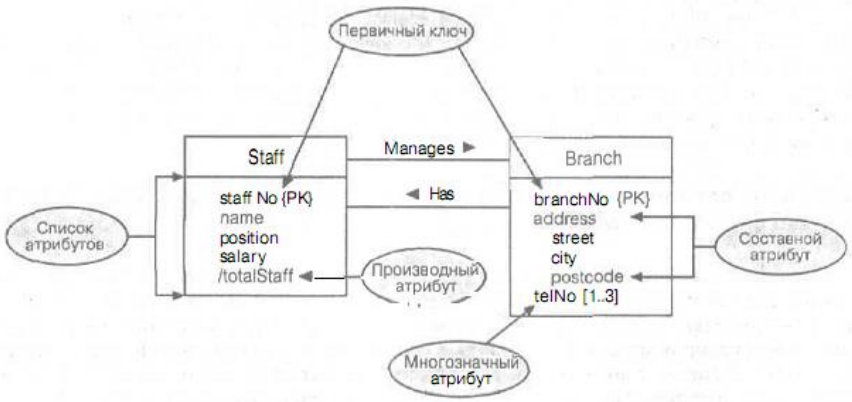
Вибір первинного ключа сутності здійснюється з урахуванням сумарної довжини атрибутів, мінімальної кількості необхідних атрибутів в ключі, а також наявності гарантій унікальності його значень в поточний момент часу і в осяжному майбутньому.

Складений ключ. Потенційний ключ, який складається з двох або декількох атрибутів.

У деяких випадках ключ сутності складається з декількох атрибутів, значення яких, взяті разом, а не окремо, унікальні для кожного екземпляра сутності.

Схематичне представлення атрибутів

Якщо сутність відображається на схемі разом зі своїми атрибутами, то прямокутник, який цю сутність визначає, ділиться на дві частини. У верхній частині відображається ім'я сутності, а в нижній - список імен атрибутів. ПЕРШИМ атрибутом (атрибутами) у списку повинен бути первинний ключ для сутності даного типу, якщо він відомий. Ім'я (імена) атрибута (атрибутів) первинного ключа має бути позначено дескриптором {PK} (скорочення від первинного ключа). В UML прийнято привласнювати атрибуту ім'я, яке починається з малої літери, а якщо воно складається з декількох слів, то перша буква кожного наступного слова пишеться з великої літери



У деяких найбільш простих додатках баз даних на ЕР-діаграмі можуть бути показані всі атрибути суті кожного типу. Але у випадку більш складних додатків баз даних на цих схемах відображається тільки один або декілька атрибутів, які утворюють первинний ключ суті кожного типу .

Імена простих атрибутів просто відображаються у вигляді списку під ім'ям сутності. А за складним атрибутом слід список складових його простих атрибутів, позначений відступом вправо.

2.2 Проектування баз даних

1) Концептуальне проектування
2) Логічне проектування
3) Нормалізація даних
4) Фізичне проектування баз даних

Проектування бази даних це процес створення проекту БД, призначеної для підтримки функціонування підприємства і підримки досягнення його мети.

Проектування баз даних - це ітераційний, багатоетапний процес прийняття обґрунтованих рішень в процесі аналізу інформаційної моделі предметної області, вимог до даних з боку прикладних програмістів і користувачів, синтезу логічних і фізичних структур даних, аналізу та обгрунтування вибору програмних і апаратних засобів. Етапи проектування баз даних пов'язані з багаторівневою організацією даних.

Відповідно до пропозицій дослідницької групи по системам управління даними Американського національного інституту стандартів ANSI / X3 / SPARC, а також CODASYL (Конференція з Data Systems мов), як правило, виділяється три рівні представлення даних:

- зовнішній рівень (з точки зору кінцевого користувача і прикладного програміста),
- концептуальний рівень (з точки зору моделі даних),
- внутрішній рівень (з погляду системного програміста).

Відповідно до цієї концепції зовнішній рівень це частина (підмножина) концептуальної моделі, необхідна для реалізації якого-небудь запиту або прикладної програми. Тобто, якщо концептуальна модель це повна схема даних, то зовнішній рівень - це деяка сукупність підсхем, необхідних для реалізації конкретної прикладної програми або запиту користувача.

Існує також інша точка зору:

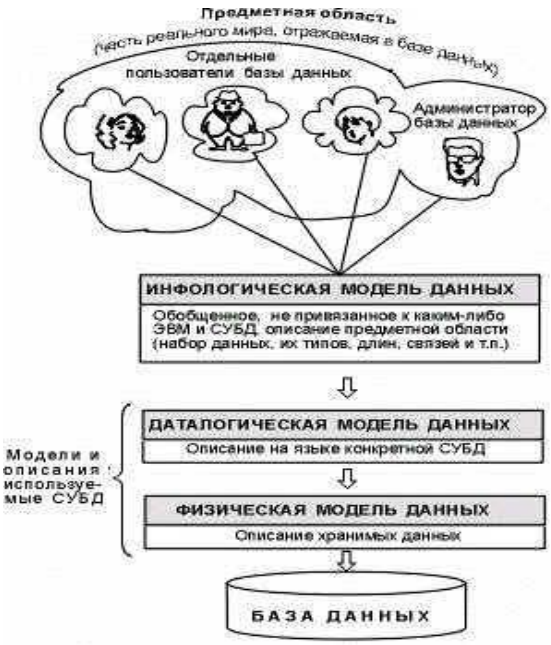
- *інфологічний рівень (з включенням зовнішнього)*
- *даталогічний рівень*
- *фізичний рівень*

Трирівнева архітектура (інфологічній, даталогічний и фізичний рівні) дозволяє забезпечити незалежність збереження даних від програм, в яких вони використовуються. Можна при необхідності переписати збережені дані на інші носії інформації і (або) реорганізувати їхню фізичну структуру. Можна підключиті до системи будь-яке число нових користувачів (нових додатків), доповнити, якщо треба, даталогічну модель. Зазначені зміни фізичної и даталогічної моделей не будуть замічені існуючімі користувачами системи, так само як не будуть замічені і нові користувачі. Отже, незалежність даних забезпечує можлівість розвитку системи баз даних без руйнування існуючіх додатків.

Таким чином, інфологічна модель відображає реальний світ у деякій зрозумілій людіні концепції, цілком незалежній від параметрів середовища збереження даних. Існує безліч підходів до побудова таких моделей: графові моделі, семантичні мережі, модель "сутність-зв'язок" і т.д. Найбільше популярну среди них віявілася модель "сутність-зв'язок".

Мета інфологічного моделювання - забезпечення найбільш природних для людини засобів збору і подання информации, что буде зберігатіся в утворюваній базі даних. Тому інфологічну модель даних намагають будуваті за аналогією з природною

мовою. Основними конструктивними елементами інфологічніх моделей є сутності, зв'язки между ними и їхні властивості (атрибути).



При проектуванні БД на зовнішньому рівні необхідно вивчити функціонування об'єкта управління, для якого проектується БД, всю первинну і вихідну документацію з погляду визначення того, які саме дані необхідно зберігати в базі даних. Зовнішній рівень це, як правило, словесний опис вхідних і вихідних повідомлень, а також даних, які доцільно зберігати в БД. Опис зовнішнього рівня не виключає наявності елементів дублювання, надмірності і неузгодженості даних. Тому для усунення цих аномалій і протиріч зовнішнього опису даних виконується інфологічне проектування. Інфологічна модель є засобом структуризації предметної області та розуміння концепції семантики даних. Інфологічну модель можна розглядати в основному як засіб документування і структурування форми подання інформаційних потреб, яка забезпечує несуперечливе спілкування користувачів і розробників системи.

Всі зовнішні уяввлення інтегруються на інфологічну рівні, де формується інфологічесна (канонічна) модель даних, яка не є простою сумою зовнішніх представлень даних.

Інфологічний рівень являє собою інформаційно-логічну модель (Илм) предметної області, з якої виключена надмірність даних і відображені інформаційні особливості об'єкта управління без урахування особливостей і специфіки конкретної СУБД. Тобто інфологічне подання даних орієнтовано переважно на людину, яка проектує або використовує базу даних.

Даталогічний рівень побудований з урахуванням специфіки та особливостей конкретної СУБД. Цей рівень представлення даних орієнтований більше на комп'ютерну обробку і на програмістів, які займаються її розробкою. На цьому рівні формується концептуальна модель даних, тобто спеціальним способом структурована модель предметної області, яка відповідає особливостям і обмеженням обраної СУБД.

Інфологичічна і даталогічна моделі, які відображають модель однієї предметної області, залежні між собою. Інфологичічна модель може легко трансформуватися в даталогічну модель.

Внутрішній рівень пов'язаний з фізичним розміщенням даних у пам'яті ЕОМ. На цьому рівні формується фізична модель БД, яка включає структури збереження даних у пам'яті ЕОМ, в т.ч. Опис форматів записів, порядок їх логічного чи фізичного

впорядкування, розміщення за типами пристроїв, а також характеристики і шляхи доступу до даних.

Від параметрів фізичної моделі залежать такі характеристики функціонування БД: обсяг пам'яті і час реакції системи. Фізичні параметри БД можна змінювати в процесі її експлуатації з метою підвищення ефективності функціонування системи. Зміна фізичних параметрів не визначає необхідність зміни інфологичної і даталогічної моделей.

Проектування баз даних складається з трьох основних етапів:
- концептуальне проектування (інфологічна модель)
- логічне проектування (даталогічна модель)
- фізичне проектування (фізична модель)

2.2.3 Концептуальне проектування

Концептуальне проектування бази даних - конструювання інформаційної моделі, що не залежить від будь-яких фізичних умов реалізації.

Концептуальне проектування – створення концептуального представленя бази даних, що включає визначення типів найважливіших сутностей і існуючих між ними зв'язків і атрибутів.

Концептуальне проектування бази даних включає:
- Створення локальної концептуальної моделі даних, виходячи з представлень про предметну області кожного з типів користувачів.
- Визначення типів сутностей.
- Визначення типів зв'язків.
- Визначення атрибутів і зв'язування їх з типами сутностей і зв'язків.
- Визначення атрибутів, що є потенційними і первинними ключами.
- Перевірка моделі на відсутність надмірності.
- Перевірка відповідності локальної концептуальної моделі конкретним транзакціям користувача.
- Обговорення локальних концептуальних моделей даних з кінцевими користувачами.

Приклад.

Спроектувати БД "Сесія". БД повинна видавати оперативну інформацію про успішність студентів в семестрі. Результатами сесії вважати лише екзамени.

Оберемо наступні сутності:

ГРУПА СТУДЕНТ ВИКЛАДАЧ ДИСЦИПЛІНА

Задамо кожну сутність набором атрибутів:

ГРУПА (*назва*).

СТУДЕНТ (*ПІБ, група*, курс, стать).

ВИКЛАДАЧ (*ПІБ*, посада, звання).

ДИСЦИПЛІНА (*назва*, число годин).

В кожному наборі атрибутів, що характеризують сутність, необхідно вибрати ключові атрибути, тобто атрибути, що роблять сутність унікальною. При заданні атрибутів ключові атрибути позначалися *курсивом*.

Розглянемо деякі семантичні обмеження в прикладі, що розглядається:
- Одному студенту може бути приписано лише одну групу.
- Один студент вивчає в семестрі декілька дисциплін.
- Однин викладач може викладати кілька дисциплін.

Проаналізуємо зв'язки між сутностями:

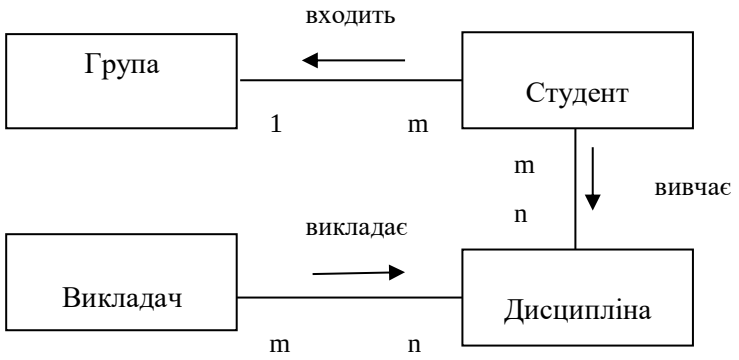
Кожен студент може навчатись лише в одній групі, але в одній группі навчається кілька студентів, тому зв'язок студент-група буде m : 1.

Кожен студент вивчає кілька дисциплін і кожну дисципліну вивчає кілька студентів, тому зв'язок студент-дисципліна m : n.

Кожен викладач може викладати кілька дисциплін і одну і ту ж дисципліну можуть викладати кілька викладачів, тому зв'язок викладач-дисципліна буде m : n.

Сутність 1	**Сутність 2**	**Назва зв'язку**	**Тип зв'язку**
студент	група	входить	m : 1
студент	дисципліна	вивчає	m : n
викладач	дисципліна	викладає	m : n

Після вибору сутностей, задання атрибутів та аналізу зв'язків ми можемо перейти до проектування інформаційної (концептуальної) схеми БД.



2.2.4 Логічне проектування

Логічне проектування бази даних - це побудова ніформационної моделі на основі існуючих конкретних моделей даних, але без урахування використовуваної СУБД і інших фізичних умов реалізації.

Логічне проектування – перетворення концептуального представлення в логічну структуру бази даних, включаючи проектування відносин.

Основним завданням логічного проектування є розробка логічної схеми, орієнтованої на обрану модель даних.

Відображення інформаційної схеми на логічну схему БД

Розглянемо відображення інформаційної схеми на реляційну, ієрархічну та мережеву моделі даних.

a) Відображення на реляційну модель даних

При відображенні інформаційної схеми кожний прямокутник схеми для моделей ER у вигляді ER-діаграми відображується в таблицю, яка є одним відношенням.

b) Відображення на ієрархічну модель даних

При відображенні інформаційної схеми на узагальнену ієрархічну модель даних враховуються наступні умови:

- В ієрархічній моделі даних має місце один корінний вузел. Він знаходиться на нульовому рівні.

- Допускається між вузлом (i+1)-го рівня та вузлом i-го рівня лише один зв’язок типу “багато до одного”.
- Число зв’язків “один до багатьох” між вузлом i-го рівня та вузлами (i+1)-го рівня може бути будь-яким (залежить від певної СУБД).
- Ніякі інші зв’язки в узагальненій ієрархічній моделі не допускаються.

При перетворенні інформаційної схеми можливе злиття сутностей різних типів в одну, при цьому необхідно розраховувати число звертань до логічних записів, об’єм передачі, об’єм пам’яті даних для кожного варіанта логічної схеми.

c) Мережевий підхід

Мережевий підхід використовує три основних поняття:

- Набір.
- Запис-власник.
- Запис-член зв’язку.

Для реалізації мережевої моделі даних СУБД мають ММД та МОД.

Логічне проектування для реляційної моделі

Процес логічного проектування складається з наступний етапів:

- Усунення особливостей локальної логічної моделі, несумісних з реляційної моделлю (необов'язковий етап).
- Визначення набору відносин, виходячи зі структури локальної логічної моделі даних.
- Перевірка відносин за допомогою правил нормалізації.
- Перевірка відповідності відносин вимогам користувальницьких транзакций.
- Визначення вимог підтримки цілісності даних.
- Створення та перевірка глобальної логічної моделі даних.
- Перевірка глобальної логічної моделі даних.

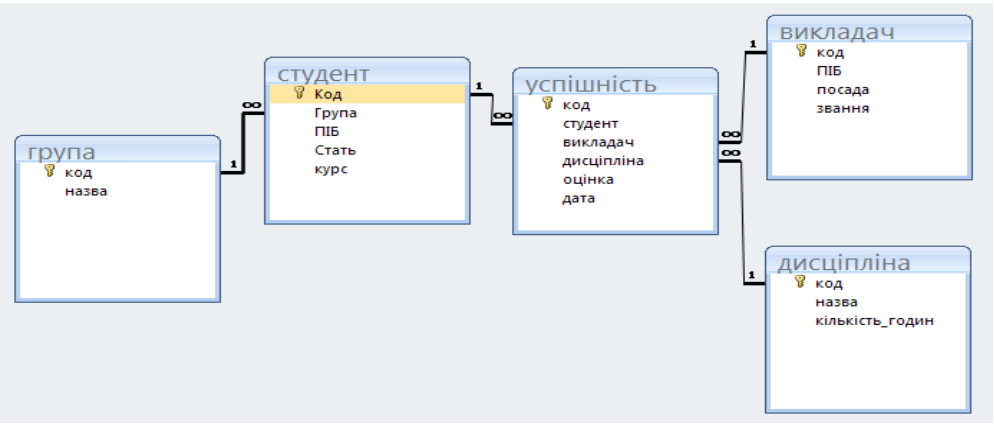
Приклад

Для реалізації оберемо реляційну модель.

Реляційна модель не підтримує зв`язки m : n, тому для усунення такого зв`язку введемо додаткову сутність УСПІШНІСТЬ атрибутами якої будуть дисципліна, студент викладач, оцінка, дата.

В якості первинного ключа оберемо ціле значення, яке буде не повторюваним кодом сутності.

В результаті буде отримана логічна модель:



2.2.3 Нормалізація даних

Нормалізація являє собою процес реорганізації даних шляхом ліквідації повторюваних груп і інших протиріч з метою приведення таблиць до виду, що дозволяє здійснювати несуперечливе і коректне редагування даних.

Остаточна мета нормалізації зводиться до одержання такого проекту бази даних, у якому кожен факт з'являється лише в одному місці, тобто виключена надмірність інформації. Таким чином, нормалізацію можна також визначити як процес, спрямований на зменшення надмірності інформації в реляционной базі даних.

2.2.3.1 Проблеми, що порушують цілісність даних

Ціль нормалізації

Надмірність інформації усувається не стільки з метою економії пам'яті, скільки для виключення можливої суперечливості збережених даних і спрощення керування ними.

Використання ненормалізованих таблиць може привести до порушення цілісності даних (суперечливості інформації) у базі даних. Звичайно розрізняють наступні проблеми, що виникають при використанні ненормалізованих таблиць:

- надмірність даних;
- аномалії відновлення;
- аномалії видалення;
- аномалії введення.

Щоб проілюструвати проблеми, що виникають при роботі з ненормалізованими базами даних, розглянемо як приклад таблицю СПІВРОБІТНИКИ, що містить інформацію про співробітників деякої організації. Структура цієї таблиці приведена на рис.1

Структура ненормалізованої таблиці СПІВРОБІТНИКИ

Код співробітника
Прізвище, Ім'я, По батькові
Дата народження
Адреса
Телефон
Посада
Розряд
Зарплата
Рейтинг
Дата прийому
Дата звільнення

Рисунок 1

Надмірність даних

Надмірність даних виявляється в тім, що в декількох записах таблиці бази даних повторюється та сама інформація. Наприклад, одна людина може працювати на двох (або навіть більш) посадах. Але в таблиці, приведеної на мал. 1, кожної посади відповідає запис, і в цьому записі утримується інформація про особисті дані співробітника, цю посаду що займає. Таким чином, якщо співробітник працює на декількох посадах, те його особисті дані будуть дублюватися кілька разів, що приведе до невиправданого збільшення займаного обсягу зовнішньої пам'яті.

Аномалії відновлення

Аномалії відновлення тісно зв'язані з надмірністю даних. Припустимо, що в

співробітника, що працює на декількох посадах, змінилася адреса. Щоб інформація, що утримується в таблиці, була коректної, необхідно буде внести зміни в кілька записів. Якщо ж виправлення буде внесено не в усі записи, то виникне невідповідність інформації, що і називається аномалією відновлення.

Аномалії видалення

Аномалії видалення виникають при видаленні записів з ненормалізованої таблиці. Нехай, наприклад, в організації проводиться скорочення штатів і деяких посад анулюються. При цьому варто видалити відповідні записи в розглянутій таблиці. Однак видалення приведе до втрати інформації про співробітника, що займав цю посаду. Така втрата інформації і називається аномалією видалення. (Для нашого випадку можна привести й інший приклад — видалення запису при звільненні співробітника приведе до втрати інформації про посаді, що він займав.)

Аномалії введення

Аномалії уведення виникають при додаванні в таблицю нових записів і звичайно виникають, коли для деяких полів таблиці задані обмеження NOT NULL. У таблиці, розглянутої як приклад, мається поле «Рейтинг», у якому утримується інформація про рівень кваліфікації співробітника, установлюваному за результатами його роботи. При прийомі на роботу нового співробітника установити рівень його кваліфікації неможливо, так він ще не виконував ніяких робіт в організації. Якщо для цього полючи задати обмеження NOT NULL, то в таблицю не можна буде ввести інформацію про нового співробітника. Це і називається аномалією введення.

2.2.3.2 Нормальні форми

Теорія нормалізації заснована на концепції *нормальних, форм.* Кожній нормальній формі відповідає деякий визначений набір обмежень, і відношення знаходиться в деякій нормальній формі, якщо воно задовольняє властивому даній формі наборові обмежень.

У теорії реляційних баз даних звичайно виділяється наступна послідовність нормальних форм:

- перша нормальна форма (1NF);
- друга нормальна форма (2NF);
- третя нормальна форма (3NF);
- нормальна форма Бойса-Кодда (УCNF);
- четверта нормальна форма (4NF);
- п'ята нормальна форма, або нормальна форма сполуки-сполучення-проекції-з'єднання (5NF)

Основні властивості нормальних форм:

- кожна наступна нормальна форма в деякому змісті краще попередньої;
- при переході до наступної нормальної форми властивості попередніх нормальних властивостей зберігаються.

В основі процесу проектування лежить метод нормалізації — декомпозиція відносини, що знаходяться в попередній нормальній формі, у двоє або більш відносин, що задовольняють вимогам наступної нормальної форми.

Найбільш важливі на практиці нормальні форми відносин ґрунтуються на фундаментальному в теорії реляційних баз даних понятті *функціональної залежності.* Функціонально залежним вважається такий атрибут, значення якого однозначно визначається значенням іншого атрибута. Функціонально залежні атрибути позначаються в такий спосіб:

$X \rightarrow$ У.

Цей запис означає, що якщо два кортежі в таблиці мають те саме значення атрибута х, то вони мають те саме значення атрибута У. Атрибут, що вказується в лівій частині, називається *детермінантом.*

Перша нормальна форма

Обмеження першої нормальної форми — відсутність множинних атрибутів (наприклад, у студента Оцінка1, Оцінка2, Оцінка3 або кілька номерів телефонів у підприємства) і значення всіх атрибутів відношення повинні бути атомарними.

Дана вимога є базовою вимогою класичної реляционной моделі даних.

Код співробітника
Прізвище
Ім'я
По батькові
Дата народження
Місто
Вулиця
Будинок
Квартира
Телефон
Посада
Розряд
Зарплата
Рейтинг
Дата прийому
Дата звільнення

Рисунок 2

Друга нормальна форма

Друга нормальна форма застосовується до відношень з складними первинними ключами.

Відношення знаходиться в другій нормальній формі в тому і тільки в тому випадку, коли це відношення знаходиться в першій нормальній формі і кожен неключовий атрибут цілком залежить від всього первинного ключа.

Для приведення таблиць до другої нормальної форми необхідно забезпечити повну залежність стовпців, які не є ключовими, від первинного ключа, а якщо цей ключ складений, то від кожного його елемента. Під повною залежністю розуміється можливість однозначного визначення значення кожного не ключового поля за допомогою значення первинного ключа.

Щоб перейти від першої нормальної форми до другого, *потрібно* виконати наступні кроки:

- Визначити, на які частини можна розбити первинний ключ, так щоб деякі з неключових полів залежали від однієї з цих частин (причому ці частини можуть містити кілька атрибутів).
- Створити нову таблицю для кожної такої частини ключа і групи залежних від неї полів і перемістити них у цю таблицю. Частина колишнього первинного ключа стане при цьому первинним ключем нової таблиці.
- Видалити з вихідної таблиці полючи, переміщені в інші таблиці, крім тих них них, що стануть зовнішніми ключами.

У нашому прикладі для приведення таблиці СПІВРОБІТНИКИ до другої нормальної форми її варто розділити на дві таблиці. Первинний ключ вихідної таблиці складається з двох атрибутів — «Код співробітника» і «Посада». Все-таки особисті дані про співробітників залежать тільки від атрибута «Код співробітника». Атрибути, що відповідають цим даним, ми і виділимо в якості однієї з таблиць, що назвемо ФІЗИЧНИХ ОСІБ. Інформацію же про посади і їхню оплату винесемо в іншу таблицю, який привласнимо ім'я СПІВРОБІТНИКИ. Схема приведення таблиці до другої нормальної форми приведена на рис. 3.

Код співробітника
Прізвище

Код фізичної особи
Прізвище

Рисунок 3

Отримані дві таблиці зв'язані між собою по полю «Код фізичної особи», що є первинним ключем для таблиці ФІЗИЧНІ ОСОБИ і зовнішній ключ для таблиці СПІВРОБІТНИКИ. Дане поле було відсутнє у вихідній таблиці і було додано при проведенні нормалізації.

Третя нормальна форма

Відношення знаходиться в третій нормальній формі в тому і тільки в тому випадку, якщо воно знаходиться в другій нормальній формі і кожен неключовий атрибут нетранзитивно залежить від первинного ключа. Тобто у третій нормальній формі стовпці, які не є ключовими, залежать від первинного ключа таблиці і не залежать від всіх інших стовпців.

Щоб перейти від другої нормальної форми до третього, потрібно виконати наступні кроки:

- Визначити всі поля (чи групи полів), від яких залежать інші поля.
- Створити нову таблицю для кожного такого поля (чи групи полів) і групи залежних від нього полів і перемістити їх у цю таблицю. Поле (чи група полів), від якого залежать всі інші переміщені полючи, стане при цьому первинним ключем нової таблиці.
- Видалити переміщені полючи з вихідної таблиці, залишивши лише ті з них, що стануть зовнішніми ключами.

Розглянемо таблицю СПІВРОБІТНИКИ, отриману після приведення вихідної таблиці до другої нормальної форми. Для цієї таблиці існує функціональний зв'язок між полями «Код співробітника» і «Зарплата». Однак цей функціональний зв'язок є *транзитивний.*

Транзитивність залежності полів «Код співробітника» і «Зарплата» означає, що заробітна плата насправді є характеристикою не співробітника, а посади, що він займає. У результаті ми не зможемо занести в базу даних інформацію, що характеризує заробітну плату посади, доти, поки не з'явиться хоча б один співробітник, що цю посаду займає (тому що первинний ключ не може містити невизначене значення). При видаленні кортежу, що описує останнього співробітника, що займає дану посаду, ми позбавимося інформації про заробітну плату, що відповідає цієї посади. Крім того, щоб погодженим образом змінити заробітну плату, що відповідає посади, буде необхідно попередньо знайти всі записи, що описують співробітників, що займають дану посаду. Таким чином, у таблиці СПІВРОБІТНИКИ як і раніше існують аномалии. Їх можна

усунути шляхом подальшої нормалізації — приведення бази даних до третьої нормальної форми.

Приведемо розглянуту як приклад базу даних до третьої нормальної форми. Для цього розділимо таблицю СПІВРОБІТНИКИ на дві — СПІВРОБІТНИКІВ і ПОСАДИ (рис. 4).

Рисунок 4

На практиці третя нормальна форма схем відносин у більшості випадків достатня, і приведенням до третьої нормальної форми процес проектування реляціоної бази даних звичайно закінчується. Тому ми не будемо розглядати інші нормальні форми, тим більше що в роботі вони використовуються порівняно рідко.

Основне правило при створенні таблиць сутностей – це кожній сутності бажано зіставити окрему таблицю. Поля таблиць сутностей можуть бути ключовими або не ключовими. Введення ключів дозволяє забезпечити унікальність значень в записі, прискорити обробку запису і виконати обробку. Якщо в таблиці є значні повторення по декількох полях і їх обсяг суттєвий, то краще їх виділити в окрему таблицю. Нову сутність легко додати і змінити, але при видаленні слід знищити всі посилання на неї з таблиць зв'язків, інакше виникає некоректність.

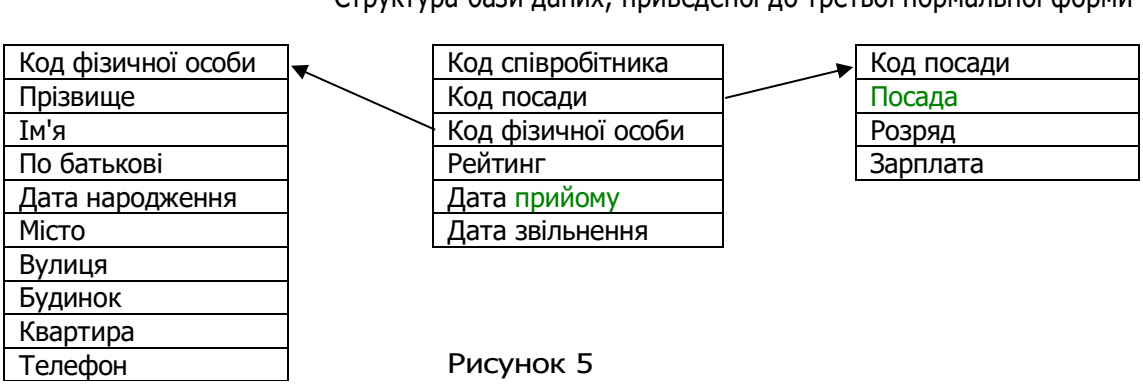


Рисунок 5

2.2.4 Фізичне проектування

Фізичне проектування є третім і останнім етапом створення проекту бази даних, при виконанні якого проектувальник приймає рішення про способи реалізації розроблюваної бази даних. Під час попереднього етапу проектування була визначена

логічна структура бази даних (яка описує відносини і обмеження в розглянутій прикладній області). Хоча ця структура не залежить від конкретної цільової СКБД, вона створюється з урахуванням обраної моделі зберігання даних, наприклад реляційної, мережевої або ієрархічної. Однак, приступаючи до фізичного проектування бази даних, перш за все необхідно вибрати конкретну цільову СКБД. Тому фізичне проектування нерозривно пов'язане з конкретною СКБД. Між логічним і фізичним проектуванням існує постійний зворотний зв'язок, так як рішення, що приймаються на етапі фізичного проектування з метою підвищення продуктивності системи, здатні вплинути на структуру логічної моделі даних.

Фізичне проектування бази даннях – опис конкретної реалізації бази даних, що розміщується в зовнішній пам'яті. Фізичний проект описує базові відносини, визначає організацію файлів і склад індексів, застосовуваних для забезпечення ефективного доступу до даних, а також регламентує всі відповідні обмеження цілісності і міри захисту.

Початковими даними для вирішення задач фізичного проектування є логічна структура БД та оцінки логічної структури.

Основні етапи фізичного проектування

Фізичне проектування бази даних (з використанням реляційної СКБД) складається з наступних основних етапів:

1) Проектування базових відношень у середовищі цільової СКБД.

Створення відношень засобами цільової СКБД. Для кожного відношення визначаються такі елементи: ім'я відношення, список простих атрибутів, визначення первинного ключа, визначення вимог посилальної цілісності для будь-яких зовнішніх ключів, допустимість значення Null для атрибута.

2) Реалізація обмежень предметної області.

Спосіб реалізації обмежень залежить від обраної СКБД. Альтернативним методом реалізації обмежень є механізм тригерів.

3) Проектування фізичного представлення бази даних.

Однією з найважливіших цілей фізичного проектування БД є організація ефективного зберігання даних. Існує кілька показників, які можуть бути використані для оцінки досягнутої ефективності:

- Продуктивність виконання транзакцій (представляє собою кількість транзакцій, які можуть бути оброблені за заданий інтервал часу);
- час відповіді (характеризує часовий проміжок, необхідний для виконання однієї транзакції. З точки зору користувача, бажано, щоб цей час був мінімальним);
- дискова пам'ять (являє собою обсяг дискового простору, необхідного для розміщення файлів БД).

Жоден з цих факторів не є самодостатнім.

Найбільший вплив на продуктивність системи становлять такі 4 компоненти апаратних засобів: оперативна пам'ять; процесор; дискове введення / виведення; мережа. Кожен з цих ресурсів здатний впливати на інші системні ресурси. Тому поліпшення показників використання одного ресурсу може позитивно позначитися на стані інших ресурсів.

4) Аналіз транзакцій.

Для успішного планування кожної транзакції необхідно знати наступне: транзакції, що виконуються найбільш часто і роблять істотний вплив на продуктивність; транзакції, найбільш важливі для роботи організації; періоди часу протягом діб / тижнів, в які навантаження БД зростає до максимуму. Для виділення областей, які з найбільшою ймовірністю можуть з'явитися джерелом проблем, необхідно: підготувати схеми відповідності шляхів виконання транзакцій і відношень; визначити відношення, найбільш часто використовувані при виконанні транзакцій; проаналізувати інтенсивності доступу до даних в деяких з транзакцій, що використовують ці відношення.

5) Визначення індексів.

Індекс – структура даних, яка допомагає СУБД швидше виявити окремі записи у файлі і скоротити час виконання запитів користувачів.

Файл, що містить логічні записи, називається файлом даних, а файл, що містить індексні записи, – індексним файлом. Значення в індексному файлі впорядковані по полю індексування, яке зазвичай будується на базі одного атрибута.

Типи індексів:

- Первинний індекс.
- Індекс кластеризації.
- Вторинний індекс

6) Розробка механізмів захисту.

Як правило, реляційні СКБД надають засоби захисту БД наступних типів: захист системи та захист даних. Засоби захисту системи регламентують доступ і експлуатацію БД на рівні системи; до них, зокрема, відноситься аутентифікація користувача по імені і паролю. Засоби захисту даних регламентують доступ і використання об'єктів БД (таких як відношення і представлення), а також дії, які можуть бути виконані користувачами з конкретними об'єктами. І в цьому випадку проект реалізації правил доступу залежить від цільової СКБД.

РОЗДІЛ 3. МОВА СТРУКТУРОВАНИХ ЗАПИТІВ TRANSACT SQL

1) Основи мови структурованих запитів SQL
2) Мова визначення даних
3) Маніпулювання даними

3.1 Основи мови структурованих запитів SQL

1) Архітектура баз даних
2) Елементи мови Transact-SQL

3.1.1 Архітектура баз даних СКБД SQL Server

1) архітектура сервера
2) архітектура бази даних в SQL Server

3.1.1.1 Архітектура сервера

SQL Server – це заснований на SQL сервер реляційних баз даних.

База даних – це звичайний файл даних, що представляє собою певну область, де зберігається інформація, її схема (schema) з описом структури і типів даних, визначеннями таблиць, представленнями, дозволами і т.п. По суті, схема містить всі допоміжні відомості, необхідні для роботи бази даних. Зазвичай база складається з двох компонентів: файлів даних з інформацією та програм доступу до бази даних. Ці програми називають системами управління базами даних або скорочено СКБД (database management system, DBMS).

SQL Server призначений для роботи в клієнт-серверних системах. Подібна архітектура має на увазі, що файли бази даних і СУБД розташовуються на комп'ютері-сервері. Додатки виконуються на окремих клієнтських комп'ютерах і взаємодіють з сервером по мережі за допомогою клієнтського інтерфейсу.

Для роботи з даними в базі використовується набір команд, які розуміє СУБД – мова (language) СКБД.

СУКД SQL Server підтримує централізовану базу даних з архітектурою *клієнт-сервер.* Технологія клієнт-сервер передбачає, що система поділяється на дві частини:

- *клієнтська частина* (front-end), яка забезпечує графічний інтерфейс і знаходиться на комп'ютері користувача і
- *серверна частина* (back-end), яка знаходиться на спеціально виділених комп'ютерах (серверах) і забезпечує зберігання бази даних, управління даними, поділ інформації, адміністрування і безпеку.

Запит на виконання операції з даними (наприклад, звичайна вибірка), що видається клієнтом (робочою станцією), породжує на сервері пошук та вилучення даних. Витягнуті дані (але не файли) транспортуються по мережі від сервера до клієнта.

Для функціонування мережі використовуються різні протоколи. SQL Server може використовувати більшість мережевих протоколів операційної системи. Для забезпечення мережевої взаємодії використовуються різні мережеві бібліотеки, що передають дані по певним мережевим протоколам. Ці бібліотеки реалізовані у вигляді DLL-файлів, що підключаються до операційної системи. Бібліотека розширює базові можливості протоколу.

У дуже великих системах підтримка баз даних здійснюється групою серверів баз

даних. Сервера можуть об'єднуватися для розподілу навантаження. При цьому кожен сервер управляється індивідуально.

Системні бази даних

При установці SQL Server створюэться 4 системні бази даних:

- master
- msdb
- model
- tempdb

Ці бази дані і будь-які інші створювані бази мыстять в обов'язковому порядку 18 системних таблиць, що зберігають інформацію, що визначає структуру й організацію відповідної бази даних.

Усі бази даних підтримують можливість автоматичного збільшення свого розміру.

База даних master зберігає всю системну інформацію системи SQL Server, включаючи системні облікові записи і системні параметри конфігурації, інформацію про існування всіх інших баз даних і місць розташування їхніх первинних файлів, що містять інформацію ініціалізації для баз даних користувачів. Фактично база даних master складається з двох файлів:

- файлу даних Master.mdf (за замовчуванням 7.5 Мбайт)
- файлу журналу транзакцій Mastlog.ldf (за замовчуванням 1.0 Мбайт)

Обидва файли зберігаються в папці Data.

База даних tempdb. Зберігає всі тимчасові об'єкти, що використовуються при роботі з SQL Server: таблиці, збережені процедури, змінні, курсори і т.д. Зберігаються і системні і користувацькі об'єкти. База створюється заново при кожнім запуску SQL Server. База даних є глобальним ресурсом, доступним усім користувачам.

База даних model. Створення нових баз даних виконується шляхом копіювання цієї системної бази даних. Весь її вміст і всі параметри конфігурації цілком переносяться в нову базу, незалежно від методу створення. За замовчуванням розмір бази 1.5 Мбайт. Якщо при створенні своєї бази даних користувач хоче, щоб його база була точною копією бази model, то при створенні не потрібно вказувати ніяких параметрів, крім імені. Якщо розмір і склад зазначені явно, то скопійована база зміниться відповідним чином.

База даних msdb. Використовується службою SQL Server для планування подій і реєстрації операторів.

3.1.1.2 Архітектура бази даних в SQL Server

Структуру бази даних можна описувати з двох сторін – логічної і фізичної.

Логічна структура бази даних являє собою сукупність логічних об'єктів баз даних, таких як таблиць, списків користувачів, збережених процедур, правил, замовчувань і т. д., а також зв'язків між ними.

Фізична структура включає в себе опис файлів для зберігання баз даних, журналів транзакцій і т.п.

Логічна архітектура баз даних

У всякій СУБД дані логічно організовані в так звані об'єкти. *Об'єкти* являють собою

те, що може "відчути" користувач при роботі з базою даних. Набір об'єктів у різних СУБД може відрізнятися.

При роботі з базами даних в SQL Server 2000 є в розпорядженні користувачів наступні об'єкти:

- *Таблиці (tables)* представляють собою двовимірні матриці, в яких зберігаються дані.
- *Збережені процедури (stored procedures)* – це набори команд Transact-SQL, збережені під певними іменами.
- *Тригери (triggers)* – спеціальний тип збережених процедур, що автоматично запускаються сервером при видаленні, додаванні або зміні даних у таблиці.
- *Представлення (views)* – так звані віртуальні таблиці, що відображають результат вибірки даних. Вони є запитом: SELECT, збережений під певним ім'ям (тому часто представлення називають збереженими запитами). Однак користувач може працювати з доступним через представлення набором даних як зі звичайною таблицею.
- *Індекси (indexes)* – додаткові структури, покликані підвищити продуктивність роботи з даними таблиць за рахунок подання цих даних у впорядкованому вигляді.
- *Користувальницькі типи даних (user-defined data types)* – типи даних, що створюються користувачами на основі вбудованих типів даних SQL Server 2000.
- *Функції користувача (user-defined functions)* – функції, створювані користувачами. Як і збережені процедури, функції складаються з набору команд Transact-SQL, збережених під певним ім'ям. Основним перевагою функцій перед збереженими процедурами є те, що вони можуть використовуватися у виразах.
- *Правила (rules)* – об'єкти бази даних, що дозволяють контролювати логічну цілісність даних. Правило являє собою логічну умову і може бути пов'язано зі стовпцем таблиці або користувальницьким типом даних. При чому одне правило може бути пов'язано з безліччю стовпців різних таблиць та користувацьких типів даних, що забезпечує централізоване управління логікою перевірки значень.
- *Обмеження цілісності (constraints)* – ці об'єкти, як і правила, покликані забезпечувати логічну цілісність даних. Однак область застосування обмежень цілісності набагато ширше, ніж правил. *Одним* з обмежень цілісності є ключі (keys), покликані забезпечити посилальну цілісність даних в пов'язаних таблицях.
- *Умовчання (defaults)* – самостійні об'єкти бази даних, які, так само як і правила, можуть зв'язуватися зі стовпцями таблиць і користувацькими типами даних. Умовчання залишені в SQL Server 2000 для забезпечення сумісності з попередніми версіями. Починаючи з SQL Server 7.0, значення за замовчуванням можуть бути вказані безпосередньо в структурі таблиці або користувальницького типу даних.

а) Таблиці

Вся інформація користувача, що зберігається в базі даних, міститься в об'єктах, званих таблицями (tables). Таблиці являють собою сукупність будь-яких відомостей про об'єкти, явища, закони і т. п. Ніякі інші об'єкти бази даних не можуть зберігати інформацію, але вони можуть звертатися до даних у таблиці.

Таблиці мають наступну структуру.

- *Стовпці (column).* Кожен стовпець представляє собою атрибут або сукупність атрибутів об'єктів (домен атрибутів). *Часто* відносно стовпця використовується термін *поле* із зазначенням імені. Поле є мінімальним елементом таблиці. Кожен стовпець у таблиці має певне ім'я, тип даних і розмір. Імена стовпців повинні бути унікальними в межах таблиці.
- *Рядки (rows).* Кожен рядок (кортеж або запис) являє собою сукупність атрибутів конкретного об'єкта. Рядки таблиці не іменуються. Щоб звернутися до конкретного рядка, користувачеві необхідно вказати якийсь атрибут (або набір атрибутів), що унікально його ідентифікує. Зазвичай в якості такого атрибута виступає первинний ключ. Також можна звертатися більш ніж до одного рядка, вказуючи логічну умову.

Системні таблиці

SQL Server зберігає дані, що визначають конфігурацію сервера і всіх його баз даних, в спеціальному наборі таблиць, відомому як *системні* таблиці (system tables).

Оскільки дані, що зберігаються в системних таблицях, відіграють важливу роль у роботі сервера, необхідно убезпечити їх від видалення або невірної зміни. Тому користувач не має напряму вносити зміни в системні таблиці. Для зміни даних в системних таблицях необхідно використовувати спеціальні системні збережені процедури, що поставляються разом з SQL Server.

Тимчасові таблиці

SQL Server підтримує так звані *тимчасові* таблиці (temporary tables), призначені для тимчасового зберігання інформації. Тимчасову таблицю можна використовувати, наприклад, для тимчасового зберігання проміжних результатів при складних розрахунках. Тимчасові таблиці зберігаються не в поточній базі даних**,** а в системній базі даних Tempdb (навіть якщо при створенні тимчасової таблиці ви явно вкажете яку іншу базу даних, це буде проігноровано і таблиця все одно буде створена в базі даних Tempdb).

У розпорядженні користувачів є два види тимчасових таблиць:

- *Локальні тимчасові таблиці (local temporary tables).* Назви цих таблиць починаються з одного символу #, наприклад #results. Локальні тимчасові таблиці існують до тих пір, поки існує з'єднання з SQL Server, в якому ці таблиці були створені, і автоматично знищуються при його закритті. Локальні тимчасові таблиці видимі тільки для з'єднання, що їх створило. Крім того, якщо локальна тимчасова таблиця була створена в збереженій процедурі (тригері, пакеті команд, користувацькій функції і т. д.), то після виходу з цієї процедури (завершенні тригера, пакета команд) така таблиця також автоматично знищується.

- *Глобальні тимчасові таблиці (global temporary tables).* Доступ до тимчасових таблиць цього типу може бути отриманий з будь-якого з'єднання, встановленого з поточним сервером, і незалежно від того, в якому саме з цих з'єднань була створена глобальна тимчасова таблиця. При цьому також не важливо, який конкретно користувач створив цю тимчасову таблицю і в контексті якої бази даних. Таким чином, одна і та ж збережена процедура, яку запускають різні користувачі, може звертатися до глобальної тимчасової таблиці для отримання інформації. Глобальні тимчасові таблиці можуть бути використані і для обміну даними між різними додатками. Глобальна тимчасова таблиця існує доти, поки вона не буде явно вилучена за допомогою команди DROP TABLE або поки не буде закрито з'єднання, в контексті якого вона була створена. Назви глобальних тимчасових таблиць починаються з символів ##, наприклад ##minutes. Так як глобальна тимчасова таблиця видна з будь-якого з'єднання, її ім'я має бути унікально в межах сервера.

б) Типи даних

Однією з властивостей стовпця є його тип даних. SQL Server має набір вбудованих типів даних, званих системними. На основі деяких з типів даних користувачі можуть створювати свої власні типи даних. Такий тип даних називається користувальницьким типом даних (user-defined data type).

в) Представлення

Представленнями (views) називають віртуальні таблиці, вміст яких визначається запитом. Для кінцевих же користувачів представлення виглядає як таблиця, однак при цьому воно насправді не містить даних, а лише *представляє* дані, розташовані в одній або більше таблицях бази даних. Подібно реальним таблицям представлення містять іменовані стовпці і рядки з даними. Інформація, яку бачить користувач через подання, не зберігається в базі даних як самостійний об'єкт. Зберігається лише код запиту, на основі якого і відбувається формування представлення. Тому в різний час одне і те ж представлення може відображати різні набори даних.

Представлення часто використовуються в таких ситуаціях:

- *Для групування стовпців різних таблиць у вигляді одного об'єкта.*
- *Для обмеження доступу користувачів до певних рядків таблиці.* Наприклад, таблиця може містити відомості про всіх співробітників фірми. Але службовцю дозволяється переглядати дані, що мають відношення тільки до нього самого.
- *Для обмеження доступу користувачів до певних стовпців таблиці.* Наприклад, можна дозволити службовцям переглядати назву відділу, робочий телефон людини, але заборонити перегляд стовпців з розміром платні або з будь-якою іншою конфіденційною інформацією.
- *Для перегляду інформації, що отримується в результаті перетворення даних стовпців.* Наприклад, за допомогою представлень можна переглянути суму значень стовпця, мінімальне або максимальне значення стовпця без відображення всіх даних цього стовпця. Також подання може містити агреговану інформацію, що отримується в результаті складних обчислень.

г) Збережені процедури

Збережені процедури (stored procedure) представляють собою групу команд Transact-SQL, об'єднаних в один РОЗДІЛ. Ця група компілюється і виконується як єдине ціле.

До складу Microsoft SQL Server входить велика кількість вбудованих системних збережених процедур. Всі вони мають префікс sp_ (від system procedure) і охоплюють практично всі аспекти управління і конфігурації сервера, дозволяючи змінювати значення в системних таблицях користувальницьких і системних баз даних.

На відміну від системних збережених процедур, які зберігаються в системній базі даних master, збережені процедури, створені користувачами, зберігаються і виконуються в контексті тієї бази даних, для якої вони були визначені.

д) Тригери

Тригерами (triggers) називається спеціальний клас збережених процедур, що запускаються автоматично при додаванні, зміну або видалення даних з таблиці.

Відповідно, залежно від виконуваних користувачем дій, що призводять до запуску тригера, вони діляться на три категорії:

- тригери зміни (update triggers);
- тригери вставки (insert triggers);
- тригери видалення (delete triggers).

е) Індекси

Індекс (index) – це структура, пов'язана з таблицею або представленням і призначена для оптимізації пошуку інформації в цій таблиці або представленні. Індекс визначається для одного або декількох стовпців, які після надання індексу називаються індексованими стовпцями. Індекс містить відсортовані значення індексованого стовпця (або стовпців) з посиланням на відповідний рядок вихідної таблиці або представлення. Коли необхідно знайти якесь значення в індексованому стовпці, то пошук здійснюється саме в індексі. Підвищення продуктивності досягається за рахунок того, що дані представляються відсортованими.

ж) Обмеження цілісності

Обмеження цілісності (constraints) – це механізм, що забезпечує автоматичний контроль відповідності даних встановленим умовам (обмеженням).

Є п'ять класів обмежень цілісності, що розрізняються по функціональності і області застосування:

- Обмеження цілісності NULL.
- Обмеження цілісності CHECK (перевірка).
- Обмеження цілісності UNIQUE (унікальність).
- Обмеження цілісності PRIMARY KEY (первинний ключ).
- Обмеження цілісності FOREIGN KEY (зовнішній ключ).

з) Функції

SQL Server підтримує два типи функцій.

- *Вбудовані функції (built-in functions).*
- *Функції користувача (user-defined functions).*

3.1.2 Елементи мови Transact-SQL

Алфавіт. Букви латинські, національного алфавіту, цифри, спеціальні символи, зарезервовані слова.

Константи. Цілі, дійсні, символьні (обмежуються одинарними лапками).

Ідентифікатори.

Існує два типи: *стандартні* й *обмежені*. Містять від 1 до 128 символів. Регістри не розрізняються.

Стандартні ідентифікатори можуть складатися із символів латинських і національного алфавіту, цифр, спеціальних символів @ $ _ #. Першим символом може бути символ латинський, національного алфавіту, спеціальний символ _ @ #.

Обмежені ідентифікатори можуть складатися з будь-якого набору символів. Такі ідентифікатори полягають в обмежники (подвійні лапки або квадратні дужки).

Підзапит. Запит, що повертає єдине значення або рядок або таьлицю. Він може бути частиною виразу або запиту. Поміщується в круглі дужки

Оператори

Це знаки операцій. Можуть бути:

- *Унарними*: + - позитивні числа, - негативні числа, ~ - доповнення числа.
- *Арифметичними*: + - додавання, - - вирахування, * - множення, / - ділення, % - узяття остачі. Для строкових значень можна використати оператор + - конкатенація, що поєднує строкові значення. Оператори + й - можна використати для значень типу дата.

  Порівняння: >, <, <=, >=, =, != | <>, !< , !>. Результатом можуть бути значення TRUE або FALSE. Якщо порівняння неможливо, повертається значення NULL.

  Бітові: & - AND, ! - OR, ^ - XOR.

- *Логічні*:

AND	конъюнкція
OR	диз'юнкція
NOT	заперечення
IN	перевірка, чи входить значення в зазначений список. <*вираження*> IN (<*список можливих значень*>) Наприклад: if a in (1,8,5) …
BETWEEN	перевірка, чи лежить значення в зазначеному діапазоні <*значення*> BETWEEN <*початкове значення*> AND <*кінцеве значення*>
EXISTS	повертає TRUE якщо підзапит повертає хоча б один рядок. EXISTS (<*підзапит*>)
LIKE	перевіряє значення на відповідність зазначеному шаблону <*вираження для перевірки*> LIKE <*шаблон*> [ESCAPE <*керуючий символ*>] У шаблоні допускається використання наступних символів-замінників: % замість може бути підставлена будь-яка кількість довільних символів (" %ом "- всі слова, що кінчаються на «ом») _ заміняє один символ рядка (всі трибуквені слова, що кінчаються на «ом») [<*список символів*>] замість символу рядка буде підставлений один з можливих. ("[слд]ом". Відповідає словам «будинок», «сом»., «лом». Можна використати діапазон ("[1-4а-д]ом".) [^<*список символів*>] замість символу рядка будуть підставлені всі символи, крім перерахованих. ("[^слд]ом", "[1-4а-д]ом".) <*керуючий символ*> пропонує сприймати наступний за ним символ як звичайний ("__20#% " ESCAPE "#" змушує символ % розглядати як звичайний)
ALL	виконує порівняння для набору даних. Повертає TRUE якщо умова виконана для всього набору даних <*значення*> <*оператор порівняння*> ALL (<*підзапит*>)

ANY	виконує порівняння для набору даних. Повертає TRUE якщо умова виконана хоча б для одного значення з набору даних *<значення> <оператор порівняння>* ANY (*<підзапит>*)
SOME	Аналог ANY

Змінні

Усе імена починаються із символу @

Перш ніж використати змінні, їх необхідно оголосити. Для цього використається команда

DECLARE <ім'я змінної> <тип> [,...]

Значення змінним можна привласнювати за допомогою команд SET й SELECT.

SET використовується для присвоєння значення однієї змінної. Значенням може бути константа або результат підзапиту.

SET <ім'я змінної> =<значення>

SELECT дозволяє привласнювати змінній значення виразу.

SELECT <ім'я змінної> =<вираз>

Для виведення значень змінних можна використати дві команди SELECT й PRINT.

SELECT виводить результат у стандартний набір рядків, а PRINT виводить інформацію в якості службової.

Типи даних

Тип даних	Опис
Bit	Один біт. У стовпці з цим типом може зберігатися або 0, або 1
binary(n)	Двійкові дані фіксованої довжини до 8000 байт
vabinary(n)	Двійкові дані змінної довжини до 8000 байт
Image	Двійкові дані довжиною до 2 Гбайт. Простір для зберігання даних цього типу відводиться сторінками, розмір яких складає 8 Кбайт
Decimal	Використовується для зберігання дрібних чисел з фіксованою кількістю знаків до і після десяткової точки
Numeric	Є аналогом типу даних Decimal
float(n)	Використовується для зберігання дрібних чисел з приблизною точністю
Real	Є аналогом float(n)
Int	Займає 4 байта і може зберігати цілі числа в діапазоні від -2^{31} до $2^{31}-1$
Smallint	Невелике ціле. Займає 2 байта і може зберігати цілі числа в діапазоні -2^{15} до $2^{15}-1$ (от -32 768 до 32 767)
Tinyint	Маленьке ціле. Займає 1 байт і може зберігати цілі числа в діапазоні від 0 до 255
Bigint	Велике ціле. Займає 8 байт і може зберігати цілі числа в діапазоні від -2^{63} до $2^{63}-1$
char(n)	Строковий тип даних фіксованої довжини без підтримки Unicode довжиною 8000 символів

nchar(n)	Строковий тип даних фіксованої довжини з підтримкою Unicode довжиною до 4000 символів
varchar(n)	Строковий тип даних змінної довжини без підтримки Unicode довжиною до 8000 символів
nvarchar(n)	Строковий тип даних змінної довжини з підтримкою Unicode довжиною 4000 символів
text	Строковий тип даних без підтримки Unicode довжиною до 2 мільярдів символів. Простір для зберігання даних цього типу відводиться сторінкам, розмір яких складає 8 Кбайт
ntext	Строковий тип даних з підтримкою Unicode довжиною до 1 мільярда символів. Простір для зберігання даних цього типу відводиться сторінкам, розмір яких складає 8 Кбайт
datetime	Тип даних для зберігання дати та часу з високою точністю в діапазоні від 1 січня 1753 і до 31 грудня 9999, що займає 8 байт
small datetime	Тип даних для зберігання дати та часу з нормальною точністю в діапазоні з 1 січня 1900 і до 6 червня 2079, що займає 4 байт
money	Використовується для зберігання грошових даних у великому діапазоні. Забезпечує точність до 4 знаків після десяткової точки і займає 8 байт
smallmoney	Використовується для зберігання грошових даних у нормальному діапазоні. Забезпечує точність до 4 знаків після десяткової точки і займає 4 байта
sql variant	Тип даних, що дозволяє зберігати дані декількох типів в одному і тому ж стовпці. Наприклад, при використанні цього типу даних у стовпці можуть одночасно перебувати дані цілочисельні, дробові, символьні і ін.
sysname	Цей тип даних, по суті, є не вбудованим, *а* для користувача (user-defined data type), створюваним в системних базах даних автоматично при установці SQL Server 2000. Використовується для вказівки імен об'єктів. Аналог NVARCHAR (128)
timestamp	Часовий штамп. Значення в стовпці цього типу змінюються SQL Server 2000 автоматично при кожній зміні рядка. Для типу даних мітка також можна використовувати псевдонім RowVersion
table	Цей тип даних призначений для тимчасового зберігання складних наборів даних. Підтримується створення змінних типу таблиці. Крім того, функції користувача можуть повертати значення типу стіл, що надає широкі можливості для розробників. Однак тип даних таблиці не може використовуватися для стовпців таблиці. Він підходить тільки для змінних Transact-SQL, параметрів збережених процедур і для значень, що повертаються функціями користувача
uniqueidentifier	Використовується для зберігання глобальних унікальних ідентифікаторів (GlobalUnique Identifier, GUIO)
cursor	Призначений для зберігання посилань на курсори

3.2 Мова визначення даних

1) Створення бази даних
2) Створення таблиці
3) Зміна структури таблиці
4) Видалення таблиці

3.2.1 Створення бази даних

CREATE DATABASE <ім'я бази даних>
[ON [PRIMARY]
[<специфікація файлу> [,.. „n]]
[, <специфікація групи> [n]]
[LOG ON { <специфікація файлу> [,...n] }]
[FOR ATTACH]

ON – слово означає, що далі треба визначення файлів бази даних.

PRIMARY – далі треба опис первинного файлу бази даних. Якщо він не визначений явно, то в цій якості буде використатися перший файл, зазначений у конструкції <специфікація файлу>. Група файлів, у яку включений первинний файл, називається первинною групою файлів (primary file group) і є групою файлів за замовчуванням (default file group), у неї будуть включені всі файли, для яких явно не зазначена *користувальницька група файлів* (user file group).

LOG ON – означає, що файли журналу транзакцій будуть визначені явно. За замовчуванням сервер автоматично створює файл розміром 25% від загальної суми розмірів файлів даних. Ім'я файлу генерується на основі імені бази даних, але наприкінці до нього додаються символи _Log.

FOR ATTACH – використається, коли необхідно виконати приєднання (attach) існуючої бази даних. Т.е., створення бази даних відбувається тільки на логічному рівні – у системну таблицю sysdatabases бази даних master вносяться відповідні записи, але створення файлів не виконується. Для підключення бази даних досить указати тільки розміщення первинного файлу бази даних. Інформація про місце розташування всіх інших файлів бази даних (вторинних і журналу транзакцій) зберігається в первинному файлі. Однак якщо місце розташування файлів бази даних з моменту її від'єднання змінилося, то необхідно буде вказати повний шлях до кожного файлу бази даних.

<специфікація файлу> – має однаковий формат для всіх типів файлів (первинного, вторинного й журналу транзакцій). Перед первинним - PRIMARY, перед журналом транзакцій - LOG ON, інакше - (указується тільки слово ON) відповідний файл буде вторинним.

Формат опису:
([NAME = <логічне ім'я > ,]
FILENAME = "<повне фізичне ім'я файлу >"
[, SIZE = <первісний розмір файлу>]
[, MAXSIZE = { <max розмір файлу> | UNLIMITED }]
[,FILEGROWTH = <крок приросту файлу>]) [....n]

SIZE Розмір може бути зазначений або в мегабайтах, або в кілобайтах. Для цього використаються приставки Mb й Kb. За замовчуванням уважається - у мегабайтах. Мінімальний розмір файлу становить 512 Кбайт, за замовчуванням буде створений файл розміром 1 Мбайт. Як розмір файлу дозволяється задавати тільки целочисленные значення.

MAXSIZE = { <max розмір файлу> | UNLIMITED }. Автоматичне збільшення файлів | не обмежувати максимальний розмір файлу. За замовчуванням розмір файлу не обмежується, і він буде рости, поки не заповнить весь доступний простір на диску.

<специфікація групи>
Додатково файли даних можуть бути включені в групи.
Формат опису:
FILEGROUP <ім'я групи файлів> <список файлів, що включають,> [...n]

3.2.2 Створення таблиці

CREATE TABLE
[*<ім'я БД>*.] *<ім'я таблиці>*
(< опис стовпчиків >
| *< формула для обчислення >* AS *< ім'я стовпця, що обчислює >*
[*CONSTRAINT <обмеження для таблиці>* **]**
)
Опис колонок
< ім'я стовпчика > <тип >
[[*DEFAULT <значення за замовчуванням>*
] | [IDENTITY [*(<поч. знач.>* **,** *<приріст>*)]] *//для автоінкрементних полів*
] *За замовчуванням <поч. знач.>* **,** *<приріст> рівні 1*
[< *обмеження цілісності для стовпчиків*>]

Обмеження цілісності для стовпчиків
[CONSTRAINT *<ім'я обмеження>*]
[NULL | NOT NULL] дозволяє чи ні зберігання значень NULL
[PRIMARY KEY | UNIQUE] створити індекс

| [[FOREIGN KEY]
REFERENCES *<ім'я табл. для кот. стовпчик буде зовнішнім ключем>* [**(** *список стовпчиків з*

указ. табл. **)**]
[ON DELETE { CASCADE | NO ACTION }] – визначають дії для рядка в створюваній таблиці при видаленні з відповідного рядка з первинного ключа або унікального стовпця головної таблиці. CASCADE - при видаленні рядка з батьківської таблиці рядок у залежній таблиці буде вилучена. NO ACTION - (за замовчуванням) видається повідомлення про помилку й дії по видаленню не виробляються.
[ON UPDATE { CASCADE | NO ACTION }] аналогічно: CASCADE – при модифікації головної таблиці рядок у підлеглій теж буде модифікована. NO ACTION - помилка.
]
[CHECK **(** *<логіч. вираз для обмеження цілісності>* **)]**

Обмеження цілісності для таблиці

[CONSTRAINT <ім'я обмеження>]
[PRIMARY KEY | UNIQUE (< ім'я стовпчика > [ASC | DESC] [,...n])]
| [FOREIGN KEY [< ім'я стовпчика > [,...n])]
REFERENCES <ім'я табл. > [(список стовпчиків з указ. табл.)]
[ON DELETE { CASCADE | NO ACTION }]
[ON UPDATE { CASCADE | NO ACTION }]

]
[CHECK **(** *<логіч. вираз для обмеження цілісності>* **)]**

3.2.3 Створення первинних та вторинних ключів

Обмеження цілісності

Обмеження цілісності (*constraints*) — це механізм, що забезпечує автоматичний контроль відповідності даних встановленим умовам (обмеженням). Наприклад, якщо є дві таблиці, що зберігають логічно зв'язані дані, то можна визначити між ними зв'язок, для того щоб гарантувати, що з головної таблиці не будуть вилучені рядки, з якими зв'язані рядки залежної таблиці. SQL Server забезпечує автоматичне відстеження зв'язків між таблицями, не дозволяючи видаляти дані доти, доки не будуть вилучені всі зв'язані рядки з залежної таблиці.

Існує п'ять класів обмежень цілісності, що розрізняються по функціональності й області застосування:

- *Обмеження цілісності NULL.* Діє для стовпця або користувальницького типу даних. Установлюючи для них обмеження цілісності *NULL* або *NOT NULL* можна відповідно дозволити або заборонити збереження значень *NULL*.
- *Обмеження цілісності PRIMARY KEY (первинний ключ).* Для стовпців, що входять у первинний ключ, автоматично встановлюється обмеження цілісності *UNIQUE*. У стовпцях, що входять у первинний ключ, неможна зберігати значення *NULL*.

```
CREATE TABLE human
(
        human_id int PRIMARY KEY,
        human_full_name char(50),
        passport_number char(15),
        human_address char(50),
          )
```

- *Обмеження цілісності CHECK (перевірка).* Це обмеження цілісності визначається для стовпця й обмежує діапазон значень, що можуть бути збережені в цьому стовпці. При визначенні обмеження цілісності *CHECK* указується логічна умова для даних, що вводяться. При введенні або зміні даних значення, що вводиться, підставляється в умову. Якщо в результаті перевірки повертається значення «*істина*» (*TRUE*), то зміни даних приймаються. Якщо ж перевірка повертає «*неправда*» (*FALSE*), то зміни даних відкидаються і генерується повідомлення про помилку. Для того самого стовпця можна створити кілька обмежень цілісності *CHECK*.

```
CREATE TABLE human
{
        human_id int PRIMARY KEY,
        human_full_name char(50),
        human_avans int,
        CONSTRAINT human_avans
        CHECK (human_avans BETWEEN 0 and 700 )
          )
```

У цьому прикладі сума авансу, що видається одній людині, не може перевищувати 700.

- *Обмеження цілісності UNIQUE (унікальність).* Діє на рівні стовпця і гарантує унікальність значень у цьому стовпці. Завдяки такому обмеженню в таблиці не

може виявитися двох рядків, що мають однакове значення в тому самому стовпці з встановленим обмеженням цілісності *UNIQUE*. На відміну від первинного ключа обмеження цілісності *UNIQUE* допускає збереження значень *NULL*.

- *Обмеження цілісності FOREIGN KEY (зовнішній ключ).* Це обмеження цілісності призначене для організації посилальної цілісності даних. Зовнішній ключ зв'язується з одним з *кандидатів* на первинний ключ в іншій таблиці (з потенційним ключем). Окремим випадком такого кандидата є стовпець (або стовпці) з обмеженням цілісності *PRIMARY KEY*. Однак зовнішній ключ також може посилатися і на стовпець (стовпці) з обмеженням цілісності *UNIQUE*. Таблицю, у якій визначений зовнішній ключ, будемо називати *залежною,* а таблицю з кандидатом на первинний ключ — *головною.* У залежну таблицю не можна вставити рядок, якщо зовнішній ключ не має відповідного значення в головній таблиці. При роботі ж з SQL Server користувачі можуть удатися до каскадної зміни і видалення значень, що входять у зовнішній ключ залежної таблиці. Тобто тепер сервер сам виконує зміну значень у зовнішньому ключі залежної таблиці, коли користувач модифікує дані в первинному ключі головної таблиці.

  Для створення залежної таблиці, зв'язаної з первинним ключем таблиці, описаної в попередньому прикладі, можна використовувати наступну команду:

```
CREATE TABLE human_order
    ( order_nmbr int,
      order_sum money FOREIGN KEY REFERENCES human_id(human_id),
      order_date date )
```

Переглянути всі дані про таблиці:

SP_help 'myTable

Приклад 1.

Є проект бази даних. Перша таблиця містить відомості про швейні матеріали: номер матеріалу, назва, склад матеріалу, кольори, кількість на складі. Дані про назву й кількість є обов'язковими. Кількість матеріалу не може бути негативним.

materials

number	*name*	*composition*	*color*	*quantity*

Друга таблиця містить відомості про швейні вироби: номер виробу, назва, модель, кольори, розмір (розмір з діапазону: L, M, XL, XXL, XXXL). Дані про назву, розмір є обов'язковими.

clothes

num	*name*	*model*	*color*	*size*

Третя таблиця містить відомості про виготовлення швейного виробу: номер виробу, номер матеріалу, операція, витрачене кількість матеріалу. При пошитті необхідно відняти кількість витраченого матеріалу з кількості на складі. При виготовленні необхідно перевірити відповідність кольорів виробу й кольори матеріалу.

manufacture

num clothes	*number mater*	*operation*	*quantity*

Запишіть послідовність інструкцій SQL, за допомогою яких можна створити описану базу даних. У базі даних повинна бути дотримана посилальна цілісність. У таблиці введіть по

одному рядку.

```
create database POSHIV          Створення бази даних
use POSHIV                      Відкриття бази даних
CREATE TABLE materials          Створення таблиць

 (number int primary key,
  name varchar(30) NOT NULL,
  composition varchar(30),
  color varchar(20),
  quantity float NOT NULL,
  check (quantity>=0)
 )

CREATE TABLE clothes
 ( num int PRIMARY KEY,
   name varchar (30) NOT NULL,
   model varchar(40),
   color varchar(20),
   SIZE varchar(10) NOT NULL
  )

CREATE TABLE MANUFACTURE
 ( num_clothes int FOREIGN KEY REFERENCES clothes(num),
   number_mater int FOREIGN KEY REFERENCES materials(number),
   operation char(40),
   quatity float
 )
```

Переглянути всі відомості про таблицю:

sp_help ‘ім’я таблиці’

3.2.4 Зміна структури таблиці

Дозволяє виконувати зміну властивостей існуючих стовпців, видаляти їх або додавати нові, а також управляти обмеженнями цілісності для стовпця або таблиці.

ALTER TABLE *<ім'я таблиці>*
[ALTER COLUMN *<ім'я стовпчика> <новий тип стовпчика>*] *зміна існуючого стовпця*
| ADD *додавання нового стовпця*
[< *опис стовпчика* >] | *<ім'я стовпчика>* AS *<вираження>* *для поля, що обчислюється*
| DROP *видалення обмеження цілісності або стовпця*
[CONSTRAINT] *<ім'я обмеження>* | COLUMN *<ім'я стовпчика>* }

Змінювати властивості існуючого стовпця в деяких випадках неможна, наприклад, якщо він є ключовим, на нього накладені обмеження CHECK, стовпець є що обчислює або має тип text, image.

Приклади:

```
ALTER TABLE doc_exa ADD column_b VARCHAR(20) NULL
ALTER TABLE doc_exb DROP COLUMN column_b
ALTER TABLE doc_exc ADD column_b VARCHAR(20) NULL
  CONSTRAINT exb_unique UNIQUE
```

3.2,5 Видалення таблиці

DROP TABLE *<ім'я таблиці>*

Приклад:

```
DROP TABLE materials
```

3.3 Маніпулювання даними

1) Робота з даними
2) Вибір даних за допомогою оператора SELECT
3) Характеристика розділу FROM
4) Підлеглі запити
5) Групування рядків
6) Агрегатні функції
7) Керуючі структури мови
8) Створення індексів і представлень
9) Розробка процедур і функцій
10) Розробка тригерів

3.3.1 Робота з даними

3.3.1.1 Додавання даних

INSERT додає одну або більше нових рядків даних до існуючої таблиці або подання.

Значення уставляються в стовпці рядка один по одному їхнього проходження, якщо не заданий список стовпців. Якщо список стовпців заданий на множині всіх доступних стовпців, в усі не перераховані стовпці автоматично уставляється або значення за замовчуванням, або *NULL*.

Якщо список стовпців упущений, пропозиція *VALUES* повинне містити значення для всіх стовпців таблиці.

Щоб вставити один рядок даних, повинне бути присутнім пропозиція *VALUES* і містити певний список значень.

Щоб вставити кілька рядків даних, визначите *<select_expr>*, що повертає вже існуючі дані з іншої таблиці. Обрані рядки повинні відповідати списку стовпців.

Формат опису

INSERT
[INTO] *<ім'я таблиці>* | *<ім'я подання>*

[(<*список стовпчиків*>)]
VALUES (<*список значень*>) | <*підзапит*>

<*список стовпчиків*> - список стовпців рядка, у які буде вироблятися вставка даних. Якщо він опущений, то запис буде вироблятися послідовно в усі стовпці, починаючи з першого.

<*список значень*> - може містити константи, вираження, функції, NULL. Якщо <*список стовпчиків*> не зазначений, то кількість значень повинне відповідати кількості стовпчиків таблиці.

<*список значень*> - Список значень для вставки в таблицю або вид. Значення повинні бути в тім же порядку, як цільові стовпці. Як значення можна використати DEFAULT або NULL, тобто уставляється значення за замовчуванням або значення буде відсутній.

< *підзапит* > - SELECT, що повертає нуль або більше рядків, де число стовпців у кожному рядку таке ж, як число елементів, які повинні бути вставлені.

Приклади

```
INSERT materials VALUES (1, 'сукно', 'вовна', 'синій', 20)

INSERT into materials (number, name, composition, color, quantity)
        VALUES (2, 'підкладка', 'віскоза', 'чорний', 10)
```

Вставка декількох рядків:

```
INSERT into materials
VALUES (2, 'верх', 'драп', 'чорний', 10), (2, 'підкладка', 'віскоза', 'синй', 12), (2, 'підкладка', 'віскоза', 'чорний', 10)
```

З підзапитом **INSERT ... SELECT**

Такой синтаксис позволяет внести в таблицу большое количество записей за один раз, причем из разных таблиц.

Следующий пример запишет в таблицу *users_new* все записи из таблицы *users*, в которых поле *country* равно "*USA*".

```
INSERT INTO users_new
    SELECT *
      FROM users
      WHERE country = 'USA'
```

/*Сформувати план виготовлення продукції на вказаний місяць. Формується в залежності від замовлення на продукцію*/

```
INSERT  INTO dbo.PlanMaking
      SELECT  products, month_, SUM(count_products) AS count_products
      FROM        dbo.OrderProducts
      GROUP BY     month_, products
      HAVING      (month_ = 1)
```

Если для таблицы, в которую происходит вставка записей, не указан список полей, то значения для всех полей будут определены на основании результата работы *SELECT*.

Если некоторые поля не определены, то для них будут принято значение по умолчанию.

3.3.1.2 Зміна даних

UPDATE

<ім'я таблиці> | *<ім'я подання>*

SET *визначає список змінюваних рядків або змінних*

<ім'я стовпчика> **=** *<вираз>* | DEFAULT | NULL

| **<***ім'я змінної***>** **=** *< вираз >* *всі зміни виконуються в змінній*

| **<***ім'я змінної***>** = *<ім'я стовпчика>* = *< вираз >* [**,**...*n*] *зміни виконуються в стовпчику й заносяться в змінну. Наприкінці змінна дорівнюватиме останній зміні*

[FROM *< список таблиць >*] *використовується, якщо необхідно врахувати дані з інших таблиць. Можна використати псевдонім і об'єднання (join)*

[WHERE *< умова >*]

Приклади:

1) UPDATE authors
   SET authors.au_fname = 'Annie'
   WHERE au_fname = 'Anne'

2) UPDATE titles
   SET t.ytd_sales = t.ytd_sales + s.qty
   FROM titles t, sales s
   WHERE t.title_id = s.title_id
   AND s.ord_date = (SELECT MAX(sales.ord_date) FROM sales)

3) UPDATE titles
   SET ytd_sales = t.ytd_sales + s.qty
   FROM titles t, sales s
   WHERE t.title_id = s.title_id
   AND s.ord_date = (SELECT MAX(sales.ord_date) FROM sales)

4) UPDATE publishers
   SET city = 'Atlanta', state = 'GA'

5) UPDATE publishers
   SET pub_name = NULL

6) Наступна інструкція змінює стовпці для всіх рядків таблиці:
   UPDATE CITIES

```
   SET POPULATION = POPULATION * 1.03;
```

7) Наступна інструкція використає пропозицію WHERE, щоб обмежити модифікацію стовпців підмножиною рядків:

```
  UPDATE COUNTRY
  SET CURRENCY = "USDollar"
  WHERE COUNTRY = "USA";
```

8) /*Визначити вартість кожної продукції в залежності від вартості складових на поточний час +35% виробничих витрат*/

```
UPDATE  Products
       SET
       Products.price_units=Price.price_prod+0.35*Price.price_prod
FROM   (SELECT Products.artycul  as artycul,
            SUM(Material.price* ManufacturingTechnology.count_material)AS price_prod
        FROM Products INNER JOIN ManufacturingTechnology ON
                Products.kod_Products = ManufacturingTechnology.products INNER JOIN
            Material ON ManufacturingTechnology.material = Material.kod_Material
        GROUP BY  dbo.Products.artycul) as Price
WHERE Products.artycul=Price.artycul
```

3.3.1.3 Видалення даних

Можна видалити одну або кілька записів таблиці.

DELETE

```
    [ TOP ( expression ) [ PERCENT ] ]
   [   FROM]  <ім'я таблиці> | <ім'я подання>
      [   FROM <опис головної таблиці>]
      [ WHERE    < умова >   ]
```

Приклади:

1) Видалення всіх записів з таблиці

```
USE pubs
DELETE authors
```

2) Видалення всіх записів, для яких задовольняється умова

```
USE pubs
DELETE FROM authors
WHERE au_lname = 'McBadden'
```

3) Видалення всіх записів, для яких у головній таблиці поле "заголовок" містить фразу 'computers'.

а) З використанням підзапита

```
USE pubs
DELETE FROM titleauthor
WHERE title_id IN
```

```
   (SELECT title_id
   FROM titles
   WHERE title LIKE '%computers%')
```

б) Або з використанням зв'язування таблиць

```
USE pubs
DELETE titleauthor
FROM titleauthor INNER JOIN titles
   ON titleauthor.title_id = titles.title_id
WHERE titles.title LIKE '%computers%'
```

4) Видалення даних із частини таблиці. У цьому випадку видаляються перші 10 записів

```
    DELETE authors
FROM (SELECT TOP 10 * FROM authors) AS t1
WHERE authors.au_id = t1.au_id
```

3.3.2 Вибірка даних

Вибір даних здійснюється за допомогою інструкції SELECT. Інструкція повертає дані з таблиці, представлення або збереженої процедури. Різні інструкції SELECT виконують наступні дії:

- Повертають одиночну стоку або частину рядка з таблиці.
- Безпосередньо повертають список рядків або список часткових рядків з таблиці.
- Повертають зв'язані рядки або часткові рядки з об'єднання двох або більше таблиць.
- Повертають всі рядки або часткові рядки з об'єднання двох або більше таблиць.

Формат

SELECT *<список стовпців, які мають бути присутніми в результаті вибірки>*
[INTO *<ім'я нової таблиці>*]
FROM *<список вихідних таблиць>*
[WHERE *<умова вибірки>*]
[GROUP BY *<вираження для групи>*]
[HAVING *<умова для групи>*]
[ORDER BY *<вираження для сортування>* [ASC | DESC]]

Розділ SELECT

Розділ SELECT управляє списком стовпців, які необхідно включити в результат. Формат опису:

SELECT [ALL | DISTINCT]
[TOP *<n>* | TOP *<n>* PERCENT [WITH TIES]]
< select_list >

ALL – значення за замовчуванням. В результат запиту дозволено включати дублюючі рядки

DISTINCT – заборонено включати дублюючі рядки

TOP – дозволяє вибирати не всі рядки, а тільки *<n>* перших.

PERCENT – означає, що треба вибирати не кількість рядків, а зазначений відсоток рядків від усієї кількості рядків

WITH TIES – будуть виведені рядки, що збігаються з останніми виведеними рядками, визначеними попереднім TOP *<n>* PERCEN

У запиті можуть використовуватися дані з різних таблиць і різних баз даних. У цьому випадку необхідно вказувати повне ім'я стовпця

<ім'я БД>.<ім'я таблиці> | <аліас таблиці>.<ім'я стовпця> |

* означає всі стовпці таблиці

Можна вказувати нове ім'я стовпця (псевдонім):

<ім'я стовпця в таблиці> [AS] <ім'я стовпця в запиті(аліас)>

Ключове слово AS не обов'язкове.

Можна створювати стовпці, що обчислюються. У цьому випадку замість <ім'я стовпця в таблиці> указується вираз, за яким обчислюється поле:

<вираз> [[AS] <ім'я стовпця в запиті>] або
<ім'я стовпця в запиті> = <вираз>

Якщо не вказати аліас, то в результаті стовпець буде без імені.

Наприклад, при наявності загальної кількості товару і його вартості, можна одержати стовпець із загальною вартістю

Кількість*ціна AS Разом або
Разом = Кількість*ціна

Приклади:

1. Наступна інструкція використовує шаблон *, щоб вибрати всі стовпці й рядки з таблиці:

```
SELECT * FROM COUNTRIES
```

2. Наступна інструкція вибирає зазначеня стовпці з таблиці:

```
SELECT JOB_GRADE, JOB_CODE, JOB_COUNTRY, MAX_SALARY
  FROM PROJECT
```

Розділ INTO

Розділ **INTO** – за замовчуванням результат вибірки виводиться для перегляду у вигляді таблиці. Фраза використовується, якщо необхідно зберегти результат у таблиці. Ім'я створюваної таблиці повинне бути унікальним у межах БД.

Розділ WHERE

Призначений для накладення фільтру на дані, оброблювані запитом. Якщо *<умова вибірки>* є істина, то рядок включається в запит.

Приклади:

1. Наступна інструкція зазначені стовпці з таблиці, які задовольняють умовам пошуку визначеним у пропозиції WHERE:

```
SELECT CAPITAL, POPULATION
  FROM COUNTRY
  WHERE POPULATION > 5000000
```

2. Наступна інструкція встановлює псевдонім таблиці в пропозиції SELECT і використовує його для ідентифікації стовпців у пропозиції WHERE:

```
SELECT C.CITY
```

```
FROM CITIES C
WHERE C.POPULATION < 1000000
```

```
select * from dbo.PlanMaking (nolock)  /* переглянути вміст таблиці з lock(заблокованими
                                                                    записами)*/
dbcc opentran  /* Перевірити чи є не завершені транзакції*/
```

3.3.3 Вибірка даних з різних таблиць

Розділ FROM

Розділ **FROM** – у розділі визначаються джерела, з якими буде працювати запит.

Формат опису
<список вихідних таблиць>
<ім'я таблиці> [[AS] *<аліас таблиці>*]
| *<ім'я подання>* [[AS] *<аліас>*]
| < *підключення таблиць* > – використовується, якщо потрібно використати зв'язані таблиці

Формат опису < *підключення таблиць* >
< *вихідна таблиця* > < *тип зв'язування* > < *підлегла таблиця* > ON < *умова зв'язування* >
| < *вихідна таблиця* > CROSS JOIN < *підлегла таблиця* >
< *вихідна таблиця* > називається лівою,
< *підлегла таблиця* > називається правою

Формат опису < *тип зв'язування* >
[INNER | LEFT | RIGHT | FULL] JOIN

INNER – використовується за замовчуванням. Вибираються пари рядків, для яких є рядки, що задовольняють критерію зв'язування в обох таблицях. Наприклад, маємо таблиці:

Gruppa

Kod_gr	Name
1	ПМ
2	ТК
3	ТХ

Student

Kod_st	FIO	Kod_gr
1	Іванов	1
2	Петров	1
3	Сидоров	2
4	Коропів	2
5	Самсонов	4

```
SELECT Gruppa.Name, Student.FIO
  FROM Gruppa INNER JOIN Student ON Gruppa.Kod_gr=Student.Kod_gr
```

Результат запиту:

ПМ	Іванов
ПМ	Петров
ТК	Сидоров
ТК	Коропів

LEFT – включаються всі рядки лівої таблиці, навіть якщо їм немає відповідного запису в правій.

```
SELECT Gruppa.Name, Student.FIO
FROM Gruppa LEFT JOIN Student ON Gruppa.Kod_gr=Student.Kod_gr
Результат запиту:
    ПМ      Іванов
     ПМ      Петров
     ТК      Сидоров
     ТК      Коропів
     ТХ      NULL
```

RIGHT – включаються всі рядки лівої таблиці, навіть якщо їм немає відповідного запису в правій.

```
SELECT Gruppa.Name, Student.FIO
 FROM Gruppa RIGHT JOIN Student ON Gruppa.Kod_gr=Student.Kod_gr
Результат запиту
    ПМ      Іванов
     ПМ      Петров
     ТК      Сидоров
     ТК      Коропів
   NULL    Самсонов
```

FULL – включаються всі рядки лівої таблиці й всі рядки правої.

```
SELECT Gruppa.Name, Student.FIO
  FROM Gruppa FULL JOIN Student ON Gruppa.Kod_gr=Student.Kod_gr
Результат запиту:
    ПМ      Іванов
     ПМ      Петров
     ТК      Сидоров
     ТК      Коропів
      ТХ      NULL
   NULL    Самсонов
```

3.3.4 Підлеглі запити

Підлеглим називається запит, що може міститись в розділах SELECT, FROM, WHERE та HAVING іншої інструкції SQL. Підзапити дозволяють природнім чином обробляти запити, що виражаються через результати інших запитів. Зовнішня інструкція SELECT використовує результат виконання внутрішньої інструкції для визначення вмісту кінцевого результату всього запиту.

Підзапит являє собою інструмент створення тимчасової таблиці, вміст якої використовує зовнішній запит. Підзапит можна вказувати безпосередньо після операторів порівняння (=, <, >, <=, >=, <>) в конструкції WHERE або HAVING.

Текст підзапиту має бути обмежений круглими дужками.

Існують три типи підзапитів.

- **Скалярний** під запит повертає єдине значення. Скалярний підзапит може бути використаним скрізь, де необхідно вказати одне значення.
- **Рядковий** підзапит повертає значення одного стовпця таблиці у вигляді множини рядків. Рядковий підзапит може бути використаним скрізь, де

застосовується конструктор рядкових значень.

- **Табличний** підзапит повертає значення одного чи декількох стовпців таблиці, що розміщаються в більш, ніж одному рядку. Табличний підзапит можна використовувати скрізь, де допускається вказувати таблицю.

До підзапитів застосовуються наступні правила та обмеження:

- В підзапиті не може бути використана конструкція впорядкування результатів ORDER BY, хоча вона може бути присутньою в зовнішній інструкції SELECT.
- Список вибірки SELECT підзапиту має складатися з імен окремих стовпців або складених з них виразів, за виключенням випадку, коли в підзапиті використовується ключове слово EXISTS.
- За умовчанням імена стовпців в підзапиті відносяться до таблиці, ім'я якої вказане в конструкції FROM підзапиту. Однак можна посилатися й на стовпці таблиці, вказаної в конструкції FROM зовнішнього запиту, для чого використовуються уточненні імена стовпців.

Приклади

. Використання підзапиту з перевіркою на рівність.

Отримати імена та прізвища співробітників, що працюють в тій самій країні, що й Ann Bennet.

```
SELECT first_name, last_name, job_country
FROM employee
WHERE job_country = (SELECT job_country FROM employee
                     WHERE first_name = 'Ann' AND last_name = 'Bennet')
```

Таблиці:

Товар(ID_товар,назва,сорт,кількість,ціна,відділ)
Продаж(ID, товар,кількість)
Відділ(ID_отдел,назва)

Перелік товарів, що не продавались

```
select назва from Товар
where not exists ( select Продаж.товар from Продаж where
Продаж.товар=Товар.ID_товар)
```

Перелік відділів в яких не було продажів

```
select назва from Відділ where not exists
 ( select товар from Продаж join Товар on Продаж.товар=Товар.ID_товар
where Товар.відділ = Відділ.ID_відділ)
```

Перелік назв товарів, які мають ту ж ціну, що і морква першого сорту

```
select назва, ціна
from Товар
where ціна= ( select ціна from Товар where сорт=1 and назва='морква')
```

3.3.5 Розділ GROUP BY

Дозволяє виконати групування рядків таблиць за певними критеріями.
Для кожної групи можна виконати функції агрегування (sum, count).

Часто в запитах потрібно формувати проміжні підсумки , що зазвичай відображається появою в запиті фрази " для кожного ..." . Для цієї мети в операторі SELECT використовується розділ GROUP BY. Запит, в якому присутня GROUP BY, називається запитом з групуванням, оскільки в ньому групуються дані, отримані в результаті виконання операції SELECT, після чого для кожної окремої групи створюється єдиний сумарний рядок.

За наявності в операторі SELECT фрази GROUP BY кожен елемент списку в розділі SELECT повинен мати єдине значення для всієї групи. Розділ SELECT може включати тільки такі типи елементів: імена полів, агрегатні функції, константи і вирази, що включають комбінації перерахованих вище елементів. Всі імена полів, наведені в розділі SELECT, повинні бути присутні і у фразі GROUP BY - за винятком випадків, коли ім'я стовпця використовується у агрегатній функції.

Якщо спільно з GROUP BY використовується умова WHERE , то вона обробляється першою, а групуванню піддаються тільки ті рядки, які задовольняють умові пошуку. Стандартом SQL визначено, що при проведенні групування усі відсутні значення розглядаються як рівні. Якщо два рядки таблиці в одному і тому ж стовпці, що групується, містять значення NULL і ідентичні значення у всіх інших непорожніх стовпцях, що групується, то вони поміщаються в одну і ту ж групу .

Формат

GROUP BY [ALL] *<умова угруповання>*

<умова угруповання> – звичайно ім'я стовпця, але можна й вираз з використанням імені стовпця.

На розділ SELECT накладаються обмеження: у безпосередньому виді дозволяється зазначення тільки імен стовпців, перерахованих у розділі GROUP BY, тобто тих стовпців, за якими відбувається групування.

ALL - якщо група не містить жодного запису, то будуть виводитися всі групи, але з порожніми значеннями (0, NULL). Така ситуація можлива при використанні розділу WHERE у запиті.

Розділ HAVING

Використовується тільки при наявності розділу GROUP BY, аналогічно розділу WHERE, тобто вказуються умови вертикальної фільтрації.

За допомогою HAVING відображаються всі попередньо згруповані за допомогою GROUP BY блоки даних, що задовольняють заданим в HAVING умовам. Це додаткова можливість "профільтрувати" вихідний набір. Умови в HAVING відрізняються від умов в WHERE:

- HAVING виключає з результуючого набору даних групи з результатами агрегованих значень;
- WHERE виключає з розрахунку агрегатних значень по угрупованню запису, що не задовольняють умові;
- в умові пошуку WHERE не можна задавати агрегатні функції.

Формат

HAVING <умова>

3.3.6 Агрегатні функції

Агрегатні функції повертають одинарне значення, обчислене зі значень в стовпці.

COUNT()	повертає кількість рядків у зазначеному стовпці з непустим значенням
COUNT(*)	повертає загальну кількість рядків, включаючи порожні
SUM ()	повертає суму значень у зазначеному стовпці
AVG()	повертає середнє значення в зазначеному стовпці
MIN ()	повертає мінімальне значення в зазначеному стовпці
MAX()	повертає максимальне значення в зазначеному стовпці

Аргументом функції є ім'я стовпця. Функція повертає єдине значення. Функції COUNT, MIN, MAX застосовні як до числових, так і до нечислових значень, тоді як функції SUM, AVG можуть бути використані тільки у випадку числових полів. За винятком COUNT(*), при обчисленні результатів будь-яких функцій спочатку виключаються всі порожні значення, після чого необхідна операція застосовується тільки до непустих значень, що залишилися, стовпця. Варіант COUNT(*) є особливим випадком використання функції COUNT – його призначення полягає в підрахунку всіх рядків у таблиці, незалежно від того, утримуються там порожні, повторювані або будь-які інші значення.

Якщо до застосування функції, що групує, необхідно виключити повторювані значення, треба перед ім'ям стовпця у визначенні функції помістити ключове слово DISTINCT. Ключове слово DISTINCT не має сенсу для функцій MIN і MAX. Однак його використання може впливати на результати виконання функцій SUM й AVG.

Функції, що групують, можуть бути використані тільки в списку вибірки SELECT й у пропозиції HAVING. У всіх інших випадках застосування цих функцій приведе до помилки. Якщо список вибірки SELECT містить функцію, що групує, та імена стовпців, то в тексті запиту повинна бути присутньою пропозиція GROUP BY, що забезпечує об'єднання даних у групи.

Приклади

1. Обчислити середній об’єм покупок, здійснених кожним покупцем.

```
SELECT Клиент.Фамилия, Avg(Сделка.Количество)
   AS Среднее_количество
FROM Клиент INNER JOIN Сделка
   ON Клиент.КодКлиента=Сделка.КодКлиента
GROUP BY Клиент.Фамилия
```

Фраза "кожним покупцем" знайшла своє відображення в SQL-запиті у вигляді пропозиції GROUP BY Клиент.Фамилия.

2. Визначити, на яку суму було продано товару кожного найменування.

```
SELECT Товар.Название,
   Sum(Товар.Цена*Сделка.Количество)
   AS Стоимость
FROM Товар INNER JOIN Сделка
   ON Товар.КодТовара=Сделка.КодТовара
GROUP BY Товар.Название
```

3. Підрахувати кількість угод, які здійснила кожна фірма.

```
SELECT Клиент.Фирма, Count(Сделка.КодСделки)
   AS Количество_сделок
FROM Клиент INNER JOIN Сделка
   ON Клиент.КодКлиента=Сделка.КодКлиента
GROUP BY Клиент.Фирма
```

4. Підрахувати загальну кількість купленого для кожної фірми товару і його вартість.

```
SELECT Клиент.Фирма, Sum(Сделка.Количество)
   AS Общее_Количество,
   Sum(Товар.Цена*Сделка.Количество)
   AS Стоимость
FROM Товар INNER JOIN
   (Клиент INNER JOIN Сделка
   ON Клиент.КодКлиента=Сделка.КодКлиента)
   ON Товар.КодТовара=Сделка.КодТовара
GROUP BY Клиент.Фирма
```

5. Визначити сумарну вартість кожного товару за кожен місяць.

```
SELECT Товар.Название, Month(Сделка.Дата)
   AS Месяц,
   Sum(Товар.Цена*Сделка.Количество)
   AS Стоимость
FROM Товар INNER JOIN Сделка
   ON Товар.КодТовара=Сделка.КодТовара
GROUP BY Товар.Название, Month(Сделка.Дата)
```

6. Визначити сумарну вартість кожного товару першого сорту за кожен місяць.

```
SELECT Товар.Название, Month(Сделка.Дата)
   AS Месяц,
   Sum(Товар.Цена*Сделка.Количество)
   AS Стоимость
FROM Товар INNER JOIN Сделка
   ON Товар.КодТовара=Сделка.КодТовара
WHERE Товар.Сорт="Первый"
GROUP BY Товар.Название, Month(Сделка.Дата)
```

7. Визначити фірми, у яких загальна кількість угод перебільшила три.

```
SELECT Клиент.Фирма, Count(Сделка.Количество)AS Количество_сделок
FROM Клиент INNER JOIN Сделка
   ON Клиент.КодКлиента=Сделка.КодКлиента
GROUP BY Клиент.Фирма
HAVING Count(Сделка.Количество)>3
```

8. Вивести список товарів, проданих на суму більше 10 000 грн.

```
SELECT Товар.Название,
   Sum(Товар.Цена*Сделка.Количество) AS Стоимость
FROM Товар INNER JOIN Сделка
   ON Товар.КодТовара=Сделка.КодТовара
```

```
GROUP BY Товар.Название
HAVING Sum(Товар.Цена*Сделка.Количество)>10000
```

9. Вивести список товарів, проданих на суму більше 10 000 грн. без указання суми.

```
SELECT Товар.Название
FROM Товар INNER JOIN Сделка
   ON Товар.КодТовара=Сделка.КодТовара
GROUP BY Товар.Название
HAVING Sum(Товар.Цена*Сделка.Количество)>10000
```

10. Наприклад, маємо таблицю

Товар

Назва	Ціна	Тип	Кількість
Туфлі	400	Взуття	10
Чоботи	700	Взуття	25
Пальто	1000	Одяг	100
Плащ	450	Одяг	50

Треба підрахувати, який розмір виручки було отримано за товар кожного типу.

```
SELECT Тип, Sum(ціна*кількість)
  FROM Товар
  GROUP BY Тип
```

Результат запиту

Тип	
Взуття	5750
Одяг	122500

Треба підрахувати, який розмір виручки було отримано за товар 'Взуття'

```
SELECT Тип, Sum(ціна*кількість)
  FROM Товар
  WHERE Тип='Взуття'
  GROUP BY All Тип
```

Результат запиту

Тип	
Взуття	5750
Одяг	NULL

Використання підзапитів з агрегуючими функціями

Приклад 1.

Отримати список всіх співробітників в США, що мають зарплатню більше середньої, вказавши, на скільки їх зарплатня перебільшує середню зарплатню співробітників у США.

```
SELECT first_name, last_name, salary,
salary-(SELECT AVG(salary) FROM employee WHERE job_country='USA') AS salary_diff
FROM employee
WHERE salary>(SELECT AVG(salary) FROM employee WHERE job_country =
'USA')
```

```
        AND job_country = 'USA'
```

Приклад 2.: Вибрати продавця у якого найвище співвідношення комісійні/зарплата.

```
SELECT first_name, last_name, commission/salary AS rate
  FROM employee
  WHERE commission IS NOT NULL
  AND commission/salary = (SELECT MAX(commission/salary)
        FROM employee
        WHERE commission IS NOT NULL)
```

Цей результат можна отримати інакше

```
SELECT TOP 1 WITH TIES first_name, last_name, commission/salary as rate
  FROM employee
  WHERE commission IS NOT NULL
  ORDER BY commission/salary DESC
```

Приклад 3.: Вибрати середньорічну суму замовлень покупця «REBOUND SPORTS».

```
SELECT s/ny AS average
  FROM
    (SELECT SUM(total) AS s
      FROM customer JOIN sales_order
      ON customer.customer_id = sales_order.customer_id
      WHERE name = 'REBOUND SPORTS') t1,
    (SELECT COUNT(DISTINCT YEAR(order_date)) AS ny
      FROM customer JOIN sales_order
      ON customer.customer_id = sales_order.customer_id
      WHERE name = 'REBOUND SPORTS') t2
```

Перелік товарів, яких найбільше

```
select T.назва,max(P.кількість)
from Товар T,
   (select назва,sum(кількість)as кількість from Товар GROUP BY назва) as P
```

Для таблиці Товар(ID_товар,назва,сорт,кількість,ціна,відділ)

Виведення даних у вказаному форматі

Для виведення вартості

convert(varchar, <ім.`я поля, що має формат money>, 1)

Для виведення дати:

convert(varchar, <ім.`я поля, що має формат daretime і містить дату>, 104)

Для виведення часу:

convert(char(5), <ім.`я поля, що має формат daretime і містить час>,108)

-- Получить дату в нормальном формате: "день.месяц.год"

```
select convert(varchar,getdate(),104) -- Только дата
, convert(varchar,getdate(),104) +' '+ convert(char(5),getdate(),108) -- Дата + часы + минуты
```

, convert(varchar,getdate(),104) +' '+ convert(varchar,getdate(),108) -- Дата + часы + минуты + секунды
, convert(varchar,getdate(),104) +' '+ stuff(convert(char(12),getdate(),114), 9, 1, '.') -- С миллисекундами

-- Вывести данные с date между 10:30 предыдущего дня и 10:30 текущего (c) iap

```
SELECT email,price,date
FROM br_reports_dc
WHERE [date]>=DATEADD(DAY,-1, CONVERT(CHAR(8), getdate(),112)+' 10:30')
AND [date]<CONVERT(CHAR(8), getdate(),112)+' 10:30'
```

-- Изъять время из даты

```
select convert(char(8),getdate(),114) [Часы минуты секунды],
convert(char(12),getdate(),114) [С миллисекундами, но они почему-то разделены двоеточием],
stuff(convert(char(12),getdate(),114), 9, 1, '.') [С миллисекундами, разделитель "точка"]
```

-- Округлить дату

```
select dateadd(dd, datediff(dd,0, getdate()), 0)
```

-- Получить дату без миллисекунд, сохраняется тип DateTime

```
select dateadd(ms,-datepart(ms, getdate()), getdate());
```

-- Получить дату без секунд и миллисекунд, сохраняется тип DateTime

```
select dateadd(mi, datepart(mi, getdate()), dateadd(hh, datediff(hh, 0, getdate()), 0))
```

-- Получить дату + часы (без минут и секунд), сохраняется тип DateTime

```
select dateadd(hh, datepart(hh, getdate()), dateadd(dd, datediff(dd, 0, getdate()), 0))
```

3.3.7 Керуючі конструкції мови

Складений оператор

BEGIN … END **-** операторні дужки, використаються для обрамлення складеного оператора.

Умовний оператор

```
IF <умова>
 <оператори>
ELSE
    <оператори>
```

Оператор вибору

```
CASE  <вхідне значення >
   WHEN  <варіант значень 1>   THEN < значення, що повертається>
```

```
        . . .
    WHEN  <варіант значень n>   THEN < значення, що повертається >
    [ELSE < значення, що повертається >]
END
```

Вибір ненульового значення

COALESCE (<список значень>)

Повертається перше значення з перерахованих через кому, не рівне NULL.

Організація циклів

WHILE <умова>
<блок команд>

У тілі циклу можна використати команди BREAK й CONTINUE.
BREAK – примусове завершення циклу
CONTINUE – примусове повернення до початку циклу

Приклади:

Вивести відтінок кольору в залежності від кольору

```
select distinct color,
    case when color='синий' or color='черный' then 'темный'
        when color='бежевый' or color='розовый' then 'светлый'
        else 'неопределенный'
    end
from clothes
```

Визначити виконання плану

```
if (select sum(quantity) from manufacture)>10
   select 'план виконано' as result
else
   select 'план не виконано' as result
```

Визначити день тижня

```
SELECT
 CASE DATEDIFF(DAY,0, GETDATE())%7
  WHEN 2 THEN 'Понедельник'
  WHEN 3 THEN 'Вторник'
  WHEN 4 THEN 'Среда'
  WHEN 5 THEN 'Четверг'
  WHEN 6 THEN 'Пятница'
  WHEN 7 THEN 'Суббота'
  WHEN 1 THEN 'Воскресенье'
 END [День недели]
```

Для товару 1,2,3 сорту вивести відповідне сповіщення

```
SELECT distinct назва, стан=
   CASE сорт
        WHEN 1 THEN ‘вищий’
        WHEN E 2 THEN ‘середній’
```

```
        WHEN E 3 THEN ‘низький’
    END
FROM Товар
```

Для таблиці Студент вивести для кожного студентарозмір стипендії

```
SELECT ПІБ,стипендія=
        if сер_бал>=4 and сер_бал<5 750 else  if сер_бал=5  500
FROM Студент
```

Якщо кількості товару більше 100, то сповістити "Товару достатньо", якщо менше 20, то "Критична маса"

```
SELECT P.назва,P.кількість,сповіщення=
        CASE
          WHERE P.кількість<20 “Критична маса. Завезти”
          WHERE P.кількістью>100 “Товару достатньо”
        END
FROM (SELECT назва, sum(кількість) as кількість
        FROM Товар
        GROUP BY назва
      ) as P
ORDER BY P.назва
```

3.3.9 Створення індексів і представлень

3.3.8.1 Представлення

Представленнями (views) називають віртуальні таблиці, вміст яких визначається запитом. Для кінцевих же користувачів представлення виглядає як таблиця, однак при цьому воно в дійсності не містить даних, а лише *представляє* дані, розташовані в одній або більш таблицях бази даних. Подібно реальним таблицям представлення містять іменовані стовпці і рядки з даними. Інформація, що бачить користувач через представлення, не зберігається в базі даних як самостійний об'єкт. Зберігається лише код запиту, на основі якого і відбувається формування представлення. Тому в різний час те саме представлення може відображати різні набори даних.

Фізично представлення реалізоване у виді запиту SELECT, на основі якого проводиться вибірка даних з однієї або декількох таблиць або представлень. Найпростіше представлення, яке створене на основі однієї таблиці і не має фільтрів, містить точно такий же набір даних, як і вихідна таблиця. Вибірка даних із представлення практично нічим не відрізняється від вибірки даних зі звичайних таблиць. Однак виконання операцій зміни, видалення і вставки даних для представлень має визначену специфіку.

Представлення часто використовуються в наступних ситуаціях:

- *Для групування стовпців різних таблиць у виді одного об'єкта.*
- *Для обмеження доступу користувачів до визначених рядків таблиці.* Наприклад, таблиця може містити зведення про всіх співробітників фірми. Але службовцеві дозволяється переглядати дані, що мають відношення тільки до нього самого.
- *Для обмеження доступу користувачів до визначених стовпців таблиці.* Наприклад, можна дозволити службовцем переглядати назву відділу, робочий телефон людини, але заборонити перегляд стовпців з розміром жалування або будь-якою іншою конфіденційною інформацією.
- *Для перегляду інформації, що виходить у результаті перетворення даних стовпців.*

Наприклад, за допомогою представлень можна переглянути суму значень стовпця, мінімальне або максимальне значення стовпця без відображення всіх даних цього стовпця. Також представлення може містити агреговану інформацію, одержувану в результаті складних обчислень.

Створення представлень

Представлення дозволяється створювати тільки в поточній базі даних. Необхідно, щоб ім'я представлення відповідало правилам для ідентифікаторів і було унікальним для кожного користувача. Ім'я представлення не повинне збігатися з ім'ям кожної з таблиць, що належать цьому користувачеві. В основі представлення можуть лежати інші представлення й процедури, які, у свою чергу, теж можуть посилатися на представлення.

При створенні представлень діють наступні обмеження:

- з представленнями не можна зв'язувати правила й визначення DEFAULT;
- до представлення дозволено прив'язати тригери INSTEAD OF, але не AFTER;
- у запитах, що визначають представлення, не повинно бути конструкцій ORDER BY, COMPUTE, COMPUTE BY і ключових слів INTO;
- на представленнях не можна створювати повнотекстові індекси;
- заборонено створення тимчасових представлень, як і представлень на основі тимчасових таблиць;
- не можна видаляти таблиці або представлення, задіяні в представленні, створеному з використанням конструкції SCHEMABINDING, поки це представлення не буде вилучене або не перестане бути прив'язаним до схеми (у результаті зміни). Крім того, заборонене виконання оператора ALTER TABLE для таблиці, задіяної в представленні, що прив'язано до схеми;
- не можна виконувати повнотекстові запити до представлення. Однак визначення представлення може включати повнотекстовий запит, якщо він посилається на таблицю, настроєну для повнотекстового індексування.

SQL Server привласнює стовпцям представлення імена й типи даних стовпців, заданих у запиті, що визначає представлення. Список вибору визначального представлення оператора – це повний або частковий перелік імен стовпців базових таблиць.

Необхідно явно задати ім'я кожного стовпця представлення, якщо хоч один з них формується на основі арифметичного виразу, вбудованої функції, константи, а також якщо є загроза, що кілька стовпців одержать однакові імена.

Стовпець представлення успадковує тип даних стовпця, похідним якого він є, незалежно від того, чи перейменований він чи ні. Це обмеження не діє, якщо представлення створене на основі запиту із зовнішнім з'єднанням, оскільки в цьому випадку стовпці, що не допускають порожніх значень, можуть втратити цю властивість.

За замовчуванням, якщо в результаті додавання або відновлення даних через представлення рядки перестали відповідати критерію визначального представленню запиту, вони виходять зі сфери видимості представлення. Наприклад, можна визначити представлення, що витягає всі рядки з відомостями про працівників, річна зарплата яких не перевищує $30000. Якщо зарплата підвищується до $32 000, то при наступних запитах представлення не буде виводити рядок для нього, оскільки значення не відповідає встановленому представленням критерію.

Визначення представлення дозволяється зашифрувати, однак після цього його не зможе одержати ніхто, навіть сам власник представлення.

Створення стандартних представлень

Існують два способи створення представлення: за допомогою *Enterprise Manager*

або мовою Transact-*SQL* засобами оператора **CREATE VIEW**.

CREATE VIEW [< *database_name* > **.**] [< *owner* > **.**] *view_name* [**(** *column* [**,**...*n*] **)**]
[WITH < view_attribute > [**,**...n]]
AS
select_statement
[WITH CHECK OPTION]

< view_attribute > ::=
{ ENCRYPTION | SCHEMABINDING | VIEW_METADATA }

Приклад:

```
USE Northwind
GO
CREATE VIEW Customer-Orders
AS
SELECT o.Orderl, c.CompanyName, c.ContactName
FROM Orders про JOIN Customers з ON o.Custonerl = c.Customer-ID
```

Модифікація представлень

Можна змінити ім'я представлення або модифікувати його визначення, не прибігаючи до видалення й повторного створення представлення (що спричиняє втрату пов'язаних з представленням прав доступу). При перейменуванні представлення варто враховувати наступне:

- представлення, яке варто перейменувати, повинне розташовуватися в поточній базі даних;
- нове ім'я повинне відповідати правилам для ідентифікаторів;
- тільки власник має право перейменовувати представлення;
- власник БД може змінювати ім'я представлення, що належить будь-якому користувачеві.

Зміна представлення не робить впливу на залежні від нього об'єкти (наприклад, на збережені процедури або тригери), якщо тільки при зміні визначення представлення залежний об'єкт не стає недійсним.

ALTER VIEW [< *database_name* > **.**] [< *owner* > **.**] *view_name* [**(** *column* [**,**...*n*] **)**]
[WITH < view_attribute > [**,**...n]]
AS
select_statement
[WITH CHECK OPTION]

< view_attribute > ::=
{ ENCRYPTION | SCHEMABINDING | VIEW_METADATA }

CHECK OPTION

Обеспечивает соответствие всех выполняемых для представления инструкций модификации данных критериям, заданным при помощи аргумента*select_statement*. Если строка изменяется посредством представления, предложение WITH CHECK OPTION гарантирует, что после фиксации изменений доступ к данным из представления сохранится.

ENCRYPTION

Выполняет шифрование элементов представления sys.syscomments, содержащего текст инструкции CREATE VIEW. Использование предложения WITH ENCRYPTION предотвращает публикацию представления в рамках репликации SQL Server.

SCHEMABINDING

Привязывает представление к схеме базовой таблицы или таблиц. Если аргумент SCHEMABINDING указан, нельзя изменить базовую таблицу или таблицы таким способом, который может повлиять на определение представления. Сначала нужно изменить или удалить само представление для сброса зависимостей от таблицы, которую требуется изменить. При использовании аргумента SCHEMABINDING инструкция *select_statement* должна включать двухкомпонентные (*schema.object*) имена таблиц, представлений или пользовательских функций, упоминаемых в предложении. Все указанные в инструкции объекты должны находиться в одной базе данных.

Представления или таблицы, входящие в представление, созданное при помощи предложения SCHEMABINDING, не могут быть сброшены, пока это представление не будет удалено или изменено таким образом, чтобы оно более не было привязано к схеме. В противном случае компонент Компонент Database Engine выдаст ошибку. Кроме того, выполнение инструкций ALTER TABLE для таблиц, которые входят в представления, привязанные к схемам, завершается ошибкой, если эти инструкции влияют на определение представления.

VIEW_METADATA

Указывает, что экземпляр SQL Server возвратит в API-интерфейсы DB-Library, ODBC и OLE DB сведения метаданных о представлении вместо базовой таблицы или таблиц, когда метаданные режима обзора затребованы для запроса, который ссылается на представление. Метаданные режима обзора — это дополнительные метаданные, которые экземпляр SQL Server возвращает вышеназванным клиентским API-интерфейсам. Эти метаданные позволяют клиентским API-интерфейсам реализовывать обновляемые клиентские курсоры. Метаданные режима обзора содержат сведения о базовой таблице, которой принадлежат столбцы в результирующем наборе.

Для представлений, созданных с применением предложения VIEW_METADATA, метаданные режима обзора возвращают имя представления, а не имена базовых таблиц при описании столбцов из представления в результирующем наборе.

В представлении, созданном с предложением WITH VIEW_METADATA, все столбцы, за исключением столбца **timestamp**, поддерживают обновление, если представление включает триггеры INSTEAD OF INSERT или INSTEAD OF UPDATE. Дополнительные сведения об обновляемых представлениях см. в разделе «Примечания».

ALTER VIEW CustomerOrders
AS
SELECT o.Qrderl, o.OrderDate, c.CompanyName, c.ContactName
FROM Orders про JOIN Customers з ON o.Custornerl = c.Customerl

Цей оператор змінює список вибору, включаючи в нього дату замовлення. Оператор ALTER VIEW заміняє запит SELECT, певний в оригінальному операторі CREATE VIEW.

Крім того, при модифікації представлення можна зашифрувати його визначення або гарантувати відповідність всіх операторів, що модифікують дані представлення, критерію оператора SELECT, що визначає представлення.

Видалення представлень

Видалення представлень не впливає на базові таблиці і їхні дані. Але якщо замість вилученого не створити інше представлення з тим же ім'ям, то наступне виконання будь-якого запиту, що використовує залежні від вилученого представлення об'єкти, закінчиться невдачею. Однак якщо в новому представленні немає посилань на об'єкти, необхідні будь-яким залежним від представлення об'єктам, то при виконанні запитів, що використовують такі залежні об'єкти, однаково виникне помилка.

DROP VIEW { *view* } [**,**...*n*]

DROP VIEW CustomerOrders

Перегляд даних через представлення

Запитувати дані через представлення можна без обмежень. Представлення використовуються для витягу даних в операторі SELECT практично так само, як звичайні таблиці.

```
SELECT Orderl, OrderDate
FROM CustomerOrders
WHERE CompanyName = 'quick-stop'
ORDER BY Orderl
```

3.3.8.2 Індекси

1) Визначення індексу
2) Організація індексів
3) Типи індексів
4) Створення індексу
5) Видалення індексу

Визначення індексу

Індекс (index) — це структура, зв'язана з таблицею або представленням і призначена для оптимізації пошуку інформації в цій таблиці або представленні. Індекс визначається для одного або декількох стовпців, що після падання індексу називаються індексованими стовпцями або *індексним ключем*.

Простий індекс визначається тільки по одній колонці таблиці.

Складений індекс – це індекс, визначений більш ніж по одній колонці.

Можна визначити індекс як унікальний або неунікальний.

В *унікальному* індексі кожне значення індексного ключа повинне бути унікальним. SQL Server створює унікальні індекси, якщо задали по таблиці обмеження PRIMARY KEY або обмеження UNIQUE.

У *неунікальному* індексі допускається дублювання індексних ключів у таблиці даних.

Індекс містить відсортовані значення індексованого стовпця (або стовпців) з посиланням на відповідний рядок вихідної таблиці або представлення. Коли необхідно знайти якесь значення в індексованому стовпці, то пошук здійснюється саме в індексі. Підвищення продуктивності досягається за рахунок того, що дані представляються відсортованими.

Здійснюється відновлення індексів при кожній операції вставки, відновлення або видалення для таблиці.

Організація індексів

Залежно від типу індексу він зберігається разом з даними або окремо від даних. Незалежно від типу всі індекси діють однаковим у своїй основі способом.

Індекси можуть бути організовані різними способами: наприклад, у вигляді B-дерева або з використанням хешування. Ширше використовують організацію у вигляді B-дерева. В цьому випадку кожна сторінка індексу називається індексною сторінкою, або *вузлом* індексу. Структура індексу починається на верхньому рівні з кореневого вузла. *Кореневий вузол* відповідає початку індексу: це перші дані, до яких здійснюється доступ при пошуку даних. Кореневий вузол містить ряд рядків індексу. Ці рядки містять значення ключа й покажчик на певну індексну сторінку (яка називається вузлом-гілкою). Ця конфігурація необхідна, оскільки у випадку таблиці даних середнього масштабу індекс складається з тисяч або мільйонів індексних сторінок. Почавши пошук з кореневого вузла й переміщаючись по вузлах індексу, SQL Server може поступово «наближатися» до потрібних даним.

Як і кореневий вузол, кожен вузол-гілка містить ряд індексних рядків у структурі індексної сторінки. Кожний індексний рядок указує на інший вузол-гілку або на вузол-лист (кінцевий вузол). Вузол-лист є останнім рівнем індексу. На відміну від кореневого вузла кожен вузол-гілка містить також зв'язаний список вузлів-гілок того ж рівня. Іншими словами, вузол «знає» про суміжні вузли й про вузли більш низького рівня.

Типи індексів

Існує два типи індексів B-Дерев: *кластеризовані* індекси й *некластеризовані* індекси*.* Кластеризований індекс зберігає у своїх вузлах-листах реальні рядки даних. Некластеризований індекс є допоміжною структурою, що вказує дані в таблиці.

В *кластеризованому* індексі у вузлі-листі зберігаються самі дані, тобто реальні рядки таблиці. Оскільки в кластеризованому індексі зберігаються реальні дані, можна створити не більше одного кластеризованого індексу по таблиці.

У *некластеризованому* індексі вузол-лист містить значення ключа, а також ідентифікатор рядка (RowID), що вказує потрібний рядок у таблиці, або ключ кластеризованого індексу, якщо є також кластеризований індекс по цій таблиці.

RowID – це покажчик, що автоматично формується системою SQL Server з ідентифікатора файлу (FileID), номера сторінки й номера рядка даних. Використовуючи RowID**,** можна зчитувати дані за допомогою всього лише однієї додаткової операції вводу-виводу.

За умовчанням створюється некластерний індекс.

Створення індексу

CREATE [UNIQUE] [CLUSTERED | NONCLUSTERED]
INDEX *ім'я_індексу* **ON** *ім'я_таблиці*
(*ім'я_колонки [ASC|DSC][, им'я_колонки, ім'я_колонки,...]*
)
[WHERE *фільтр***]**
[WITH *параметри]*

Колонки з типами даних **ntext**, **text**, **varchar(max)**, **nvarchar(max)**, **varbinary(max)**, **xml**, **image** не можуть бути ключовими для індекса. Крім того,

WHERE <filter_predicate>

Створює відфільтрований індекс шляхом зазначення термін для включення в індекс. Відфільтрований індекс повинен бути некластерізованний. Умова фільтра не може посилатися на обчислюваний стовпець, стовпець визначається користувачем типу, стовпець типу просторових даних або стовпець типу ID. Порівняння за допомогою літералів NULL з операторами порівняння неприпустимі. Замість цього використовуються оператори IS NULL і IS NOT NULL.

Наприклад:

```
WHERE StartDate > '20000101' AND EndDate <= '20000630'
WHERE ComponentID IN (533, 324, 753)
WHERE StartDate IN ('20000404', '20000905') AND EndDate IS NOT NULL
```

Параметри:

```
FILLFACTOR =fillfactor
  | IGNORE_DUP_KEY = { ON | OFF }
  | DROP_EXISTING = { ON | OFF }
  | ONLINE = { ON | OFF }
  | ALLOW_ROW_LOCKS = { ON | OFF }
  | ALLOW_PAGE_LOCKS = { ON | OFF }]
```

FILLFACTOR =fillfactor

Вказує, чи зберігати тимчасові результати сортування в базі даних tempdb.Значеніе за замовчуванням - OFF.

ON Проміжні результати сортування зберігаються в базі даних tempdb.

OFF Проміжні результати сортування зберігаються в тій же базі даних, де і індекс..

IGNORE_DUP_KEY = { ON | OFF }

Визначає відповідь на помилку, трапляється, коли операція вставки намагається вставити в унікальний індекс повторювані значення ключа..Значеніе за замовчуванням - OFF.

ON Якщо в унікальний індекс вставляються повторювані значення ключа, виводиться застережливе повідомлення.

OFF - виводиться повідомлення про помилку. Буде виконаний відкат всієї операції INSERT.

DROP_EXISTING = { ON | OFF }

Вказує, що названий існуючий кластерізованний або некластерізованний індекс віддаляється і перестраівается.Значеніе за замовчуванням - OFF.

ON Існуючий індекс віддаляється і перебудовується.

OFF Видається помилка, якщо індекс з вказаним ім'ям вже існує.

ONLINE = { ON | OFF }

Визначає, будуть базові таблиці і пов'язані індекси доступні для запитів і зміни даних під час операцій з індексамі.Значеніе за замовчуванням - OFF.

ALLOW_ROW_LOCKS = { ON | OFF }

Вказує, чи дозволено блокування строк.Значеніе за замовчуванням - ON.

ALLOW_PAGE_LOCKS = { ON | OFF }

Вказує, чи дозволено блокування страніц.Значеніе за замовчуванням - ON.

Найпростіша форма команди create index (створення індексу) має наступний вигляд:

```
create index назву індексу
on назва таблиці (назва стовпчика)
```

Для складеного індексу:

```
create index nmind
```

on friends_etc (pname, sname)

Видалення індексу

drop index назва табліци.названіе індексу
[, назва табліци.названіе індексу] ...

Коли видається ця команда, SQL Сервер видаляє зазначені індекси з бази даних і звільняє, займану ними пам'ять.

Тільки власник індексу може видалити його. Права на видалення індексів не можуть передаватися іншим користувачам. Не можна видаляти індекси системних таблиць, розташованих в базі даних master, або в базі даних користувача.

Індекс корисно видалити в тому випадку, якщо він не використовується в більшості запитів.

Наприклад:

drop index friends_etc.phone_ind

Щоб підключити індекс

SELECT * FROM myTable
With (index(myInd))

3.3.9. Розробка процедур і функцій

3.3.9.1 Збережені процедури

1) Загальний опис
2) Створення
3) Виклик

Загальний опис

Збережені процедури (stored procedure) являють собою групу команд Transact-SQL, об'єднаних в один РОЗДІЛ. Ця група компілюється і виконується як єдине ціле. Кожна збережена процедура має унікальне в межах бази даних ім'я, по якому користувачі можуть до неї звертатися. У збережені процедури також можуть входити команди виклику інших збережених процедур.

До складу Microsoft SQL Server входить велика кількість вбудованих системних збережених процедур. Усі вони мають префікс sp_ (від system procedure) і охоплюють практично всі аспекти керування і конфігурування сервера, дозволяючи змінювати значення в системних таблицях користувальницьких і системних баз даних.

Існує кілька категорій збережених процедур:

– Системні збережені процедури призначені для виконання різних адміністративних дій. Практично всі дії з адміністрування серверу виконуються з їх допомогою. Системні збережені процедури мають префікс sp_, зберігаються в системній базі даних і можуть бути викликані в контексті будь-який інший бази даних.

– Користувалцькі збережені процедури реалізують ті чи інші дії. Збережені процедури - повноцінний об'єкт бази даних. Внаслідок цього кожна збережена

процедура розташовується в конкретній базі даних, де і виконується.

– Тимчасові збережені процедури існують лише деякий час, після чого автоматично знищуються сервером. Вони діляться на локальні та глобальні. Локальні тимчасові збережені процедури можуть бути викликані тільки з того з'єднання, в якому створені. При створенні такої процедури їй необхідно дати ім'я, що починається з одного символу #. Як і всі тимчасові об'єкти, збережені процедури цього типу автоматично видаляються при відключенні користувача, перезапуску або зупинці сервера. Глобальні тимчасові збережені процедури доступні для будь-яких з'єднань сервера, на якому є така ж процедура. Для її визначення достатньо дати їй ім'я, що починається з символів ##. Видаляються ці процедури при перезапуску або зупинці сервера, а також при закритті з'єднання, в контексті якого вони були створені.

Створення

CREATE PROC *procedure_name* [**;** *number*]
[{ @*parameter data_type* }
[VARYING] [**=** *default*] [OUTPUT]
] [**,**...*n*]
[WITH
{ RECOMPILE | ENCRYPTION | RECOMPILE **,** ENCRYPTION }]
[FOR REPLICATION]
AS *sql_statement* [...*n*]

procedure_name

Ім'я нової збереженої процедури. Локальну або глобальну процедуру можна створити, указавши символ # перед *procedure_name* (*#procedure_name*) у випадку локальних тимчасових процедур і символ ## у випадку глобальних тимчасових процедур (*##procedure_name*).

; *number*

Необов'язкове ціле число, використовуване для угруповання процедур з тим самим ім'ям.

@ *parameter*

Параметр процедури. Можна оголосити один або більше параметрів. При виконанні процедури значення кожного з оголошених параметрів повинне бути зазначене користувачем. Якщо процедура містить параметри, що повертають табличне значення, а у виклику відсутній параметр, передається порожня таблиця.

Ім'я параметра повинне починатись з символу **@**.

data_type

Тип даних параметру.

VARYING

Указує результуючий набір, підтримуваний як вихідний параметр. Цей аргумент динамічно формується збереженою процедурою, і його вміст може розрізнятися. Застосовується тільки до аргументів типу **cursor**.

default

Значення за замовчуванням для аргументу. Якщо значення *default* визначене, процедуру можна виконати без вказівки значення відповідного аргументу. Значення за замовчуванням повинне бути константою або може рівнятися NULL. Якщо в процедурі використовується аргумент із ключовим словом LIKE, він може включати символи-шаблони %, _, [] і [^].

OUTPUT

Показує, що аргумент процедури є вихідним. Значення цього аргументу можна одержати за допомогою інструкції EXECUTE.

READONLY

Указує, що параметр не може бути оновлений або змінений у тілі процедури.

RECOMPILE

Показує, що не кеширується план виконання процедури й що процедура компілюється під час виконання.

ENCRYPTION

Показує, що SQL Server виконає затемнення вихідного тексту інструкції CREATE PROCEDURE. Користувачі, що не мають доступу до системних таблиць або файлів баз даних, не зможуть одержати затемнений текст. Однак цей текст буде доступний привілейованим користувачам.

EXECUTE AS

Визначає контекст безпеки, у якому повинна бути виконана збережена процедура.

FOR REPLICATION

Збережена процедура використовується як процедура-фільтр і виконується тільки під час реплікації. Якщо зазначено аргумент FOR REPLICATION, параметри не можуть бути оголошені. Аргумент RECOMPILE не враховується для процедур, створених з аргументом FOR REPLICATION.

<sql_statement>

Одна або кілька інструкцій мови Transact-SQL, які будуть включені до складу процедури.

Виклик

Для запуску збереженої процедури використовується команда Transact-SQL EXECUTE: EXECUTE <ім'я збереженої процедури> [<список параметрів>] Елементи списку розділяються комами.

3.3.9.2 Функції

SQL Server підтримує два типи функцій.

- *Вбудовані функції (built-in functions).* Вбудовані функції є складовою частиною середовища програмування і виконують заздалегідь визначену послідовність команд Transact-SQL. Вони не можуть бути модифіковані користувачем https://msdn.microsoft.com/ru-ru/library/ms174318(v=sql.120).aspx
- *Функції користувача (user-defined functions).* Ім'я таких функцій і послідовність вхідних у них команд задає сам користувач.

Користувальницька функція являє собою підпрограму Transact-SQL, що повертає значення. Користувальницька функція не може виконувати дії, що змінюють стан бази даних. Вона, як і системна функція, може бути викликана із запиту. Скалярні функції, як і збережені процедури, можуть бути виконані інструкцією EXECUTE.

Визначені користувачем функції створюються інструкцією CREATE FUNCTION, змінюються інструкцією ALTER FUNCTION і видаляються інструкцією DROP FUNCTION. Повне ім'я кожної визначеної користувачем функції повинне бути унікальним.

Користувальницька функція може приймати 0 або більше вхідних параметрів і повертати або скалярне, або табличне значення. Максимальне число вхідних параметрів для функції дорівнює 1024. Якщо для параметра функції встановлене

значення за замовчуванням, необхідно вказати ключове слово DEFAULT при виклику функції, щоб одержати встановлене за замовчуванням значення.

В визначених користувачем функціях не підтримуються вихідні параметри.

Типи користувацьких функцій:

- Скалярні, які повертають скалярне значення
- Табличні, які повертають результат у вигляді таблиці

Скалярні функції (Scalar Functions)

CREATE FUNCTION [*owner_name***.**] *function_name*
([{ @*parameter_name* [AS] *scalar_parameter_data_type* [= *default*] } [**,**...*n*]] **)**
RETURNS *scalar_return_data_type*
[WITH < ї
BEGIN

Приклад:

```
CREATE FUNCTION dbo.AveragePrice()
RETURNS money
WITH SCHEMABINDING
AS
BEGIN
   RETURN (SELECT AVG(Price) FROM dbo.Titles)
END
GO
```

Функція з табличним значенням (Inline Table-valued Functions)

CREATE FUNCTION [*owner_name***.**] *function_name*
([{ @*parameter_name* [AS] *scalar_parameter_data_type* [= *default*] } [**,**...*n*]] **)**
RETURNS TABLE
[WITH < function_option > [[**,**] ...*n*]]
[AS]
RETURN [**(**] select-*stmt* [**)**]

Приклад:

```
CREATE FUNCTION dbo.fnAuthorList()
RETURNS TABLE
AS
RETURN (SELECT au_id, au_lname + ', ' + au_fname AS au_name, address AS address1, city
+ ', ' + state + ' ' + zip AS address2 FROM authors)
GO
```

Функція з табличним значенням з декількох інструкцій (Multi-statement Table-valued Functions)

CREATE FUNCTION [*owner_name***.**] *function_name*
([{ @*parameter_name* [AS] *scalar_parameter_data_type* [= *default*] } [**,**...*n*]] **)**
RETURNS @*return_variable* TABLE < table_type_definition >
[WITH < function_option > [[**,**] ...*n*]]
[AS]
BEGIN

function_body
RETURN
END

function_name

Ім'я користувальницької функції. Імена функцій повинні задовольняти правилам побудови ідентифікаторів і повинні бути унікальними в межах бази даних і схеми. Дужки після імені функції обов'язкові навіть при відсутності параметрів.

@ *parameter_name*

Параметр користувальницької функції. Може бути оголошений один або кілька параметрів.

При виконанні функції значення кожного з оголошених параметрів повинне бути зазначене користувачем, якщо для цього параметра не визначене значення за замовчуванням.

Визначайте ім'я аргументу, використовуючи знак (**@**) як перший символ. Ім'я параметра повинне відповідати правилам побудови ідентифікаторів. Параметри локальні в межах функції, тобто в різних функціях можуть бути використані однакові імена параметрів. Параметри можна використовувати тільки замість констант. Вони не можуть використовуватися замість імен таблиць, імен стовпців або імен інших об'єктів бази даних.

parameter_data_type

Тип даних параметра.

[**=** *default*]

Значення параметра за замовчуванням. Якщо визначено значення *default*, функція виконується навіть у тому випадку, якщо для даного параметра значення не зазначене.

return_data_type

Значення скалярної користувальницької, що повертається, функції.

function_body

Указує серію інструкцій Transact-SQL, що у сукупності не викликає побічних ефектів начебто зміни вмісту таблиць і формує значення функції, що повертається. Аргумент *function_body* використовується тільки в скалярних функціях і у функціях, що повертають табличне значення з декількох інструкцій.

Для скалярних функцій аргумент *function_body* являє собою серію інструкцій Transact-SQL, які в сукупності обчислюють скалярне значення.

Для функцій, що повертають табличне значення з декількох інструкцій, аргумент *function_body* являє собою серію інструкцій Transact-SQL, що заповнюють змінну TABLE, що повертається.

scalar_expression

Указує скалярне значення, що повертається скалярною функцією.

TABLE

Указує, що повертається значенням, що, функції, що повертає табличне значення, є таблиця. Функціям, що повертають табличне значення, можуть передаватися тільки константи й **@***local_variables*.

< table_type_definition > :: =
({ *column_definition* | *table_constraint* } [,...*n*])

Визначає табличний тип даних для функції Transact-SQL. Оголошення таблиці включає визначення стовпців, а також обмежень для стовпців і таблиць.

< function_option > ::=

{ ENCRYPTION | SCHEMABINDING }

Указує, що функція буде мати один або трохи з наступних параметрів.

ENCRYPTION

Указує, що компонент Database Engine перетворить вихідний текст інструкції CREATE PROCEDURE у схований формат. Вихідні дані заплутування не видні безпосередньо в жоднім поданні каталогу. Користувачі, що не мають доступу до системних таблиць або файлів баз даних, не зможуть одержати схований текст. Проте цей текст буде доступний привілейованим користувачам, які можуть звертатися до системних таблиць або через порт DAC, або за допомогою безпосереднього доступу до файлів баз даних. Крім того, користувачі, які можуть підключити відладчик до серверного процесу, можуть одержати вихідний текст процедури з пам'яті під час виконання.

SCHEMABINDING

Указує, що функція прив'язується до об'єктів бази даних, на які в ній є посилання. Це запобігає зміні функції, якщо на неї є посилання з інших об'єктів, прив'язаних до схеми.

Прив'язка функції до об'єктів, на які вона посилається, віддаляється тільки в наступних випадках:

- При видаленні функції.
- При зміні функції інструкцією ALTER, якщо не зазначений параметр SCHEMABINDING.
- Функція може бути прив'язана до схеми тільки в тому випадку, якщо виконуються наступні умови.
- Функція є функцією Transact-SQL.
- Користувальницькі функції й подання, на які посилається дана функція, також прив'язані до схеми.
- Об'єкти, на які посилається функція, вказуються двокомпонентними іменами.
- Функція й об'єкти, на які вона посилається, ставляться до однієї й тій же базі даних.
- Користувач, що виконує інструкцію CREATE FUNCTION, має дозвіл REFERENCES на об'єкти бази даних, на які посилається функція.

Виконання визначеним користувачем функцій, що повертають скалярне значення

Визначені користувачем функції, що повертають скалярні значення, можна викликати тим же способом, що й збережені процедури. При виконанні визначеної користувачем функції, що повертає скалярне значення, параметри вказуються тим же способом, що й для збережених процедур:

- значення аргументів у дужки не поміщають;
- можна вказувати імена параметрів;
- якщо вказуються імена параметрів, значення аргументів не потрібно вказувати в тій же послідовності, що й параметри.

У наступному прикладі наведене створення визначеної користувачем функції, що повертає десяткове скалярне значення

```
CREATE FUNCTION dbo.ufn_CubicVolume
---i Input dimensions in centimeters.
 (@CubeLength decimal(4,1), @CubeWidth decimal(4,1),
 @CubeHeight decimal(4,1) )
RETURNS decimal(12,3) ---i Cubic Centimeters.
WITH SCHEMABINDING
```

```
AS
BEGIN
 RETURN ( @CubeLength * @CubeWidth * @CubeHeight )
END;
GO
```

Виклик визначених користувачем функцій, що повертають значення табличного типу даних

Визначену користувачем функцію, що повертає значення типу **table**, можна викликати там, де дозволені табличні вирази: у пропозиції FROM інструкцій SELECT, INSERT, UPDATE і DELETE. Виклик обумовленої користувачем функції, що повертає таблицю, може супроводжуватися додатковим псевдонімом таблиці. У наступному прикладі ілюструється виклик функції, що повертає табличне значення, ufnGetContactInformation у пропозиції FROM інструкції SELECT

```
USE AdventureWorks;
GO
SELECT ContactID, FirstName, LastName, JobTitle, ContactType
FROM dbo.ufnGetContactInformation(2200);
GO
SELECT ContactID, FirstName, LastName, JobTitle, ContactType
FROM dbo.ufnGetContactInformation(5);
GO
```

Якщо визначена користувачем функція, що повертає таблицю, викликається в пропозиції FROM вкладеного запиту, аргументи функції не можуть посилатися на які-небудь стовпці зовнішнього запиту.

В інструкції SELECT можна викликати визначену користувачем функцію, що повертає значення типу **table**, тільки один раз.

3.3.9.3 Вбудовані функції

Вбудовані функції, наявні в розпорядженні користувачів при роботі з SQL, можна умовно розділити на наступні групи:

- Математичні функції;
- Строкові функції;
- Функції для роботи з датою і часом;
- Функції конфігурування;
- Функції системи безпеки;
- Функції управління метаданими;
- Статистичні функції.

Математичні функції

ABS	обчислює абсолютне значення числа
ACOS	обчислює арккосинус
ASIN	обчислює арксинус
ATAN	обчислює арктангенс
ATN2	обчислює арктангенс з урахуванням квадратів
CEILING	виконує округлення вгору
COS	обчислює косинус кута
COT	повертає котангенс кута

DEGREES перетворює значення кута з радіан в градуси
EXP повертає експоненту
FLOOR виконує округлення вниз
LOG обчислює натуральний логарифм
LOG10 обчислює десятковий логарифм
PI повертає значення "пі"
POWER з водить число до степеня
RADIANS перетворює значення кута з градуса в радіани
RAND повертає випадкове число
ROUND виконує округлення із заданою точністю
SIGN про пределяет знак числа
SIN обчислює синус кута
SQUARE виконує зведення числа в квадрат
SQRT витягує квадратний корінь
TAN повертає тангенс кута

```
SELECT Товар.Название, Сделка.Количество,
Round(Товар.Цена*Сделка.Количество
   *0.05,1)
AS Налог
FROM Товар INNER JOIN Сделка
   ON Товар.КодТовара=
   Сделка.КодТовара
```

Пример Использование функции округления до одного знака после запятой для расчета налога. (html, txt)

Рядкові функції

ASCII повертає код ASCII лівого символу рядка
CHAR за кодом ASCII повертає символ
CHARINDEX визначає порядковий номер символу, з якого починається входження підрядка в рядок
DIFFERENCE повертає показник збігу рядків
LEFT повертає вказане число символів з початку рядка
LEN повертає довжину рядка
LOWER переводить всі символи рядка в нижній регістр
LTRIM видаляє пробіли на початку рядка
NCHAR повертає за кодом символ Unicode
PATINDEX виконує пошук підрядка в рядку за вказаною шаблоном
REPLACE замінює входження підрядка на вказане значення
QUOTENAME конвертує рядок в формат Unicode
REPLICATE виконує тиражування рядки певне число разів
REVERSE повертає рядок, символи якій записані у зворотному порядку
RIGHT повертає вказане число символів з кінця рядка
RTRIM видаляє прогалини в кінці рядка
SOUNDEX повертає код звучання рядка
SPAC повертає вказане число прогалин
STR виконує конвертування значення числового типу в символьний формат
STUFF видаляє вказане число символів, замінюючи нової підрядком

SUBSTRING повертає для рядка підрядок зазначеної довжини з заданого символу
UNICODE повертає Unicode-код лівого символу рядка
UPPER переводить всі символи рядка у верхній регістр

Приклад. Використання функції LEFT для отримання ініціалів клієнтів.

```
SELECT Фирма, [Прізвище]+""
   +Left([Ім'я],1)+"."
   +Left([По батьколві],1)
   +"." AS ПІБ
FROM Клієнт
```

Функції для роботи з датою і часом

DATEADD (*datepart*, *number*, *date*) додає до дати зазначена значення днів, місяців, годин. Результат в форматі ***datetime***

DATENAME (*datepart*, *date*) виділяє з дати вказану частину і повертає її в символьному форматі

DATEPART (*datepart*, *date*) виділяє з дати вказану частину і повертає її в числовому форматі

DAY (*date*) повертає число з вказаної дати

GETDATE () повертає поточнн системну дату і час в форматі ***datetime***

ISDATE (*expression*) перевіряє правильність виразу на відповідність одному з форматів введення дати

MONTH (*date*) повертає значення місяця з вказаної дати

YEAR (*date*) повертає значення року з вказаної дати

DATETIMEFROMPARTS (*year*, *month*, *day*, *hour*, *minute*, *seconds*, *milliseconds*) Повертає значення типу ***datetime***, що відповідає вказаній даті і часу

DATEDIFF (*datepart*, *startdate* , *enddate*) повертає різницю між вказаними частинами двох дат

```
SELECT Year(Дата) AS Год, Month(Дата)
  AS Месяц,
  Sum(Количество) AS Общ_Количество
FROM Сделка
GROUP BY Year(Дата), Month(Дата)
```

Використання функцій YEAR і MONTH для визначення загальної кількості товару, проданого за кожен місяць кожного року.

```
DECLARE @d DATETIME
DECLARE @y INT
SET @d='29.10.03'
SET @y=DATEPART(yy,@d)
SELECT @y
```

Перетворення дати в зручний для виведення вид

```
CONVERT(varchar(12), GETDATE(), 101)
```

3.3.10 Розробка тригерів

3.3.10.1 Визначення триггера в стандарті мови SQL

Тригер є спеціальний тип збережених процедур, що запускаються сервером автоматично при спробі зміни даних в таблицях, з якими тригери пов'язані. Кожен тригер прив'язується до конкретної таблиці. Всі вироблені їм модифікації даних розглядаються як одна транзакція. У разі виявлення помилки або порушення цілісності даних відбувається відкат цієї транзакції. Тим самим внесення змін забороняється. Скасовуються також всі зміни, вже зроблені тригером.

За допомогою обмежень цілісності, правил і значень за замовчуванням не завжди можна домогтися потрібного рівня функціональності. Часто потрібно реалізувати складні алгоритми перевірки даних, що гарантують їх достовірність і реальність. Тригери можна розглядати як свого роду фільтри, котрі вступають в дію після виконання всіх операцій відповідно до правил, стандартними значеннями і т.д.

Створює тригер тільки власник бази даних. Це обмеження дозволяє уникнути випадкового зміни структури таблиць, способів зв'язку з ними інших об'єктів і т.п.

На відміну від звичайної підпрограми, тригер виконується неявно в кожному випадку виникнення тригерній події, до того ж він не має аргументів. Приведення його в дію іноді називають запуском тригера. За допомогою тригерів досягаються наступні цілі:

- Перевірка коректності введених даних і виконання складних обмежень цілісності даних, які важко, якщо взагалі можливо, підтримувати за допомогою обмежень цілісності, встановлених для таблиці;
- Видача попереджень, що нагадують про необхідність виконання деяких дій при оновленні таблиці, реалізованому певним чином;
- Накопичення аудиторської інформації за допомогою фіксації відомостей про внесені зміни і тих особах, які їх виконали;
- Підтримка реплікації.
- Існує три типи тригерів:
  - INSERT TRIGGER - запускаються при спробі вставки даних за допомогою команди INSERT.
  - UPDATE TRIGGER - запускаються при спробі зміни даних за допомогою команди UPDATE.
  - DELETE TRIGGER - запускаються при спробі видалення даних за допомогою команди DELETE.

3.3.10.2 Реализація тригерів в MS SQL Server

В реализації СУБД MS SQL Server використовується оператор створення або зміни тригера:

```
<Определение_триггера>::=
{CREATE | ALTER} TRIGGER имя_триггера
ON {имя_таблицы | имя_просмотра }
[WITH ENCRYPTION ]
{
{ { FOR | AFTER | INSTEAD OF }
{ [ DELETE] [,] [ INSERT] [,] [ UPDATE] }
[ WITH APPEND ]
[ NOT FOR REPLICATION ]
AS
```

```
  sql_оператор[...n]
} |
{ {FOR | AFTER | INSTEAD OF } { [INSERT] [,]
  [UPDATE] }
[ WITH APPEND]
[ NOT FOR REPLICATION]
AS
{ IF UPDATE(имя_столбца)
[ {AND | OR} UPDATE(имя_столбца)] [...n]
|
IF (COLUMNS_UPDATES(){оператор_бит_обработки}
  бит_маска_изменения)
{оператор_бит_сравнения }бит_маска [...n]}
sql_оператор [...n]
}
}
```

Тригер може бути створений тільки в поточній базі даних, не допускається звернення всередині тригера до інших баз даних, в тому числі і розташованим на віддаленому сервері.

Призначення аргументів

Имя триггера должно быть уникальным в пределах базы данных. Дополнительно можно указать имя владельца.

При вказівці аргументу WITH ENCRYPTION сервер виконує шифрування коду тригера, щоб ніхто, включаючи адміністратора, не міг отримати до нього доступ і прочитати його.

Параметри, що визначають поведінку тригерів:

- AFTER
- INSTEAD OF

AFTER. Тригер виконується після успішного виконання викликали його команд. Якщо ж команди не можуть бути успішно завершені, тригер не виконується. Зміни даних в результаті виконання запиту користувача і виконання тригера здійснюється в тілі однієї транзакції, тому якщо станеться відкат тригера, то будуть відхилені і призначені для користувача зміни.

Можна визначити кілька AFTER-тригерів для кожної операції (INSERT, UPDATE, DELETE). В цьому випадку можна за допомогою системної збереженої процедури sp_settriggerorder вказати, який з них буде виконуватися першим, а який останнім.

За замовчуванням в SQL Server все тригери є AFTER-тригерами.

INSTEAD OF. Тригер викликається замість виконання команд. На відміну від AFTER-тригера INSTEAD OF-тригер може бути визначений як для таблиці, так і для перегляду. Для кожної операції INSERT, UPDATE, DELETE можна визначити тільки один INSTEAD OF-тригер.

Конструкції ***[DELETE] [,] [INSERT] [,] [UPDATE]*** и

FOR | AFTER | INSTEAD OF } { [INSERT] [,] [UPDATE]

визначають, на яку команду буде реагувати тригер. При його створенні має бути вказана хоча б одна команда. Допускається створення тригера, що реагує на дві або на все три команди.

Аргумент WITH APPEND дозволяє створювати кілька тригерів кожного типу.

При створенні тригера з аргументом NOT FOR REPLICATION забороняється його запуск під час виконання модифікації таблиць механізмами реплікації.

Конструкція AS sql_оператор [... n] визначає набір SQL- операторів і команд, які будуть виконані при запуску тригера.

3.3.10.3 Програмування тригера

При виконанні команд додавання, зміни і видалення записів сервер створює дві спеціальні таблиці: inserted і deleted. У них містяться списки рядків, які будуть вставлені або видалені по завершенні транзакції. Структура таблиць inserted і deleted ідентична структурі таблиць, для якої визначається тригер. Для кожного тригера створюється свій комплект таблиць inserted і deleted, тому ніякої іншої тригер не зможе отримати до них доступ. Залежно від типу операції, що отримала виконання тригера, вміст таблиць inserted і deleted може бути різним:

1. команда INSERT - в таблиці inserted містяться всі рядки, які користувач намагається вставити в таблицю; в таблиці deleted не буде ні одного рядка; після завершення тригера всі рядки з таблиці inserted перемістяться в вихідну таблицю;

2. команда DELETE - в таблиці deleted будуть міститися всі рядки, які користувач спробує видалити; тригер може перевірити кожен рядок і визначити, чи дозволено її видалення; в таблиці inserted бракуватиме жодного рядка;

3. команда UPDATE - при її виконанні в таблиці deleted знаходяться старі значення рядків, які будуть видалені при успішному завершенні тригера. Нові значення рядків містяться в таблиці inserted. Ці рядки додадуться в вихідну таблицю після успішного виконання тригера.

Для отримання інформації про кількість рядків, яке буде змінено при успішному завершенні тригера, можна використовувати функцію@@ROWCOUNT; вона повертає кількість рядків, оброблених останньою командою. Слід підкреслити, що тригер запускається ні до спробі змінити конкретну рядок, а в момент виконання команди зміни. Одна така команда впливає на безліч рядків, тому тригер повинен обробляти всі ці рядки.

Якщо тригер виявив, що з 100 вставляються, змінюваних чи видалення рядків тільки одна не задовольняє тих чи інших умов, то ніяка рядок не буде вставлена, змінена або видалена. Така поведінка обумовлена вимогами транзакції - повинні бути виконані або всі модифікації, або жодної.

Тригер виконується як неявно певна транзакція, тому всередині тригера допускається застосування команд управління транзакціями. Зокрема, при виявленні порушення обмежень цілісності для переривання виконання тригера і скасування всіх змін, які намагався виконати користувач, необхідно використовувати команду ROLLBACK TRANSACTION.

Для отримання списку стовпців, змінених при виконанні команд INSERT або UPDATE, що викликали виконання тригера, можна використовувати функцію COLUMNS_UPDATED (). Вона повертає двійковечисло, кожний біт якого, починаючи з молодшого, відповідає одному стовпцю таблиці (в порядку проходження стовпців при створенні таблиці). Якщо біт встановлено у значення "1", то відповідний стовпець був змінений. Крім того, факт зміни стовпчика визначає і функція UPDATE (ім'я_стовпця).

3.3.10.4 Видалення тригкра

DROP TRIGGER {имя_триггера} [,...n]

3.3.10.5 Приклади використання тригерів

Приклад 1. Використання тригера для реалізації обмежень на значення. В додається в таблицю Угода записи кількість проданого товару має бути не більше, ніж його залишок з таблиці Склад.

Команда вставки запису в таблицю Угода може бути, наприклад, такий:

```
INSERT INTO Угода
  VALUES (3,1, -299, '01 / 08/2002 ')
```

Створюваний тригер повинен відреагувати на її виконання таким чином: необхідно скасувати команду, якщо в таблиці Склад величина залишку товару виявилася менше продаваного кількості товару з введеним кодом (в прикладі код товару = 3). У вставляється записи кількість товару вказується зі знаком "+", якщо товар поставляється, і зі знаком "-", якщо він продається. Представлений тригер налаштований на обробку тільки однієї додається записи.

```
CREATE TRIGGER Триггер_ins
ON Сделка FOR INSERT
AS
IF @@ROWCOUNT=1
BEGIN
  IF NOT EXISTS
     (SELECT *
      FROM inserted
      WHERE -inserted.количество<=
                ALL(SELECT Склад.Остаток
                   FROM Склад,Сделка
                   WHERE Склад.КодТовара=Сделка.КодТовара)
     )
    BEGIN
      ROLLBACK TRAN
      PRINT 'Отмена поставки: товара на складе нет'
    END
END
```

Приклад 2. Використання тригера для збору статистичних даних.

Створити тригер для обробки операції вставки запису в таблицю Угода, наприклад, такої команди:

```
INSERT INTO Сделка
  VALUES (3,1,200,'01/08/2002')
```

поставляється товар з кодом 3 від клієнта з кодом 1 в кількості 200 одиниць.

При продажу або отриманні товару необхідно відповідним чином змінити кількість його складського запасу. Якщо товару на складі ще немає, необхідно додати відповідний запис в таблицю Склад. Тригер обробляє тільки одну додається рядок.

```
ALTER TRIGGER Триггер_ins
ON Сделка FOR INSERT
AS
DECLARE @x INT, @y INT
```

```
IF @@ROWCOUNT=1
--в таблицу Сделка добавляется запись
--о поставке товара
BEGIN
--количество проданного товара должно быть не
--меньше, чем его остаток из таблицы Склад
IF NOT EXISTS(SELECT *
          FROM inserted
          WHERE -inserted.количество<
  =ALL(SELECT Склад.Остаток
        FROM Склад,Сделка
        WHERE Склад.КодТовара=
            Сделка.КодТовара))
    BEGIN
      ROLLBACK TRAN
        PRINT 'откат товара нет '
     END
--если записи о поставленном товаре еще нет,
--добавляется соответствующая запись
--в таблицу Склад
  IF NOT EXISTS ( SELECT *
          FROM Склад C, inserted i
          WHERE C.КодТовара=i.КодТовара )
      INSERT INTO Склад (КодТовара,Остаток)
  ELSE
--если запись о товаре уже была в таблице
--Склад, то определяется код и количество
--товара из добавленной в таблицу Сделка записи
  BEGIN
    SELECT @y=i.КодТовара, @x=i.Количество
    FROM Сделка C, inserted i
    WHERE C.КодТовара=i.КодТовара
--и производится изменения количества товара в
--таблице Склад
        UPDATE Склад
        SET Остаток=остаток+@x
        WHERE КодТовара=@y
    END
END
```

Приклад 3. Створити тригер для обробки операції видалення запису з таблиці Угода, наприклад, такої команди:

```
DELETE FROM Сделка WHERE КодСделки=4
```

Для товара, код которого указан при удалении записи, необходимо откорректировать его остаток на складе. Триггер обрабатывает только одну удаляемую запись.

```
CREATE TRIGGER Триггер_del
ON Сделка FOR DELETE
AS
IF @@ROWCOUNT=1 -- удалена одна запись
```

```
BEGIN
  DECLARE @y INT,@x INT
--определяется код и количество товара из
--удаленной из таблицы Склад записи
  SELECT @y=КодТовара, @x=Количество
  FROM deleted
--в таблице Склад корректируется количество
--товара
     UPDATE Склад
     SET Остаток=Остаток-@x
     WHERE КодТовара=@y
END
```

РОЗДІЛ 4. ЗАХИСТ БАЗ ДАНИХ

1) Адміністрування системи безпеки
2) Захист даних
3) Приклад

4.1 Адміністрування системи безпеки

1) Основи забезпечення безпеки
2) Компоненти структури безпеки

4.1.1 Основи забезпечення безпеки

Система збереження інформації повинна бути максимально захищена від крадіжки, знищення, від випадкового і зловмисного пошкодження і спотворення інформації

Стратегія всебічного захисту з рівнями безпеки, що перекриваються - це кращий спосіб боротьби з погрозами безопеки. SQL Server надає архітектуру безпеки, яка дозволяє адміністраторам баз даних і розробникам створювати захищені додатки баз даних і боротися з загрозами. В кожній версії SQL Server є удосконалення в порівнянні з попередніми версіями у вигляді додавання нових функцій і можливостей. Однак система безпеки не поставляється в готовому вигляді. Кожний додаток унікальний за своїми вимогам до безопеки.

Примірник SQL Server містить ієрархічну колекцію сутностей, починаючи з сервера. Кожний сервер складається з декількох баз даних, а кожна база даних - з колекції об'єктів, що захищаються. У кожного об'єкта, що захищається, у SQL Server є пов'язані права доступу, які можуть надаватися учаснику, що є окремимою особою, групою або процесом, що отримали доступ до SQL Server. Платформа безпеки SQL Server управляє доступом до захищуваних сутностей за допомогою перевірки *автентичності(подлинности)* та *авторизації*.

Перевірка *автентичності* - це процес входу в SQL Server, в рамках якого учасник запитує доступ шляхом подачі облікових даних, які перевіряє сервер. В час перевірки автентичності відбувається ідентифікація користувача або процесу.

Авторизація - це процес, який застосовується після перевірки того, що користувач відповідає аутентифікації. Протягом цього процесу система визначає, які ресурси може використовувати даний користувач. Іншими словами, інформація із структурного і системного каталогу про конкретну сутність тепер доступна лише довіреним особам - тим людям, які мають повноваження доступу до даної сутності.

Перевірка автентичності

Аутентифікацією (встановленням автентичності) називається перевірка приналежності суб'єкту доступу пред'явленого ним ідентифікатора і підтвердження його автентичності. Іншими словами, аутентифікація полягає в перевірці права користувача на доступ до ресурсу. Найчастіше аутентифікація здійснюється за допомогою введення імені і пароля.

SQL Server підтримує два режими перевірки автентичності:

- режим перевірки автентичності Windows (режим аутентифікації Windows) і
- змішаний режим перевірки автентичності.

Режим перевірки автентичності Windows є режимом за замовчуванням. В цьому випадку використовується система безпеки Windows і її механізм облікових записів

Визначеним обліковим записам користувачів і груп Windows дозволяється входити в SQL Server. Користувачі Windows, що пройшли перевірку автентичності, не повинні надавати додаткові облікові дані.

Змішаний режим дає користувачу можливість з'єднуватись з базою даних використовуючи аутентифікацію Windows або аутентифікацію SQL Server. Це означає, що деякі облікові записи користувача можуть бути встановлені для використання і в підсистемі безпеки SQL Server і в підсистемі безпеки Windows.

4.1.2 Компоненти структури безпеки

Основою системи безпеки є облікові записи(accounts), користувачі(users), ролі і групи(groups).

Користувачі

Після проектування логічної структури бази даних, зв'язків між таблицями, обмежень цілісності та інших структур необхідно визначити коло користувачів, які матимуть доступ до бази даних.

В системі SQL-сервер організована дворівневе налаштування обмежень доступу до даних:

– На першому рівні необхідно створити обліковий запис користувача (login), що дозволяє йому підключитися до самого серверу, але не дає автоматичного доступу до баз даних.

– На другому рівні для кожної бази даних SQL-сервера на підставі облікового запису необхідно створити запис користувача.

На основі прав, виданих користувачеві як користувачеві бази даних (user), його реєстраційне ім'я (login) отримує доступ до відповідної бази даних. У різних базах даних login одного і того ж користувача може мати однакові або різні імена user з різними правами доступу. Інакше кажучи, за допомогою облікового запису користувача здійснюється підключення до SQL-серверу, після чого визначаються його рівні доступу для кожної бази даних окремо.

Для створення користувача в середовищі MS SQL Server слід зробити наступні кроки:

– Створити в базі даних обліковий запис користувача, вказавши для нього пароль і прийняте за умовчанням ім'я бази даних (процедура sp_addlogin).

– Додати цього користувача в усі необхідні бази даних (процедура sp_adduser).

– Надати йому в кожній базі даних відповідні привілеї (команда GRANT).

Створення нового облікового запису може бути зроблене за допомогою системної збереженої процедури:

sp_addlogin
[@ login =] 'учетная_запись'
[, [@ Password =] 'пароль']
[, [@ Defdb =] 'база_данних_по_умолчанію']

Після завершення аутентифікації і отримання ідентифікатора облікового запису (login ID) користувач вважається зареєстрованим, і йому надається доступ до сервера. Для кожної бази даних, до об'єктів якої він має намір отримати доступ, обліковий запис користувача (login) асоціюється з користувачем (user) конкретної бази даних, що здійснюється за допомогою процедури:

sp_adduser
[@ loginame =] 'учетная_запись'
[, [@ Name_in_db =] 'имя_пользователя']

[, [@ Grpname =] 'ім'я_ролі']

Користувач, який створює об'єкт в базі даних (таблицю, збережену процедуру, перегляд), стає його власником. Власник об'єкта (database object owner dbo) має всі права доступу до створеного ним об'єкту. Щоб користувач міг створити об'єкт, власник бази даних (dbo) повинний надати йому відповідні права. Повне ім'я створюваного об'єкта включає в себе ім'я створив його користувача.

Ролі

Для спрощення управління правами доступу в більшості серверних СУБД застосовується механізм ролей - наборів прав доступу до об'єктів бази даних, що привласнюються деякій сукупності користувачів (групі). При використанні ролей управління розподілом прав доступу до об'єктів між користувачами, які виконують однакові функції та застосовують одні й ті ж додатки, істотно спрощується: створення ролі і одноразове призначення їй відповідних прав здійснюється набагато швидше, ніж визначення прав доступу кожного користувача до кожного об'єкту.

Ролі можуть бути вбудованими і користувацькими. Перші створюються автоматично при установці сервера, а другі створюються користувачем.

В SQL Server реалізовані два види стандартних ролей:

- на рівні сервера і
- на рівні баз даниж

Можна включити будь-який обліковий запис SQL Server (login) або обліковий запис Windows в будь-яку роль сервера.

Ролі баз даних дозволяють об’єднувати користувачів в одну одиницю і працювати з нею як з користувачем.

Іноді необхідно мати постійний набір прав для доступу до бази даних з додатку. В цьому випадку використовують ролі додатку.

Різні дії по відношенню до ролі здійснюються за допомогою спеціальних процедур:

- створення нової ролі:
  sp_addrole
  [@ rolename =] 'ім'я_ролі'
  [, [@ Ownername =] 'ім'я_владельца']
- додавання користувача до ролі:
  sp_addrolemember
  [@ rolename =] 'ім'я_ролі',
  [@ membername =] 'имя_пользователя'
- видалення користувача з ролі:
  sp_droprolemember
  [@ rolename =] 'ім'я_ролі',
  [@ membername =] 'имя_пользователя'
- видалення ролі:
  sp_droprole
  [@ rolename =] 'ім'я_ролі'

4.2 Захист даних

1) Шифрування даних
2) Права доступу
3) Приклад

4.2.1 Шифрування даних

Як би добре не була спланована система безпеки SQL Server, залишається можливість несанкційованого доступу, копіювання файлів з даними для перегляду на іншому комп'ютері, або отримання даних при їх передачі по мережі. Організації, в чиїх базах даних зберігається конфіденційна інформація, повинні дотримуватися вимог численних законодавчих актів. Ці закони вимагають шифрування конфіденційної інформації на рівні бази даних і операційної системи.

SQL Server, як і інші поширені комерційні системи управління базами даних, володіє безліччю варіантів шифрування, в тому числі на рівні комірок, бази даних і файлів через Windows, а також на транспортному рівні.

Шифрування - процес кодування конфіденційних даних з використанням ключа і пароля. Шифрування надійно захищає дані і скорочує вірогідність несанкціонованого розкриття конфіденційної інформації, оскільки без відповідного ключа або пароля дані недоступні. В розпорядженні SQL Server багато режимів шифрування, в тому числі на рівні комірок, бази даних, файлів через Windows і шифрування на транспортному рівні.

В пізніх версіях з'явилась можливість зашифрувати всю базу даних за допомогою прозорого шифрування. При такому шифруванні можна захистити бази даних без зміни існуючих додатків, структур баз даних або процесів.

Прозоре шифрування даних шифрує бази даних в реальному часі, в міру внесення записів у файли (* .mdf) бази даних SQL Server і файли (* .ldf) журналу транзакцій. Записи також шифруються в реальному часі під час резервного копіювання бази даних, а потім формуються моментальні знімки. Дані шифруються перед записом на диск і розшифровуються перед отриманням. Процес повністю прозорий для користувача або програми, оскільки виконується на рівні SQL Server Service.

При прозорому методі SQL Server шифрує базу даних за допомогою ключа шифрування бази даних.

Модель шифрування SQL Server основується на ключах шифрування, які використовуються для шифрування інших ключів. Використовується кілька рівнів ключів шифрування:

- ключ верхнього рівня шифрується за допомогою Windows API і є асиметричним ключем
- другий рівень – ключ, за допомогою якого шифруються симетричні ключі, асиметричні ключі і сертифікати. Кожна база даниз має один такий ключ.
- третій рівень - симетричні ключі, асиметричні ключі і сертифікати

У SQL Server підтримується використання механізмів політики паролів Windows. Політика паролів застосовується до імені входу, яке використовує перевірку автентичності SQL Server, і до користувача автономної бази даних з паролем.

У SQL Server можуть застосовуватися політики складності та закінчення терміну дії, які застосовуються в Windows до паролів, використовуваним в SQL Server.

4.2.2 Права доступу

Визначення привілеїв в стандарті мови

Кожна СУБД повинна підтримувати механізм, що гарантує, що доступ до бази даних зможуть отримати тільки ті користувачі, які мають відповідний дозвіл. Мова SQL включає оператори GRANT і REVOKE, призначені для організації захисту таблиць в базі даних. Механізм захисту побудований на використанні ідентифікаторів користувачів, наданих їм прав володіння і привілеїв.

Ідентифікатор користувача – це логін користувача, який пов'язаний з паролем. Ідентифікатор користувача визначає, на які об'єкти бази даних користувач може

посилатися і які операції з цими об'єктами він має право виконувати. Для надання набору прав можна використовувати ролі.

Привілеями, або ***правами***, називаються дії, які користувач має право виконувати відносно даної таблиці бази даних або подання. У стандарті SQL визначається наступний набір привілеїв:

SELECT - право вибирати дані з таблиці;

INSERT - право вставляти в таблицю нові рядки;

UPDATE - право змінювати дані в таблиці;

DELETE - право видаляти рядки з таблиці;

REFERENCES - право посилатися на стовпці вказаної таблиці в описах вимог підтримки цілісності даних;

USAGE - право використовувати домени, перевірки та набори символів.

Привілеї INSERT і UPDATE можуть обмежуватися лише окремими стовпцями таблиці, в цьому випадку користувачеві дозволяється модифікувати значення тільки зазначених стовпців. Аналогічним чином привілей REFERENCES може поширюватися виключно на окремі стовпці таблиці, що дозволить використовувати їх імена в формулюваннях вимог захисту цілісності даних - наприклад, у пропозиціях CHECK і FOREIGN KEY, що входять у визначення інших таблиць, тоді як застосування для подібних цілей інших стовпців буде заборонено.

Коли користувач за допомогою оператора CREATE TABLE створює нову таблицю, він автоматично стає її власником і отримує по відношенню до неї повний набір привілеїв, яких решта користувачів початково не мають. Щоб забезпечити їм доступ, власник повинен явним чином надати необхідні права, для чого використовується оператор GRANT.

Створюючи представлення за допомогою оператора CREATE VIEW, користувач автоматично стає власником цього подання і також отримує повний набір прав. Для створення уявлення користувачеві досить мати привілей SELECT для всіх назв таблиць і привілей REFERENCES для всіх стовпців, згадуваних у визначенні цього подання. Привілеї INSERT, UPDATE і DELETE відносно створеного уявлення користувач отримає тільки в тому випадку, якщо має відповідні привілеї стосовно всіх використовуваних в поданні таблиць.

Надання привілеїв

```
GRANT
  ALL [ PRIVILEGES ] |
SELECT | DELETE | INSERT
    [(имя_столбца[,...n])]
| UPDATE [(имя_столбца[,...n])]}
| REFERENCES [(имя_столбца[,...n])] |
    USAGE
[ ( column [ ,...n ] ) ] ON  table | view
 | ON { table | view } [ ( column [ ,...n ] ) ]
 | ON  stored_procedure | extended_procedure
 | ON  user_defined_function
TO security_account [ ,...n ]
[ WITH GRANT OPTION ]
[ AS  group | role ]
```

WITH GRANT OPTION

Вказує, що учаснику також дається можливість надати зазначений дозвіл іншим

учасникам.

security_account

Вказує учасника, якому надається дозвіл.

AS group | role

Визначає учасника, у якого інший учасник, що виконує даний запит, успадковує право надавати даний дозвіл

Відміна привілеїв

```
REVOKE[GRANT OPTION FOR]
   {<привилегия>[,...n]
   | ALL PRIVILEGES}
ON имя_объекта
FROM {<идентификатор_пользователя>
   [,...n]| PUBLIC}
[RESTRICT | CASCADE]
```

Ключове слово ALL PRIVILEGES означає, що для зазначеного користувача скасовуються всі привілеї, надані йому раніше тим користувачем, який ввів даний оператор. Необов'язкова фраза GRANT OPTION FOR дозволяє для всіх привілеїв, переданих у вихідному операторі GRANT фразою WITH GRANT OPTION, скасовувати можливість їх передачі незалежно від самих привілеїв.

Якщо в операторі вказано ключове слово RESTRICT, успішне виконання команди REVOKE можливо лише в тому випадку, коли перераховані в операторі привілеї не можуть послужити причиною появи у будь-яких інших користувачів так званих "залишених" привілеїв. За допомогою параметра CASCADE видаляються всі привілеї, які інакше могли б залишитися у інших користувачів.

"Залишеними" є привілеї, що збереглися у користувача, якому вони свого часу були надані за допомогою параметра GRANT OPTION.

4.3 Приклад

Створити групу доступу для бази даних з правами на здійснення запитів за допомогою системної процедури ***sp_addrole***:

```
sp_addrole [ @rolename = ] 'role'
 [ , [ @ownername = ] 'owner' ]
```

```
use pubs
go
sp_addrole FirstGroup
```

Надамо створеній групі право здійснення запитів за таблиці за допомогою команди ***GRANT***:

```
grant select on TABLE to FirstGroup
```

Створити нового користувача за допомогою процедури ***sp_addlogin***:

```
sp_addlogin [ @loginame = ] 'login'
 [ , [ @passwd = ] 'password' ]
 [ , [ @defdb = ] 'database' ]
```

[, [@deflanguage =] 'language']
[, [@sid =] sid]
[, [@encryptopt =] 'encryption_option']

sp_addlogin 'ZZZ', '123456', 'pubs'

Зробимо прив'язку користувача до бази даних за допомогою системної процедури ***sp_adduser***:

sp_adduser [@loginame =] 'login'
[, [@name_in_db =] 'user']
[, [@grpname =] 'group']

use pubs
sp_adduser 'ZZZ', 'ZZZ'

Включимо створеного користувача до групи FirstGroup:

sp_addrolemember [@rolename =] 'role' ,
[@membername =] 'security_account'

EXEC sp_addrolemember 'FirstGroup', 'ZZZZZ'

Для перевірки наявних прав доступу необхідно створити нове підключення до серверу БД, використовуючи нового користувача.

Відображення плану запиту за допомогою команд SQL Server-а
set showplan_text on.

Використовуючи поточне підключення створюємо запит до бази в межах наданих раніше повноважень, та встановлюємо режим відображення результату в текстовому вигляді за допомогою клавіш Ctrl+T:

set showplan_text on
go
select top 100 * from TABLE

Відображення плану запиту можна також отримати в графічному вигляді. Для цього необхідно виконати наступне:

- Вимкнути опцію showplan_text — set showplan_text off
- Перейти в режим відображення результату запиту в вигляді таблиці клавішами ***Ctrl+D***
- Виділити створений запит та натиснути комбінацію ***Ctrl+L***

Отриману інформацію про використання запитом фізичного диску можна за допомогою опції "statistics io":

set statistics io on
go
select top 100 * from Table

РОЗДІЛ 5. БАЗИ ДАНИХ В ІНТЕРНЕТІ

З появою технології баз даних було накопичено більше інформації, ніж за всю попередню історію. Однак, доступ користувача до баз даних обмежений з цілого ряду причин:

- для отримання інформації необхідний фізичний доступ до відповідної СУБД;
- користувач повинен бути в курсі моделі даних, знати схему бази даних;
- потрібне вміння користуватися мовою запитів до БД.

Які можливості взаємодії Web-додатків і СУБД? З одного боку, технології Internet / Intranet мають зручний мова розробки розподілених гіпертекстових документів, включаючи прості, зручні, розвинуті та уніфіковані інтерфейси для доступу до інформації. З іншого боку - наявність великої кількості цінних баз даних, керованих різнорідними СУБД, а також прагнення збільшити доступність даних для корпоративних користувачів. Виникає природне бажання схрестити ці дві технології і забезпечити доступ до баз даних в інтерфейсі Web. Ще два роки тому існували тільки ідеї такого схрещування і не дуже ретельно розроблені підходи до реалізації. На сьогодні є два класу механізмів такої взаємодії: 1) забезпечують доступ до БД (за запитом клієнта) на стороні Web-сервера; 2) працюють безпосередньо на стороні клієнта.

Доступ до бази даних на стороні сервера

Механізм доступу до БД на стороні сервера реалізується за рахунок наявності стандартизованих засобів:

- підтримки діалогових форм на рівні гіпертекстового документа (мова HTML);
- можливості запуску серверних програм, взаємодія яких відбувається через стандартний інтерфейс CGI або прикладні інтерфейси Web-сервера.

При реалізації на основі CGI загальна схема реалізації доступу до бази даних на стороні Web-сервера виглядає наступним чином:

- при перегляді документа користувач зустрічає посилання на сторінку, що містить одну або декілька форм, призначених для запиту даних з бази даних;
- користувач запитує цю сторінку, крім незаповнених форм сторінка може містити загальну інформацію про базу даних та про призначення пропонованих форм;
- якщо користувача цікавить інформація з БД, яку можна отримати на основі запропонованих форм, то він заповнює одну з форм і відправляє заповнену форму на сервер;
- отримавши заповнену форму, сервер запускає відповідну зовнішню програму, передаючи їй параметри й одержуючи результати на основі протоколу CGI;
- зовнішня програма перетворить запит, виражений за допомогою заповненої форми, в запит мовою, зрозумілою сервера баз даних (зазвичай це мова SQL).

При використанні CGI вся інтерпретація користувальницького запиту здійснюється серверною програмою. Вона може бути гранично жорсткою, орієнтованої на виконання запиту до фіксованої таблиці фіксованої бази даних, або відносно гнучкою, здатною виконати довільний запит до однієї або декількох таблиць бази даних, ідентифікованої в параметрах клієнта.

API - це, фактично, дешевий, але небезпечний спосіб виконати в адресному просторі сервера WWW програму, яка відповідає специфікаціям на мові HTML. Така програма повинна бути заздалегідь підготовлена і включена в бібліотеку, з якої сервер може проводити динамічну завантаження (DLL-модулі в Windows або Колективна бібліотека sharedlibrary в Unix).

Доступ до бази даних на стороні клієнта

Мабуть, найбільш потужні засоби забезпечення доступу до баз даних на стороні Web-клієнта забезпечує мову Java. Java - це об'єктно-орієнтована мова програмування, що є, по суті справи, "безпечним" підмножиною мови C ++. Зокрема, Java не містить засобів адресної арифметики, не підтримує механізм множинного спадкоємства і т.д. Тому стверджується, що коректність Java-програми можна перевірити до її реального виконання (це абсолютно недоведене твердження). Розрізняють:

- мову Java як такий, для якого існують компілятори в так званий "мобільний код" (машинно-незалежний код, який може інтерпретуватися або з якого можуть генеруватися машинні коди на різних платформах);
- мову JavaScript, який зазвичай використовується для розширення можливостей мови HTML за рахунок додавання процедурної складової;
- і програмний продукт HotJava, є, по суті, інтерпретатором мобільних кодів Java.

Для забезпечення доступу до баз даних на стороні Web-клієнта найбільш істотна наявність мови Java. Технологія розробки HTML-документа дозволяє написати довільну кількість додаткових Java-програм, відкомпілювати їх в мобільні коди і поставити посилання на відповідні коди в тілі HTML-документа. Такі додаткові Java-програми називаються апплетами (Java-applets). Отримавши доступ до документа, який містить посилання на аплети, клієнтська програма перегляду запитує у Web-сервера всі мобільні коди. Коди можуть почати виконуватися відразу після розміщення в комп'ютері клієнта або бути активізовані за допомогою спеціальних команд.

Оскільки аплет являє собою довільну Java-програму, то, зокрема, він може бути спеціалізований для роботи із зовнішніми базами даних. Більш того, система програмування Java включає розвинений набір класів, призначених для підтримки графічного інтерфейсу користувача. Спираючись на використання цих класів, аплет може отримати від користувача інформацію, що характеризує його запит до бази даних, в тому ж вигляді, як якби використовувався стандартний механізм форм мови HTML, а може застосовувати будь-який інший інтерфейс.

Для взаємодії Java-аплета із зовнішнім сервером баз даних розроблено спеціалізований протокол JDBC, який, фактично, поєднує функції шлюзування між інтерпретатором мобільних Java-кодів і ODBC, а також включає ODBC.

Методичний посібник з дисципліни ”Організація баз даних і баз знань”, для здобувачів освітньо-кваліфікаційного рівня фаховий молодший бакалавр спеціальності 122 «Комп’ютерні науки» денної форми навчання /уклад. О.В. Присада.. – Ковель: КПЕК Луцького НТУ, 2020. – 103 с.

Комп’ютерний набір і верстка: **Олександр ПРИСАДА**
Редактор: **Леся ПРОКОПЧУК**

Підп. до друку «___»_____2023. Формат 60х84/16. папірофіс.
Гарн.Таймс. Ум.друк. арк. ____ Обл.-вид. арк. ___ Зам. ___
Тираж ____ прим.

ВСП «КПЕФК ЛНТУ»
45000 м. Ковель, вул. Заводська, 23
Друк – ВСП «КПЕФК ЛНТУ»