Міністерство освіти і науки, молоді та спорту України
Тернопільський державний економічний університет
Факультет комп'ютерних інформаційних технологій

Конспект лекцій з дисципліни

«Організація баз даних»

Освітньо-кваліфікаційний рівень - бакалавр
Галузь знань – 0501 "Інформатика і обчислювальна техніка"
Напрям підготовки – 6.050102 „Комп'ютерна інженерія"

Фахове спрямування - "Спеціалізовані комп'ютерні системи "
Фахове спрямування - "Комп'ютерні системи та мережі"

Кафедра комп'ютерної інженерії

Тернопіль– 2012

ТЕМА 1.

Поняття інформації та інформаційної системи
Класифікація інформаційних систем
Архітектура інформаційної системи
Бази даних та системи управління БД
Архітектура СУБД
Функції СУБД
Розподілені інформаційні системи

1.1. Поняття інформації та інформаційної системи.

Інформа́ція — абстрактне поняття, що має різні значення залежно від контексту. Походить від латинського слова «informatio», яке має декілька значень:
Роз'яснення; Виклад фактів, подій; Витлумачення;
Представлення, поняття;

Ознайомлення, просвіта.

Властивості інформації

Найважливішими, з практичної точки зору, властивостями інформації є цінність, достовірність та актуальність.
Цінність інформації — визначається користю та здатністю її забезпечити суб’єкта необхідними умовами для досягнення ним поставленої мети.
Достовірність — здатність інформації об'єктивно відображати процеси та явища, що відбуваються в навколишньому світу. Як правило достовірною вважається насамперед інформація яка несе у собі безпомилкові та істинні дані. Під безпомилковістю слід розуміти дані які не мають, прихованих або випадкових помилок. Випадкові помилки в даних обумовлені, як правило, неумисними спотвореннями змісту людиною чи збоями технічних засобів при переробці даних в інформаційній системі.Тоді як під істинними слід розуміти дані зміст яких неможливо оскаржити або заперечити.
Актуальність — здатність інформації відповідати вимогам сьогодення (поточного часу або певного часового періоду).

Часові властивості

Часові властивості визначають здатність даних передавати динаміку зміни ситуації (динамічність). При цьому можна розглядати або час запізнення появи в даних відповідних ознак об'єктів, або розходження реальних ознак об'єкта і тих же ознак, що передаються даними. Відповідно можна виділити:
Актуальність — властивість даних, що характеризує поточну ситуацію;
Оперативність — властивість даних, яка полягає в тому, що час їхнього збору та переробки відповідає динаміці зміни ситуації;
Ідентичність — властивість даних відповідати стану об'єкта.

Властивість недоступності

При розгляді захищеності даних можна виділити технічні аспекти захисту даних від несанкціонованого доступу та соціально-психологічні аспекти класифікації даних за мірою їхньої конфіденційності та секретності (властивість конфіденційності)2.

Інші властивості інформації

Суспільна природа — джерелом інформації є пізнавальна діяльність людей, суспільства.
Мовна природа — інформація виражається за допомогою мови — знакової системи будь-якої природи, яка служить засобом спілкування, мислення, висловлювання думки. Мова може бути природною, що використовується у повсякденному житті та служить формою висловлення думок і засобом спілкування між людьми а також штучною, створеною людьми з певною метою (наприклад, мова математичної символіки, інформаційно-пошукова, алгоритмічна та ін. мови).

Невідривність від мови носія

Дискретність — одиницями інформації як засобами висловлювання є слова, речення, уривки тексту, а у плані змісту — поняття, висловлювання, описання фактів, гіпотези, теорії, закони тощо.

Незалежність від творців

Старіння — головною причиною старіння інформації є не сам час, а поява нової інформації, з надходженням якої попередня інформація виявляється невірною, перестає адекватно передавати явища та закономірності матеріального світу, людського спілкування та мислення.
Розсіювання — існування у багатьох джерелах.
Види інформації

Інформацію можна поділити на види за кількома ознаками:
За способом сприйняття
Для людини інформація поділяється на види залежно від типу рецепторів, що сприймають її.
Візуальна — сприймається органами зору. Ми бачимо все довкола.
Аудіальна — сприймається органами слуху. Ми чуємо звуки довкола нас.
Тактильна — сприймається тактильними рецепторами.
Нюхова — сприймається нюховими рецепторами. Ми відчуваємо аромати довкола.
Смакова — сприймається смаковими рецепторами. Ми відчуваємо смак.
За формою подання
За формою подання інформація поділяється на такі види:
Текстова — що передається у вигляді символів, призначених позначати лексеми мови;
Числова — у вигляді цифр і знаків, що позначають математичні дії;
Графічна — у вигляді зображень, подій, предметів, графіків;
Звукова — усна або у вигляді запису передачі лексем мови аудіальним шляхом.

За призначенням

Масова — містить тривіальні відомості і оперує набором понять, зрозумілим більшій частині соціуму
Спеціальна — містить специфічний набір понять, при використанні відбувається передача відомостей, які можуть бути не зрозумілі основній масі соціуму, але необхідні і зрозумілі в рамках вузької соціальної групи, де використовується дана інформація
Особиста — набір відомостей про яку-небудь особистість, що визначає соціальний стан і типи соціальних взаємодій всередині популяції.

1.2. Класифікація інформаційних систем. Архітектура інформаційної системи

Структуру інформаційної системи становить сукупність окремих її частин, які називаються підсистемами.
Підсистема - це частина системи, виділена за якою-небудь ознакою.
Загальну структуру інформаційної системи можна розглядати як сукупність підсистем незалежно від сфери застосування. У цьому випадку говорять про структурний ознаці класифікації, а підсистеми називають забезпечують. Таким чином, структура будь-якої інформаційної системи може бути представлена сукупністю забезпечують підсистем.
Серед забезпечують підсистем зазвичай виділяють інформаційне, технічне, математичне, програмне, організаційне і правове забезпечення.
Інформаційне забезпечення
Призначення підсистеми інформаційного забезпечення полягає у своєчасному формуванні та видачу достовірної інформації для прийняття управлінських рішень.
Інформаційне забезпечення - сукупність єдиної системи класифікації і кодування інформації, уніфікованих систем документації, схем інформаційних потоків, що циркулюють в організації, а також методологія побудови баз даних.
Примітка. Системи класифікації і кодування інформації розглянуті в гл.2.
Уніфіковані системи документації створюються на державному, республіканському, галузевому та регіональному рівнях. Головна мета - це забезпечення порівнянності показників різних сфер суспільного виробництва. Розроблено стандарти, де встановлюються вимоги:
до уніфікованих систем документації;
до уніфікованих форм документів різних рівнів управління;
до складу та структурі реквізитів і показників;
до порядку впровадження, ведення та реєстрації уніфікованих форм документів.
Однак, незважаючи на існування уніфікованої системи документації, при обстеженні більшості організацій постійно виявляється цілий комплекс типових недоліків:
надзвичайно великий обсяг документів для ручної обробки;
одні і ті ж показники часто дублюються в різних документах;

робота з великою кількістю документів відволікає фахівців від рішення безпосередніх завдань;
є показники, які створюються, але не використовуються, та ін
Тому усунення зазначених недоліків є одним із завдань, що стоять при створенні інформаційного забезпечення.
Схеми інформаційних потоків відображають маршрути руху інформації і її обсяги, місця виникнення первинної інформації та використання результатної інформації. За рахунок аналізу структури подібних схем можна виробити заходи щодо вдосконалення всієї системи управління.
Як приклад найпростішої схеми потоків даних можна навести схему, де відображені всі етапи проходження службової записки або записи в базі даних про прийом на роботу співробітника - від моменту її створення до виходу наказу про його зарахування на роботу.
Побудова схем інформаційних потоків, дозволяють виявити обсяги інформації і провести її детальний аналіз, забезпечує:
виключення дублюючої і невживаною інформації;
класифікацію і раціональне представлення інформації.
При цьому докладно повинні розглядатися питання взаємозв'язку руху інформації за рівнями управління (див. рис.3.2). Слід виявити, які показники необхідні для прийняття управлінських рішень, а які ні. До кожного виконавця повинна надходити тільки та інформація, яка використовується.
Методологія побудови баз даних базується на теоретичних засадах їх проектування. Для розуміння концепції методології приведемо основні її ідеї у вигляді двох послідовно реалізованих на практиці етапів:
1-й етап - обстеження всіх функціональних підрозділів фірми з метою:
зрозуміти специфіку та структуру її діяльності;
побудувати схему інформаційних потоків;
проаналізувати існуючу систему документообігу;
визначити інформаційні об'єкти й відповідний склад реквізитів (параметрів, характеристик), що описують їх властивості та назначеніе.2-й етап - побудова концептуальної інформаційно-логічної моделі даних для обстеженої на 1-му етапі сфери діяльності. У цій моделі повинні бути встановлені й оптимізовані всі зв'язки між об'єктами і їхніми реквізитами.
Інформаційно-логічна модель є фундаментом, на якому буде створена база даних.
Для створення інформаційного забезпечення необхідно:
чітке розуміння цілей, завдань, функцій всієї системи управління організацією;
виявлення руху інформації від моменту виникнення і до її використання на різних рівнях управління, представленої для аналізу у вигляді схем інформаційних потоків;
вдосконалення системи документообігу;
наявність і використання системи класифікації і кодування;
володіння методологією створення концептуальних інформаційно-логічних моделей, що відбивають взаємозв'язок інформації;
створення масивів інформації на машинних носіях, що вимагає наявності сучасного технічного забезпечення.

Технічне забезпечення

Технічне забезпечення - комплекс технічних засобів, призначених для роботи інформаційної системи, а також відповідна документація на ці засоби і технологічні процеси.
Комплекс технічних засобів становлять:
комп'ютери будь-яких моделей;
пристрої збору, накопичення, обробки, передачі й виводу інформації;
пристрої передачі даних і ліній зв'язку;
оргтехніка й пристрої автоматичного знімання інформації;
експлуатаційні матеріали та ін

Документацією оформляються попередній вибір технічних засобів, організація їх експлуатації, технологічний процес обробки даних, технологічне оснащення. Документацію можна умовно розділити на три групи:
загальносистемну, що включає державні та галузеві стандарти з технічного обслуговування;
спеціалізовану, що містить комплекс методик по всіх етапах розробки технічного забезпечення;
нормативно-довідкову, використовувану при виконанні розрахунків по технічному забезпечення.
До теперішнього часу склалися дві основні форми організації технічного забезпечення (форми використання технічних засобів): централізована і частково або повністю децентралізована.
Централізоване технічне забезпечення базується на використанні в інформаційній системі великих ЕОМ і обчислювальних центрів.
Децентралізація технічних засобів передбачає реалізацію функціональних підсистем на персональних комп'ютерах безпосередньо на робочих місцях.
Перспективним підходом слід вважати, мабуть, частково децентралізований підхід - організацію технічного забезпечення на базі розподілених мереж, що складаються з персональних комп'ютерів і великий ЕОМ для зберігання баз даних, загальних для будь-яких функціональних підсистем.

Математичне та програмне забезпечення

Математичне та програмне забезпечення - сукупність математичних методів, моделей, алгоритмів і програм для реалізації цілей і завдань інформаційної системи, а також нормального функціонування комплексу технічних засобів.
До засобів математичного забезпечення відносяться:
засоби моделювання процесів управління;
типові завдання управління;
методи математичного програмування, математичної статистики, теорії масового обслуговування та ін
До складу програмного забезпечення входять загальносистемні та спеціальні програмні продукти, а також технічна документація.

Класифікації інформаційних систем

Найпростішою і очевидною класифікацією є класифікація по областях застосування. У цьому зв'язку можна говорити про інформаційних системах в економіці (АСЕ - автоматизовані системи в економіці), в освіті (АСО), у наукових дослідженнях (АСНИ) і т.д.
Ще одним класифікаційним ознакою може виступати характер інформації, якою оперує ІС. З цієї точки зору всі інформаційні системи прийнято ділити на фактографічні та документальні. Під фактографическим типом даних прийнято розуміти дані представляють собою опис деяких фактів предметної області. Наприклад, фактом є дані на конкретну людину (ПІБ, адресу, паспортні дані тощо), книгу (автор, назва, рік видання тощо), машину (марка, рік випуску, виробник і т.п.) і т.д. Іншими словами, факт в інформаційній системі постає у вигляді набору деяких властивостей (атрибутів), кількісне значення яких, як правило, виражається простим типом даних. Характерним представником фактографічних інформаційних систем є широко відома в бухгалтерських колах "1С бухгалтерія".
Документ, на відміну від факту, не може бути виражений простою структурою.

Визначення

Під документом будемо розуміти що зберігається в інформаційній базі, об'єкт довільної структури, що містить інформацію довільного характеру, доступ, до якого можна отримати за його реквізитами.
Під реквізитами документа будемо розуміти сукупність властивостей цього документа, що дозволяють однозначно його ідентифікувати. Прикладами реквізитів можуть служити назва документа, його номер, дата створення, імена творців, електронний підпис і т.д. В якості

прикладів документів можна назвати статті, тексти наказів і розпоряджень, бухгалтерські документи, карти місцевості, звукові записи і т.д. Важливо ще раз підкреслити, що структура об'єкта, який ми назвали документом, може носити самий довільний характер: формати для текстових документів (звичайний текстовий формат, формат Word, формат PDF, формат DJVu, формат HTML і т.д.), таблиці, графічні файли і т.п.
Типовим прикладом документальних інформаційних систем є довідкові юридичні системи типу Гаранта, Консультант + і т.п. Пошукові інтернет системи також є представниками документальних систем. Реальні інформаційні системи часто оперують деякою сумішшю фактографічної і документальної інформації, тим більше що сучасні СУБД, на основі яких, як правило, і будуються сучасні ІС, надають потужні інструментальні засоби для маніпулювання інформацією того й іншого типу.
Нарешті, інформаційні системи можна класифікувати і за тієї ролі, яку вони відіграють у професійній діяльності. Таким чином, можна виділити
Системи управління. ІС даного типу призначені для вирішення завдань автоматизації процесів управління. Виділяють також класи систем управління персоналом і систем управління технологічними процесами.
Обчислювальні інформаційні системи. Дані системи призначені для проведення оперативних розрахунків та обміну інформацією між робочими місцями в рамках однієї організації. У цьому класі виділяються також системи автоматичного проектування (САПР).
Пошуково-довідкові інформаційні системи. Дані системи призначені для збору, зберігання і пошуку інформації довідкового характеру. Такі системи не замінні в конкретних галузях знань: медицині, юриспруденції, програмування та ін
Системи прийняття рішення. Системи цього класу призначені для автоматизації пошуку рішення керівного складу. Особливістю задач прийняття рішень є: недостатність наявної інформації, її суперечливість і не чіткість, слабка формалізація та наявність якісних оцінок. Як ІВ для прийняття рішень використовуються системи, побудовані на основі алгоритмів штучного інтелекту і баз знань. Часто такі системи підтримують природно-мовний інтерфейс.
Інформаційні навчальні системи. До інформаційних навчальним систем відносять: системи програмного навчання, системи для ділових ігор та тренажерні комплекси.
З визначення зокрема слід, що в інформаційній системі є два компоненти: програмне забезпечення та електронне інформаційне сховище. Звернемося до малюнка 1.1, де сказане представлено в графічному вигляді.
Будь-яка інформаційна система розрахована на використання її в будь-якої професійної області. Значить ІС розрахована на взаємодію, з якими або користувачами. Причому під користувачами в загальному випадку слід розуміти не тільки людей, але й інші інформаційні системи, з якими дана ІС обмінюється інформацією.
Блок ПЗ (програмне забезпечення) поділений на малюнку 1.1 на три частини: ІП - інтерфейс користувача, ІД - інтерфейс з даними, БЛ - бізнес логіка. Звичайно, даний розподіл загальному випадку є умовним, і не означає, що в реальному програмному забезпеченні можна явно виділити всі три частини. Однак умоглядне наявність таких частин випливає з дуже простого міркування. Якщо програма взаємодіє з користувачем і даними, значить, якась її частина (логіка) відповідає за цю взаємодію. З іншого боку, як ми згодом будемо неодноразово переконуватися, структура зберігання даних практично ніколи не збігається зі структурою даних, що подаються користувачу. Отже, у програмному забезпеченні повинно бути передбачено перетворення інформації з одного формату до іншого і назад. Ось ця частина програмного забезпечення, яку ми виділили чисто логічно і прийнято називати бізнес логікою.
Формально інформаційні системи можна розділити на автономні і мережеві. Причому до мережевих інформаційних систем ми відносимо й такі, які епізодично синхронізують свої інформаційні сховища з іншими сховищами допомогою будь-яких каналів зв'язку (у тому числі і за допомогою переносних пристроїв зовнішньої пам'яті). Попит на автономні ІВ не великий, в силу майже повної відсутності немережевих комп'ютерів. Таким чином, постає питання про те, яку роль відіграє комп'ютерна мережа в побудові інформаційних систем.

Тут же нам хотілося б поговорити про різні підходи побудови архітектури ІС в мережі. У результаті ми отримаємо ще один показник, на основі якого можна класифікувати інформаційні системи.
Основним сервісом локальних комп'ютерних мереж є файловий сервіс, здійснюваний файловими серверами. Файловий сервер призначений для того, щоб зберігати файли і надавати до них доступ користувачам мережі. Тому природним рішенням побудови інформаційної системи це розташування інформаційного сховища на файловому сервері. Все програмне забезпечення інформаційної системи буде розташовуватися, таким чином, на мережевих комп'ютерах. Користувачі комп'ютерної мережі, на комп'ютерах яких буде встановлено програмне забезпечення інформаційної системи, отримають, таким чином, одночасний доступ до інформаційного сховища. Така архітектура інформаційної системи називається файл-серверної. Дана архітектура широко застосовується для створення інформаційних систем з відносно не великою кількістю одночасно працюючих користувачів (кілька десятків). Причина такого обмеження закладена в тому, що все програмне забезпечення, у тому числі ті його модулі, які відповідають за обробку даних, розташовуються на кожному з мережних комп'ютерів. Таким чином, для виконання операцій з даними необхідно отримати копію цих даних на мережний комп'ютер. Зрозуміло, це призводить до збільшення мережевого трафіку. До того ж в обробці даних, таким чином, виявляються, задіяні і мережеві комп'ютери, і локальна мережа і, звичайно, сам файловий сервер.
Для побудови інформаційних систем з великою кількістю користувачів застосовується інша архітектура. Ця архітектура базується на використанні серверів баз даних. Особливістю серверів баз даних полягає в їх здатності виконувати спеціальні запити до даних. Мова запитів влаштований таким чином, що одна команда цієї мови може містити в собі безліч елементарних операцій над даними. Таким чином можна значно знизити мережевий трафік, а для збільшення продуктивності інформаційної системи буде потрібно збільшення продуктивності тільки сервера баз даних. Крім цього сучасні сервера баз даних дозволяють зберігати на стороні сервера програмні модулі (збережені процедури, тригери та ін), які за командою з боку користувача (клієнта) можуть бути запущені на виконання. У результаті, з'являється реальна можливість виконувати на стороні сервера не тільки обробку даних (див. Малюнок 1.1), а й інші дії. Тепер, знову подивившись на малюнок 1.1, ми бачимо, що програмне забезпечення ІС може бути реально, а не умоглядно, розділене на дві половини. На стороні користувача теоретично може залишитися тільки ПО, яке відповідає за інтерфейс користувача. Така побудова архітектури ІС, коли програмне забезпечення ділиться на дві половини між користувальницьким комп'ютером і сервером баз даних називають технологій "клієнт-сервер", а архітектура ІС - клієнт-серверної. Клієнт, в якому реалізований тільки користувальницький інтерфейс називається тонким клієнтом, у противному випадку клієнт називається товстим.

Зауваження

Інформаційні системи, побудовані з централізованого принципом, називають також банками даних. Таким чином, і файл-серверні і клієнт-серверні інформаційні системи можна формально назвати банками даних.
Важливим плюсом використання серверів баз даних є можливість вбудувати розвинену систему безпеки сервера в систему безпеки інформаційної системи. Зокрема сервера баз даних дозволяють чітко розмежувати доступ різних користувачів до об'єктів інформаційного сховища, журналіровать всі дії вироблені користувачем, інтегрувати систему безпеки ІС з системою безпеки комп'ютерної мережі і т.д.

3. Бази даних та системи управління БД. Архітектура СУБД. Функції СУБД

Систе́ма керування ба́зами да́них (СКБД) — комп'ютерна програма чи комплекс програм, що забезпечує користувачам можливість створення, збереження, оновлення, пошук інформації та контролю доступу в базах даних.

Першим поколінням СКБД прийнято вважати ієрархічні й мережеві системи. Ці системи отримали широке поширення в 1970-х роках, а першою комерційною системою цього типу була система IMS компанії IBM.
У 1980-х роках ці системи були витіснені системами другого покоління — повсюдно використовуваними і донині реляційними СКБД. У цих системах використовувалися непроцедурні мови управління даними (SQL) і передбачався значний ступінь незалежності даних. Реляційні системи внесли значні удосконалення в управління даними: графічний користувацький інтерфейс (GUI), клієнт-серверні застосунки, розподілені бази даних, паралельний пошук даних та інтелектуальний аналіз даних.
Але вже до кінця 1980-х років існуюча тоді реляційна модель перестала задовольняти розробників в силу низки обмежень. Відповіддю на зростаючу складність програм баз даних стали два нових напрямки розвитку СКБД: об'єктно-орієнтовані СКБД і об'єктно-реляційні СКБД.
У 1991 був утворений консорціум ODMG (en:Object Data Management Group), основною метою якого стало вироблення промислового стандарту об'єктно-орієнтованих баз даних. Між 1993 та 2001 роками ODMG опубоіковала п'ять ревізій своїх специфікацій. Остання версія стандарту має індекс 3.0, після чого група розпустилася. До кінця 1990-х існувало близько десяти компаній, що виробляли комерційні продукти, що позиціонуються на ринку як ООСКБД. Найбільш відомими системами даного класу стали Objectivity, Versant виробництва однойменних компаній, а також СКБД Jasmine, випущена компанією CA.
Незважаючи на переваги, що дозволяють ефективніше вирішувати певний ряд завдань, об'єктно-орієнтовані системи так і не змогли завоювати значущу частку ринку СКБД, залишившись «нішевим» продуктом.
Постачальниками традиційних реляційних СКБД також була проведена значна робота з об'єднання об'єктно-орієнтованих і реляційних систем. Розробники постаралися розширити мову SQL, щоб включити в неї концепції об'єктно-орієнтованого підходу, зберігаючи переваги реляційної моделі (об'єктні розширення мови SQL були зафіксовані в стандарті SQL:1999). Основний принцип — це еволюційний розвиток можливостей СКБД без корінний ломки попередніх підходів та зі збереженням наступності з системами попереднього покоління.
Поняття СКБД третього покоління, якими, власне кажучи, і є об'єктно-реляційні СКБД, з'явилося після опублікування групою відомих фахівців в області баз даних «Маніфесту систем баз даних третього покоління». Основні принципи СКБД третього покоління, позначені в маніфесті:
Крім традиційних послуг з управління даними, СКБД третього покоління повинні забезпечити підтримку розвиненіших структур об'єктів і правил. Розвинутіша структура об'єктів характеризує засоби, необхідні для зберігання і маніпулювання нетрадиційними елементами даних (тексти, просторові дані, мультимедіа).
СКБД третього покоління повинні включити в себе СКБД другого покоління. Системи другого покоління внесли вирішальний вклад у двох областях — непроцедурний доступ за допомогою мови запитів SQL і незалежність даних. Ці досягнення обов'язково повинні враховуватися в системах третього покоління.
СКБД третього покоління повинні бути відкриті для інших підсистем. Це включає оснащення різноманітними інструментами підтримки прийняття рішень, доступом з багатьох мов програмування, інтерфейсами до існуючих популярних систем і бізнес-застосунків, можливістю запуску програм з бази даних на іншій машині і розподілені СКБД. Весь набір інструментів і СКБД має ефективно функціонувати на різноманітних апаратних платформах з різними операційними системами. Крім того, СКБД, що розраховує на широку сферу застосування, повинна бути оснащена мовою четвертого покоління (4GL).
У середині 1990 років було лише кілька дослідних прототипів СКБД, які поєднали найкращі риси реляційних і об'єктно-орієнтованих СКБД. Першим комерційним продуктом, якому були властиві об'єктно-реляційні риси, став Universal Server компанії Informix (згодом була поглинена IBM). В даний час більшість цих ідей вже втілено в реальних комерційних

рішеннях, в тому числі і в продуктах основних постачальників СКБД (Oracle Database і IBM DB2).
Розвиток індустрії систем керування базами даних базується на значних фундаментальних наукових дослідженнях. Найчастіше, між самими дослідженнями та їхньою конкретною реалізацією в прикладних рішеннях минають роки, а іноді й десятиліття. Роботу в області управління даними проводять як університетські дослідницькі групи (MIT, Berkeley), так і центри розробок основних постачальників СКБД (Oracle, IBM, Microsoft). Інвестування в управління даними — це довгострокове, і разом з тим, вигідне вкладення коштів. В даний час дослідники мають у своєму розпорядженні засоби, що дозволяють ефективно реалізувати найскладніші запити, що маніпулюють терабайтами й петабайтами різних даних.
Основними тенденціями, які дали привід для проведення різних масштабних досліджень в області баз даних стали:
Експонентний ріст даних. Обсяг даних, у тому числі синтетичних, що генеруються автоматизованими системами, значно зріс. Збільшилося і число прикладних областей, в яких вимагається обробка великих обсягів даних. До таких областей тепер відносяться не тільки традиційні корпоративні програми, пошук у веб, але також і наукові дослідження, обробка природних мов, аналіз соціальних мереж тощо.
Значне ускладнення структур використовуваних даних. Прості види даних у вигляді чисел і символьних рядків стали доповняться численною мультимедійною інформацією, просторовими, процедурними даними та великою кількістю інших складних форматів.
Широке поширення дешевих високопродуктивних апаратних засобів. Щорічно ми спостерігаємо зростання обчислювальних можливостей мікропроцесорів, збільшення ємності і зниження вартості доступних і зручних в експлуатації пристроїв дискової і оперативної пам'яті.
Активний розвиток засобів комунікації та «всесвітньої павутини» World Wide Web. WWW стає єдином інформаційним середовищем, що пронизує весь світ і об'єднує величезне число користувачів та електронних пристроїв.
Поява нових важливих областей застосування СКБД. У першу чергу, це пов'язано з інтелектуальним аналізом даних, сховищами даних, а останнім часом - з паралельними обчисленнями і хмарними технологіями.

Основні характеристики СКБД

Контроль за надлишковістю даних
Несуперечливість даних
Підтримка цілісності бази даних (коректність та несуперечливість)
Цілісність описується за допомогою обмежень
Незалежність прикладних програм від даних
Спільне використання даних
Підвищений рівень безпеки
Можливості СКБД
Дозволяється створювати БД (здійснюється за допомогою мови визначення даних DDL (Data Definition Language))
Дозволяється додавання, оновлення, видалення та читання інформації з БД (за допомогою мови маніпулювання даними DML, яку часто називають мовою запитів)
Можна надавати контрольований доступ до БД за допомогою:
Системи забезпечення захисту, яка запобігає несанкціонованому доступу до БД;
Системи керування паралельною роботою прикладних програм, яка контролює процеси спільного доступу до БД;
Система відновлення — дозволяє відновлювати БД до попереднього несуперечливого стану, що був порушений в результаті збою апаратного або програмного забезпечення
Основні компоненти середовища СКБД
апаратне забезпечення
програмне забезпечення
дані

процедури — інструкції та правила, які повинні враховуватись при проектуванні та використанні БД
користувачі
адміністратори даних (керування даними, проектування БД, розробка алгоритмів, процедур) та БД (фізичне проектування, відповідальність за безпеку та цілісність даних)
розробники БД
прикладні програмісти
кінцеві користувачі

Архітектура СКБД

Існує трирівнева система організації СКБД ANSI-SPARC, при якій існує незалежний рівень для ізоляції програми від особливостей представлення даних на нижчому рівні.
Рівні:
Зовнішній — представлення БД з точки зору користувача.
Концептуальний — узагальнене представлення БД, описує які дані зберігаються в БД і зв'язки між ними. Підтримує зовнішні представлення, підтримується внутрішнім рівнем.
Внутрішній — фізичне представлення БД в комп'ютері.
Логічна незалежність — повна захищеність зовнішніх моделей від змін, що вносяться в концептуальну модель.
Фізична незалежність — захищеність концептуальної моделі від змін, які вносяться у внутрішню модель.

4. Розподілені інформаційні системи.

Різновидом баз даних з погляду їх формування, зберігання й використання є *розподілені бази даних*. Ці бази даних широко використовуються в організації комплексів взаємопов'язаних АРМ менеджерів, де встановлено ПЕОМ, а також у системі об'єктивного інформаційного забезпечення менеджменту.
В інформаційно-обчислювальних системах, які використовують ПЕОМ, потреба переходу від традиційних до розподілених баз даних зумовлена прагненням розв'язати суперечність між перевагою розподіленого зберігання і ведення баз даних та потребою їх інтегрованого використання як цілого.
Розподілена база даних — це сукупність логічно пов'язаних баз даних або частин однієї бази, які розпаралелені між кількома територіально розподіленими ПЕОМ і забезпечені відповідними можливостями для управління цими базами чи їхніми частинами. Тож розподілена база даних реалізується на різних просторово розосереджених обчислювальних засобах разом з організаційними, технічними і програмними засобами її створення та ведення.
До *основних переваг* розподіленої бази даних можна віднести:
підвищення продуктивності системи за рахунок розпаралелення процесів збирання та обробки даних;
підвищення ефективності управління базами даних і поліпшення експлуатаційних характеристик системи управління даними;
досягнення збалансованості навантаження й синхронізації процесів збирання й обробки даних;
підвищення надійності й живучості системи;
вдосконалення гнучкості, нарощуваності та модифікованості бази даних;
скорочення вартості організації і затрат на експлуатацію бази даних;
збільшення обсягу збережених і доступних для обробки даних;
зменшення обсягів даних, що пересилаються.
Розподілені бази даних можна *ефективно використовувати* в предметних областях, які характеризуються:
надто великими обсягами даних, що збираються, зберігаються й обробляються;
фізичною розосередженістю місць збирання, зберігання й використання даних;

наявністю розвинених засобів обчислювальної техніки та мереж передачі даних;
можливістю збирання й обробки більшої частини інформації в місцях, де вона виникає чи зберігається;
необхідністю одночасного масового збирання й обробки інформації тощо.
Ці особливості притаманні передусім виробничому об'єкту управління та його предметним областям. Як предметні області можна використовувати ресурси (матеріальні, трудові, фінансові тощо).
За способом розміщення розподілені бази даних поділяють на зосереджені і розосереджені.
Зосереджені (або централізовані) розподілені бази даних фізично розміщені в одному місці.
Для обміну інформацією між окремими (локальними) підбазами використовуються канали зв'язку прямого доступу. Термін «*канал зв'язку*» (channel) щодо обміну інформацією означає засіб передавання інформації (письмової, усної, формальної, неформальної тощо), придатної для електронних засобів зв'язку. Обмін даними між взаємопов'язаними підбазами здійснюється без помітних обмежень на обсяги й характер інформації, що передається. Такі бази даних мають цілу *низку переваг*:
простоту побудови;
зведене до мінімуму дублювання інформації;
максимальну уніфікацію методів зберігання, коригування та пошуку інформації.
Проте бази даних, зосереджені в одному місці — вузлі мережі, — мають чимало недоліків:
при централізації зберігання значно збільшується час на передавання інформації, а через це зростає й час реакції системи;
централізована система обмежена обсягами пам'яті ЕОМ тощо.
Розосереджені (або децентралізовані) розподілені бази даних фізично розміщені в різних місцях — вузлах обчислювальної мережі. Обмін інформацією між підбазами здійснюється з використанням каналів зв'язку. Як підбази розподіленої бази даних можуть використовуватися зосереджені (централізовані) бази даних і окремі (локальні) підбази. Обмін між взаємозв'язаними підбазами здійснюється здебільшого результатною (обробленою, узагальненою) інформацією. При виконанні запиту в таких системах використовується декомпозиція запиту на підзапит до локальних підбаз і паралельне виконання виділених підзапитів у різних вузлах обчислювальної мережі.
Ці бази даних мають безперечні *переваги* порівняно з централізованими:
обсяги пам'яті обмежені пам'яттю не однієї ЕОМ, а сумарною пам'яттю ЕОМ, які містяться в усіх вузлах мережі;
зменшуються затрати на передавання інформації, оскільки в кожному вузлі перебуває та інформація, яка необхідна конкретному користувачеві і за змогою забезпечує всі його інформаційні потреби.
Проте розосереджена база даних призводить до неминучого дублювання певної частини інформації, безконтрольності зростання її обсягів, ускладнюючи водночас проблему зберігання несуперечливості інформації.
Розглянута концепція розподілених баз даних набула значного поширення при функціонуванні розподіленої обробки інформації. При цьому організація такої обробки інформації відбувається, як правило, в рамках системи автоматизованого збирання й обробки інформації як в окремих структурних ланках, так і по об'єкту управління в цілому.
Організаційною передумовою застосування розподіленої обробки інформації є те, що в умовах ринкової економіки, коли широко впроваджуються в практику господарського механізму госпрозрахунок, оренда тощо, розвиваються процеси децентралізації управління об'єктом.
Технічною передумовою зазначених процесів є те, що набувають поширення (і вельми ефективно) ПЕОМ, які прості в експлуатації та обслуговуванні, мають відносно низьку вартість і малі габаритні розміри тощо. Усе це дає змогу наблизити їх до місць масового виникнення, первинного збирання, обробки й використання інформації щодо процесів, які відбуваються у виробничо-господарській діяльності об'єкта управління, а також розподілити ці ПЕОМ на всіх рівнях управління цього об'єкта.

В умовах системної обробки інформації, коли концепція розподіленої обробки інформації реалізується на базі комплексів ПЕОМ і локальних мереж, широко застосовуються АРМ керівників і фахівців різних рівнів об’єкта управління.
На великих об’єктах управління, наприклад, на промислових підприємствах, комплекси АРМ доцільніше будувати за *ресурсозабезпечувальною ознакою* (див. п. 3.1), тобто для збирання й обробки інформації за окремими блоками (ресурсами): готовою продукцією, основними засобами (фондами), матеріальними, трудовими й фінансовими ресурсами. Крім того, виділяються центральні ПЕОМ (сервери) як для внутрішнього збирання зведеної інформації по об’єкту управління в цілому, створення і зберігання розподіленої бази (баз) даних тощо, так і для зв’язків з об’єктами управління зовнішнього середовища. Окремі АРМ менеджерів вищого рівня пов’язані як із центральними і зовнішніми ПЕОМ, так і з ПЕОМ керівників середнього й низового рівнів.
За наявності зосередженої (централізованої) розподіленої бази (баз) даних у них формуються не лише всі масиви з умовно-постійною інформацією, а також і єдині масиви регламентуючої інформації та бібліотеки описаних форм первинних і вихідних документів, стандартних текстів-заготовок і т. ін. АРМ менеджерів середнього та низового рівнів дістають у готовому вигляді потрібну їм інформацію від цієї бази (баз) даних. Такий спосіб створення й використання розподілених баз даних вигідний за невеликих обсягів виробництва й коротких відстаней ПЕОМ низових і середніх рівнів від центральної ПЕОМ.
Розосереджена розподілена база (бази) даних, де комплекси ПЕОМ згруповані за ресурсозабезпечувальною ознакою і становлять основу системи об’єктивного інформаційного забезпечення менеджерів різних рівнів на великих об’єктах управління, у ринкових відносинах має велике значення. За цих умов є реальна можливість своєчасно забезпечувати всіх користувачів об’єктивною і вірогідною внутрішньою та зовнішньою інформацією.

ТЕМА 2.

Ієрархічна та мережна моделі даних

Проблеми маніпулювання даними та обмеження цілісності даних.

2.1. Моде́ль да́них

Моде́ль да́них (англ. Data model) — абстрактне представлення реального світу, що відображає тільки ті об'єкти, що безпосередньо стосуються програми. Це, як правило, визначає специфічну групу об'єктів, їх атрибутивне значення і відношення між ними. В випадку ГІС, використовується механізм представлення і організації просторової моделі даних, або растрової моделі даних. Вона не залежить від комп'ютерної системи і пов'язана тільки з структурою даних.
Основою бази даних є модель даних — фіксована система понять і правил для представлення даних структури, стану і динаміки проблемної області в базі даних. У різний час послідовне застосування одержували ієрархічна, мережна і реляційна моделі даних. У наш час усе більшого поширення набуває об'єктно-орієнтований підхід до організації баз даних ГІС.

Ієрархічна модель даних

Ієрархічні бази даних можуть бути представлені як дерево, що складається з об'єктів різних рівнів. Верхній рівень займає один об'єкт, другий - об'єкти другого рівня і т. д.
Між об'єктами існують зв'язки, кожен об'єкт може включати в себе декілька об'єктів нижчого рівня. Такі об'єкти перебувають у відношенні предка (об'єкт більш близький до кореня) до нащадка (об'єкт більш низького рівня), при цьому можлива ситуація, коли об'єкт-предок не має нащадків або має їх кілька, тоді як в об'єкта-нащадка обов'язково тільки один предок. Об'єкти, що мають загального предка, називаються близнюками.
Наприклад, якщо ієрархічна база даних містила інформацію про покупців та їх замовленнях, то буде існувати об'єкт "покупець" (батько) і об'єкт "замовлення" (дочірній). Об'єкт "покупець" матиме покажчики від кожного замовника до фізичного розташування замовлень покупця на об'єкт "замовлення".

У цій моделі запит, направлений вниз по ієрархії, простий (наприклад: які замовлення належать цьому покупцеві), а проте запит, направлений вгору по ієрархії, більш складний (наприклад, який покупець помістив це замовлення). Також, важко уявити не-ієрархічні дані при використанні цієї моделі.
Ієрархічної базою даних є файлова система, що складається з кореневої директорії, в якій є ієрархія піддиректорій і файлів.

Ієрархічна модель даних

Перші системи управління базами даних використовували ієрархічну модель даних, і в часі їх поява передує появі мережевої моделі.
Структурна частина ієрархічної моделі
Основними інформаційними одиницями в ієрархічній моделі даних є сегмент і поле. Поле даних визначається як найменша неподільна одиниця даних, доступна користувачеві. Для сегмента визначаються тип сегмента та примірник сегмента. Примірник сегмента утвориться з конкретних значень полів даних. Тип сегмента - це пойменована сукупність вхідних у нього типів полів даних.
Як і мережева, ієрархічна модель даних базується на графових формі побудови даних, і на концептуальному рівні вона є просто окремим випадком мережевої моделі даних. В ієрархічній моделі даних вершині графа відповідає тип сегмента або просто сегмент, а дуг - типи зв'язків предок - нащадок. В ієрархічних структурах сегмент - нащадок повинен мати в точності одного предка.
Ієрархічна модель являє собою зв'язний неорієнтований граф деревовидної структури, що об'єднує сегменти. Ієрархічна БД складається з упорядкованого набору дерев.

Перетворення концептуальної моделі в ієрархічну модель даних

Перетворення концептуальної моделі в ієрархічну структуру даних багато в чому схоже з перетворенням її в мережеву модель, але й має деякі відмінності у зв'язку з тим, що ієрархічна модель вимагає організації всіх даних у вигляді дерева.
Перетворення зв'язку типу "один до багатьох" між предком і нащадком здійснюється практично автоматично в тому випадку, якщо нащадок має одного предка, і відбувається це таким чином. Кожен об'єкт з його атрибутами, що бере участь в такого зв'язку, стає логічним сегментом. Між двома логічними сегментами встановлюється зв'язок типу "один до багатьох". Сегмент з боку "багато" стає нащадком, а сегмент з боку "один" стає предком.
Ситуація значно ускладнюється, якщо нащадок в зв'язку має не одного, а двох і більше предків. Так як подібне положення є неможливим для ієрархічної моделі, то відображена структура даних потребує перетвореннях, які зводяться до заміни одного дерева, наприклад, двома (якщо є два предка). В результаті такого перетворення в базі даних з'являється надмірність, так як єдино можливий вихід з цієї ситуації - дублювання даних.

Керуюча частина ієрархічної моделі

У рамках ієрархічної моделі виділяють мовні засоби опису даних (ЯОД) та засоби маніпулювання даними (ММД). Кожна фізична база описується набором операторів, що обумовлюють як її логічну структуру, так і структуру зберігання БД. При цьому спосіб доступу встановлює спосіб організації взаємозв'язку фізичних записів.
Визначено такі способи доступу:
ієрархічно послідовний;
ієрархічно індексного-послідовний;
ієрархічно прямій;
ієрархічно індексного-прямій;
індексний.
Крім завдання імені БД і способу доступу опису повинні містити визначення типів сегментів, що складають БД, відповідно до ієрархії, починаючи з кореневого сегмента. Кожна фізична БД містить тільки один кореневий сегмент, але в системі може бути декілька фізичних БД.

Серед операторів маніпулювання даними можна виділити оператори пошуку даних, оператори пошуку даних з можливістю модифікації, оператори модифікації даних. Набір операцій маніпулювання даними в ієрархічній БД невеликий, але цілком достатній.
Приклади типових операторів пошуку даних
знайти вказане дерево БД;
перейти від одного дерева до іншого;
знайти примірник сегмента, що задовольняє умовам пошуку;
перейти від одного сегмента до іншого всередині дерева;
перейти від одного сегмента до іншого в порядку обходу ієрархії.
Приклади типових операторів пошуку даних з можливістю модифікації:
знайти і утримати для подальшої модифікації єдиний екземпляр сегмента, що задовольняє умовам пошуку;
знайти і утримати для подальшої модифікації наступний примірник сегменту з тими ж умовами пошуку;
знайти і утримати для подальшої модифікації наступний примірник для того ж батька.
Приклади типових операторів модифікації ієрархічно організованих даних, які виконуються після виконання одного з операторів другої групи (пошуку даних з можливістю модифікації):
вставити новий екземпляр сегмента в зазначену позицію;
оновити поточний екземпляр сегмента;
видалити поточний екземпляр сегмента.
В ієрархічній моделі автоматично підтримується цілісність посилань між предками і нащадками. Основне правило: ніякої нащадок не може існувати без свого батька.

Відомі ієрархічні СУБД

Типовим представником (найбільш відомим і поширеним) є Information Management System (IMS) фірми IBM. Перша версія з'явилася в 1968 р.
Time-Shared Date Management System (TDMS) компанії Development Corporation;
Mark IV Multi - Access Retrieval System компанії Control Data Corporation;
System 2000 розробки SAS-Institute;
Сервери каталогів, такі, як LDAP і Active Directory (допускають чітке уявлення у вигляді дерева)
За принципом ієрархічної БД побудовані ієрархічні файлові системи та Реєстр Windows.
InterSystems Cach
Google App Engine DataStore API

2.3. Мережна модель даних

Мережева модель даних - логічна модель даних, яка є розширенням ієрархічного підходу, строга математична теорія, що описує структурний аспект, аспект цілісності й аспект обробки даних у мережевих базах даних.
Різниця між ієрархічної моделлю даних і мережевої полягає в тому, що в ієрархічних структурах запис-нащадок повинна мати в точності одного предка, а в мережевій структурі даних у нащадка може матися будь-яке число предків.
Мережева БД складається з набору примірників певного типу запису і набору примірників певного типу зв'язків між цими записами.
Тип зв'язку визначається для двох типів запису: предка і нащадка. Примірник типу зв'язку складається з одного примірника типу запису предка і впорядкованого набору примірників типу запису нащадка. Для даного типу зв'язку L з типом запису предка P і типом запису нащадка C повинні виконуватися наступні дві умови:
кожен екземпляр типу запису P є предком тільки в одному екземплярі типу зв'язку L;
кожен екземпляр типу запису C є нащадком не більше ніж в одному примірнику типу зв'язку L.

Аспект маніпуляції

Зразковий набір операцій маніпулювання даними:
знайти певний запис в наборі однотипних записів;

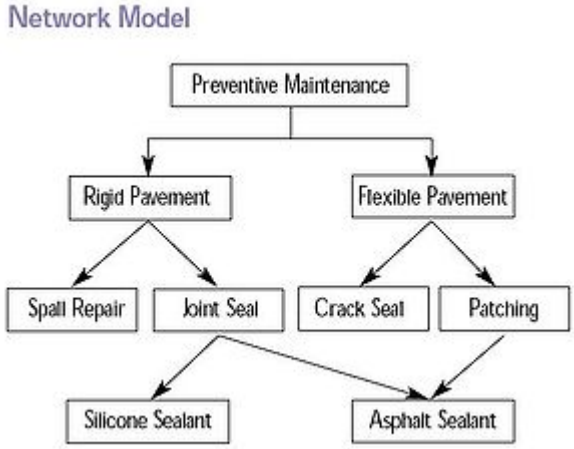
перейти від предка до першого нащадку за деякою зв'язку;
перейти до наступного нащадку у деякому зв'язку;
перейти від нащадка до предка по деякому зв'язку;
створити новий запис;
знищити запис;
модифіковані запис;
включити у зв'язок;
виключити із зв'язку;
переставити в інший зв'язок і т. д.

Аспект цілісності

Є (необов'язкова) можливість вимагати для конкретного типу зв'язку відсутність нащадків, не беруть участь ні в одному примірнику цього типу зв'язку (як в ієрархічній моделі).
Переваги
Перевагою мережевої моделі даних є можливість ефективної реалізації за показниками витрат пам'яті й оперативності.

Недоліки

Недоліком мережевої моделі даних є висока складність і жорсткість схеми БД, побудованої на її основі.
Мережева СУБД, графічне представлення зв'язків



До основних понять мережевої моделі бази даних належать: рівень, елемент (вузол), зв'язок.
Вузол - це сукупність атрибутів даних, що описують деякий об'єкт. На схемі ієрархічного дерева вузли представляються вершинами графа. У мережній структурі кожен елемент може бути пов'язаний з будь-яким іншим елементом.
Мережні бази даних подібні ієрархічним, за винятком того, що в них є покажчики в обох напрямках, які з'єднують споріднену інформацію.
Незважаючи на те, що ця модель вирішує деякі проблеми, пов'язані з ієрархічною моделлю, виконання простих запитів залишається досить складним процесом.
Також, оскільки логіка процедури вибірки даних залежить від фізичної організації цих даних, то ця модель не є повністю незалежною від програми. Іншими словами, якщо необхідно змінити структуру даних, то потрібно змінити і додаток.

Приклади мережевих СУБД
СООБЗ Cerebrum [1]
ІСУБД CronosPRO [2]
dbVista
Cach
GT.M

2.4. Проблеми маніпулювання даними та обмеження цілісності даних

Цілісність інформації (англ. data integrity, information integrity) — властивість інформації, яка полягає в тому, що інформація не може бути модифікована неавторизованим користувачем і/або процесом. Інформація зберігає цілісність, якщо дотримуються встановлені правила її модифікації та видалення.

Цілісність інформації — умови, за яких інформація зберігається, передається та приймається без змін.[1]

Цілісність інформації — задачі забезпечення цілісності і доступності інформаційних об'єктів у обчислювальних мережах, модель системи зв'язку. Методи захисту від помилок: мажоритарний (принцип Вердана) і метод передачі інформаційними блоками з кількісними характеристиками блоку.

У кожний момент часу існування бази даних відомості, що містяться в ній, повинні бути повними, несуперечливими і адекватно відображають предметну область. У цьому й полягає її цілісність. Цілісність бази даних досягається внаслідок введення обмеження цілісності (вказівка діапазону допустимих значень, співвідношення між значеннями даних, обмеження на видалення інформації і т.д.). Обмеження реалізуються різними засобами СУБД, наприклад, за допомогою декларативних (повідомлених при розробці бази даних її розробником) обмежень цілісності.

У другій частині реляційної моделі даних визначаються два обмеження, які повинні виконуватися в будь-якої реляційної бази даних. Це:

Цілісність сутностей.

Цілісність зовнішніх ключів.

Перш, ніж говорити про цілісність сутностей, опишемо використання null-значень в реляційних базах даних.

Null-значення

Основне призначення баз даних полягає в тому, щоб зберігати і надавати інформацію про реальний світ. Для представлення цієї інформації в базі даних використовуються звичні для програмістів типи даних – Рядкові, чисельні, логічні і т.п. Однак у реальному світі часто зустрічається ситуація, коли дані невідомі або не повні. Наприклад, місце проживання або дата народження людини можуть бути невідомі (База даних розшукуваних злочинців). Якщо замість невідомої адреси доречно було б вводити порожній рядок, то що вводити замість невідомої дати? Відповідь – порожню дату – не цілком задовільний, тому що найпростіший запит "видати список людей у порядку зростання дат народження" дасть явно неправильних відповідь.

Для того щоб обійти проблему неповних або невідомих даних, в базах даних можуть використовуватися типи даних, поповнені так званим null-значенням. Null-значення – це, власне, не значення, а якийсь маркер, який показує, що значення невідоме.

Таким чином, у ситуації, коли можлива поява невідомих або неповних даних, розробник має на вибір два варіанти.

Перший варіант полягає в тому, щоб обмежитися використанням звичайних типів даних і не використовувати null-значення, а замість невідомих даних вводити або нульові значення, або значення спеціального виду – наприклад, домовитися, що рядок "Адреса невідома" і є ті дані, які потрібно вводити замість невідомої адреси. У будь-якому випадку на користувача (або на розробника) лягає відповідальність на правильне трактування таких даних. Зокрема, може знадобитися написання спеціального програмного коду, що у потрібних випадках "виловлював" би такі дані. Проблеми, що виникають при цьому очевидні – Не всі дані стають рівноправні, потрібен додатковий програмний код, "відслідковує" цю нерівноправність, в результаті чого ускладнюється розробка і супровід додатків.

Другий варіант полягає у використанні null-значень замість невідомих даних. За здавалося б природністю такого підходу ховаються менш очевидні і більш глибокі проблеми. Найбільш кидається в очі проблемою є необхідність використання тризначної логіки при оперуванні з даними, які можуть містити null-значення. У цьому випадку при неакуратному формулюванні запитів, навіть самі природні запити можуть давати неправильні відповіді. Є більш фундаментальні проблеми, пов'язані з теоретичним обгрунтуванням коректності

введення null-значень, наприклад, незрозуміло взагалі, чи входять null-значення в домени чи ні.
Докладне обговорення проблем використання null-значень виходить за межі даної роботи. Можна тільки сказати про те, що це питання в теорії реляційних баз даних остаточно не вирішене. Основоположник реляційного підходу Кодд вважав null-значення невід'ємною частиною реляційної моделі. К. Дейт, один з найбільших теоретиків реляційної моделі виступає категорично проти null-значень (докладне обговорення проблем, що виникають при використанні null-значень наведено у книзі.
Практично всі реалізації сучасних реляційних СУБД дозволяють використовувати null-значення, незважаючи на їх недостатню теоретичну обгрунтованість. Таку ситуацію можна порівняти з ситуацією, що склалася на початку століття з теорією множин. Майже відразу після створення Кантором теорії множин, в ній були виявлені внутрішні суперечності (антиномії). Були розроблені більш суворі теорії, що дозволяють уникнути цих протиріч (конструктивна теорія множин). Однак у реальній роботі більшість математиків користується класичною теорією множин, тому що більш суворі теорії більш обмежені і негнучкий в застосуванні саме в силу своєї більшої строгості.
Думка автора (дуже скромне порівняно з думкою корифеїв реляційної теорії) полягає в тому, що бажано уникати null-значень. Тим не менш, приведемо тут опис тризначної логіки, необхідної для роботи з null-значеннями.
Тризначна логіка (3VL)
Оскільки null-значення позначає насправді той факт, що значення невідоме, то будь-які алгебраїчні операції (додавання, множення, конкатенація рядків і т.д.) повинні давати також невідоме значення, тобто null. Дійсно, якщо, наприклад, вага деталі невідома, то невідомо також, скільки важать 10 таких деталей.
При порівнянні виразів, що містять null-значення, результат також може бути невідомий, наприклад, значення істинності для виразу є null, якщо один або обидва аргументи є null. Таким чином, визначення істинності логічних виразів базується на тризначної логіці (three-valued logic , 3VL), В якій крім значень T – ІСТИНА і F – БРЕХНЯ, введено значення U – НЕВІДОМО. Логічне значення U – це те ж саме, що і null-значення.

ТЕМА 3.

Реляційна модель та її характеристики
Структура реляційних даних
Домени
Декартовий добуток доменів та відношення
Схема БД
Таблиці БД.

3.1. Реляційна модель та її характеристики

Реляційна модель даних (РМД) - логічна модель даних, прикладна теорія побудови баз даних, яка є додатком до завдань обробки даних таких розділів математики як теорії множин і логіка першого порядку.
На реляційної моделі даних будуються реляційні бази даних.
Реляційна модель даних включає такі компоненти:
Структурний аспект (складова) - дані в базі даних є набором відносин.
Аспект (складова) цілісності - відносини (таблиці) відповідають певним умовам цілісності. РМД підтримує декларативні обмеження цілісності рівня домену (типу даних), рівня відносини і рівня бази даних.
Аспект (складова) обробки (маніпулювання) - РМД підтримує оператори маніпулювання відносинами (реляційна алгебра, реляційне числення).
Крім того, до складу реляційної моделі даних включають теорію нормалізації.
Термін "реляційний" означає, що теорія заснована на математичному понятті ставлення (relation). Як неформального синоніма терміну "відношення" часто зустрічається слово таблиця. Необхідно пам'ятати, що "таблиця" є поняття нестроге і неформальне і часто означає не "ставлення" як абстрактне поняття, а візуальне уявлення відносини на папері або

екрані. Некоректне і нестрогое використання терміну "таблиця" замість терміна "ставлення" нерідко призводить до нерозуміння. Найбільш часта помилка полягає в міркуваннях про те, що РМД має справу з "плоскими", або "двовимірними" таблицями, тоді як такими можуть бути тільки візуальні представлення таблиць. Відносини ж є абстракціями, і не можуть бути ні "плоскими", ні "неплоским".

Для кращого розуміння РМД слід відзначити три важливі обставини:

модель є логічною, тобто відносини є логічними (абстрактними), а не фізичними (збереженими) структурами;

для реляційних баз даних вірний інформаційний принцип : все інформаційне наповнення бази даних представлено одним і тільки одним способом, а саме - явним завданням значень атрибутів у кортежі відносин; зокрема, немає ніяких покажчиків (адрес), що зв'язують одне значення з іншим;

наявність реляційної алгебри дозволяє реалізувати декларативне програмування і декларативне опис обмежень цілісності, на додаток до навігаційного (процедурним) програмування і процедурної перевірці умов.

Принципи реляційної моделі були сформульовані в 1969 - 1970 роках Е. Ф. Коддом (EF Codd). Ідеї Кодда були вперше публічно викладені в статті "A Relational Model of Data for Large Shared Data Banks" [1], що стала класичною.

Суворе виклад теорії реляційних баз даних (реляційної моделі даних) в сучасному розумінні можна знайти в книзі К. Дж. Дейта. "CJ Date. An Introduction to Database Systems" ("Дейт, К. Дж. Введення в системи баз даних").

Найбільш відомими альтернативами реляційної моделі є ієрархічна модель, і мережева модель. Деякі системи, що використовують ці старі архітектури, використовуються досі. Крім того, можна згадати про об'єктно-орієнтованої моделі, на якій будуються так звані об'єктно-орієнтовані СУБД, хоча однозначної і загальноприйнятого визначення такої моделі немає.

Як правило, розрізняють два рівні абстракції представлення даних у вигляді інформаційної та фізичної моделей. Користувач мало звертає увагу на організацію фізичного зберігання інформації - його цікавить логічне представлення даних. Інформаційна модель повинна відображати предметну ділянку в зрозумілих і звичних для користувача термінах. Змістовно база даних містить структуровану певним чином інформацію про факти, події, об’єкти та їх властивості і зв’язки між ними.

Концепція реляційної моделі бази даних запропонована Е.Ф.Коддом в 1970 році для розв’язання наступної задачі - забезпечити незалежність представлення та опису даних від прикладних програм. В основі цієї моделі лежать поняття відношення (relations), подане у вигляді таблиці з дотриманням деяких обмежувальних умов. Основні взаємопов’язані поняття фізичного, спеціального прикладного та математичного рівнів побудови реляційної бази даних та їх взаємовідношення показані в таблиці, в рядках якої знаходяться еквівалентні поняття

Стовпці відношення називають атрибутами і їм присвоюють імена. Список імен атрибутів відношення називається схемою відношення. Таблиці не дозволяють проводити узгодження наступних трьох способів маніпулювання даними: впорядкування, групування по певних ознаках, доступ по дереву параметрів. Це пов’язано з тим, що в таблиці всі три способи маніпулювання жорстко закріплені: дані впорядковані по одному параметру і не впорядковані по іншому.

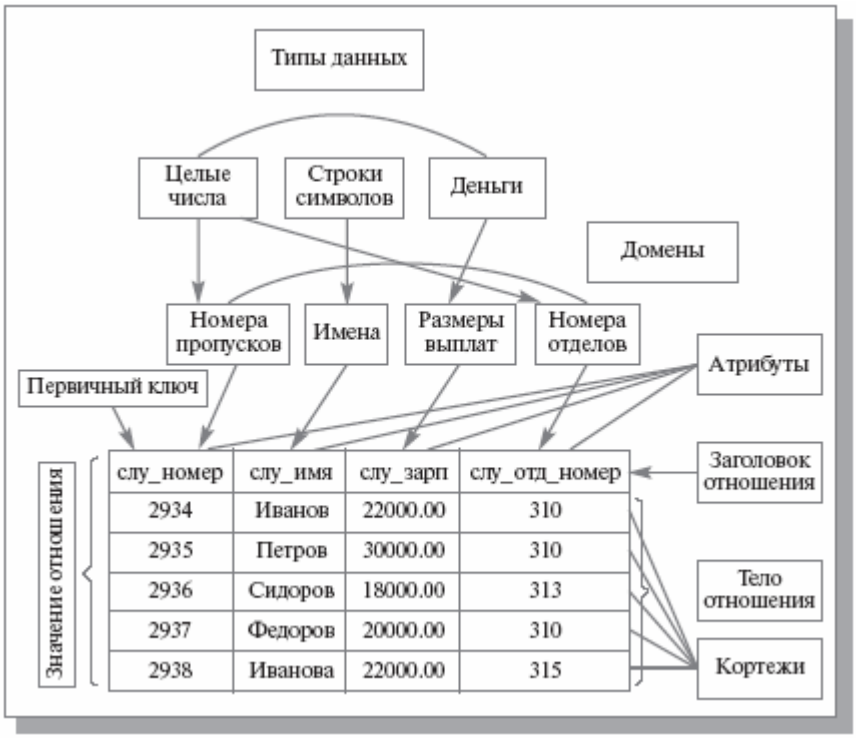
Відношення реляційних баз даних.

Відношення реляційної бази даних діляться на два класи: об’єктні та зв’язні. Об’єктне відношення зберігає дані про об’єкти (екземпляри сутності). В об’єктному відношенні один (або декілька) з атрибутів однозначно ідентифікують об’єкт. Такий ключовий атрибут називається (одиничним чи множинним) ключем відношень або первинним атрибутом. Ключ, як правило, знаходиться у першому стовпці. Інші атрибути функціонально залежать від даного ключа. Ключ може включати кілька атрибутів (складний ключ). В об’єктному відношенні атрибути не повинні дублюватися. Це основне обмеження в реляційній базі

даних для збереження цілісності даних. Зв’язне відношення зберігає ключі двох чи більше об’єктних відношень, тобто по ключах встановлюються зв’язки між об’єктами відношень. Зв’язне відношення може мати і інші атрибути, які функціонально залежать від цього зв’язку. Ключі в зв’язних відношеннях називаються зовнішніми (сторонніми) ключами, оскільки вони є первинними ключами інших відношень.
Умови і обмеження, які накладаються на відношення реляційних баз даних на табличному рівні представлення, можна сформулювати наступним чином:
· не може бути однакових первинних ключів, тобто всі рядки (записи) повинні бути унікальними;
· всі рядки повинні мати однакову типову структуру;
· імена стовпців в таблиці повинні бути різними, а значення стовпців повинні бути однотиповими;
· значення стовпців повинні бути атомарними, тобто не можуть бути компонентами інших відношень;
· повинна зберігатися цілісність для зовнішніх ключів;
· порядок розміщення рядків у таблиці неістотний - він впливає лише на швидкість доступу до потрібного рядка.
Кожен запис таблиці складається із окремих полів (атрибутів), які діляться на три групи: ключові, вказівні та допоміжні. Ключовими є ті атрибути, якими однозначно ідентифікується даний запис. Двох записів з однаковими ключовими атрибутами бути не може. Вказівні атрибути виконують роль ключових атрибутів в інших базах даних, на які посилається даний запис. З їх допомогою можна отримати додаткову інформацію для заданого запису. Допоміжні атрибути - це характеристики для заданого запису і вони, як правило, можуть повторюватись.
Файл бази даних може бути індексованим або ні. Наявність індексації означає, що всі записи в файлі баз даних перевпорядковані у відповідності із наперед заданим принципом (алфавітний порядок, порядок зростання чи спадання і так далі) по одному з полів. Інформація про індексацію міститься в індексному файлі у вигляді набору взаємопов’язаних пар чисел. Система індексації застосовується для зручності використання бази даних: для заданих полів створюється впорядкований перелік пар чисел, перше з яких вказує фізичний реальний порядковий номер запису в файлі, друге - порядковий номер даного запису у перевпорядкованому списку. Розрізняють так звані одиничні і множинні індексні файли. Для кожного фізичного запису одиничний індексний файл містить тільки одну пару чисел, в той час як множинний - множину з декількох впорядкованих пар чисел. Кожне друге з наступної пари чисел в наборі означає порядковий номер у новому списку.

3.2. Вступ в реляційну модель даних

Прийнято вважати, що реляційний підхід до організації баз даних був закладений в кінці 1960-х рр. Едгаром Коддом. Переваги реляційного підходу привели до того, що до кінця 80-х років реляційні системи зайняли на світовому ринку СУБД домінуюче положення. В останні десятиліття цей підхід є найбільш поширеним.
Перевагами реляційного підходу прийнято рахувати наступні властивості:
реляційний підхід грунтується на невеликому числі інтуїтивно зрозумілих абстракцій, на основі яких можливо просте моделювання найбільш поширених наочних областей;
ці абстракції можуть бути точно та формально визначені;
реляційний підхід забезпечує можливість ненавігаційного маніпулювання даними без необхідності знання конкретної фізичної організації баз даних в зовнішній пам’яті.
Основні поняття реляційних баз даних: тип даних, домен, атрибут, кортеж, відношення, первинний ключ.
Визначати поняття будемо на прикладі відношення СЛУЖБОВЦІ, що містить інформацію про службовців певного підприємства



Мал. 2.1. Співвідношення основних понять реляційного підходу

Тип даних

Значення даних, що зберігаються в реляційній базі даних, є такими, що типізуються, тобто відомий тип кожного значення, що зберігається. Поняття типу даних в реляційній моделі даних повністю відповідає поняттю типу даних в мовах програмування. В сучасних реляційних БД допускається зберігання символьних, числових даних (точних і приблизних), спеціалізованих числових даних (таких, як «гроші»), а також спеціальних «темпоральних» даних (дата, час, часовий інтервал).
У прикладі на мал. 2.1 присутні дані трьох типів: рядки символів, цілі числа і «гроші».

Домен

Поняття домена більш специфічно для баз даних, хоч і є аналогії з підтипами в деяких мовах програмування. У загальному вигляді домен визначається шляхом завдання деякого базового типа даних, до якого відносяться елементи домена. З кожним доменом зв'язується ім'я, унікальне серед імен всіх доменів відповідної бази даних.
Найбільш правильним інтуїтивним трактуванням поняття домена є його сприйняття як допустимої потенційної, обмеженої підмножини значень даного типа. Наприклад, домен ІМЕНА в прикладі визначений на базовому типі символьних рядків, але до числа його значень можуть входити тільки ті рядки, які можуть представляти імена (зокрема, для можливості представлення російських імен такі рядки не можуть починатися з м'якого або твердого знаку і не можуть бути довше, наприклад, 20 символів).
Якщо деякий атрибут відношення визначається на деякому домені (як, наприклад, на мал. 2.1 атрибут СЛУ_ИМЯ визначається на домені ІМЕНА), то надалі обмеження домена грає роль обмеження цілісності, що накладається на значення цього атрибуту.
Слід зазначити також семантичне навантаження поняття домена: дані вважаються порівнянними тільки у тому випадку, коли вони відносяться до одного домена. У нашому прикладі значення доменів НОМЕРА ПРОПУСКІВ і НОМЕРА ВІДДІЛІВ відносяться до типу цілих чисел, але не є порівнянними (допускати їх порівняння було б безглуздо).

Заголовок відношення, кортеж, тіло відношення, значення відношення, змінна відношення

Поняття відношення є найбільш фундаментальним в реляційному підході до організації баз даних. Це відбито і в загальній назві підходу – термін реляційний (relational) походить від relation (відношення). Для уточнення терміну відношення виділяються поняття заголовка відношення, значення відношення і змінної відношення. Крім того, для пояснення потрібно допоміжне поняття кортежу.

Отже, заголовком (або схемою) відношення r (Hr) називається кінцева безліч впорядкованих пар виду <A, T>, де A називається ім'ям атрибуту, а T позначає ім'я деякого базового типа або раніше певного домена. За визначенням потрібно, щоб всі імена атрибутів в заголовку відношення були різні. У прикладі на мал. 2.1 заголовком відношення СЛУЖБОВЦІ є безліч пар {<слу_номер, номера_пропусков> <слу_имя, имена> <слу_зарп, размеры_выплат> <слу_отд_номер, номера_отделов>}.

Кортежем tr, відповідним заголовку Hr, називається безліч впорядкованих триплетів вигляду <A, T v>. Третій елемент триплета <A, T v> повинен бути допустимим значенням типу даних або домена T. Заголовку відношення СЛУЖБОВЦІ відповідають, наприклад, наступні кортежі: (<слу_номер, номера_пропусков, 2934> <слу_имя, імена, Иванов> <слу_зарп, размеры_выплат, 22.000> і т.д.).

Тілом Br відношення r називається довільна безліч кортежів tr. Одне з можливих тіл відношення СЛУЖБОВЦІ показане на мал. 2.1. Відмітимо, що в загальному випадку, як це демонструють, зокрема, мал. 2.1, можуть існувати такі кортежі tr, які відповідають Hr, але не входять в Br.

Значенням Vr відношення r називається пара безлічі Hr і Br. Одне з допустимих значень відношення СЛУЖБОВЦІ показане на мал. 2.1.

У мінливій реляційній базі даних зберігаються відношеньи, значення яких змінюються в часі. Змінною VARr називається іменований контейнер, який може містити будь-яке допустиме значення Vr. Природно, що при визначенні будь-якої VARr потрібно указувати відповідний заголовок відношення Hr.

Тут варто підкреслити, що будь-яка прийнята на практиці операція оновлення бази даних – INSERT (вставка кортежу в змінну відношення), DELETE (видалення кортежу із значення-відношення змінної відношення) і UPDATE (модифікація кортежу, значення-відношення, змінної відношення) – з модельної точки зору є операцією привласнення змінної відношення деякого нового значення-відношення.

Відмітимо, що надалі в тих випадках, коли точний сенс терміну зрозумілий з контексту, можна використовувати термін відношення як в сенсі значення відношення, так і в сенсі змінна відношення.

За визначенням, ступенем, або «арностью» є потужність заголовка відношення. Наприклад, ступінь відношення СЛУЖБОВЦІ рівний чотирьом, тобто воно є 4-арным (кватернарним).

Інтуїтивна інтерпретація реляційних понять

Звичайним життєвим представленням відношення є таблиця, заголовком якої є схема відношення, а рядками – кортежі відношення-екземпляра; в цьому випадку імена атрибутів відповідають іменам стовпців даної таблиці. Тому іноді говорять про «стовпці таблиці», маючи на увазі «атрибути відношення».

Звичайно, це достатньо груба термінологія, оскільки у звичайних таблиць і рядки, і стовпці впорядковані, тоді як атрибути і кортежі відношень є елементами неврегульованих множин.

Фундаментальні властивості відношень

Зупинимося тепер на деяких важливих властивостях відношень:

Відсутність кортежів-дублікатів, первинний і можливі ключі відношень

Та властивість, що тіло будь-якого відношення ніколи не містить кортежів-дублікатів, виходить з визначення тіла відношення як безліч кортежів. У класичній теорії множин за визначенням будь-яка множина складається з різних елементів.

Саме з цієї властивості витікає наявність у кожного значення відношення первинного ключа – мінімальної безлічі атрибутів, які унікально визначають кортеж відношення. Дійсно, оскільки у будь-який час всі кортежі тіла будь-якого відношення різні, у будь-якого значення відношення властивістю унікальності володіє, принаймні, повний набір його атрибутів. Проте у формальному визначенні первинного ключа потрібне забезпечення його «мінімальності», тобто в набір атрибутів первинного ключа не повинні входити такі атрибути, які можна відкинути без збитку для основної властивості – однозначного визначення кортежу.

Поняття первинного ключа є виключно важливим у зв'язку з поняттям цілісності баз даних. На практиці первинні (і можливі) ключі змінних відношень з'являються в результаті явних вказівок проектувальника відношення. Визначаючи змінну відношення, проектувальник моделює частину наочної області, дані з якої міститиме база даних. І звичайно, проектувальник повинен знати природу цих даних. Наприклад, йому повинно бути відомо, що ніякі два службовців ні в який момент часу не можуть мати посвідчення з одним і тим же номером. Тому він може (і навіть повинен) явно оголосити {СЛУ_НОМЕР} можливим ключем. Якщо на підприємстві встановлено, що у всіх службовців повинні бути різні повні імена, то проектувальник може (і знову ж таки повинен) оголосити можливим ключем і {СЛУ_ИМЯ}. Потім проектувальник повинен оцінити, який з можливих ключів є надійнішим (властивість його унікальності ніколи не буде скасовано) і вибрати найбільш надійний можливий ключ як первинного (у нашому випадку природним вибором був би ключ {СЛУ_НОМЕР}, тому що рішення про унікальність повних імен службовців виглядає штучним і може бути легко скасовано керівництвом підприємства).

Тепер пояснимо, чому проектувальнику слід явно оголошувати первинний і можливі ключі змінних відношень. Річ у тому, що в результаті цього оголошення СУБД одержує інформацію, яка надалі використовуватиметься як обмеження цілісності. СУБД ніколи не допустить появи в змінній відношення значення-відношення, що містить два кортежі з однаковим значенням атрибуту. Тим самим оголошення первинного і можливих ключів дають СУБД можливість підтримувати цілісність бази даних навіть у разі спроб занесення в неї некоректних даних.

Нарешті, повернемося до властивості мінімальності первинного і можливих ключів. Як наголошувалося вище, ця властивість є критично важливою, і важливість виявляється саме при трактуванні первинного і можливих ключів як обмежень цілісності. У нашому прикладі з відношенням СЛУЖБОВЦІ властивістю унікальності володітиме не тільки безліч атрибутів { СЛУ_НОМЕР}, але і, наприклад, множина { СЛУ_НОМЕР, СЛУ_ОТД_НОМЕР}. Але якби ми виставили як обмеження цілісності вимогу унікальності { СЛУ_НОМЕР, СЛУ_ОТД_НОМЕР}, то СУБД гарантувала б відсутність кортежів з однаковим значенням атрибуту СЛУ_НОМЕР не у всьому значенні відношення СЛУЖБОВЦІ, а тільки в групах кортежів з одним і тим же значенням атрибуту СЛУ_ОТД_НОМЕР. Зрозуміло, що це не відповідає сенсу модельованої наочної області.

Відсутність впорядкованості кортежів

Звичайно, формально властивість відсутності впорядкованості кортежів в значенні відношення також є слідством визначення тіла відношення як безліч кортежів. Проте на цю властивість можна поглянути і з іншого боку. Достатньо часто у користувачів реляційних СУБД і розробників інформаційних систем викликає роздратування той факт, що вони не можуть зберігати кортежі відношення на фізичному рівні в потрібному їм порядку. Можна було б розробити іншу теорію, в якій допускалися б впорядковані «відношеньи». Проте зберігати впорядковані списки кортежів в умовах інтенсивно оновлюваної бази даних

набагато складніше технічно, а підтримка впорядкованості спричиняє за собою істотні накладні витрати.
Відсутність вимоги до підтримки впорядкування безлічі кортежів відношення додає СУБД додаткову гнучкість при зберіганні баз даних в зовнішній пам'яті і при виконанні запитів до бази даних. Це не суперечить тому, що при формулюванні запиту до БД, наприклад, на мові SQL можна зажадати сортування результуючої таблиці відповідно до значень деяких стовпців.

Відсутність впорядкованості атрибутів

Атрибути відношень не впорядковані, оскільки за визначенням заголовок відношення є безліч пар <имя атрибуту, ім'я домена>. Для посилання на значення атрибуту в кортежі відношення завжди використовується ім'я атрибуту. СУБД сама приймає рішення про те, в якому фізичному порядку слід зберігати значення атрибутів кортежів. Крім того, ця властивість полегшує виконання операції модифікації схем існуючих відношень не тільки шляхом додавання нових атрибутів, але і шляхом видалення існуючих.

Атомарність значень атрибутів, перша нормальна форма відношення

Значення всіх атрибутів є атомарними (вірніше, скалярними). Це витікає з визначення домена як потенційної безлічі значень скалярного типа даних, тобто серед значень домена не можуть міститися значення з видимою структурою, зокрема безліч значень (відношеньи). Головне в атомарності значень атрибутів полягає в тому, що реляційна СУБД не повинна забезпечувати користувачам явної видимості внутрішньої структури значення. До всіх значень можна звертатися тільки за допомогою операцій, визначених у відповідному типі даних.
Прийнято говорити, що в реляційних базах даних допускаються тільки нормалізовані відношення, або відношення, представлені в першій нормальній формі.
Відношення знаходиться в першій нормальній формі тоді і тільки тоді, коли в будь-якому допустимому значенні відношення кожен його кортеж містить тільки одне значення для кожного з атрибутів.
Нормалізовані відношення складають основу класичного реляційного підходу до організації баз даних. Вони володіють деякими обмеженнями (не всяку інформацію зручно представляти у вигляді плоских таблиць), але істотно спрощують маніпулювання даними.

Реляційна модель даних

Хоча поняття реляційної моделі даних першим ввів основоположник реляційного підходу Едгар Кодд, найбільш поширене трактування реляційної моделі даних, мабуть, належить відомому популяризатору ідей Кодда Крістоферу Дейту, який відтворює її (з різними уточненнями) практично у всіх своїх книгах. Згідно трактуванню Дейта, реляційна модель складається з трьох частин, що описують різні аспекти реляційного підходу: структурної частини, маніпуляційної частини і цілісної частини.

У структурній частині моделі фіксується, що єдиною родовою структурою даних, використовуваної в реляційних БД, є нормалізовані n-арное відношеньи. Визначаються поняття доменів, атрибутів, кортежів, заголовка, тіла і змінної відношення.
У маніпуляційній частині моделі визначаються два фундаментальні механізми маніпулювання реляційними БД – реляційна алгебра і реляційне числення. Основною функцією маніпуляційної частини реляційної моделі є забезпечення міри реляційності будь-якої конкретної мови реляційних БД: мова називається реляційною, якщо він володіє не меншою виразністю і потужністю, чим реляційна алгебра або реляційне числення.

Цілісність сутності і посилань

Нарешті, в цілісній частині реляційної моделі даних фіксуються дві базові вимоги цілісності, які повинні підтримуватися в будь-якій реляційній СУБД. Перша вимога називається вимогою цілісності сутності (entity integrity). Об'єкту або сутності реального світу в

реляційних БД відповідають кортежі відношень. Вимога цілісності сутності повністю звучить таким чином: у будь-якої змінної відношення повинен існувати первинний ключ, і ніяке значення первинного ключа в кортежах значення-відношення змінної відношення не повинне містити невизначених значень. Щоб це формулювання було повністю зрозуміле, потрібно обговорити поняття невизначеного значення (NULL).
Звичайно, теоретично будь-який кортеж, що заноситься у відношення, що зберігається, повинен містити всі характеристики модельованої їм сутності реального світу, які потрібно зберегти в базі даних. Проте на практиці не всі ці характеристики можуть бути відомі до того моменту, коли потрібно зафіксувати сутність в базі даних. Простим прикладом може бути процедура ухвалення на роботу людини, розмір заробітної платні якого ще не визначений. Едгар Кодд запропонував використовувати в таких випадках невизначені значення. Невизначене значення не належить ніякому типу даних і може бути присутнім серед значень будь-якого атрибуту, визначеного на будь-якому типі даних (якщо це явно не заборонено при визначенні атрибуту). Якщо a – це значення деякого типу даних або NULL, op – будь-яка двомісна «арифметична» операція цього типу даних (наприклад +), а lop – операція порівняння значень цього типа (наприклад =), то за визначенням:
a op NULL = NULL
NULL op a = NULL
a lop NULL = unknown
NULL lop a = unknown
Тут unknown – це третє значення логічного, або булевого, типу, що володіє наступними властивостями:
NOT unknown = unknown
true AND unknown = unknown
true OR unknown = true
false AND unknown = false
false OR unknown = unknown
Друга вимога, яка називається вимогою цілісності по посиланнях (referential integrity), є складнішою. Очевидно, що при дотриманні нормалізованості відношень складна сутність реального світу представляється в реляційній БД у вигляді декількох кортежів декількох відношень. Вимога цілісності по посиланнях, або вимога цілісності зовнішнього ключа, полягає в тому, що для кожного значення зовнішнього ключа, що з'являється в кортежі значення-відношення змінної відношення, що посилається, або в значенні-відношенні змінної відношення, на яку указує посилання, повинен знайтися кортеж з таким же значенням первинного ключа, або значення зовнішнього ключа повинне бути повністю невизначеним (тобто ні на що не указувати).
Обмеження цілісності сутності і по посиланнях повинні підтримуватися СУБД.
Для дотримання цілісності сутності досить гарантувати відсутність в будь-якій змінній відношення значень-відношень, що містять кортежі з одним і тим же значенням первинного ключа (і забороняти входження в значення первинного ключа невизначених значень). З цілісністю по посиланнях справа йде дещо складніше. Зрозуміло, що при оновленні відношення (вставці нових кортежів або модифікації значення зовнішнього ключа в існуючих кортежах), що посилається, досить стежити за тим, щоб не з'являлися некоректні значення зовнішнього ключа. Але як бути при видаленні кортежу з відношення, на яке існує посилання?
Тут існують три підходи, кожний з яких підтримує цілісність по посиланнях. Перший підхід полягає в тому, що взагалі забороняється проводити видалення кортежу, для якого існують посилання (тобто спочатку потрібне або видалити кортежі, що посилаються, або відповідним чином змінити значення їх зовнішнього ключа). При другому підході при видаленні кортежу, на який є посилання, у всіх кортежах, що посилаються, значення зовнішнього ключа автоматично стає повністю невизначеним. Нарешті, третій підхід (каскадне видалення) полягає в тому, що при видаленні кортежу з відношення, на яке веде посилання, з відношення, що посилається, автоматично віддаляються кортежі, що посилаються.

3.3. Схема БД. Таблиці.

Теорія реляційних баз даних (БД) оперує з багатьма термінами і поняттями. Але в реальних системах управління базами даних (СУБД) часто щось спрощується, щось додається своє. Тому ми розгляутмо тільки деякі основні, що реально будуть використані у проектах Delphi.

Таблиця

Таблиця - сукупність рядків і стовпців. Майже повна аналогія з таблицями на папері. Важливі уточнення: Кожен стовпець повинен мати ім'я, унікальне в межах конкретної таблиці. У теорії баз даних вважається, що рядки можуть іти в будь-якому порядку і не мають номерів чи якоїсь іншої ідентифікації. Хоча деякі СУБД й додають до кожного рядка номер, але при вибірці даних за допомогою SQL, у загальному випадку, їх одержати не можна. Тому до кожного рядка прийнято додавати якусь ідентифікацію - ключ, для того, щоб потім можна було легко знайти потрібну інформацію.

Відношення

У теорії БД розрізняють поняття таблиці і відношення. Наприклад, у відношенні не обумовлюється порядок стовпців, а тільки їх набір. Ще для відношень вводять певні обмеження, як неможливість мати два зовсім однакові рядки. Можна вважати, що розходження між ними полягає в наступному: відношення - абстрактне представлення інформації про об'єкти предметної області, а таблиця - більш конкретне. Але у першому наближенні у подальшому будемо вважати, що це те саме.

ТЕМА 4.

Потенційні, первинні та зовнішні ключі
Цілісність реляційних даних

4.1. Ключі.

Ключ - це набір стовпчиків, може складатися з одного стовпчика, чи охоплювати всі стовпчики таблиці. Ключ призначений для ідентифікації рядків таблиці. У теорії БД це єдиний спосіб послатися на рядок. Ключі бувають різні - потенційні, первинні, альтернативні, зовнішні, індексні, хеш-ключі, ключі сортування, вторинні ключі, ключі шифрування і розшифрування та ін. Але ми будемо розглядати тільки такі, що нам знадобиться в роботі над проектами.

Потенційні ключі. Потенційним ключем будемо називати таку комбінацію стовпців, що має наступні властивості:

Унікальність. У таблиці немає двох різних рядків з однаковими значеннями в потенційному ключі. Таким чином буде забезпечено однозначність, наприклад, у процесі можливого пошуку за ключем.

Ненадлишковість. Не можна забрати один зі стовпчиків із ключа, так, щоб ключ не втратив унікальності. Цим буде забезпечена своєрідна мінімальність інформації, що потрібна для пошуку.

Розглянемо, наприклад, таку таблицю:

Табельний №	Прізвище	Ім'я	По батькові	Посада
123456	Мороз	Іван	Іванович	Декан
234567	Зима	Петро	Іванович	Заступник
345678	Снігуронька	Марія	Іванівна	Методист

У цій таблиці потенційним ключем може бути будь-який стовпець. Але вона буде розширюватися, тому з великою вірогідністю більшість стовпчиків не зможуть бути потенційним ключем, бо можливі збіги. Так, стовпчик Прізвище зможе виконувати цю функцію, якщо тільки ніколи не буде нових рядків у таблиці із тим же прізвищем. Можна взяти комбінацію прізвища і посади, хоча й тут є таж сама небезпека. Табельний номер також підходить на роль потенційного ключа. Як висновок зауважимо, що до кожної конкретної таблиці потенційних ключів може бути багато. Вибір потенційного ключа - справа програміста-розробника прикладної програми.

Первинні ключі.

Первинний ключ - це один з потенційних ключів. Той, котрий під час аналізу та оцінки виявиться більш оптимальним. У простій реальній ситуації можна обрати табельний номер чи номер паспорта. Вибір табельного номера є більш надійним та професійним за умови, що ніколи не буде повторно комусь надано той же номер. Професіонал з БД додасть ще один поле - код, що гарантовано буде містити унікальне для кожного запису значення (це може бути лічильник). У Paradox такий тип поля називається AutoIncrement, у SQL Server є зразу 2 варіанти - TimeStamp та властивість Identity поля.

Альтернативні ключі.

Первинний ключ може бути тільки один на всю таблицю. Після вибору первинного ключа з набору потенційних ключів, ключі, що залишилися, називаються альтернативними.

Зовнішні ключі.

Коли створюється нова база даних, наприклад для нарахування зарплати, не варто всю інформацію про всіх працівників записати в одну таблицю (хоча можна було б зробити у такий спосіб, але це стане джерелом багатьох роблем). Якщо, наприклад, хтось із працівників згадується у такій таблиці не один раз (зарплата, премія, надбавки, вирахування, податки та ін.), то при зміні його/її прізвища треба буде провести зміни у всіх рядках (або ж як варіант забезпечити змінну кількість сповпчиків). Це незручно. Це є приводом розділити таку таблицю на кілька з метою оптимізації як обсягів даних, так і способів їх обробки. Тому доцільно створити дві таблиці:Код працівника

Вигляд руху		Сума
1	Оклад	100
1	Премія	30
1	Податки	-25
2	Оклад	90
...	...	...

Код працівника	Прізвище	Ім'я	По батькові
1	Мороз	Іван	Іванович
2	Зима	Петро	Іванович
3	Снігуронька	Марія	Іванівна

У першій таблиці стовпчик "Код працівника" називається зовнішнім ключем. Зрозуміло, що він не може існувати без відповідного рядка другої таблиці, у якій стовпчик "Код працівника" є первинним ключем. Друга таблиця є своєрідним довідником прізвищ для першої.

Хоча теорія БД вимагає, щоб зовнішні ключі завжди посилалися на первинні ключі, ми цю вимогу зведемо до простої рекомендації: бувають випадки, коли одна й та ж таблиця може служити довідником для різних інших таблиць, причому в різній якості.

Референціальна цілісність.У попередньому викладі ми згадували питання "А що буде, якщо не знайдеться працівника з певним кодом, що є у першій таблиці?" Відповідь є очевидною: Такої ситуації треба всіляко уникати, бо можливі непередбачувані ситуації у процесі обробки інформації.

Референційною цілісністю (англ. Refential Integriey) називається такий стан БД, коли в її таблицях є не тільки посилання на якусь інформацію, але й сама інформація (іншими словами, усе, що потрібно у процесі обробки, правильно знаходиться). Контроль референціальної цілісності - забезпечення такого стану

Зауваження: А якщо користувач захоче видалити якусь інформацію (наприклад, про одного з працівників, то що? Виходи такі:- просто заборонити такі дії, - разом з записами основної таблиці видаляти усі відповідні записи з іншої таблиці (так зване "каскадне видалення"). Але ні при яких ситуаціях не можна допускати порушення референційної цілісності

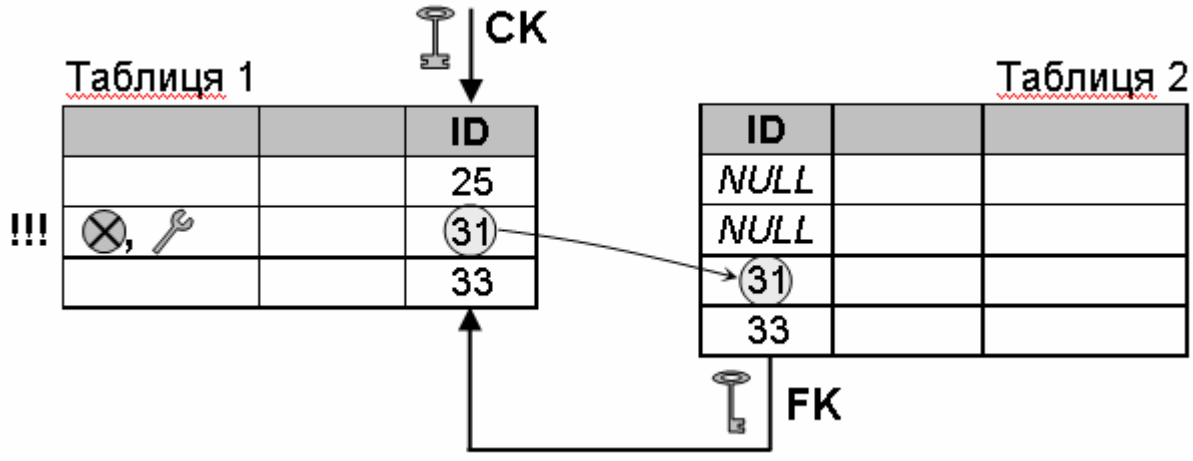
База даних. Базою даних ми будемо називати сукупність таблиць, індексів, збережених процедур, тригерів та інших складових, що пов'язані певною спільністю (предметної області, прикладної автоматизованої системи).

4.2. Цілісність даних

Для БД задаються два базові правила цілісності: Цілісність сутностей. 2. Цілісність посилань.

4.2.1. Цілісність сутностей

Сутність (entity) – абстрактний об’єкт певного виду.
Правило: у базовій таблиці жоден стовпець первинного ключа не може містити NULL - значення (пусте значення).
Примітка: пусте значення вказує на те, що значення стовпця у даний момент невідоме або не прийняте для цього запису. Пусте значення (NULL) слід розглядати, як логічну величину «невідомо». Однак NULL не слід розуміти, як нульове числове значення, заповнену пробілами текстову стрічку чи нульову стрічку, бо вони представляють собою певні значення, у той час, коли NULL позначає відсутність будь-якого значення.
Призначення: гарантує, що кожна сутність (логічний об’єкт) буде мати унікальну ідентифікацію.
Приклад: рахунок у банку не може мати декілька дублюючих значень та не може мати пусте значення (NULL). Іншими словами, усі рахунки унікально ідентифікуються своїм номером.
Примітка: це правило стосується лише первинного ключа. Якщо у таблиці існують інші потенційні (альтернативні) ключі, то їм дозволяється повністю або частково приймати значення NULL , хоча розробник може і заборонити пусті значення на етапі проектування БД.
4.2.2. Цілісність значень
Правило: якщо у таблиці існує зовнішній ключ, то його значення повинні або співпадати зі значеннями потенційного ключа (первинного чи унікального) деякої базової таблиці, або містити пусті значення (NULL).
Приклад: клієнту може бути не призначений (ще) торговий агент, однак не можливо призначити клієнту неіснуючого агента.
Призначення: допускається, що стовпець може не мати відповідного значення, однак стовпець не може приймати недопустимі значення. Виконання правила цілісності посилання, як правило, робить неможливим видалення стрічки чи зміни значення потенційного ключа однієї таблиці, якщо цей ключ має співпадіння зі значенням зовнішнього ключа у іншій таблиці.

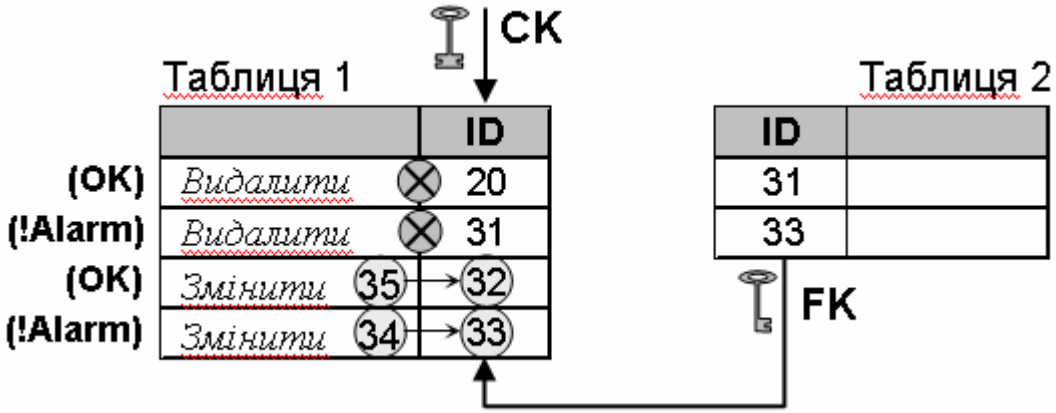


Структура цілісності посилання

Для налаштування механізму цілісності посилань є передбачено три основні стратегії:

1. ЗАБОРОНА.

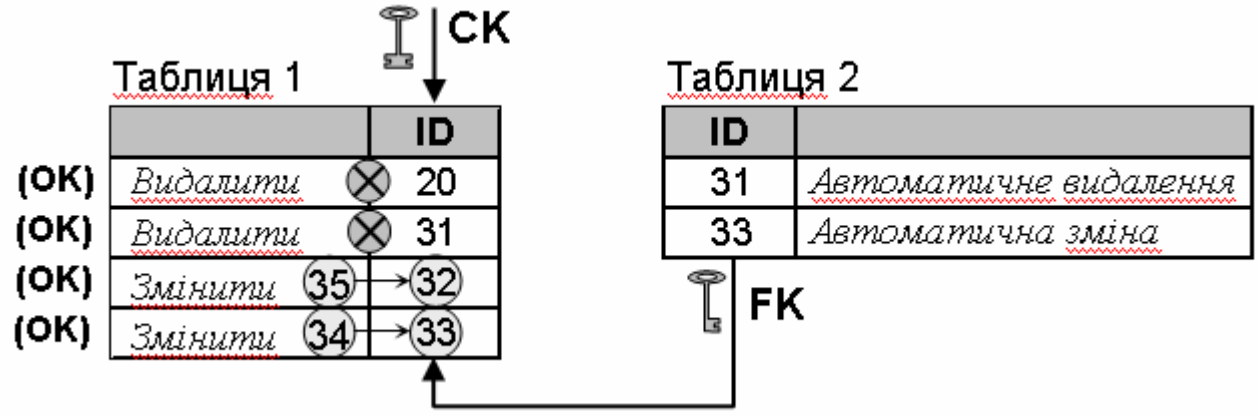
Згідно цієї стратегії, накладається заборона на усі зміни потенційного ключа, якщо існують зовнішні ключі, що посилаються на нього. Тобто, видаляти стрічки чи змінювати значення у стовпці ID таблиці 1 можна лише для тих значень ID , для яких не існує відповідних їм елементів у таблиці 2.



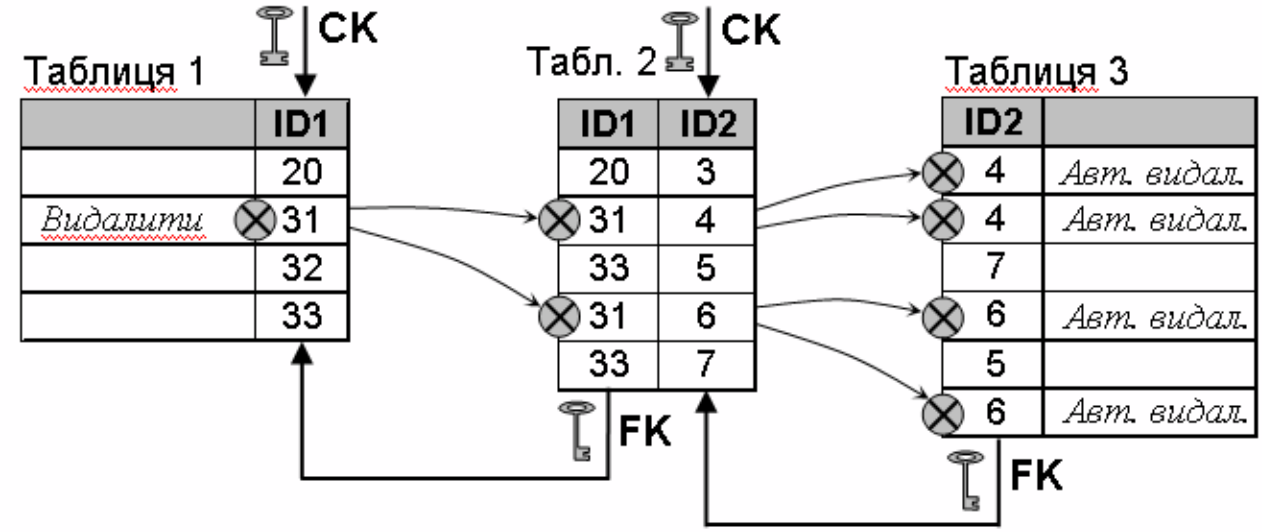
Ілюстрація стратегії «заборони»

2. КАСКАДНІ ЗМІНИ.

Виконання операцій над вихідною стрічкою базової таблиці з потенційним ключем «каскадним» чином розповсюджується на усі стрічки інших таблиць, зовнішні ключі яких посилаються на базову стрічку. Тобто, видалення стрічки з базової таблиці 1 спонукає видаленню усіх стрічок з таблиці 2, для яких значення зовнішнього ключа співпадає зі значеннями потенційного ключа таблиці 1. Зміна ж значення поля ID таблиці 1 генерує зміну відповідного значення поля ID таблиці 2.



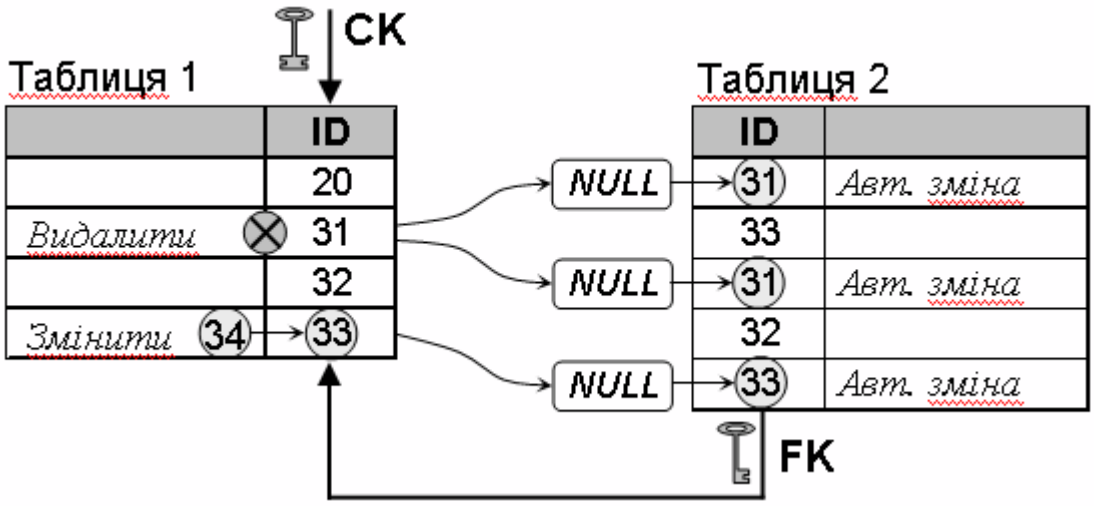
Ілюстрація стратегії виконання каскадних операцій



Ілюстрація виконання множинного каскадного видалення

3. ПРИСВОЄННЯ NULL -ЗНАЧЕНЬ.

У цьому випадку, при видаленні стрічок базової таблиці чи зміні значень потенційного ключа, відповідні значення зовнішніх ключів замінюються значеннями NULL .



Ілюстрація стратегії встановлення NULL -значень

Модифікацією цієї стратегії є варіант встановлення не NULL -значень, а значень за замовчуванням, які визначені для стовпців зі зовнішнім ключем. При невідповідності значень за замовчуванням потенційному ключу встановлюється NULL -значення.
Висновок: правило цілісності посилань вимагає, щоб кожне значення зовнішнього ключа посилалося на реально існуюче значення потенційного ключа, в іншому випадку, для нього повинно бути встановлене значення NULL .

ТЕМА 5.

Операції реляційної алгебри та реляційне числення

5.1. Реляційна алгебра

Основна ідея реляційної алгебри полягає в тому, що оскільки відношення є множинами, засоби маніпулювання відношеннями можуть базуватися на традиційних теоретико-множинних операціях, додатково до деяких спеціальних операцій, специфічних для баз даних. Набір основних алгебраїчних операцій, запропонований Коддом, складається з восьми операцій, які діляться на два класи, – теоретико-множинні операції і спеціальні реляційні операції. До складу теоретико-множинних операцій входять операції:

об2єднання відношень;
перетин відношень;
взяття різниці відношень;
Декартовий добуток відношень.

Спеціальні реляційні операції включають:

обмеження відношення;
проекція відношення;
з2єднання відношень;
ділення відношень.

Крім того, до складу алгебри входить операція присвоювання, що дозволяє зберігати в базі даних результати обрахунку алгебраїчних виразів, і операція перейменування атрибутів, яка дає можливість коректно сформувати заголовок (схему) результуючого відношення.

Загальна інтерпретація реляційних операцій

Якщо не вдаватися в деякі тонкощі, майже всі операції з запропонованих вище володіють очевидною і простою інтерпретацією:

Операція перейменування дає відношення, тіло якого співпадає з тілом операнда, але імена атрибутів змінені.

Операція присвоювання дозволяє зберегти результат обчислення реляційного виразу в існуючому відношенні БД. Оскільки результатом будь-якої реляційної операції (крім операції присвоювання) є деяке відношення, можна відобразити реляційні вирази, в яких замість відношення-операнда деякої реляційної операції знаходиться вкладений реляційний вираз.

Замкненість реляційної алгебри

Кожне відношення характеризується схемою (або заголовком) і набором кортежів (або тілом). Тому, якщо реалізовувати алгебру, операції якої замкнені відносно поняття відношення, кожна операція повинна породжувати відношення в повній мірі, тобто воно повинно володіти і тілом, і заголовком. Лише в цьому випадку буде можливість будувати вкладені вирази. Заголовок відношення є множиною пар <ім2я-атрибута, ім2я-домена>. Якщо зробити загальний огляд реляційних операцій, можна побачити, що домени атрибутів результуючого відношення однозначно визначаються доменами відношень-результатів. Обов2язкова наявність заголовку диктується реляційною замкненістю, інакше кажучи результат реляційної операції повинен обов2язково мати повністю визначений тип відношення, тобто мати наперед визначений набір атрибутів. Основна причина цієї вимоги – необхідність мати можливість посилатися на ці імена атрибутів у наступних операціях

Особливості теоретико-множинних операцій реляційної алгебри

Хоча в основі теоретико-множинної частини реляційної алгебри лежить класична теорія множин, відповідні операції реляційної алгебри володіють деякими особливостями. Почнемо з операції об’єднання (вертикальне) (все, що ми говоритимемо з приводу об’єднання переноситься на операції перетину і різниці).

При виконанні операції об2єднання двох відношень одержується відношення, щ містять всі кортежі, що входять хоча б в одне з відношень-операндів.

Операція перетину двох відношень дозволяє одержати відношення, що включає всі кортежі, які входять до обох відношень-операндів.

Відношення, що є різницею двох відношень, включає всі кортежі, що входять у відношення – перший операнд; такі, що жоден з них не входить у відношення, яке є другим операндом.

Результат операцій: відношення. При виконанні цієї операції необхідно пам2ятати про сумісність відношень по об’єднанню: два відношення сумісні по об’єднанню в тому і лише в

тому випадку, коли володіють однаковими заголовками. Більш точно, це означає, що в заголовках обох відношень міститься один і той самий набір імен атрибутів, і одночасно атрибути визначені на одному і тому самому домені. Якщо два атрибути сумісні по об'єднанню, то при звичайному виконанні над ними операцій об'єднання:

SELECT * FROM A UNION SELECT * FROM B,

перетину

SELECT * FROM A INTERSECT SELECT * FROM B,

і взяття різниці

SELECT * FROM A EXCEPT SELECT * FROM B

в мові SQL результатом операції є відношенням є відношення з коректно визначеним заголовком, що співпадає з заголовком кожного з відношень операндівв. Зауважимо, що залучення до складу операцій реляційної алгебри трьох операцій об'єднання, перетину і різниці є надлишковим, оскільки будь-яка з цих операцій виражається через дві інші. Тим не менш, Кодд в свій час вирішив включити всі три операції, виходячи з інтуїтивних потреб потенциального користувача системи реляційних БД, далекого від математики.

Приклад:

```
SELECT P.P#
FROM P
WHERE P.WEIGHT>16.0
UNION
SELECT SP.P#
FROM SP
WHERE SP.P#='S2';
```

Операція взяття Декартового (або прямого) добутку двох відношень.

При виконанні прямого добутку двох відношень одержується відношення, кортежі якого є конкатенацією (счепленням) кортежів першого і другого операндів (A CROSS JOIN B).

A B

S#		P#
S1		P1
S2		P2
S3		P3
S4		P4

A TINES B

S#	P#		…	…		…	…	
S1	P1		S2	P1		S3	P1	
S1	P2		S2	P2		S3	P2	
S1	P3		S2	P3		S3	P3	
…	…		…	…		…	…	

```
SELECT A.S#, B.P#
FROM A, B
```

В реляційній алгебрі виконується спеціалізована форма операції взяття прямого добутку – розширений прямий добуток відношень. При взятті розширеного прямого добутку двох

відношень елементом результуючого відношення є кортеж, що є конкатенацією (або злиттям) одного кортежа першого відношення і одного кортежа другого відношення. Для того щоб одержати результуюче відношенням з коректно сформованим заголовком потрібно притримуватись поняття сумісності по взяттю розширеного прямого добутку. Два відношення сумісні при цьому в тому і лише в тому випадку, якщо множини імен атрибутів цих відношень не перетинаються, тобто немає атрибутів з однаковими назвами. Будь-які два відношення можна зробити сумісними по взяттю прямого добутку шляхом застосування операції перейменування до одного з цих відношень. Слід зауважити, що операція взяття прямого добутку не є виправданою на практиці. По-перше, потужність її результату досить велика навіть при допустимих потужностях операндів, а по-друге, результат операції не є більш інформативним, ніж взяті в сукупності операнди.
Вибірка (обмеження). Результатом обмеження відношення за деякою умовою є відношення, що містить кортежі відношення-операнда, яке задовольняє цій умові.
Нехай маємо відношення А з атрибутами Х і У., а символ ∪ означає будь-який оператор порівняння (=, ¯, >, >= та ін.), такий що умова Х ∪ У при заданих Х та У дає значення істина або хибність. Тоді ∪-вибіркою з відношення А по атрибутам Х та У називається відношення, що має той самий заголовок, що і відношення А і тіло, що містить множину всіх кортежів t відношення А, для яких перевірка умови Х ∪ У дає значення істина. Умова найчастіше задається за допомогою ключового слова WHERE.

```
SELECT *
FROM S
WHERE S#='S1';
```

З використанням цих визначень можна використовувати операції обмеження, в яких умова обмеження є довільною булевим виразом, складеним з простих умов з використанням логічних зв'язок AND, OR і AND та дужок. На інтуїтивному рівні операцію обмеження краще за все уявити як взяття деякої «горизонтальної» вирізки з відношення-операнда.

```
SELECT *
FROM P
WHERE P.CITY<>'Paris'
AND P.WEIGHT>10,0;
```

Проекція.
При виконанні проекції відношення на заданий набір його атрибутів одержується відношення, кортежі якого одержуються шляхом взяття відповідних значень з заданих стовпців кортежів відношення-операнда.
Нехай задане відношення А з атрибутами X, Y,…, Z. Тоді проекцією відношення А за атрибутами X, Y, …, Z (А {X, Y,…, Z}) називається відношення, що задовольняє наступним вимогам.
– Його заголовок одержується з заголовку А шляхом видалення з нього всіх атрибутів, що не входять до множини А {X, Y,…, Z}.
– Його тіло містить множину всіх кортежів виду {X:x, Y:y,…, Z:z}, таких, що x, y, z у відношенні А є значеннями відповідних атрибутів (SELECT DISINCT X, Y,… Z FROM A).

```
SELECT S.S#, S.SNAME
FROM S
```

Таким чином, за допомогою проекції можна створити вертикальну підмножину заданого відношення.

Спеціальні реляційні операції

Операція з2єднання відношень (горизонтальна)

При з2єднанні двох відношень за деякою умовою утворюється результуюче відношення, кортежі якого є конкатенацією кортежів першого і другого відношень і задовольняють цій умові.
Загальна операція з'єднання (яку також називають з'єднанням за умовою) потребує наявності двох операндів – відношень, що з'єднуються і третього операнда – простої умови.
Нехай з'єднуються відношення А і В, що не мають спільних імен атрибутів. Тоді ∪-

з’єднання відношення A за атрибутом X з відношенням B за атрибутом У називається результат наступного виразу
(A JOIN B) WHERE X∪Y;
S JOIN P USING CITY
Є часткові випадки з’єднання – просте і еквіз’єднання. Операція природного з’єднання застосовується до пари відношень A і B, що володіють спільним атрибутом c (тобто, атрибутом з одним і тим самим іменем і визначеним на тому самому домені). Нехай відношення A і B мають заголовки
{X_1, X_2, …, X_m, Y1, Y2, … Yn}
та
{Y1, Y2, …Yn, Z1, Z2, …Zp}
відповідно, тобто атрибути Y1, Y2, …Yn (і лише вони) – спільні для двох цих відношень, X1, X2,… Xм – решта атрибутів відношення A і Z1, Z2, …Zp – решта атрибутів для B. далі ми будемо їх називати як X, Y, Z. Тоді операція природного з’єднання відношень A і B називається відношення з заголовком {X, Y, Z} і тілом, що містить множину всіх кортежів вигляду {X:x, Y:y, Z:z}. Таке з’єднання необов’язково виконується по значенню первинного і відповідного зовнішнього ключа.
S NATURAL JOIN P
Операція з’єднання називається операцією еквіз’єднаня, якщо умова з2єднання має вигляд (a = b), де a і b – атрибути різних операндів з’єднання. Цей випадок важливий тому, що (a) він часто зустрічається на практиці, і (b) для нього існують ефективні алгоритми реалізації.
SELECT S.*, P.P#, P.PNAME
FROM S, P
WHERE S.CITY=P.SITY
Операція ділення відношень
У операції реляційного ділення два операнди – бінарне і унарне відношення. Результуюче відношення складається з одноатрибутних кортежів, що містять значення першого атрибута кортежів першого операнда таких, що множина значень другого атрибута (при фіксованому значенні першого атрибута) співпадає з множиною значень другого операнда.
Ця операція є найменш очевидною з увсіх операцій реляційної алгебри і тому потребує детального пояснення. Нехай задані два відношення – A з заголовком {a_1, a_2,…, a_n, b_1, b_2,…, b_m} і B з заголовком {b_1, b_2,…, b_m}. Будемо вважати, що атрибут b_i відношення A і атрибут b_i відношення B не лише володіють одним і тим самим іменем, але й визначені на одному і тому самому домені. Назвемо множину атрибутів {a_j} складовим атрибутом a, а множину атрибутів {b_j} – складовим атрибутом b. Після цього будемо говорити про реляційне ділення бінарного відношення A (a, b) на унарне відношення B(b). Результатом ділення A на B є унарне відношення C(a), що складається з кортежів v таких, що у відношенні A є кортежі <v, w> такі, що множина значень {w} включає множину значень атрибута b у відношенні B.
Припустимо, що в базі даних робітників підтримуються два відношення: РОБІТНИКИ (ВІД_НОМЕР, ІМЯ) і ІМЕНА (ІМЯ), причому унарне відношення ІМЕНА містить всі прізвища, які мають робітники організації. Тоді після виконання операції реляційного ділення відношення РОБІТНИКИ на відношенне ІМЕНА буде отримане унарне відношення, що міститиме номера відділів, робітники яких мають всі можливі в цій організації імена.

5.2. Реляційне числення

Розглянемо запит «Вибрати номери постачальників і назви міст, в яких знаходяться постачальники деталі з номером P2». Якщо б для формулювання такого запиту використовувалася реляційна алгебра, ми б отримали алгебраїчний вираз, який читався б, наприклад, наступним чином:
виконати з’єднання відношення S і SP за атрибутом S#;
вибрати з результату цього з2єднання кортежі з номером деталі 2P22;
спроектувати результат попередньої операції за атрибутами S# і CITY
Ми чітко сформулювали послідовність кроків виконання запиту, кожен з яких відповідає одній реляційній операції. Якщо ж сформулювати той самий запит з використанням реляційного числення, про яке йтиметься зараз, то ми одержимо формулу, яку можна було б

прочитати, наприклад, наступним чином: Видати S# і CITY для робітників таких, що постачають деталь з номером P2. У другому способі формулювання ми вказуємо лише характеристики результуючого відношення, але нічого не говоримо про спосіб його формування. В цьому випадку система повинна сама вирішити, які операції і в якому порядку потрібно виконати над відношеннями. Зазвичай говорять, що алгебраїчне формулювання є процедурним, тобто таким, що задає правила виконання запиту, а логічне – описовим (або декларативним), оскільки воно лишень описує властивості бажаного результату. Як ми зазначали, на справді ці два механізми еквівалентні і існують не дуже складні правила перетворення одного формалізму в інший.

Кортежні змінні і правильно побудовані формули

Реляційне числення є прикладною гілкою формального механизму числення предикатів першого порядку. Базисними поняттями числення є поняття змінної з визначеною для неї областю допустимих значень і поняття правильно побудовано формули, яка спирається на змінні, предикати і квантори. В залежності від того, що є областю визначення змінної, розрізняють числення кортежів і числення доменів.

В численні кортежів областями визначення змінних є відношення бази даних, тобто допустимим значенням кожної змінної є кортеж деякого відношення.

В численні доменів областями визначення змінних є домени, на яких визначені атрибути відношень бази даних, тобто допустимим значенням кожної змінної є значення деякого домена.

Для визначення кортежної змінної, областю визначення якої є деяке базове відношення потрібно вжити конструкцію (наприклад використану в операторі SELECT):

```
Select PX.COLOR, PX.CITY
FROM P AS PX
WHERE PX.CITY<>’Paris’
AND PX.WEIGHT>10.0;
```

З цього визначення випливає, що в будь-який момент часу змінна PX представлятиме змінну кортежа, областю значень якої буде поточне значення відношення P. При використанні кортежних змінних в формулах можна посилатися на значення атрибута змінної (це аналогично тому, як, наприклад, при програмуванні на мові Сі можна посилатися на значення поля структурної змінної).

```
PX.CITY
```

В мові SQL допускається неявне звернення до змінних кортежів, що дозволяє переписати наш запит у наступному вигляді:

```
Select P.COLOR, P.CITY
FROM P
WHERE P.CITY<>’Paris’
AND P.WEIGHT>10.0;
```

Основа ідея полягає в тому, щоб дозволити використовувати ім2я відношення для позначення неявної змінної кортежа, областю значення якої буде дана таблиця.

Правильно побудовані формули (WFF – Well-Formed Formula) служать для виразу умов, що накладаються на кортежні змінні. Основою WFF є прості порівняння, які є операціями порівняння скалярних значень (значень атрибутів змінних або літерально заданих констант). Наприклад, конструкція

«A.CITY=B.CITY»

є простим порівнянням. Більш складні варіанти WFF будуються за допомогою логічних зв’язок NOT, AND, OR. Нарешті, допускається побудова WFF за допомогою кванторів. Якщо form – це WFF, в якій бере участь змінна var, то конструкції EXISTS var (form) і FORALL var (form) є wff. Змінні, що входять до WFF, можуть бути вільними або зв’язаними. Всі змінні, що входять до WFF, при побудові якого не використовується квантор, є вільними.

Вказати імена постачальників деталі з номером 2P22:

```
SELECT DISTINCT S.SNAME
FROM S
WHERE EXISTS
```

(SELECT*
FROM SP
WHERE SP.S#=S.S#
AND SP.P#=’P2’);
SQL EXISTS (SELECT … FROM) буде мати значення істина тоді і лише тоді, коли результат обчислення виразу SELECT … FROM буде непустим.
SQL не містить деякої безпосередньої підтримки універсального квантора FORALL; відповідно запити типу «ДЛЯ ВСІХ» звичайно виражаються через заперечення квантора існування, як у наступному прикладі вибрати імена постачальників, які не постачають деталь з номером 2P22:
SELECT DISTINCT S.NAME
FROM S
WHERE NOT EXISTS
(SELECT *
FROM SP
SP.S#=S.S#,
AND SP.P#=’P2’);
В численні доменів областю визначення змінних є не відношення, а домени. По відношенню до БД Постачання деталей можна говорити, наприклад, про домени-змінні S# (значення – допустимі номери постачальників) або SNAME (значення – допустимі імена постачальників). Основною формальною відмінністю числення доменів від числення кортежів є наявність додаткового набору предикатів, що дозволяють виражати так звані умови входження. Якщо R – це n-арне відношення з атрибутами a_1, a_2,…, a_n, тоді умова членства має вигляд R (ai1:vi1, ai2:vi2,…, aim:vim) (m <= n), де vij – це або літеральна константа, або ім’я доменої змінної. Умова членства приймає значення true в тому і лише тому випадку, якщо у відношенні R існує кортеж, що містить вказані значення вказаних атрибутів.

ТЕМА 6.

Основні поняття SQL
Запити на читання даних
Склеювання таблиць
Умови відбору рядків таблиць
Агрегатні функції
Запити з групуванням
Складні запити
Запити на оновлення даних.

6.1. Основні поняття

Всі мови маніпулювання даними (ММД), створені до появи реляційних баз даних і розроблені для багатьох систем управління базами даних (СКБД) персональних комп'ютерів, були орієнтовані на операції з даними, представленими у вигляді логічних записів файлів. Це вимагало від користувачів детального знання організації зберігання даних і достатніх зусиль для вказівки не лише того, які дані потрібні, але й того, де вони розміщені і як крок за кроком отримати їх.
Розглянутий ж нижче непроцедурного мова SQL (Structured Query Language - структуризовано мова запитів) орієнтований на операції з даними, представленими у вигляді логічно взаємозалежних сукупностей таблиць. Особливість пропозицій цієї мови полягає в тому, що вони орієнтовані більшою мірою на кінцевий результат обробки даних, ніж на процедуру цієї обробки. SQL сам визначає, де знаходяться дані, які індекси і навіть найбільш ефективні послідовності операцій слід використовувати для їх отримання: не треба вказувати ці деталі в запиті до бази даних.

Для ілюстрації відмінностей між ММД розглянемо таку ситуацію. Нехай, наприклад, ви збираєтеся подивитися кінофільм і хочете скористатися для поїздки в кінотеатр послугами таксі. Одному шоферу таксі достатньо сказати назву фільму - і він сам знайде вам кінотеатр, в якому показують потрібний фільм. (Подібним же чином, самостійно, відшукує запитані дані SQL.)

Для іншого шофера таксі вам, можливо, потрібно буде самому дізнатися, де демонструється потрібний фільм і назвати кінотеатр. Тоді водій повинен знайти адресу цього кінотеатру. Може статися й так, що вам доведеться самому дізнатися адресу кінотеатру і запропонувати водієві проїхати до нього по таким-то і таким-то вулицях. У найгіршому випадку вам, може бути, навіть доведеться по дорозі давати вказівки: "Повернути наліво ... проїхати п'ять кварталів ... повернути праворуч ...". (Аналогічно більший або менший рівень деталізації запиту доводиться створювати користувачеві в різних СУБД, що не мають мови SQL.)

Поява теорії реляційних баз даних і запропонованого Коддом мови запитів "alpha", заснованого на реляційному обчисленні [2, 3], ініціювало розробку ряду мов запитів, які можна віднести до двох класів:

Алгебраїчні мови, що дозволяють виражати запити засобами спеціалізованих операторів, що застосовуються до відносинам (JOIN - з'єднати, INTERSECT - перетнути, SUBTRACT - відняти і т.д.).

Мови числення предикатів представляють собою набір правил для запису виразу, що визначає нове відношення з заданої сукупності існуючих відносин. Іншими словами числення предикатів є метод визначення того ставлення, яке нам бажано отримати (як відповідь на запрос) з відносин, вже наявних в базі даних.

Розробка, в основному, йшла у відділеннях фірми IBM (мови ISBL, SQL, QBE) і університетах США (PIQUE, QUEL) [3]. Останній створювався для СУБД INGRES (Interactive Graphics and Retrieval System), яка була розроблена на початку 70-х років в Університеті шт. Каліфорнія і сьогодні входить до п'ятірки кращих професійних СУБД. Сьогодні з усіх цих мов повністю збереглися і розвиваються QBE (Query-By-Example - запит по зразку) та SQL, а з решти взяті в розширення внутрішніх мов СУБД тільки найцікавіші конструкції.

На початку 80-х років SQL "переміг" інші мови запитів і став фактичним стандартом таких мов для професійних реляційних СУБД. У 1987 році він став міжнародним стандартом мови баз даних і почав впроваджуватися у всі поширенням СУБД персональних комп'ютерів.

Чому ж це сталося?

Безперервне зростання швидкодії, а також зниження енергоспоживання, розмірів і вартості комп'ютерів привели до різкого розширення можливих ринків їх збуту, кола користувачів, різноманітності типів і цін. Природно, що розширився попит на різноманітне програмне забезпечення.

Борючись за покупця, фірми, що виробляють програмне забезпечення, стали випускати на ринок все більш і більш інтелектуальні і, отже, об'ємні програмні комплекси. Купуючи (бажаючи придбати) такі комплекси, багато організацій та окремі користувачі часто не могли розмістити їх на власних ЕОМ, однак не хотіли і відмовлятися від нового сервісу. Для обміну інформацією та її усуспільнення були створені мережі ЕОМ, де усуспільнюватись програми і дані стали розміщувати на спеціальних обслуговуючих пристроях - файлових серверах.

СУБД, що працюють з файловими серверами, дозволяють безлічі користувачів різних ЕОМ (іноді розташованих досить далеко один від одного) отримувати доступ до одних і тих же баз даних. При цьому спрощується розробка різних автоматизованих систем управління організаціями, навчальних комплексів, інформаційних та інших систем, де безліч співробітників (учнів) повинні використовувати загальні дані й обмінюватися створюваними в процесі роботи (навчання). Однак при такій ідеології вся обробка запитів з програм або з терміналів користувача ЕОМ виконується на цих же ЕОМ. Тому для реалізації навіть простого запиту ЕОМ часто повинна зчитувати з файлового сервера і (або) записувати на сервер цілі файли, що веде до конфліктних ситуацій і перевантаження мережі.

Для виключення зазначених та деяких інших недоліків була запропонована технологія "Клієнт-Сервер", по якій запити користувальницьких ЕОМ (Клієнт) обробляються на спеціальних серверах баз даних (Сервер), а на ЕОМ повертаються лише результати обробки запиту. При цьому, природно, потрібен єдину мову спілкування з Сервером і в якості такої мови обраний SQL. Тому всі сучасні версії професійних реляційних СУБД (DB2, Oracle, Ingres, Informix, Sybase, Progress, Rdb) і навіть нереляційних СУБД (наприклад, Adabas) використовують технологію "Клієнт-Сервер" і мову SQL. До того ж приходять розробники СУБД персональних ЕОМ, багато з яких вже сьогодні забезпечені мовою SQL.
Існує думка: Оскільки велика частина запитів формулюється на SQL, практично байдуже, що це за СУБД - був би SQL.
Реалізація в SQL концепції операцій, орієнтованих на табличне представлення даних, дозволило створити компактний мову з невеликим (менше 30) набором пропозицій. SQL може використовуватися як інтерактивний (для виконання запитів) і як вбудований (для побудови прикладних програм). У ньому існують:

пропозиції визначення даних (визначення баз даних, а також визначення та знищення таблиць і індексів);

запити на вибір даних (пропозиція SELECT);

пропозиції модифікації даних (додавання, видалення і зміна даних);

пропозиції керування даними (надання та скасування привілеїв на доступ до даних, управління транзакціями і інші). Крім того, він надає можливість виконувати в цих пропозиціях:

арифметичні обчислення (включаючи різноманітні функціональні перетворення), обробку текстових рядків і виконання операцій порівняння значень арифметичних виразів і текстів;

упорядкування рядків і (або) стовпців при виведенні вмісту таблиць на друк або екран дисплея;

створення уявлень (віртуальних таблиць), що дозволяють користувачам мати свій погляд на дані без збільшення їх обсягу в базі даних;

запам'ятовування виведеного за запитом вмісту таблиці, декількох таблиць або уявлення в іншій таблиці (реляційна операція присвоювання).

агрегатування даних: групування даних і застосування до цих груп таких операцій, як середнє, сума, максимум, мінімум, число елементів і т.п.

У SQL використовуються наступні основні типи даних, формати яких можуть дещо відрізнятися для різних СУБД:

INTEGER
- Ціле число (зазвичай до 10 значущих цифр і знак);

SMALLINT
- "Коротке ціле" (зазвичай до 5 значущих цифр і знак);

DECIMAL (p, q)
- Десяткове число, яке має p цифр (0 <p <16) і знак; за допомогою q задається число цифр праворуч від десяткової точки (q <p, якщо q = 0, воно може бути опущено);

FLOAT
- Дійсне число з 15 значущими цифрами і цілочисловим порядком, визначеним типом СУБД;

CHAR (n)
- Символьний рядок фіксованої довжини з n символів (0 <n <256);

VARCHAR (n)
- Символьний рядок змінної довжини, що не перевищує n символів (n> 0 і різне в різних СУБД, але не менше 4096);

DATE
- Дата в форматі, визначеному спеціальною командою (за замовчуванням mm / dd / yy); поля дати можуть містити тільки реальні дати, що починаються за кілька тисячоліть до н.е. і обмежені п'ятим-десяти тисячоліть н.е.;

TIME
- Час у форматі, визначеному спеціальною командою, (за замовчуванням hh.mm.ss);

DATETIME
- Комбінація дати і часу;
MONEY
- Гроші в форматі, який визначає символ грошової одиниці ($, руб, ...) і його розташування (суфікс чи префікс), точність дробової частини і умова для показу грошового значення.
У деяких СУБД ще існує тип даних LOGICAL, DOUBLE та ряд інших. СУБД INGRES надає користувачеві можливість самостійного визначення нових типів даних, наприклад, площинні або просторові координати, одиниці різних метрик, п'яти-або шестиденні тижня (робочий тиждень, де відразу після п'ятниці або суботи слід понеділок), дробу, графіка, великі цілі числа (що стало дуже актуальним для російських банків) і т.п.
Орієнтований на роботу з таблицями SQL не має достатніх коштів для створення складних прикладних програм. Тому в різних СУБД він або використовується разом з мовами програмування високого рівня (наприклад, такими як Сі чи Паскаль), або включений до складу команд спеціально розробленої мови СУБД (язик систем dBASE, R: BASE і т.п.). Уніфікація повних мов сучасних професійних СУБД досягається за рахунок впровадження об'єктно-орієнтованої мови четвертого покоління 4GL. Останній дозволяє організовувати цикли, умовні речення, меню, екранні форми, складні запити до баз даних з інтерфейсом, орієнтованим як на алфавітно-цифрові термінали, так і на віконний графічний інтерфейс (X-Windows, MS-Windows).

6.2. Запити на читання даних.

Всі запити на отримання практично будь-якої кількості даних з однієї або декількох таблиць виконуються за допомогою єдиного пропозиції SELECT. У загальному випадку результатом реалізації пропозиції SELECT є інша таблиця (див. приклади п. 1.3). До цієї нової (робочої) таблиці може бути знову застосована операція SELECT і т.д., тобто такі операції можуть бути вкладені одна в одну. Представляє історичний інтерес той факт, що саме можливість включення одного речення SELECT всередину іншого послужила мотивуванням використання прикметника "структуризувати" в назві мови SQL.
Пропозиція SELECT може використовуватися як:
самостійна команда на отримання і вивід рядків таблиці, сформованої із стовпців і рядків однієї або декількох таблиць (подань);
елемент WHERE-або HAVING-умови (скорочений варіант пропозиції, званий "вкладений запит");
фраза вибору в командах CREAT VIEW, DECLARE CURSOR або INSERT;
засіб привласнення глобальним змінним значень із рядків сформованої таблиці (INTO-фраза).
У даній і наступній главах будуть розглянуті тільки два перші функції пропозиції SELECT, а тут - його синтаксис, обмежений конструкціями, що використовуються при реалізації цих функцій. Тут (так само як і в інших розділах книги) в синтаксичних конструкціях використовуються такі позначення:
зірочка (*) для позначення "все" - вживається в звичайному для програмування сенсі, тобто "Всі випадки, задовольняють визначенню";
квадратні дужки ([]) - означають, що конструкції, укладені в ці дужки, є необов'язковими (тобто можуть бути опущені);
фігурні дужки ({}) - означають, що конструкції, укладені в ці дужки, повинні розглядатися як цілі синтаксичні одиниці, тобто вони дозволяють уточнити порядок розбору синтаксичних конструкцій, замінюючи звичайні дужки, які використовуються в синтаксисі SQL;
многоточие (...) - вказує на те, що безпосередньо передує йому синтаксична одиниця факультативно може повторюватися один або більше разів;
пряма риса (|) - означає наявність вибору з двох або більше можливостей. Наприклад позначення ASC | DESC вказує, можна вибрати один з термінів ASC або DESC; коли ж один із елементів вибору укладений у квадратні дужки, то це означає, що він вибирається за замовчуванням (так, [ASC] | DESC означає, що відсутність усієї цієї конструкції буде сприйматися як вибір ASC);
крапка з комою (;) - завершальний елемент пропозицій SQL;

кома (,) - використовується для розділення елементів списків;
прогалини () - можуть вводитися для підвищення наочності між будь-якими синтаксичними конструкціями пропозицій SQL;
прописні жирні латинські літери та символи - використовуються для написання конструкцій мови SQL і повинні (якщо це спеціально не обумовлено) записуватися в точності так, як показано;
малі літери - використовуються для написання конструкцій, які повинні замінюватися конкретними значеннями, обраними користувачем, причому для визначеності окремі слова цих конструкцій зв'язуються між собою символом підкреслення (_);
терміни таблиця, стовпець, ... - Замінюють (з метою скорочення тексту синтаксичних конструкцій) терміни ім'я_таблиці, ім'я_стовпця, ..., відповідно;
термін таблиця - використовується для узагальнення таких видів таблиць, як базовая_табліца, уявлення або псевдонім; тут псевдонім служить для тимчасового (на момент виконання запиту) перейменування та (або) створення робочої копії базовой_табліци (подання).
Пропозиція SELECT (вибрати) має наступний формат:
підзапит [UNION [ALL] підзапит] ...
[ORDER BY {[таблиця.] Стовпець | номер_елемента_SELECT} [[ASC] | DESC]
[, {[Таблиця.] Стовпець | номер_елемента_SELECT} [[ASC] | DESC]] ...;
і дозволяє об'єднати (UNION) а потім упорядкувати (ORDER BY) результати вибору даних, отриманих за допомогою декількох "підзапитів". При цьому упорядкування можна проводити в порядку зростання - ASC (ASCending) або убування DESC (DESCending), а за замовчуванням приймається ASC.
У цьому реченні підзапит дозволяє вказати умови для вибору потрібних даних і (якщо потрібно) їх обробки
SELECT
(Вибрати) дані із зазначених стовпців і (якщо необхідно) виконати перед виведенням їх перетворення відповідно до зазначеними виразами і (або) функціями
FROM
(З) перерахованих таблиць, в яких розташовані ці стовпці
WHERE
(Де) рядки із зазначених таблиць повинні задовольняти зазначеного переліку умов відбору рядків
GROUP BY
(Групуючи по) зазначеного переліку стовпців з тим, щоб отримати для кожної групи єдине агреговане значення, використовуючи у фразі SELECT SQL-функції SUM (сума), COUNT (кількість), MIN (мінімальне значення), MAX (максимальне значення) або AVG (середнє значення)
HAVING
(Маючи) в результаті лише ті групи, які задовольняють зазначеному переліку умов відбору груп
і має формат
SELECT [[ALL] | DISTINCT] {* | елемент_SELECT [, елемент_SELECT] ...}
FROM {базовая_табліца | уявлення} [псевдонім]
[, {Базовая_табліца | уявлення} [псевдонім]] ...
[WHERE фраза]
[GROUP BY фраза [HAVING фраза]];
Елемент_SELECT - це одна з наступних конструкцій:
[Таблиця.] * | Значення | SQL_функція | сістемная_переменная
де значення - це:
[Таблиця.] Стовпець | (вираз) | константа | змінна
Синтаксис виразів має вигляд
({[[+] | -] {Значення | функція_СУБД} [+ | - | * | **]} ...)
а синтаксис SQL_функцій - одна з наступних конструкцій:

{SUM | AVG | MIN | MAX | COUNT} ([[ALL] | DISTINCT] [таблиця.] Стовпець)
{SUM | AVG | MIN | MAX | COUNT} ([ALL] вираз)
COUNT (*)

Фраза WHERE включає набір умов для відбору рядків:

WHERE [NOT] WHERE_условіе [[AND | OR] [NOT] WHERE_условіе] ..

де WHERE_условіе - одна з наступних конструкцій:

значення {= | <> | <| <= |> |> =} {значення | (підзапит)}
значення_1 [NOT] BETWEEN значення_2 AND значеніе_3
значення [NOT] IN {(константа [, константа] ...) | (підзапит)}
значення IS [NOT] NULL
[Таблиця.] Стовпець [NOT] LIKE 'строка_сімволов' [ESCAPE 'символ']

EXISTS (підзапит)

Крім традиційних операторів порівняння (= | <> | <| <= |> |> =) в WHERE фразі використовуються умови BETWEEN (між), LIKE (схоже на), IN (належить), IS NULL (не визначено) і EXISTS (існує), які можуть передувати оператором NOT (не). Критерій відбору рядків формується з одного або декількох умов, з'єднаних логічними операторами:

AND
- Коли повинні задовольнятися обидва поділюваних за допомогою AND умови;

OR
- Коли має задовольнятися одне з поділюваних за допомогою OR умов;

AND NOT
- Коли має задовольнятися перша умова і не повинно друге;

OR NOT
- Коли або має задовольнятися перша умова або не має задовольнятися друге,

причому існує пріоритет AND над OR (спочатку виконуються всі операції AND і тільки після цього операції OR). Для отримання бажаного результату WHERE умови мають бути введені в правильному порядку, який можна організувати введенням дужок.

При обробці умови числа порівнюються алгебраїчно - негативні числа вважаються меншими, ніж позитивні, незалежно від їх абсолютної величини. Рядки символів порівнюються відповідно до їх поданням в коді, використовуваному в конкретної СУБД, наприклад, в коді ASCII. Якщо порівнюються два рядки символів, що мають різні довжини, більш короткий рядок доповнюється праворуч пробілами для того, щоб вони мали однакову довжину перед здійсненням порівняння.

Нарешті, синтаксис фрази GROUP BY має вигляд

GROUP BY [таблиця.] Стовпець [, [таблиця.] Стовпець] ... [HAVING фраза]

GROUP BY ініціює перекомпоновку формованої таблиці по групах, кожна з яких має однакове значення в стовп-цях, включених до переліку GROUP BY. Далі до цих груп застосовуються агрегуються функції, зазначені у фразі SELECT, що призводить до заміни всіх значень групи на єдине значення (сума, кількість тощо).

За допомогою фрази HAVING (синтаксис якої майже не відрізняється від синтаксису фрази WHERE)

HAVING [NOT] HAVING_условіе [[AND | OR] [NOT] HAVING_условіе] ...

можна виключити з результату групи, не задовольняють заданим умовам:

значення {= | <> | <| <= |> |> =} {значення | (підзапит)
| SQL_функція}

{Значення_1 | SQL_функція_1} [NOT] BETWEEN
{Значення_2 | SQL_функція_2} AND {значеніе_3 | SQL_функція_3}

{Значення | SQL_функція} [NOT] IN {(константа [, константа] ...)
| (Підзапит)}

{Значення | SQL_функція} IS [NOT] NULL

[Таблиця.] Стовпець [NOT] LIKE 'строка_сімволов' [ESCAPE 'символ']

EXISTS (підзапит)

6.2. Агрегування даних

SQL-функції

У SQL існує низка спеціальних стандартних функцій (SQL-функцій). Крім спеціального випадку COUNT (*) кожна з цих функцій оперує сукупністю значень стовпця деякої таблиці і створює єдине значення, яке визначається так:

COUNT
- Число значень в стовпці,

SUM
- Сума значень в стовпці,

AVG
- Середнє значення в стовпці,

MAX
- Найбільше значення в стовпці,

MIN
- Найменше значення в стовпці.

Для функцій SUM і AVG розглянутий стовпець повинен містити числові значення.
Слід зазначити, що тут стовпець - це стовпець віртуальної таблиці, в якій можуть міститися дані не тільки з шпальти базової таблиці, але й дані, отримані шляхом функціонального перетворення і (або) зв'язування символами арифметичних операцій значень з одного або декількох стовпців. При цьому вираз, що визначає стовпець такої таблиці, може бути як завгодно складним, але не повинно містити SQL-функцій (вкладеність SQL-функцій не допускається). Однак з SQL-функцій можна складати будь-які вирази.
Аргументу всіх функцій, крім COUNT (*), може передувати ключове слово DISTINCT (різний), що вказує, що надлишкові дублюючі значення повинні бути виключені перед тим, як буде застосовуватися функція. Спеціальна ж функція COUNT (*) служить для підрахунку всіх без винятку рядків у таблиці (включаючи дублікати).

6.3. Функції без використання фрази GROUP BY

Якщо не використовується фраза GROUP BY, то в перелік елементов_SELECT можна включати лише SQL-функції або вирази, що містять такі функції. Іншими словами, не можна мати в списку стовпці, які не є аргументами SQL-функцій.
Наприклад, видати дані про масу лука (ПР = 10), проданого постачальниками, і вказати кількість цих постачальників:

```
SELECT SUM (К_во), COUNT (К_во)
FROM Поставки
WHERE ПР = 10;
```

Результат:

SUM (К_во)	COUNT (К_во)
220	2

Якби для виведення в результат ще й номери продукту був сформований запит

```
SELECT ПР, SUM (К_во), COUNT (К_во)
    FROM Поставки
    WHERE ПР = 10;
```

то було б отримано повідомлення про помилку. Це пов'язано з тим, що SQL-функція створює єдине значення з безлічі значень стовпця-аргументу, а для "вільного" стовпця повинно бути видано всі безліч його значень. Без спеціальної вказівки (воно задається фразою GROUP BY) SQL не буде з'ясовувати, однакові значення цієї множини (як в даному прикладі, де ПР = 10)

або різними (як було б при відсутності WHERE фрази). Тому подібний запит відкидається системою.

Правда, ніхто не забороняє дати запит

```
SELECT 'Кількість лука =', SUM (К_во), COUNT (К_во)
    FROM Поставки
    WHERE ПР = 10;
```

Результат:

'Кількість лука ='	SUM (К_во)	COUNT (К_во)
Кількість лука =	220	2

Відзначимо також, що в стовпці-аргументі перед застосуванням будь-якої функції, крім COUNT (*), виключаються всі невизначені значення. Якщо виявляється, що аргумент - порожня множина, функція COUNT приймає значення 0, а інші - NULL.
Наприклад, для отримання суми цін, середньої ціни, кількості продуктів і кількості різних цін продуктів, проданих коопторгом УРОЖАЙ (ПС = 5), а також для отримання кількості продуктів, які можуть поставлятися цим коопторгом, можна дати запит

```
SELECT SUM (Ціна), AVG (Ціна), COUNT (Ціна),
    COUNT (DISTINCT Ціна), COUNT (*)
    FROM Поставки
    WHERE ПС = 5;
```

і отримати

SUM (Ціна)	AVG (Ціна)	COUNT (Ціна)	COUNT (DISTINCT Ціна)	COUNT (*)
6.2	1.24	5	4	7

В іншому прикладі, де треба дізнатися "Скільки поставлено моркви і скільки постачальників її постачають?":

```
SELECT SUM (К_во), COUNT (К_во)
    FROM Поставки
    WHER ПР = 2;
```

буде отримана відповідь:

SUM (К_во)	COUNT (К_во)
-0 -	0

Нарешті, спробуємо отримати суму маси поставленого лука з його середньою ціною ("Чоботи з яєчнею"):

```
SELECT (SUM (К_во) + AVG (Ціна))
FROM Поставки
WHERE ПР = 10;
```

Результат:

SUM (К_во) + AVG (Ціна)
220.6

6.4. Фраза GROUP BY

Ми показали, як можна обчислити масу певного продукту, що поставляється постачальниками. Припустимо, що тепер потрібно обчислити загальну масу кожного з продуктів, що поставляються в даний час постачальниками. Це можна легко зробити за допомогою пропозиції

```
SELECT ПР, SUM (К_во)
    FROM Поставки
    GROUP BY ПР;
```

Результат показаний на рис. 2.3, а.

а)		б)				в)		г)	
ПР		**ПС**	**ПР**	**Ціна**	**К_во**	**ПР**		**ПР**	
9	0	1	9	-0 -	-0 -	1	370	9	0
11	150	3	9	-0 -	-0 -	2	0	11	150
12	30	5	9	-0 -	-0 -	3	250	12	30
15	370	1	11	1.50	50	4	100	15	70
1	370	5	11	-0 -	-0 -	5	170	1	370
3	250	6	11	-0 -	-0 -	6	220	3	250
5	170	8	11	1.00	100	7	200	5	70
6	220	1	12	3.00	10	8	150	6	140
8	150	3	12	2.50	20	9	0	8	150
7	200	6	12	-0 -	-0 -	10	220	7	200
2	0	1	15	2.00	170	11	150	2	0
4	100	3	15	1.50	200	12	30	4	100
13	190	2	1	3.60	300	13	190	13	190
14	70	7	1	4.20	70	14	70	14	70
16	250	2	3	-0 -	-0 -	15	370	16	250
17	50	7	3	4.00	250	16	250	17	50
10	220			...		17	50	10	220

Рис. 2.3. Ілюстрації до фрази GROUP BY

Фраза GROUP BY (групувати по) ініціює перекомпоновку зазначеної у FROM таблиці по групах, кожна з яких має однакові значення в стовпці, зазначеному в GROUP BY. У розглянутому прикладі рядки таблиці Поставки групуються так, що в одній групі містяться всі рядки для продукту з ПР = 1, в іншій - для продукту з ПР = 2 і т.д. (Див. мал. 2.3.б). Далі до кожної групи застосовується фраза SELECT. Кожен вираз в цій фразі повинно приймати єдине значення для групи, тобто воно може бути або значенням стовпця, зазначеного в GROUP BY, або арифметичним виразом, що включає це значення, або константою, або однією з SQL-функцій, яка оперує всіма значеннями стовпця в групі і зводить ці значення до єдиного значенню (наприклад, до суми) .

Відзначимо, що фраза GROUP BY не припускає ORDER BY. Щоб гарантувати впорядкування по ПР результату розглянутого прикладу (рис. 2.3, в) слід дати запит

```
SELECT ПР, SUM (К_во)
    FROM Поставки
    GROUP BY ПР
    ORDER BY ПР;
```

Нарешті, відзначимо, що рядки таблиці можна групувати по будь-якій комбінації її стовпців. Так, за запитом

```
SELECT Т, БЛ, COUNT (БЛ)
    FROM Замовлення
    GROUP BY Т, БЛ;
```

можна дізнатися коди і кількість порцій страв, замовлених відпочиваючими пансіонату (32 особи) на кожну з трапез наступного дня:

Т	БЛ	COUNT (БЛ)
1	3	18
1	6	14
1	19	17
1	21	15

...

Якщо в запиті використовуються фрази WHERE і GROUP BY, то рядки, які не задовольняють фразі WHERE, виключаються до виконання групування.
Наприклад, видати для кожного продукту його код і загальний обсяг можливих поставок, враховуючи тимчасову недієздатність постачальника з ПС = 2:

```
SELECT ПР, SUM (К_во)
    FROM Поставки
    WHERE ПС <> 2
    GROUP BY ПР;
```

Результат, наведений на рис. 2.3, г, відрізняється від результату (рис. 2.3, а) аналогічного запиту для всіх постачальників обсягом поставок продуктів з кодами 15, 5 і 6.

6.5. Вкладені підзапити

Види вкладених підзапитів

Вкладений підзапит - це підзапит, взятий у круглі дужки і вкладений в WHERE (HAVING) фразу пропозиції SELECT або інших пропозицій, що використовують WHERE фразу. Вкладений підзапит може містити в своїй WHERE (HAVING) фразі інший вкладений підзапит і т.д. Неважко здогадатися, що вкладений підзапит створений для того, щоб при відборі рядків таблиці, сформованої основним запитом, можна було використовувати дані з інших таблиць (наприклад, при відборі страв для меню використовувати дані про наявність продуктів у коморі пансіонату).
Існують прості і корельовані вкладені підзапити. Вони включаються в WHERE (HAVING) фразу з допомогою умов IN, EXISTS або однієї з умов порівняння (= | <> | <| <= |> |> =).
Прості вкладені підзапити обрабативаютя системою "знизу вгору". Першим обробляється вкладений підзапит самого нижнього рівня. Безліч значень, отримане в результаті його виконання, використовується при реалізації підзапиту більш високого рівня і т.д.
Запити з корельованими вкладені підзапити обробляються системою в зворотному порядку. Спочатку вибирається перший рядок робочої таблиці, сформованої основним запитом, і з неї вибираються значення тих стовпців, які використовуються у вкладеному підзапит (вкладених підзапит). Якщо ці значення задовольняють умовам вкладеного підзапиту, то вибрана рядок включається в результат. Потім вибирається другий рядок і т.д., поки в результат не будуть включені всі рядки, що задовольняють вкладені підзапити (послідовності вкладених підзапитів).
Слід зазначити, що SQL володіє великою надмірністю в тому сенсі, що він часто надає декілька різних способів формулювання одного і того ж запиту. Тому в багатьох прикладах даної глави будуть використані вже знайомі нам по попередньому розділі концептуальні формулювання запитів. І незважаючи на те, що частина з них успішніше реалізується за допомогою з'єднань, тут все ж будуть приведені їхні варіанти з використанням вкладених підзапитів. Це пов'язано з необхідністю детального знайомства зі створенням і принципом виконання вкладених підзапитів, так як існує чимало завдань (особливо на видалення і зміна даних), які не можуть бути реалізовані іншим способом. Крім того, різні формулювання одного і того ж запиту вимагають для свого виконання різних ресурсів пам'яті і можуть значно відрізнятися за часом реалізації в різних СУБД.

6.6. Прості вкладені підзапити

Прості вкладені підзапити використовуються для представлення множини значень, дослідження яких має здійснюватися в будь-якому предикат IN, що ілюструється в наступному прикладі: видати назву і статус постачальників продукту з номером 11, тобто помідорів.

```
SELECT Назва, Статус
FROM Постачальники
WHERE ПС IN
  (SELECT ПС
        FROM Поставки
        WHERE ПР = 11);
```

Результат:

Назва	Статус
Ситний	ринок
УРОЖАЙ	кооптorг
ЛІТО	агрофірма
Корюшка	кооператив

Як вже зазначалося в п. 3.3.1 , при обробці повного запиту система виконує насамперед вкладений підзапит. Цей підзапит видає безліч номерів постачальників, які постачають продукт з кодом ПР = 11, а саме безліч (1, 5, 6, 8). Тому початковий запит еквівалентний такому простому запиту:

```
SELECT Назва, Статус
    FROM Постачальники
    WHERE ПС IN (1, 5, 6, 8);
```

Підзапит з декількома рівнями укладення можна проілюструвати на тому ж прикладі. Нехай потрібно дізнатися не постачальників продукту 11, як це робилося в попередніх запитах, а постачальників помідорів, що є продуктом з номером 11. Для цього можна дати запит

```
SELECT Назва, Статус
    FROM Постачальники
    WHERE ПС IN
    (SELECT ПС
        FROM Поставки
        WHERE ПР IN
            (SELECT ПР
                FROM Продукти
                WHERE Продукт = 'Помідори'));
```

В даному випадку результатом самого внутрішнього підзапиту є тільки одне значення (11). Як вже було показано вище, підзапит наступного рівня в свою чергу дає в результаті безліч (1, 5, 6, 8). Останній, самий зовнішній SELECT, обчислює наведений вище остаточний результат. Взагалі допускається будь-яка глибина вкладеності підзапитів.

Той же результат можна отримати за допомогою з'єднання

```
SELECT Назва, Статус
    FROM Постачальники, Поставки, Продукти
    WHERE Поставщікі.ПС = Поставкі.ПС
    AND Поставкі.ПР = Продукти.ПР
    AND Продукт = 'Помідори';
```

При виконанні цього компактного запиту система повинна одночасно обробляти дані з трьох таблиць, тоді як у попередньому прикладі ці таблиці обробляються по черзі. Природно, що для їх реалізації тебуются різні ресурси пам'яті і часу, однак цього неможливо відчути при роботі з обмеженим обсягом даних в ілюстративної базі ПАНСІОН.

6.7. Використання однієї і тієї ж таблиці в зовнішньому і вкладених підзапитів

Видати номери постачальників, які постачають хоча б один продукт, що поставляється постачальником 6.

	Результат:
SELECT DISTINCT ПС	ПС
FROM Поставки	1
WHERE ПР IN	3
(SELECT ПР	5
FROM Поставки	6
WHERE ПС = 6);	8

Відзначимо, що посилання на Поставки у вкладеному підзапит означає не те ж саме, що посилання на Поставки в зовнішньому запиті. В дійсності, два імені Поставки позначають різні значення. Щоб цей факт став явним, корисно використовувати псевдоніми, наприклад, X і Y:

```
SELECT DISTINCT X.ПС
     FROM Поставки X
     WHERE X.ПР IN
     (SELECT Y.ПР
          FROM Поставки Y
          WHERE Y.ПС = 6);
```

Тут X і Y - довільні псевдоніми таблиці Поставки, які визначаються у фразі FROM і використовувані як явні уточнітелі у фразах SELECT і WHERE. Нагадаємо, що псевдоніми визначені лише в межах одного запиту.

6.8. Запити на оновлення даних

Пропозиція UPDATE

Оновлення єдиною записи

Змінити назву блюда з кодом БЛ = 5 на Форшмак, збільшити його вихід на 30 г і встановити NULL-значення в стовпець Праця.

```
UPDATE Страви
     SET Блюдо = 'Форшмак', Вихід = (Вихід +30), Праця =
NULL
     WHERE БЛ = 5;
```

Оновлення безлічі записів

Потроїти ціну всіх продуктів таблиці поставки (крім ціни кави - ПР = 17).

```
UPDATE Поставки
     SET Ціна = Ціна * 3
     WHERE ПР <> 17;
```

Оновлення з підзапитом

Встановити рівною нулю ціну і К_во продуктів для постачальників з Паневежиса і Резекне.

```
UPDATE Поставки
     SET Ціна = 0, К_во = 0
     WHERE ПС IN
     (SELECT ПС
          FROM Постачальники
          WHERE Місто IN ('Паневежис', 'Резекне'));
```

Оновлення декількох таблиць

Змінити номер продукту ПР = 13 на ПР = 20.

```
UPDATE Продукти UPDATE Склад
    SET ПР = 20 SET ПР = 20
    WHERE ПР = 13; WHERE ПР = 13;

    UPDATE Поставки UPDATE Наявність
    SET ПР = 20 SET ПР = 20
    WHERE ПР = 13; WHERE ПР = 13;
```

На жаль в єдиним запиті неможливо оновити більш однієї таблиці, а так як код продукту входить в чотири таблиці, то довелося видати чотири подібних запиту. Це може призвести до протиріччя бази даних (порушення цілісності по посиланнях), оскільки після виконання першого речення таблиці Склад, Поставки і Наявність посилаються на вже неіснуючий продукт. База стає несуперечливою тільки після виконання четвертого запиту.

ТЕМА 7.

Запити на створення та оновлення схеми БД, таблиць та представлень.

7.1. Системний каталог

Системний каталог - це набір таблиць, в яких міститься інформація, необхідна для правильного функціонування СУБД: про підтримуваних базах даних і їх базових таблицях, представлених, курсор, індексах, користувачів і їх права доступу до інформації, правилах модифікації даних і т.д. У різних СУБД, що підтримують SQL, існує від десятка до декількох десятків системних таблиць, структура яких нічим не відрізняється від уже знайомої нам структури користувальницьких таблиць.

Так, в кожному рядку системної таблиці SYSTABLES зберігається опис однієї з таблиць користувальницьких або системної баз даних. Для кожної з них вказується ім'я таблиці, ім'я користувача, який створив цю таблицю, число стовпців в ній і ряд інших елементів інформації. У таблиці SYSCOLUMNS міститься рядок для кожного стовпця кожної таблиці, в якій вказано ім'я стовпця, ім'я таблиці, частиною якої є даний стовпець, тип даних для цього стовпця та багато іншої інформації про стовпці.

За допомогою пропозиції SELECT користувач може отримати інформацію з будь системної таблиці. Наприклад, він може дати запит на отримання імен таблиць, числа їх стовпців і рядків, власника і короткого опису (якщо таке вводилося в базу даних):

```
SELRCT Tab_name, N_col, N_row, Tab_owner, Comments
    FROM SYSTABLES;
```

і отримати результат, показаний на рис. 5.1, а.

Для отримання ж деяких даних про стовпцях таблиці Страви можна дати запит

```
SELECT Col_name, Type, Length, Comments
    FROM SYSCOLUMNS
    WHERE Tab_name = 'Страви';
```

і отримати результат, показаний на рис. 5.1, б.

а)

Tab_name	N_col	N_row	Tab_owner	Comments
...				
SYS_TABLES	11		SYSTEM	
SYS_COLUMNS	14		SYSTEM	
...				
Страви	6	33	KIRILLOW	Перелік страв, відомих шеф-кухарю
Поставки	4	37	GROMOW	Дані про поставляються продуктах

Від_блюд	2	5	KIRILLOW	Перелік видів страв
Трапези	2	3	GROMOW	Перелік трапез в пансіонаті
Склад	3	148	KIRILLOW	Склад блюд
Продукти	11	17	KIRILLOW	Таблиця продуктів

...

б)

Col_name	Type	Length	Comments
БЛ	INTEGER	4	Код страви
Блюдо	TEXT	16	Назва страви
В	TEXT	1	Код виду страви (З, С, ...)
Основа	TEXT	6	Основний продукт у страві
Вихід	REAL	4	Маса порції готової страви
Праця	INTEGER	4	Вартість приготування страви (коп)

Рис. 5.1. Результати запитів по системним таблицям

Користувач, не знайомий зі структурою бази даних, може за допомогою подібного роду запитів одержати інформацію про такий структурі. Для цього йому треба володіти мовою SQL і трохи подумати.

На закінчення слід зазначити, що СУБД не дозволяє оновлювати каталог з допомогою пропозицій DELETE, INSERT і UPDATE. Оновлення проводиться тільки при створенні, модифікації або знищення таблиць, індексів, правил і т.п. за допомогою пропозицій, що розглядаються нижче.

7.2. Створення і знищення базових таблиць

Базові таблиці описуються в SQL за допомогою пропозиції CREATE TABLE (створити таблицю), синтаксис якого має невеликі відмінності в різних СУБД. Проте всі вони підтримують наступну мінімальну форму:

```
CREATE TABLE базовая_табліца (стовпець тип_даних [NOT NULL]
                        [, Стовпець тип_даних [NOT NULL]] ...);
```

де тип_даних повинен належати до одного з типів даних, підтримуваних СУБД (наприклад, одному з типів даних, перелічених у п.1.2).

Так, опис таблиці Страви може бути записано у вигляді

```
CREATE TABLE Страви
       (БЛ SMALLINT NOT NULL,
             Страви CHAR (70) NOT NULL,
             У CHAR (1),
             Основа CHAR (10),
             Вихід FLOAT,
             Праця SMALLINT);
```

У результаті створюється порожня базова таблиця Страви, а в системний каталог поміщається рядок, що описує цю таблицю. Відзначимо, що в професійних СУБД ім'я таблиці доповнюється ім'ям користувача, який видав пропозицію CREATE TABLE. Якщо цей користувач зареєстрований в системі під ім'ям Kirillov, то в каталозі буде зареєстрована таблиця Kirillov.Блюда і зазначений користувач може звертатися до неї по імені Kirillov.Блюда або за скороченим імені Страви, яке використовувалося у всіх попередніх прикладах і буде використовуватися далі.

Конструкція NOT NULL забороняє використання невизначеного значення, тобто спеціального значення, яке вводиться для представлення "невідомого значення" або "незастосовні значення". Наприклад, рядок поставки таблиці Поставки може містити невизначене значення в стовпці Ціна і (або) К_во (извесно, що постачальник поставляє

зазначений продукт, але на даний момент невідома ціна цього продукту і (або) обсяг поставки).
Існуючу базову таблицю можна в будь-який момент знищити за допомогою пропозиції DROP TABLE (знищити таблицю):

```
DROP TABLE базовая_табліца;
```

по якому віддаляється опис таблиці, її дані, пов'язані з нею уявлення і індекси, побудовані для стовпців таблиці (див. п. 5.3).
У SQL існує також пропозицію ALTER TABLE (змінити таблицю), яке дозволяє додати праворуч до таблиці новий стовпець, тобто модифікувати опис табіци. Так як без нього "можна жити", а обсяг книги обмежений, то ми не будемо тут описувати цю пропозицію.

7.3 . Індекси продуктивності

Для прискорення пошуку даних можна створювати індекси. Індекс - це системна таблиця, побудована по значеннях заданого стовпця заданої таблиці. У ньому розміщується перелік унікальних значень зазначеного стовпця таблиці з посиланнями на ті її рядки, де зустрічаються ці значення (структура, схожа на предметний покажчик книги). Наприклад, індекс, побудований для стовпця Основа таблиці Блюда, буде містити такі відомості:

Значення стовпця	**Рядки, в яких зустрічається таке значення**							
Кава	32	33						
Крупа	20	21						
Молоко	7	8	12	18	22	24	28	31
М'ясо	2	6	9	13	14			
Овочі	1	3	17	23	15			
Риба	4	5	10	11				
Фрукти	25	26	27	29	30			
Яйця	16	19						

Відзначимо, що такий індекс вже існував (у дещо іншій формі) у базі даних, хоча ця обставина ніяк не вплинуло на текст ілюстраційні пропозицій SELECT, DELETE, INSERT і UPDATE. SQL навмисно не включає в свої конструкції посилання на індекси. Рішення про те, використовувати або не використовувати будь-якої індекс при обробці деякого конкретного запиту приймається не користувачем, а оптимізатором СУБД, який враховує безліч факторів - розмір таблиць, тип використовуваних структур зберігання даних, статистичний розподіл даних у таблицях і індексах і т.д . Однак щоб оптимізатор зміг використовувати індекси, їх потрібно побудувати (щоб виграти в лотерею потрібно, принаймні, мати лотерейний квиток).
Природно, що пошук якого-небудь значення шляхом послідовного перебору невпорядкованих даних буде у багато разів повільніше, ніж пошук з використанням впорядкованого списку (індексу). Ясно також, що таблицю можна впорядкувати лише за даними одного стовпця, тоді як пошук часто доводиться здійснювати за даними декількох стовпців. По декількох стовп-ЦАМ виробляється і з'єднання таблиць. Тому, незважаючи на те, що індекси збільшують обсяг бази даних, їх слід використовувати як для окремих стовпців таблиці, так і для комбінації декількох її стовпців (наприклад, для комбінації: Прізвище, Ім'я, По батькові).
Для побудови індексу в SQL існує пропозиція CREATE INDEX (створити індекс), що має формат

```
CREATE [UNIQUE] INDEX імя_індекса ON базовая_табліца
       (Стовпець [[ASC] | DESC] [, стовпець [[ASC] | DESC]] ...);
```

де UNIQUE (унікальний) вказує, що ніяким двом рядкам в индексируемой базовій таблиці не дозволяється приймати одне і те ж значення для індексуємого стовпця (або комбінації стовпців) в один і той же час.
Наприклад, індекси для стовпців БЛ і Основа таблиці Страви створюються за допомогою пропозицій

```
CREATE UNIQUE INDEX Блюда_БЛ ON Страви (БЛ);
       CREATE INDEX Блюда_Основа ON Страви (Основа);
```

а індекс для первинного ключа (стовпці БЛ і ПР) таблиці Склад - за допомогою пропозиції
```
CREATE UNIQUE INDEX Состав_БЛ_ПР ON Склад (БЛ, ПР);
```
У великих (понад 1000 рядків) таблицях пошук індексованих значень виконується на порядок швидше, ніж пошук неіндексованих, а в дуже великих таблицях - на два-три порядки.

Так може бути, якщо дозволяє пам'ять, слід побудувати індекси для всіх стовпців всіх таблиць бази даних?

Якщо база даних не повинна модифікуватися, то на це питання можна дати позитивну відповідь. Однак при видаленні або додаванні рядка таблиці повинні бути перебудовані всі індекси, побудовані для її стовпців, а при зміні значення індексованого стовпця - індекс цього стовпця. Коли модифікується багато - кілька сотень (тисяч) рядків - і після модифікації кожного рядка перебудовуються всі її індекси, час модифікації може бути на порядок (кілька порядків) більше часу модифікації рядків з неіндексовані стовп-цями. Тому перед модифікацією безлічі рядків таблиці доцільно знищити індекси її стовпців, що можна зробити за допомогою пропозиції DROP INDEX (знищити індекс), що має наступний формат:
```
DROP INDEX імя_індекса;
```
Так як індекси можуть створюватися або знищуватися в будь-який час, то перед виконанням запитів доцільно будувати індекси лише для тих стовпців, які використовуються в WHERE і ORDER BY фразах запиту, а перед модифікацією великого числа рядків таблиць з індексованими стовпцями ці індекси слід знищити.