Лекція 3
Реляційні СУБД та мова SQL

1 Основні поняття реляційних баз даних

Дамо визначення реляційної бази даних.

Реляційною називається база даних, у якій усі дані, доступні користувачу, і організовані у вигляді таблиць, а всі операції над даними зводяться до операцій над цими таблицями.

Наведене визначення не залишає місця вбудованим покажчикам, що є в ієрархічних і мережних СУБД. Незважаючи на це, реляційна СУБД також здатна реалізувати відносини предок/нащадок, однак ці відносини представлені винятково значеннями даних, що містяться в таблицях.

У реляційній базі даних інформація організована у вигляді таблиць, розділених на рядки і стовпці, на перетинанні яких містяться значення даних. Кожна таблиця має унікальне ім'я.

Більш наочно структуру таблиці ілюструє рис 3.1, на якому зображена таблиця ДЕТАЛЬ. Кожен горизонтальний *рядок* цієї таблиці представляє окрему фізичну сутність — одну деталь. Кожен вертикальний *стовпець* таблиці ДЕТАЛЬ представляє один елемент даних для кожної з деталей. На перетині кожного рядка з кожним стовпцем таблиці міститься одне значення даних. Усі значення, що містяться в тому самому стовпці, є даними одного типу. Наприклад, у стовпці **КОЛІР** містяться тільки слова, у стовпці **ВАГА** - числа. Стовпці таблиці упорядковані з ліва на право, їхній порядок визначається при створенні таблиці. У будь-якій таблиці завжди є як мінімум один стовпець. У стандарті ANSI/ISO не вказується максимально припустиме число стовпців у таблиці, однак майже у всіх комерційних БД цю межу існує і звичайно складає приблизно 255 стовпців.

На відміну від стовпців, рядки таблиці не мають визначеного порядку. У таблиці може міститися будь-яка кількість рядків. Цілком припустиме

існування таблиці з нульовою кількістю рядків. Така таблиця називається *порожньою.*

Основними поняттями реляційних баз даних є:

- тип даних,
- домен,
- атрибут,
- кортеж,
- первинний ключ
- таблиця.

Для початку покажемо зміст цих понять на прикладі бази даних ДЕТАЛЬ.

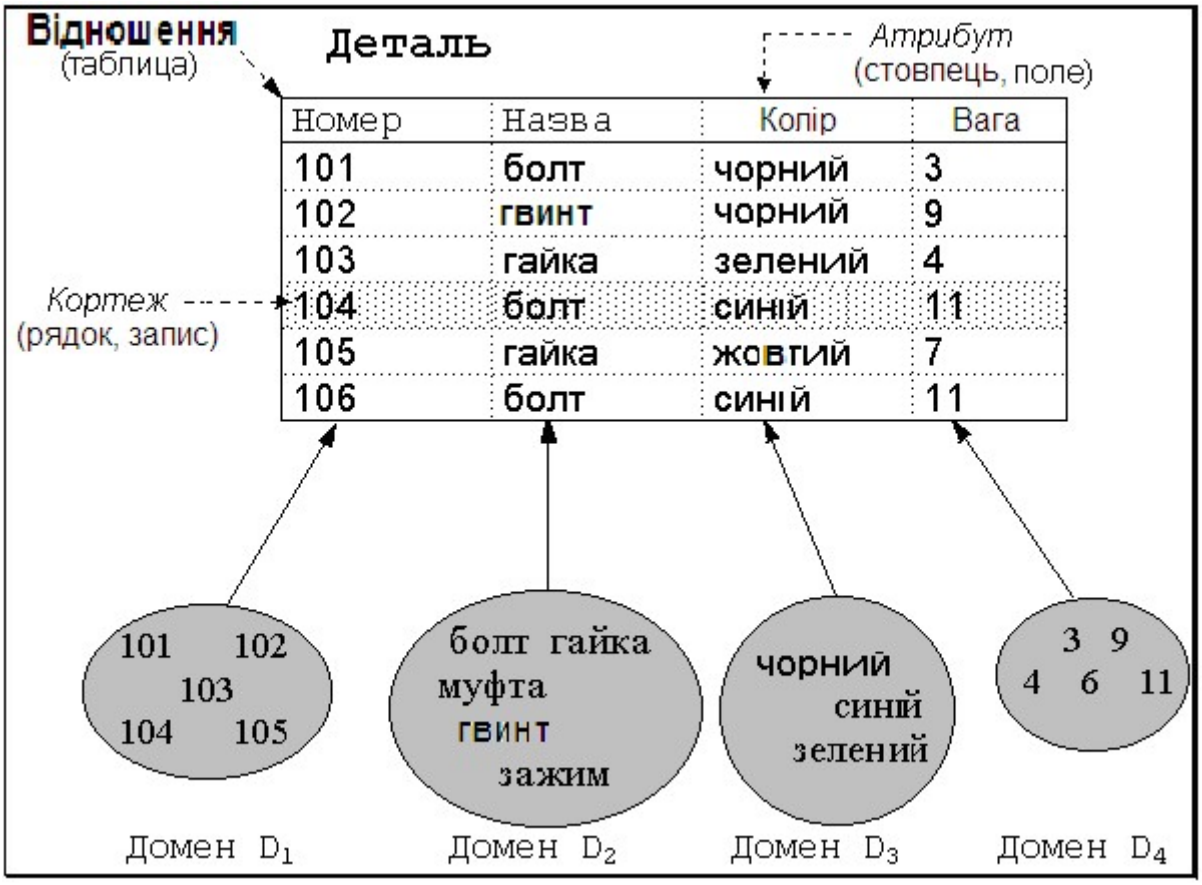

Рис. 3.1. Приклад бази даних «Деталь».

Поняття тип даних у реляційної моделі даних цілком адекватно поняттю типу даних у мовах програмування. Звичайно в сучасних реляційних БД допускається збереження символьних, числових даних,

бітових рядків, спеціалізованих числових даних (таких як "гроші"), а також спеціальних даних (дата, час, часовий інтервал). Досить активно розвивається підхід до розширення можливостей реляційних систем абстрактними типами даних. Відповідними можливостями володіють, наприклад, системи сімейства Ingres/Postgres. ***У нашому прикладі ми маємо справу з даними трьох типів: рядки символів, цілі числа і "гроші***".

Поняття *домена* більш специфічно для баз даних, хоча і має деякі аналогії з підтипами в деяких мовах програмування. ***У самому загальному вигляді домен визначається завданням деякого базового типу даних, до якого відносяться елементи домена, і довільного логічного виразу, застосовуваного до елемента типу даних.*** Якщо обчислення цього логічного вираження дає результат "істина", то елемент даних є елементом домена.

Найбільш правильним інтуїтивним трактуванням поняття домена є розуміння домена як припустимої потенційної множини значень даного типу. Наприклад, домен "Імена" у нашому прикладі визначений на базовому типі рядків символів, але в число його значень можуть входити тільки ті рядки, що можуть зображувати ім'я (зокрема, такі рядки не можуть починатися з м'якого знака).

Слід зазначити також семантичне навантаження поняття домена: дані вважаються порівнянними тільки в тому випадку, коли вони відносяться до одному домену. У нашому прикладі значення доменів "Номера пропусків" і "Номера груп" відносяться до типу цілих чисел, але не є порівнянними. Помітимо, що в більшості реляційних СУБД понять домена не використовується, хоча в Oracle V.7 воно вже підтримується.

Схема відносини - це іменована множина пара {ім'я атрибута, ім'я домена (чи типу, якщо поняття домена не підтримується)}. Ступінь чи "арность" схеми відносини - потужність цієї множини. Ступінь відносини ДЕТАЛЬ дорівнює чотирьом, тобто воно є 4-арным. Якщо всі атрибути одного відношення визначені на різних доменах, осмислено

використовувати для іменування атрибутів імена відповідних доменів (не забуваючи, звичайно, про те, що це є усього лише зручним способом іменування і не усуває розходження між поняттями домену й атрибуту). Схема БД (у структурному змісті) - це набір іменованих схем відносин.

Кортеж, що відповідає даній схемі відносини, - це множина пар {ім'я атрибута, значення}, що містить одне входження кожного імені атрибута, що належить схемі відносини. "Значення" є припустимим значенням домену даного атрибуту (чи типу даних, якщо поняття домена не підтримується). Тим самим, ступінь або "арність" кортежу, тобто число елементів у ньому, збігається з "арністю" відповідної схеми відносини**.** ***Попросту говорячи, кортеж - це набір іменованих значень заданого типу. Відношення - це множина кортежів, що відповідають одній схемі відносини. Іноді, щоб не плутатися, говорять "відношення-схема" і "відношення-екземпляр", іноді схему відносини називають заголовком відносини, а відношення як набір кортежів - тілом відносини***. Насправді, поняття схеми відносини ближче усього до поняття структурного типу даних у мовах програмування. Було б цілком логічно дозволяти окремо визначати схему відносини, а потім одне чи кілька відносин з даною схемою.

Однак у реляційних базах даних це не прийнято. Ім'я схеми відносини в таких базах даних завжди збігається з ім'ям відповідного відношення - екземпляра. У класичних реляційних базах даних після визначення схеми бази даних змінюються тільки відношення-екземпляри. У них можуть з'являтися нові і віддалятися чи модифікуватися існуючі кортежі. Однак у багатьох реалізаціях допускається і зміна схеми бази даних: визначення нових і зміна існуючих схем відносини. ***Це прийнято називати еволюцією схеми бази даних.***

Звичайним життєвим представленням відносини є таблиця, заголовком якої є схема відносини, а рядками - кортежі відношення-екземпляра; у цьому випадку імена атрибутів іменують стовпці цієї таблиці. Тому іноді говорять "стовпець таблиці", маючи на увазі "*атрибут*

відносини". Коли ми перейдемо до розгляду практичних питань організації реляційних баз даних і засобів управління, ми будемо використовувати цю життєву термінологію. Цієї термінології дотримують у більшості комерційних реляційних СУБД.

Реляційна база даних - це набір відносин, імена яких збігаються з іменами схем відносин у схемі БД.

Як видно, основні структурні поняття реляційної моделі даних (якщо не вважати поняття домену) мають дуже просту інтуїтивну інтерпретацію, хоча в теорії реляційних БД усі вони визначаються абсолютно формально і точно. Оскільки рядка в реляційної таблиці не упорядковані, не можна вибрати рядок по її номеру в таблиці. У таблиці ні "першої", "останньої" чи "тринадцятої" рядка. Тоді яким же образом можна вказати в таблиці конкретний рядок?

У правильно побудованій реляційної базі даних у кожній таблиці є один чи кілька стовпців, значення в який у всіх рядках різні. Цей стовпець (стовпці) називається первинним ключем таблиці.
Первинний ключ – унікальний ідентифікатор для таблиці. Його наявність припускає, що існує чи стовпець сукупність стовпців, що не містять однакових значень.

Таблиця products, фрагмент якої показаний на рис. 3.2, являє приклад таблиці, у якій первинний ключ являє собою комбінацію стовпців. Такий первинний ключ називається ***складеним.*** Стовпець mfr_id містить ідентифікатори виробників усіх товарів, перерахованих у таблиці, а стовпець product_id містить номера, привласнені товарам виробниками. Може показатися, що стовпець product_id міг би й один виконувати роль первинного ключа, однак ніщо не заважає двом різним виробникам привласнити своїм виробам однакові номери. Таким чином, як первинний ключ таблиці products необхідно використовувати комбінацію стовпців mfs_id і product_id. Для кожного з товарів, що містяться в таблиці, комбінація значень у цих стовпцях буде унікальною.

Первинний ключ для кожного рядка таблиці є унікальним, тому в таблиці з первинним ключем немає двох зовсім однакових рядків. ***Таблиця, у якій усі рядки відрізняються друг від друга, у математичних термінах називається відношенням***. Саме цьому терміну реляційні бази даних і зобов'язані своєю назвою, оскільки в їхній основі лежать відносини (таблиці в яких відрізняються друг від друга рядки).

Хоча первинні ключі є важливою частиною реляційної моделі даних, у перших реляційних СУБД (System/R, DB2, Oracle і інших) не була забезпечена явно їхня підтримка. Як правило, проектувальники бази даних самі стежили за тим, щоб у всіх таблиць були первинні ключі, однак у самих СУБД не було можливості визначити для таблиці первинний ключ. І тільки в СУБД DB2 Version 2, що з'явилася в квітні 1988 року, компанія IBM реалізувала підтримку первинних ключів. Після цього подібна підтримка була додана в стандарт ANSI/ISO.

Стовпець однієї таблиці, значення в який збігаються зі значеннями стовпця, що є первинним ключем іншої таблиці, називається зовнішнім ключем.

Одним з відмінностей реляційної моделі від перших моделей представлення даних було те, що в ній були відсутні явні покажчики, які використовуються для реалізації відносин предок/нащадок в ієрархічній моделі даних. Однак цілком очевидно, що відносини предок/нащадок існують і в реляційних базах даних. Наприклад, кожен службовець закріплений за конкретним офісом, тому ясно, що між рядками таблиці ФІРМА і таблиці СЛУЖБОВЦІ існує відношення.

Відношення предок/нащадок, яке існує між офісами і працюючими в них людьми, у реляційної моделі не загублено; просто воно реалізовано у вигляді однакових значень даних, що зберігаються в двох таблицях, а не у вигляді явного покажчика. Усі відносини, що існують між таблицями реляційної бази даних, реалізуються в такому вигляді. Сукупно первинний і

зовнішній ключі створюють між таблицями, у яких вони містяться, таке ж відношення предок/нащадок, як і в ієрархічній базі даних.

Зовнішній ключ, як і первинний ключ, теж може являти собою комбінацію стовпців. На практиці зовнішній ключ завжди буде складеним (що складається з кількох стовпців), якщо він посилається на складений первинний ключ в іншій таблиці. Очевидно, що кількість стовпців і їхній типи даних у первинному і зовнішньому ключах збігаються.
Якщо таблиця зв'язана з кількома іншими таблицями, вона може мати кілька зовнішніх ключів. Реляційна модель даних має всі можливості мережевої моделі по частині виразу складних відносин.

Зовнішні ключі є невід'ємною частиною реляційної моделі, оскільки реалізують відносини між таблицями бази даних. До нещастя, як і у випадку з первинними ключами, підтримка зовнішніх ключів була відсутня в перших реляційних БД. Вона була введена в системі DB2 Version 2 і тепер є у всіх комерційних БД.

2 Фундаментальні властивості відносин

Зупинимося тепер на деяких важливих властивостях відносин, що випливають із приведених раніше визначень. Та властивість, що відносини не містять кортежів-дублікатів, випливає з визначення відносини як множини кортежів. У класичній теорії множин по визначенню кожна множина складається з різних елементів.

З цієї властивості випливає наявність у кожного відношення первинного ключа - набору атрибутів, значення яких однозначно визначають кортеж відносини. Для кожного відношення принаймні повний набір його атрибутів має цю властивість. Однак при формальному визначенні первинного ключа потрібно забезпечення його "мінімальності", тобто в набір атрибутів первинного ключа не повинні входити такі атрибути, які можна відкинути без збитку для основної властивості - однозначно

визначати кортеж. Поняття первинного ключа є винятково важливим у зв'язку з поняттям цілісності баз даних**.**

Властивість відсутності упорядкованості кортежів відносини також є наслідком визначення відношення-екземпляра як множини кортежів. Відсутність вимоги до підтримки порядку на множині кортежів відносини дає додаткову гнучкість СУБД при збереженні баз даних у зовнішній пам'яті і при виконанні запитів до бази даних. Це не суперечить тому, що при формулюванні запиту до БД, наприклад, мовою SQL можна зажадати сортування результуючої таблиці у відповідності зі значеннями деяких стовпців. Такий результат, узагалі говорячи, не відношення, а деякий упорядкований список кортежів.

Атрибути відносин не упорядковані, оскільки по визначенню схема відносини є множина пар {ім'я атрибута, ім'я домену}. Для посилання на значення атрибута в кортежі відносини завжди використовується ім'я атрибута. Це властивість теоретично дозволяє, наприклад, модифікувати схеми існуючих відносин не тільки шляхом додавання нових атрибутів, але і шляхом видалення існуючих атрибутів. Однак у більшості існуючих систем така можливість не допускається, і хоча упорядкованість набору атрибутів відносини явно не потрібно, часто як неявний порядок атрибутів використовується їхній порядок у лінійній формі визначення схеми відносини.

Значення всіх атрибутів є атомарними. Це випливає з визначення домена як потенційної множини значень простого типу даних, тобто серед значень домена не можуть міститися множини значень (відносини). У прикладі приведеному на рис. 3.2 атрибут ВІДДІЛ не є атомарним, тому що він має внутрішню структуру.

Рис. 3.2. Атрибут ВІДДІЛ не є атомарним, він має внутрішню структуру Значення даного атрибута можна представити наступним відношенням (табл. 2), у якому кожен атрибут тепер містить атомарні значення. Таблиця 2.

СП_НОМЕР	СП_ІМ'Я	СП_ЗАРП	НОМЕР_ВІДД
2934	Іванов	112,000	310
2935	Петров	144,000	310
2936	Сидоров	92,000	313
2937	Федоров	110,000	310
2938	Іванова	112,000	315

Нормалізовані відносини складають основу класичного реляційного підходу до організації баз даних. Вони мають деякі обмеження (не будь-яку інформацію зручно представляти у вигляді плоских таблиць), але істотно спрощують маніпулювання даними.

3 Реляційна модель даних

Коли в попередніх розділах ми говорили про основні поняття реляційних баз даних, ми не спиралися на яку-небудь конкретну реалізацію. Ці міркування в однаковій мірі відносилися до будь-якої системи, при побудові якої використовувався реляційний підхід.

Іншими словами, ми використовували поняття так називаної реляційної моделі даних. Модель даних описує деякий набір родових понять і ознак, якими повинні володіти всі конкретні СУБД і керовані ними бази даних, якщо вони ґрунтуються на цій моделі. Наявність моделі даних дозволяє порівнювати конкретні реалізації, використовуючи одну загальну мову.

Найбільш розповсюджене трактування реляційної моделі даних належать Дейту, що відтворює її (з різними уточненнями) практично у всіх своїх книгах. Згідно Дейту реляційна модель складається з трьох частин, що описують різні аспекти реляційного підходу:

- структурної частини;
- маніпуляціоної частини;
- цілісної частини.

У ***структурній частині*** моделі фіксується, що єдиною структурою даних, використовуваної в реляційних БД, є так називане *нормалізоване* ***(нормалізація – покроковий процес приведення даних до двомірної табличної форми***) n-арне відношення. По суті справи, у попередніх двох розділах цієї лекції ми розглядали саме поняття і властивості структурної складової реляційної моделі.

У маніпуляціоній частині моделі затверджуються два фундаментальних механізми маніпулювання реляційними БД - реляційна алгебра і реляційне числення. Перший механізм базується в основному на класичній теорії множин (з деякими уточненнями), а другий - на класичному логічному апараті числення предикатів першого порядку. Основною функцією *маніпуляціоної* частини реляційної моделі є забезпечення міри реляційності будь-якої конкретної мови реляційних БД: мова називається реляційною, якщо він має не меншу виразність і потужність, чим реляційна алгебра або реляційне числення.

Нарешті, у цілісній частині реляційної моделі даних фіксуються дві базових вимоги цілісності, що повинні підтримуватися в будь-який реляційної БД.

Перша вимога називається вимогою цілісності сутностей. Об'єкту або сутності реального світу в реляційних БД відповідають кортежі відносин. ***Конкретна вимога полягає в тому, що будь-який кортеж будь-якого відношення відрізнимо від будь-якого іншого кортежу цього відношення,*** тобто іншими словами***, будь-яке відношення повинне мати первинний ключ***. Як ми бачили в попередньому розділі, ця вимога автоматично задовольняється, якщо в системі не порушуються базові властивості відносин.

Друга вимога називається *вимогою цілісності по посиланнях* і є трохи більш складною. Очевидно, що при дотриманні нормалізованості відносин складні сутності реального світу представляються в реляційної БД у вигляді кількох кортежів кількох відносин. Наприклад, представимо, що нам потрібно представити в реляційної базі даних сутність ВІДДІЛ з атрибутами НОМЕР_ВІДД (номер відділу), ВІДД_КІЛ (кількість співробітників) і СП_НОМЕР (набір співробітників відділу). Для кожного співробітника потрібно зберігати СП_НОМЕР (номер співробітника), СП_ІМ'Я (ім'я співробітника) і СП_ЗАРП (заробітна плата співробітника). Як ми незабаром побачимо, при правильному проектуванні відповідної БД у ній з'являться двоє відносин: ВІДДІЛИ (НОМЕР_ВІДД, ВІДД_КІЛ) (первинний ключ – НОМЕР_ВІДД) і СПІВРОБІТНИКИ (СП_НОМЕР, СП_ІМ'Я, СП_ЗАРП, СП_ВІДД_НОМ) (первинний ключ - СП_НОМЕР).

Як видно, атрибут СП_ВІДД_НОМ з'являється у відношенні СПІВРОБІТНИКИ не тому, що номер відділу є власною властивістю співробітника, а лише для того, щоб мати можливість відновити при необхідності повну сутність ВІДДІЛ. Значення атрибута СП_ВІДД_НОМ у будь-якому кортежі відносини СПІВРОБІТНИКИ повинні відповідати значенню атрибуту ВІДД_НОМ у деякому кортежі відносини ВІДДІЛИ. Атрибут такого роду називається, як ми вже знаємо, *зовнішнім ключем*, оскільки його значення однозначно характеризують сутності, представлені кортежами деякого іншого відношення (тобто задають значення їхнього

первинного ключа). *Говорять, що відношення, у якому визначений зовнішній ключ, посилається на відповідне відношення, у якому такий же атрибут є первинним ключем.*

Вимога цілісності по посиланнях, чи вимога зовнішнього ключа полягає в тому, що для кожного значення зовнішнього ключа, що з'являється у відношенні, що посилається, у відношенні, на якому веде посилання, повинний знайтися кортеж з таким же значенням первинного ключа, або значення зовнішнього ключа повинне бути невизначеним (тобто ні на що не вказувати). Для нашого приклада це означає, що якщо для співробітника зазначений номер відділу, те цей відділ повинний існувати.

Обмеження цілісності сутності і по посиланнях повинні підтримуватися СУБД. ***Для дотримання цілісності сутності досить гарантувати відсутність у будь-якому відношенні кортежів з тим самим значенням первинного ключа***. З цілісністю по посиланнях справи йдуть трохи більш складно.

Зрозуміло, що при відновленні відношення, що посилається, (уставці нових кортежів чи модифікації значення зовнішнього ключа в існуючих кортежах) досить стежити за тим, щоб не з'являлися некоректні значення зовнішнього ключа. Але як бути при видаленні кортежу з відношення, на яке веде посилання?

Тут існують три підходи, кожний з який підтримує цілісність по посиланнях. Перший підхід полягає в тім, що забороняється робити видалення кортежу, на який існують посилання (тобто спочатку потрібно або видалити кортежі, що посилаються, або відповідним чином змінити значення їхнього зовнішнього ключа).

Другий підхід -- при видаленні кортежу, на який є посилання, у всіх кортежах, що посилаються, значення зовнішнього ключа автоматично стає невизначеним. Третій підхід (каскадне видалення) полягає в тому, що при

видаленні кортежу з відношення, на яке веде посилання, з відношення, що посилається, автоматично видаляються всі кортежі, що посилаються.

У розвинених реляційних СУБД звичайно можна вибрати спосіб підтримки цілісності по посиланнях для кожної окремої ситуації визначення зовнішнього ключа. Звичайно, для прийняття такого рішення необхідно аналізувати вимоги конкретної прикладної області.