Частина I
Теорія баз даних

1.1. Вступ в бази даних ... 1
1.1.1. Що таке база даних ... 1
1.1.2. Структура бази даних ... 4
1.2. Реляційна модель бази даних ... 7
1.2.1. Домени і відносини ... 8
1.2.2. Цілісність даних ... 11
1.2.3. Реляційна алгебра ... 13
1.2.4.Реляційне обчислення ... 16
1.3. Проектування логічної структури бази даних ... 19
1.3.1. Концепція функціональної залежності ... 19
1.3.2. Нормалізація бази даних ... 20
1.3.3. Об'єктне моделювання ... 26
1.4. Функції захисту бази даних ... 29
1.4.1. Транзакції і паралелізм ... 29
1.4.2. Безпека і цілісність баз даних ... 32
1.5. Додаткові аспекти реляційної технології ... 36
1.5.1. Уявлення ... 36
1.5.2. Підвищення продуктивності за допомогою оптимізації ... 41
1.5.3. Домени, відношення і типи даних ... 45
1.5.4. Невизначені значення і тризначна логіка ... 48
1.5.5. Розподілені бази даних ... 49
1.6. Технологія фізичного зберігання і доступу до даних ... 56
1.6.1. Основні етапи доступу до бази даних ... 56
1.6.2. Управління сторінками ... 57
1.6.3. Процедура індексування і хешування ... 59
1.6.4. Стиснення даних ... 64

1.1. Вступ в бази даних

1.1.1. Що таке база даних

В даний час життя людини настільки сильно насичене різного роду інформацією, що для управління ним потрібне створення величезної кількості баз і банків даних різного призначення. Зараз практично будь-яка задача пов'язане з маніпуляцією інформації і даними. З цієї причини в останні роки появилася безліч різних комп'ютерних систем – систем управління базами даних – призначених для цих цілей.

Терміни база даних (БД) і система управління базами даних (СУБД) найчастіше використовуються як ті, що відносяться до комп'ютерів. Поняття БД можна застосувати до будь-якої зв'язаної між собою за певною ознакою інформації, що зберігається і організована особливим чином – як правило, у вигляді таблиць. По суті БД – це деяка подоба електронної картотеки, електронного сховища даних, яке зберігається в комп'ютері у вигляді одного або декількох файлів. При цьому виникає необхідність у виконанні ряду операцій із БД, в першу чергу це:

- додавання нової інформації в існуючі файли БД;
- додавання нових порожніх файлів в БД;
- зміна (модифікація) інформації в існуючих файлах БД;
- пошук інформації в БД;

- видалення інформації із існуючих файлів БД;
- видалення файлів із БД.

Комп'ютеризована інформаційна система є програмним комплексом, задачі якої полягають в підтримці надійного зберігання БД в комп'ютері, виконанні перетворень інформації і відповідних обчислень, наданні користувачам зручного і легко освоюваного інтерфейсу. Традиційно об'єми інформації, з якими доводиться мати справу таким системам, досить великі, а самі БД мають достатньо складну структуру. Прикладами інформаційних систем є системи замовлення залізничних або авіаційних квитків, банківські системи і багато інших.

Як ілюстрацію розглянемо невелику БД приведену на рис.1.1. в якій зберігається інформація про стипендію студентів.

KOD	NAME	GROUP	BALL	STIP
525122	Варварук В.	РФ-31	4,5	110,20
525123	Возняк Ю.	РФ-31	4,0	105,50
525124	Глотов О.	РФ-31	5,0	150,00
525125	Гуменяк М.	РФ-31	4,5	110,20
525126	Деркевич Ю.	РФ-31	3,0	0,00
525127	Довгий В.	РФ-31	4,5	110,20
525128	Кудря А.	РФ-31	4,0	105,50
525129	Лавер В.	РФ-31	4,5	110,20
525130	Мицак У.	РФ-31	3,0	0,00
525131	Піцик Б.	РФ-31	4,5	110,20

Запис

Поле

Рис. 1.1. Приклад таблиці БД "Стипендія"

Дані про студентів записуються в таблицю, що має строгу структуру даних. Інформація всередині таблиці складається з полів, що мають своє ім'я. У нашому прикладі – це поля, призначені для:

- KOD – номер студентського квитка;
- NAME – прізвище і ініціали студента;
- GROUP – найменування навчальної групи студента;
- BALL – середній бал успішності студента;
- STIP – розмір нарахованої стипендії.

Кожен рядок таблиці потрібно розглядати як запис, при цьому інформація заноситься у відповідні поля. В той же час всі записи складаються з однакових полів, а характер інформації для одного поля у всіх записах однаковий, але різні поля мають різний характер даних, що зберігаються.

Основним призначенням БД в першу чергу є швидкий пошук інформації, що міститься в ній. При великому розмірі БД ручний пошук, а також модифікація інформації, що міститься, займає значний час. Використання комп'ютера для ведення БД усуває перераховані вище проблеми – пошук і вибірка інформації, її модифікація здійснюється достатньо швидко і ефективно, а сама БД, що складається з тисяч записів, може легко вміщатися на дискеті. Існує велика кількості програм, які призначені для організації інформації, розміщення її в таблиці і маніпуляції з нею – такі програми і отримали назву СУБД. Основна особливість СУБД – це наявність засобів для вводу і зберігання не тільки самих даних, але і описів їх структури.

Якщо говорити детальніше, то до функцій СУБД відносять наступні:

управління даними безпосередньо в БД – функція, що забезпечує зберігання даних, що безпосередньо входять до БД. і службової інформації, що забезпечує роботу СУБД;

управління даними в пам'яті комп'ютера – функція, пов'язана в першу чергу з тим, що СУБД працюють із БД великого розміру. В цілях прискорення роботи СУБД використовується буферизація даних в оперативній пам'яті комп'ютера. При цьому користувач СУБД використовує тільки необхідну для його конкретної задачі частину БД, а при необхідності одержує нову "порцію" даних;

управління транзакціями – функція СУБД, яка проводить ряд операцій над БД як над єдиним цілим. Як правило, такі операції проводяться в пам'яті комп'ютера. В першу чергу транзакції необхідні для підтримки логічної цілісності БД в розрахованих на багато користувачів системах. Якщо транзакція (маніпуляція над даними) успішно виконується, то СУБД вносить відповідні зміни в БД. У зворотному випадку жодна із зроблених змін ніяк не впливає на стан БД;

управління змінами до БД і протоколювання – функція, пов'язана з надійністю зберігання даних, тобто можливістю СУБД відновлювати стан БД в аварійних ситуаціях, наприклад, при випадковому виключенні живлення або збої носія інформації. Очевидно, що для відновлення БД потрібно мати в своєму розпорядженні додаткову інформацію, по якій і здійснюється відновлення. З цією метою ведеться протокол змін БД в який перед маніпуляціями з даними робиться відповідний запис. Для відновлення БД після збою СУБД використовується протокол і архівна копія БД – повна копія БД до моменту початку заповнення протоколу;

підтримка мов БД – для роботи з БД використовуються спеціальні мови, які в цілому називаються мовами баз даних. У СУБД звичайно підтримується єдина мова, що містить всі необхідні засоби – від створення БД до забезпечення призначеного для користувача інтерфейсу при роботі з даними. Найбільш поширеною в даний час мовою СУБД є мова SQL (Structured Query Language).

В даний час в більшості БД використовуються реляційні моделі даних (про них поговоримо нижче), проте спершу стисло розглянемо інші основні моделі даних, а саме:

- моделі, засновані на інвертованих списках;
- ієрархічні моделі даних;
- мережні моделі даних;

1 БД, організована за допомогою інвертованих списків, побудована таким чином, що таблиці і шляхи доступу до них видно користувачам, при цьому рядки таблиць фізично впорядковані в деякій послідовності. У цих БД підтримуються оператори, що встановлюють адресу запису абсолютною (наприклад, перший запис, останній запис таблиці) або відносною (наприклад, наступний за поточним запис таблиці);

БД, заснована на ієрархічній моделі, складається з впорядкованого набору дерев. Кожне дерево складається із одного "кореневого" (предок) і впорядкованого набору із нуля або більше пов'язаних з ним піддерев (нащадки). Цілісність зв'язку між ними підтримується автоматично.

У таких БД підтримуються наступні оператори маніпулювання даними:

- знайти дерево БД за заданою ознакою;
- перейти від одного дерева до іншого;
- перейти до запису всередині дерева або у порядку обходу ієрархії (зверху вниз, зліва направо);
- вставити новий запис у вказану позицію;
- видалити поточний запис.

У БД із мережевою структурою даних, піддерева можуть мати будь-яке число кореневих. Фактично мережева БД складається з набору записів і безлічі зв'язків між цими записами. Типовий перелік операцій для мережевих БД може бути наступним:

- знайти запис за заданою ознакою;
- перейти від предка до нащадка по вказаному зв'язку;
- перейти від нащадка до предка по деякому зв'язку;
- створити новий запис або видалити той, що існує;
- модифікувати заданий запис;
- включити в зв'язок або виключити із зв'язку;
- переставити в інший зв'язок.

Реляційні БД в даний час використовуються найширше, тому на цій моделі даних ми зупинимося докладніше трохи нижче.

1.1.2. Структура бази даних

Поняття архітектури і структури є одним з найважливіших в теорії БД і служить основою для розуміння можливостей сучасних СУБД. Розрізняють три рівні архітектури БД:

- внутрішній рівень – найбільш наближений до фізичної системи безпосереднього зберігання даних Він описує, яким чином розміщуються дані на пристроях зберігання інформації. Для традиційного користувача БД внутрішній рівень, як правило, не доступний до перегляду і модифікації;
- зовнішній рівень – пов'язаний із способами представлення даних безпосередньо для користувачів. На зовнішньому рівні користувачу надається можливість маніпуляції даними в СУБД за допомогою спеціальної мови. Така орієнтованість на кінцевого користувача робить БД незалежного від фізичних параметрів середовища зберігання даних;
- концептуальний рівень – є перехідним від внутрішнього до зовнішнього рівням і, по суті, є узагальнене представлення даних для безлічі користувачів. На цьому рівні зміст БД представляється в цілому, на відміну від зовнішнього рівня – де конкретні дані представляються конкретному користувачу.

Очевидно, що структура БД не повинна модифікуватися до тих пір, поки зміни в реальному світі не зажадають відповідного коректування в ній для того, щоб ця модель продовжувала відповідати предметній області БД. Природно, що проектування БД починають з аналізу предметної області і виявлення вимог до неї кінцевих користувачів. Проектування, як правило, доручається адміністратору бази даних (АБД) – людині, добре знайомому з машинною обробкою даних. Саме АБД визначає, які саме дані зберігатимуться в процесі концептуального проектування БД.

Об'єднуючи вимоги майбутніх користувачів про вміст БД АБД спочатку створює узагальнений опис створюваної БД з використанням природної мови, математичних формул, графіків і інших засобів. Тільки після формування такого узагальненого уявлення АБД приступає безпосередньо до формалізації представлення даних в БД.

В процесі проектування АБД стикається з проблемою управління передачею даних. Запити до БД від кінцевих користувачів повинні відбуватися під управлінням і контролем спеціального програмного компоненту – диспетчера. Дійсно, в загальному випадку робоча станція користувача від самої БД може бути фізично віддалена на значну відстань. Програма-диспетчер передачі даних не є частиною СУБД, а є окремим додатком, що функціонує спільно і узгоджено.

Тут виникає необхідність ввести поняття архітектура клієнт/сервер, яка надає справжню свободу вибору і узгодження різних типів компонентів для клієнта, сервера і всіх проміжних ланок. Правда, це веде до ускладнення системи.

Що ж означає термін «архітектура клієнт/сервер?» Основою такої системи є сервер БД, що є додатком, що здійснює комплекс дій по управлінню даними – виконання запитів, зберігання і резервне копіювання даних, відстежування цілісності, перевірку прав користувачів, ведення журналу транзакцій. Як робоче місце (клієнта)

при цьому може бути використаний звичайний персональний комп'ютер, що дозволяє не відмовлятися від звичного робочого середовища.

Таким чином, інформаційна система побудована за принципом: клієнт/сервер, складається звичайно з трьох основних компонентів:

- сервер БД, який і є власне СУБД і управляє зберіганням даних, доступом, захистом, резервним копіюванням, відстежує цілісність даних і виконує запити клієнта;
- клієнти, що є різними додатками користувачів і що виконують запити до сервера, який перевіряє допустимість даних і одержує відповіді від нього;
- мережа і комунікаційне програмне забезпечення, що здійснює взаємодію між клієнтом і сервером за допомогою мережевих протоколів.

У функції сервера БД входить не тільки безпосереднє обслуговування даних. Обов'язково передбачаються системи блокування і управління доступом розрахованим на багато користувачів, елементи захисту даних від несанкціонованого доступу, структури оптимізації запитів до БД.

Крім того, в задачі серверної частини СУБД входить забезпечення посилальної цілісності даних і контроль завершення транзакцій. Посилальна цілісність даних – це система і набір спеціальних правил, що забезпечують єдність зв'язаних даних в БД. Контроль завершення транзакцій – задача СУБД по контролю і попередженню пошкодження даних в нештатних ситуаціях, наприклад, при апаратному збої.

Ці функції реалізуються за допомогою зберігаючих процедур, тригерів і правил. Зберігаючі процедури – це набір особливих дій і маніпуляцій з даними, які зберігаються на сервері, причому програми-клієнти здатні їх виконувати. Тригери – це вид зберігаючих процедур. Вони пов'язані з подіями, і запускаються автоматично, як тільки на сервері БД з даними відбувається така подія. Правило – це такий тип тригера, який перевіряє дані до внесення їх в БД.

У задачі комунікаційного програмного забезпечення входить в першу чергу забезпечення можливості програмі-клієнту швидко і легко підключитися до ресурсів сервера. Існують різноманітні варіанти цього програмного забезпечення, але всі вони повинні звільняти прикладні програми від складної взаємодії з операційною системою, мережними протоколами і серверами ресурсів.

У системах, побудованих за принципом клієнт/сервер, переважають два типи архітектури процесорів серверів: RISC і Intel. RISC є традиційним варіантом вибору для Unix-серверів, Взагалі кажучи, RISC-сервери по продуктивності не набагато перевершують сервери на базі Intel-процесорів, але вони коштують від 15 до 50 тисяч доларів.

Сервери на базі процесорів Intel зараз швидко нарощують обчислювальну здатність. З появою процесора Pentium, що використовує RISC-подібну технологію, платформа Intel набуває все більш широкої популярності. При цьому багато сучасних операційних системи працюють на Intel-машинах. З врахуванням цього Unix вже поступається місцем Windows NT, в якому широкі можливості Unix поєднуються з простотою використання Windows, проте на противагу Windows все більше стає операційна Linux.

Комунікаційне програмне забезпечення може бути, наприклад, у вигляді спеціального програмного забезпечення, яке здійснює віддалений виклик сервісних функцій СУБД. Воно управляє передачею запиту і отриманням результату, але не пропонує клієнту власного інтерфейсу і обробку даних на сервері – адже це задача власне клієнтської частини.

Спрощено архітектуру клієнт/сервер можна представити у вигляді схеми, приведеної на рис. 1.2.

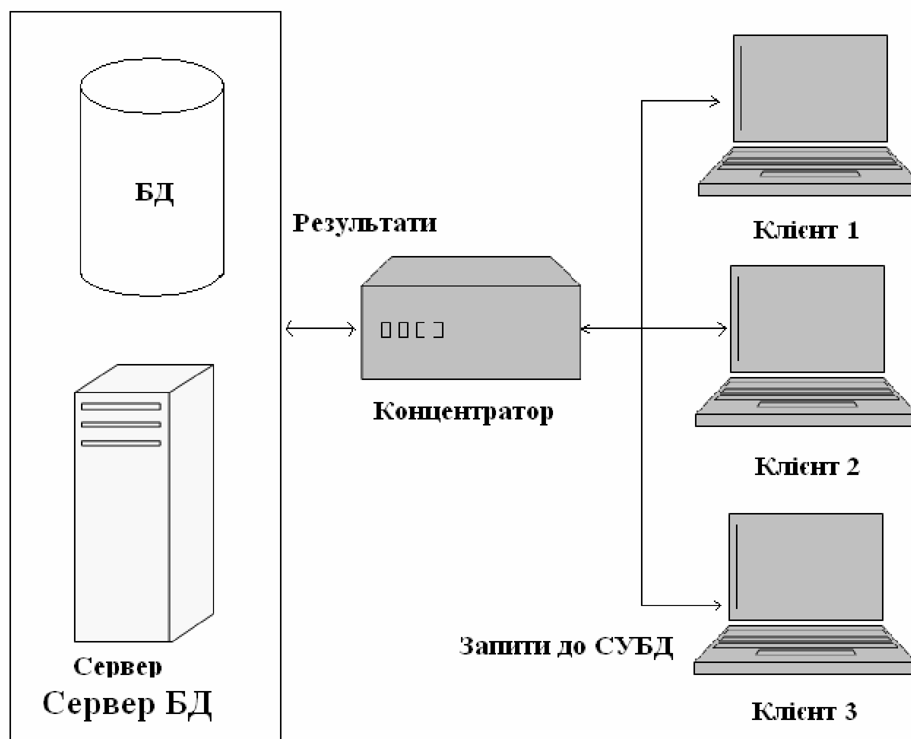


Рис. 1.2. Спрощена схема СУБД в архітектурі клієнт/сервер.

Найбільш простий вигляд БД побудованої в архітектурі клієнт/сервер – це розділення обчислювального навантаження між двома окремими системами: клієнтом і сервером. У принципі і клієнтська, і серверна частини можуть фізично знаходитися на одному і тому ж комп'ютері, але більшість систем цієї архітектури запускають клієнтське забезпечення на одному комп'ютері, а додаток-сервер – на другому. Зрозуміло, для обміну інформацією використовується мережа і мережне програмне забезпечення. При цьому один додаток може працювати незалежно від іншого, виконувати різні завдання і розділяти обчислювальне навантаження.

Основні переваги клієнт/серверних в порівнянні з аналогічними інформаційними системами полягають в наступному. По-перше, це зниження кількості передаваної по комп'ютерній мережі інформації. Це відбувається тому, що. скажімо, при вибірці з великої БД декількох записів сервер обробляє запит і як результат передає клієнту інформацію, що тільки цікавить, а не всю БД. По-друге, перевагою архітектури клієнт/сервер є можливість зберігання правил доступу і обробки на сервері, що дозволяє уникнути дублювання коду в різних додатках, що використовують спільну БД. Крім того, будь-яка маніпуляція із даними може бути проведена тільки в рамках цих правил. Частина коду, пов'язаного з обробкою даних, як і правило, реалізується у вигляді зберігаючих процедур сервера, що дозволяє ще більш прискорити роботу клієнтського додатку за рахунок зменшення його розмірів, а це в свою чергу означає, що вимоги до робочих станцій можуть бути не такими високими. Це зрештою знижує загальну вартість інформаційної системи навіть при використанні дорогої СУБД і могутнього сервера БД.

По третє, сучасні СУБД, реалізовані на платформі клієнт/сервер, володіють могутніми можливостями управління доступу до елементів БД, резервного копіювання, архівації і паралельної обробки даних, що значно покращує роботу.

Використовуючи безліч комп'ютерів, системи на платформі клієнт/сервер розподіляють прикладну задачу по різних робочих станціях і серверах. Кожен елемент при цьому бере на себе свою частину обчислювального навантаження, використовуючи інформацію спільно із іншими комп'ютерами мережі, при цьому потужність системи

підвищується без нарощування продуктивності одного окремого комп'ютера, а виходить як результат підсумовування можливостей багатьох. Крім всього, архітектура клієнт/сервер є технологією, що надає велику самостійність користувачам і можливість прояву творчості в створенні клієнтських додатків.

Традиційно в клієнт/серверних системах використовуються дві ланки – клієнт і сервер. При такій побудові велике програмне навантаження несе на собі клієнтська частина, а сервер в основному використовують для обслуговування даних. Дволанкові системи виходять достатньо простими в побудові і обслуговуванні, але таку модель неможливо наростити для спільної роботи тисяч користувачів.

Для забезпечення роботи при великих обчислювальних навантаженнях використовується трьох- або багатоланкова архітектура, що спричиняє за собою розподіл системи по додаткових ланках, що забезпечують додаткову обчислювальну потужність. Створена могутня обчислювальна система, на жаль, володіє підвищеною складністю, а при її створенні розробники витрачають більше часу.

1.2. Реляційна модель бази даних

Реляційні моделі даних, як вже було сказано, в даний час набули найбільшу популярність і практично всі сучасні СУБД орієнтовані саме на таке представлення даних.

Реляційну модель можна представити як особливий метод розгляду даних, що містить і власне дані (у вигляді таблиць), і способи роботи і маніпуляції із ними (у вигляді зв'язків). Реляційна модель передбачає три концептуальні елементи: структура, цілісність і обробка даних, як втім, і більшість нереляційних моделей. У цих елементах є свої спеціальні поняття, які для подальшого викладу необхідно стисло пояснити.

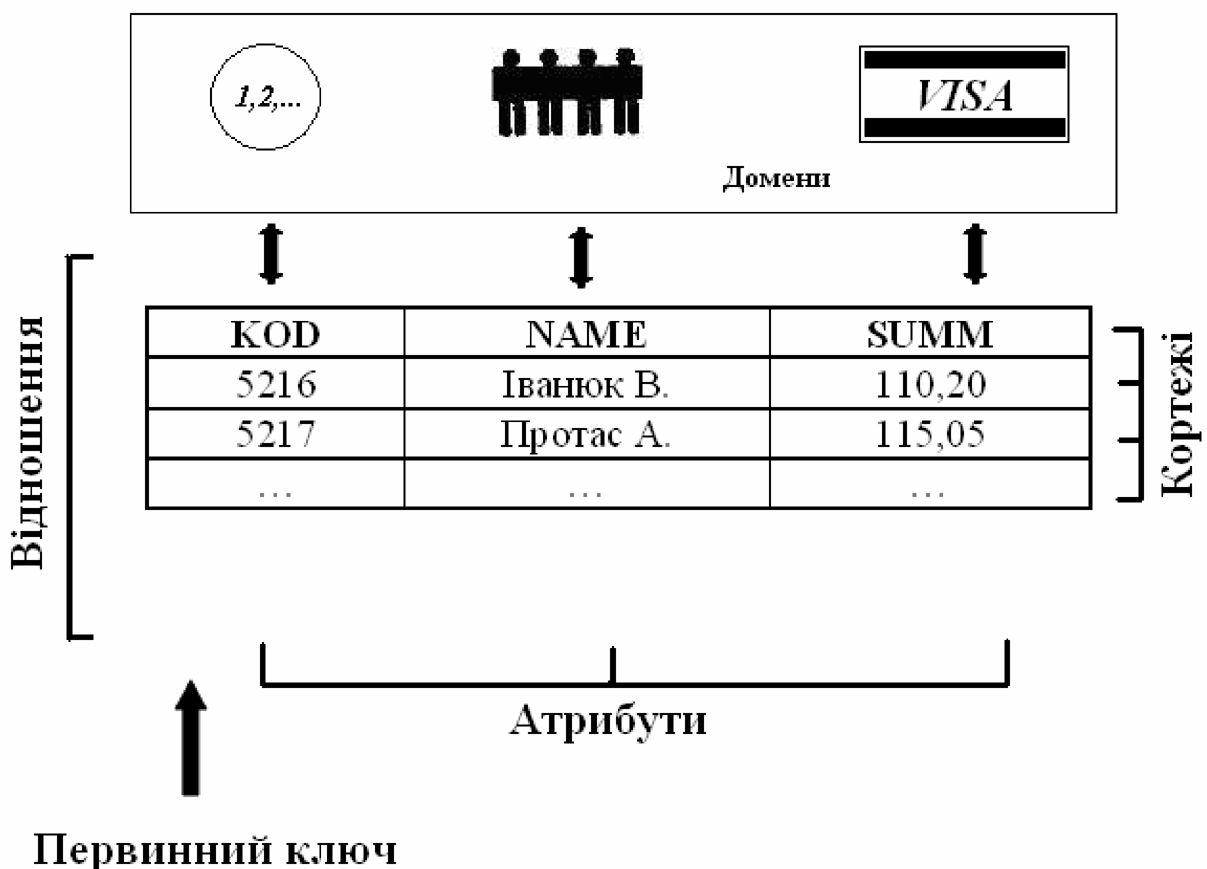


Рис. 2.1. Пояснення понять реляційних БД.

Таблиця розглядається як безпосереднє «сховище» даних, Традиційно в реляційних системах таблицю називають відношенням. Рядок таблиці називають кортежем, а стовпець – атрибутом. При цьому атрибути мають унікальні (в межах відношення) імена. Кількість кортежів в таблиці називають кардинальним числом, а кількість атрибутів – степенем. Для відношення передбачають унікальний ідентифікатор, тобто один або декілька атрибутів, значення яких в один і той же час не бувають однаковими – ідентифікатор, називають первинним ключем. Домен – це безліч допустимих однорідних значень для того або іншого атрибуту. Таким чином, домен можна розглянути як іменовану множину даних, причому складові частини цієї множини є логічно неподільними одиницями (як домен можуть виступати, наприклад, перелік прізвищ співробітників установи, проте не всі прізвища можуть бути присутніми в таблиці).

Відношення містить дві частини – заголовок і власне змістовну частину. Заголовок містить кінцеву множину атрибутів, а змістовна частина (тіло відношення) – множину пар імені атрибуту і його значення. Наприклад, на рис. 2.1 KOD, NAME і SUMM, що містяться в заголовку, є атрибутами, а скажемо, пари SUMM – 25.50 або KOD – 5216 є елементами тіла відношення.

У реляційних БД, на відміну від інших моделей, користувач вказує, які дані для нього необхідні, а не те, як це робити. З цієї причини процес переміщення і навігації по БД в реляційних системах є автоматичним, а цю задачу в таких СУБД виконує так званий оптимізатор. Його робота полягає, наприклад, в тому, щоб найбільш ефективним способом провести вибірку даних із БД за запитом. Таким чином, оптимізатор, принаймні, повинен зуміти визначити, з яких таблиць вибираються дані, наскільки багато інформації в цих таблицях, який фізичний порядок записів в таблицях і як вони згруповані и.тд.

Крім того, реляційна СУБД виконує і функції каталога. У каталозі зберігаються описи всіх об'єктів, з яких складається БД – таблиць, індексів, тригерів і т.п. Очевидно, що це життєво необхідно для правильної роботи всієї системи – так, наприклад, оптимізатор використовує в своїй роботі інформацію, що зберігається в каталозі. Цікавий той факт, що каталог сам є набором таблиць, тому СУБД може маніпулювати ними традиційними засобами, не вдаючись до якихось особливих прийомів і методів.

1.2.1. Домени і відносини

Основні означення: домени; види відносин; предикати. Відношення мають ряд основних властивостей, а саме:

- у найзагальнішому випадку у відношенні не буває двох однакових кортежів. Це випливає з самого формулювання відношення, проте для деяких СУБД у ряді випадків допускається відступ від цієї властивості. Дійсно, оскільки у відношенні має місце первинний ключ, то однакові кортежі виключені;
- кортежі не впорядковані зверху вниз – у відношенні просто відсутнє поняття позиційного номера. У відношенні без втрати інформації можна з успіхом розташувати кортежі у будь-якому порядку;
- атрибути не впорядковані зліва направо – атрибути в заголовку відношення можна розташовувати у будь-якому порядку, при цьому цілісність даних не порушується. Тому поняття позиційного номера відносно атрибуту те ж не існує;
- значення атрибутів складаються з, логічно неподільних одиниць – це властивість є наслідок того, що значення беруться із доменів. Інакше, можна сказати, що відносини не містять груп повторення, тобто є нормалізованими (про що ми говоритимемо нижче).

У реляційних системах підтримуються декілька видів відносин.

Іменоване – є змінне відношення, яке визначається в СУБД шляхом операторів створення і, як правило, необхідне для зручнішого представлення інформації для користувача.

Вазове відношення – що є безпосередньою важливою частиною БД, тому при проектуванні їм дають власне найменування.

Похідне відношення – те, яке було означене через інші (як правило, базові) відносини шляхом використання засобів СУБД.

Представлення – фактично є іменованим похідним відношенням, при цьому представлення виражається виключно через оператори СУБД, що застосовані до іменованих відносин, тому їх фізично в БД не існує.

Результат запиту – це неіменоване похідне відношення, що містить дані – результат конкретного запиту. Результат запиту в БД не зберігається, а існує тільки до тих пір, поки він необхідний користувачу.

Відношення, що зберігається – те, яке фізично підтримується в пам'яті комп'ютера. До тих, що зберігаються, в більшості випадків, відносяться базові відносини.

Виходячи з вищесказаного, можна тепер визначити реляційну БД як набір відносин, зв'язаних між собою.

Зв'язок в даному випадку – це асоціювання двох або більше відносин БД, що не має зв'язків між відносинами, має дуже просту структуру і повною мірою реляційною називатися не може. Проте одна з основних вимог до організації реляційної БД – це забезпечення можливості пошуку одних кортежів по значеннях інших, для чого необхідно встановити між ними зв'язок. А оскільки в реальних інформаційних системах часто містяться тисячі кортежів, то теоретично між ними може бути встановлено більше мільйона зв'язків. Наявність такої безлічі зв'язків і визначає складність реляційних моделей БД. Існують наступні основні види зв'язків:

- один до одного;
- один до багатьох;
- багато до одного;
- багато до багатьох.

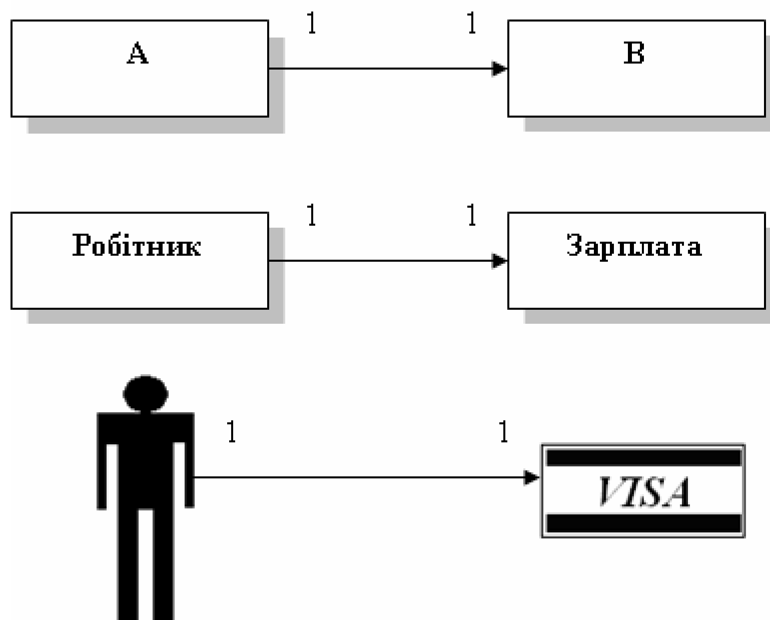


Рис. 2.2. Зв'язок "один до одного".

Зв'язок "один до одного" припускає, що в кожен момент часу кожному елементу (кортежу) А відповідає 0 або 1 елементів (кортежів) В (див. рис. 2.2). Наприклад, працівник одержує зарплату, і лише одну.

Зв'язок "один до багатьох" полягає в тому, що в кожен момент часу кожному елементу (кортежу) А відповідає декілька елементів (кортежів) В (див. рис. 2.3). Як приклад можна сказати, що в будинку проживає багато мешканців.

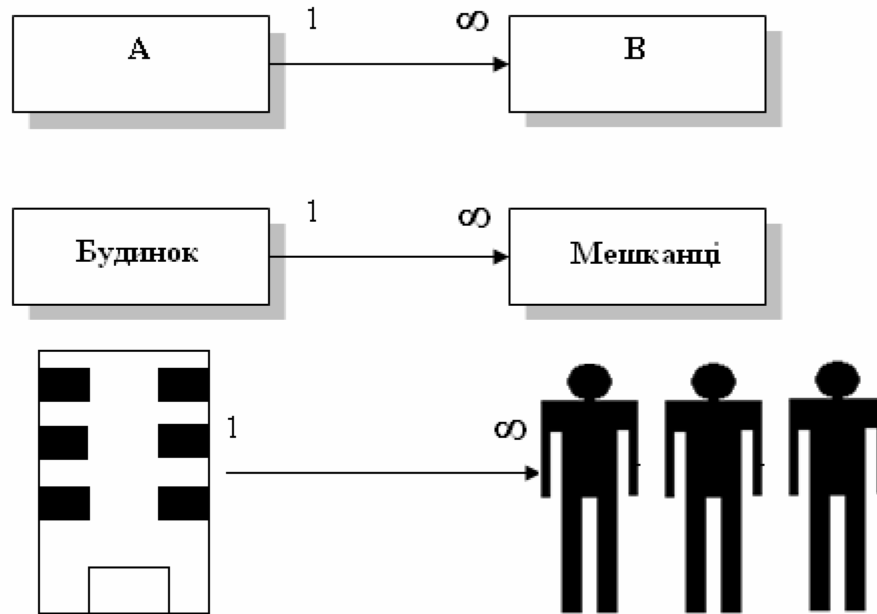


Рис.2.3. Зв’язок «один до багатьох».

Зв'язок "багато до одного" припускає, що в кожен момент часу безлічі елементів А відповідає 1 елемент В. Наприклад, декілька студентів є студентською навчальною групою (див. рис. 2.4).

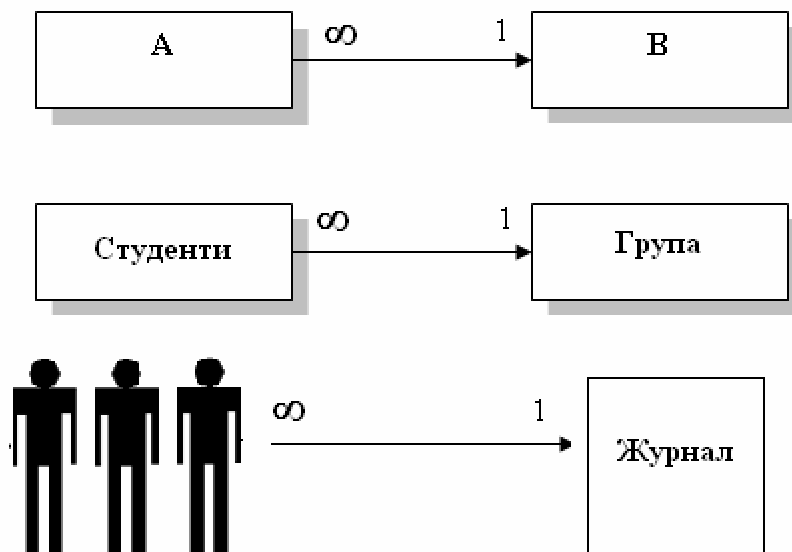


Рис. 2.4. Зв'язок "багато до одного".

Нарешті, зв'язок "багато до багатьох" полягає в тому, що в кожен момент часу безлічі елементів А відповідає безліч елементів В (див. рис. 2.5). На жаль, цей тип зв'язку в реляційних БД безпосередньо не підтримується. Прикладом такого зв'язку

може служити той факт, що у студентів навчальні заняття по дисциплінах ведуть безліч викладачів.

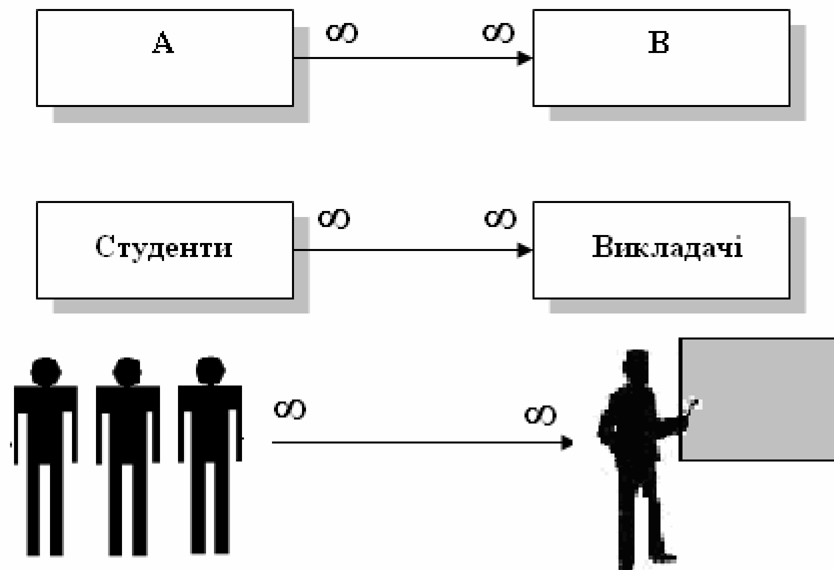


Рис. 2.5. Зв'язок "багато до багатьох".

Крім вищеперелічених, ще можуть існувати множинні зв'язки між одними і тими ж елементами, тренарні зв'язки (див. рис. 2.6), які, втім, можуть бути виражені через вже розглянуті.

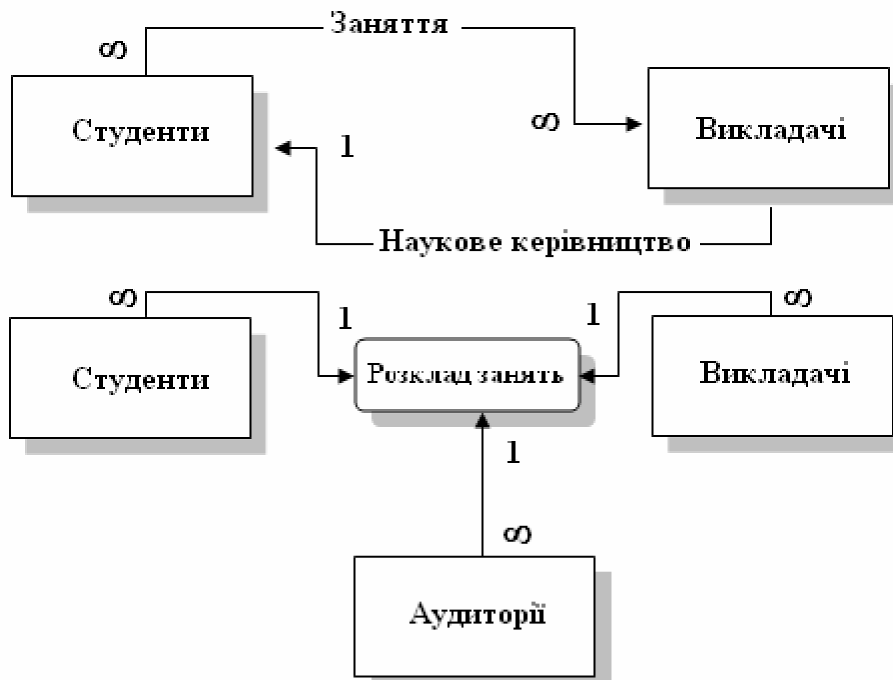


Рис. 2.6. Множинні і тренарні зв'язки.

1.2.2. Цілісність даних

У реляційних моделях питанню цілісності даних відводиться особливе місце. Нагадаємо, що ключ або потенційний ключ – це мінімальний набір атрибутів, по значеннях яких можна однозначно знайти необхідний кортеж. Мінімальність означає, що виключення з набору будь-якого атрибуту не дозволяє ідентифікувати кортеж по атрибутах, що залишилися.

Кожне відношення володіє хоч би одним можливим ключем. Один з них береться за первинний ключ. При виборі первинного ключа слід віддавати перевагу нескладеним ключам або ключам, складеним із мінімального числа атрибутів. Небажано також використовувати ключі із довгими текстовими значеннями (переважно використовувати цілочисельні атрибути). Так, для ідентифікації працівника можна використовувати або унікальний табельний номер або номер паспорта, або набір із прізвища, імені, по батькові і номера відділу.

Не допускається, щоб первинний ключ відношення, тобто будь-який атрибут, що бере участь в первинному ключі, приймав невизначене значення.

Тепер поговоримо про зовнішні ключі. Варто мати на увазі, що якщо відношення С зв'язує відношення А і В, то воно повинно включати зовнішні ключі, що відповідають первинним ключам відношень А і В, що представлено на рис. 2.7. Таким чином, при розгляді проблеми вибору способу зв'язку відношень в БД виникає питання про те, які ж повинні бути зовнішні ключі. При цьому для кожного зовнішнього ключа необхідно вирішити проблему, пов'язану з можливістю (або неможливістю) появи в зовнішніх ключах невизначених значень (NULL-значень – значень атрибуту для відсутньої інформації). Іншими словами, чи може існувати деякий кортеж у відношенні, для якого невідомий кортеж в пов'язаному з ним відношенні.

З іншого боку, необхідно наперед обдумати питання про те. що відбудеться при спробі видалення кортежів із відношення, на яке посилається зовнішній ключ. При цьому існують наступні вірогідні можливості:

- операція *каскадує* – тобто видалення кортежів у відношенні приводить до видалення відповідних кортежів в зв'язаному відношенні. Наприклад, видалення інформації про прізвище, ім'я і т.п. співробітника в одному відношенні приводить до видалення інформації про його заробітну платню в другом;

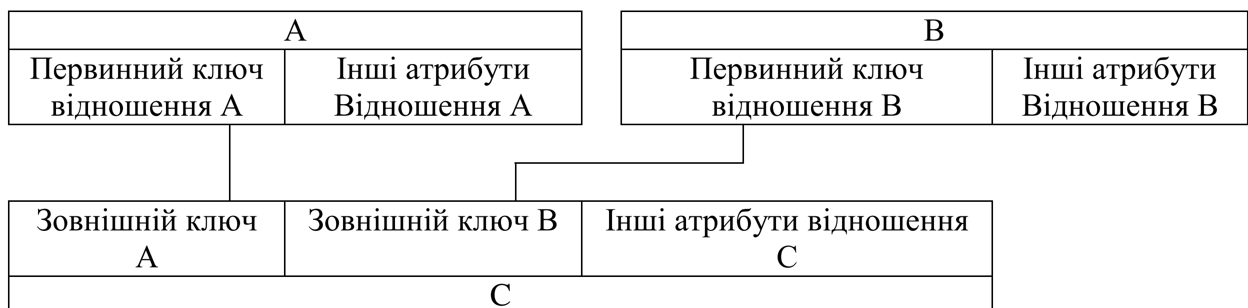


Рис. 2.7. Зовнішні ключі.

- операція *обмежується* – тобто знищуються лише ті кортежі, для яких зв'язаної інформації в другом відношенні немає. Якщо така інформація є, то видалення здійснити не можна. Наприклад, видалення інформації про прізвище, ім'я і т.п. співробітника можливе лише в тому випадку, якщо інформація про його заробітну платню в зв'язаному відношенні відсутня.

Нарешті, потрібно передбачити технологію того, що відбуватиметься при спробі оновлення первинного ключа відношення, на яке посилається деякий зовнішній ключ. Тут є ті ж можливості, як і при видаленні:

- операція *каскадує* – тобто при оновленні первинного ключа відбувається оновлення зовнішнього ключа в зв'язаному відношенні. Наприклад, оновлення первинного ключа у відношенні, де зберігається інформація про співробітника приводить до оновлення зовнішнього ключа у відношенні з інформацією про його заробітну платню;
- операція *обмежується* – тобто оновлюються лише ті первинні ключі, для яких зв'язаної інформації в другом відношенні немає. Якщо така інформація є, то

оновлення зробити не можна. Наприклад, оновлення первинного ключа у відношенні, де зберігається інформація про співробітника, можливе в тому випадку, якщо інформація про його заробітну платню в зв'язаному відношенні відсутня.

Таким чином, для кожного зовнішнього ключа в БД повинні передбачатися не тільки атрибут або комбінація атрибутів, що становлять цей зовнішній ключ, і відношення, що ідентифікується цим ключем, але також і варіанти "поведінки" БД в розглянутих вище випадках.

1.2.3. Реляційна алгебра

Формальною основою реляційної моделі БД є реляційна алгебра, що базується на теорії множин і розглядає спеціальні оператори над відношеннями, і реляційне обчислення, що базується на математичній логіці.

Основних операторів в реляційній алгебрі вісім, і схематично їх можна представити так, як це показано на рис. 2.8. Треба відзначити, що реляційна алгебра володіє великою потужністю – складні запити до БД можуть бути виражені за допомогою одного виразу. Саме з цієї причини ці механізми включені в реляційну модель даних. Конкретна мова маніпулювання реляційними БД називається реляційно-повною, якщо будь-який запит, що виражається за допомогою одного виразу реляційної алгебри або однієї формули реляційного обчислення, може бути виражений за допомогою одного оператора цієї мови.

Реляційна алгебра володіє важливою властивістю – вона закрита щодо поняття відношення. Це означає, що вирази реляційної алгебри виконуються над відношеннями реляційних БД і результати їх обчислення також є відношення. Тому будь-який вираз може бути представлений як відношення, що дозволяє використовувати його в інших виразах реляційної алгебри.

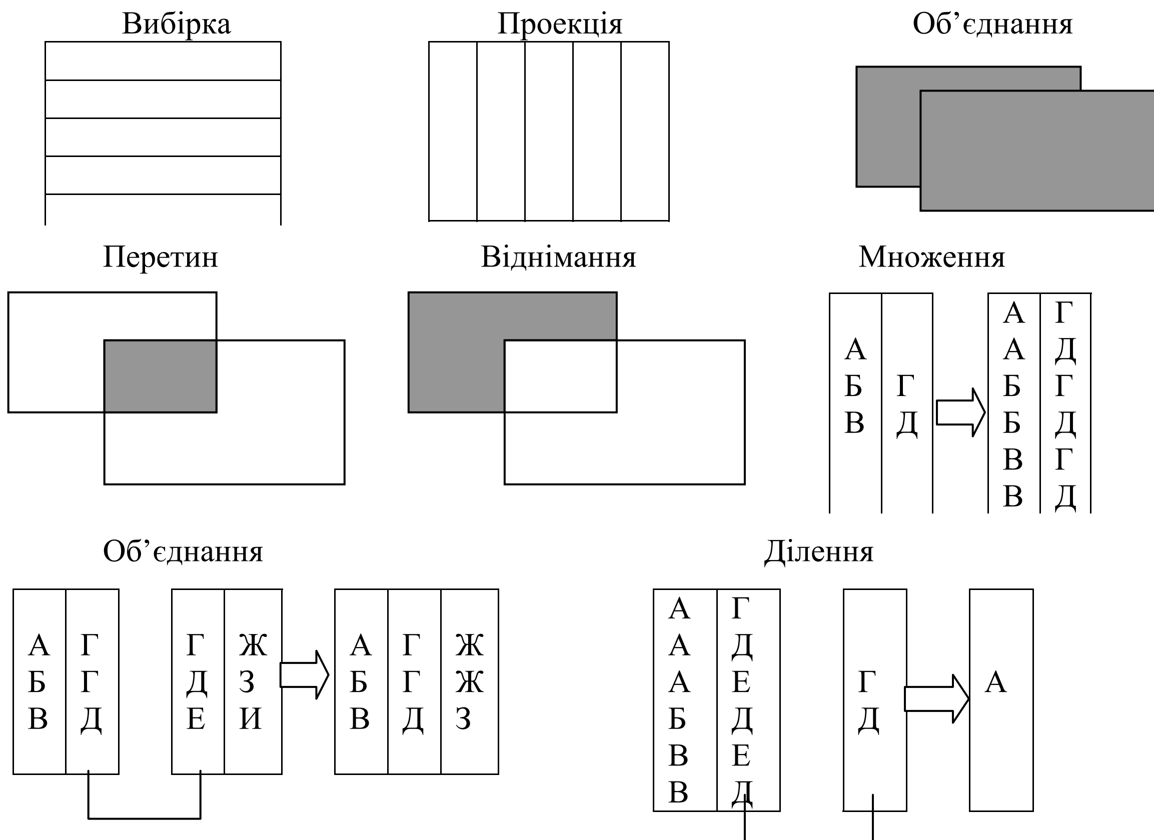


Рис. 2.8. Основні оператори реляційної алгебри.

Основна ідея реляційної алгебри полягає в тому, що засоби маніпулювання відношеннями, що розглядаються як множини, базуються на традиційних множинних операціях, доповнених деякими специфічними операціями для БД.

Існує багато підходів до означення реляційної алгебри, які розрізняються набором операцій і способами їх інтерпретації, але в принципі всі вони більш менш рівносильні. Опишемо варіант алгебри, який був запропонований Коддом. У цьому варіанті, як вже було показано вище, набір операцій алгебри, складається з восьми основних:

- вибірка відношення;
- проекція відношення;
- об'єднання відношень;
- перетин відношень;
- віднімання відношень;
- добуток відношень;
- об'єднання відношень;
- ділення відношень.

Ці операції можна пояснити таким чином:

- результатом вибірки відношення по деякій умові є відношення, яке включає тільки ті кортежі первинного відношення, які задовольняють цій умові;
- при здійсненні проекції відношення на заданий набір його атрибутів буде отримане відношення, кортежі якого взяті із відповідних кортежів первинного відношення;
- при виконанні операції об'єднання двох відношень буде отримане відношення, що включає всі кортежі, що входять хоч би в одне з тих, що беруть участь в операції відношень;
- як результат операції перетину двох відношень виходить відношення, що включає всі кортежі, які входять в обидва первинні відношення;
- відношення, що є різницею двох відношень, включає всі кортежі, що входять в перше відношення і одночасно такі, що жоден з них не входить у відношення, що є другим;
- при виконанні прямого добутку двох відношень виходить відношення, кортежі якого є поєднанням кортежів першого і другого відношення;
- при з'єднанні двох відношень по деякій умові утворюється результуюче відношення, кортежі якого є поєднанням кортежів першого і другого відношень, що задовольняє цій умові;
- операція реляційного ділення має два операнди – бінарне (тобто що складається з двох атрибутів) і унарне (містить один атрибут) відношення. Результат операції – відношення, що складається з кортежів, що включають значення першого атрибуту кортежів першого відношення, при чому таких, що множина значень другого атрибуту співпадає із множиною значень другого відношення.

Крім вищеперелічених, є ряд особливих операцій, характерних для роботи із БД:

- як результат операції перейменування виходить відношення, набір кортежів якого співпадає із тілом первинного відношення, але імена атрибутів змінені;
- операція привласнення дозволяє зберегти результат обчислення реляційного виразу в існуючому відношенні БД.

Звідси випливає, що якщо результатом реляційної операції є деяке відношення, то є можливість утворювати реляційні вирази, в яких замість первинного відношення (відношення-операнда) використовуватиметься вкладений реляційний вираз. Це відбувається завдяки тому факту, що операції реляційної алгебри дійсно закриті щодо поняття відношення.

Почнемо з операції об'єднання відношень, проте це в рівній мірі відноситься до операцій перетину і віднімання. У реляційній алгебрі результатом операції об'єднання

повинно бути відношення. Якщо допустити в реляційній алгебрі можливість об'єднання довільних двох відношень із різними наборами атрибутів, то результатом такої операції буде множина, проте множина різнотипних кортежів, тобто, взагалі кажучи, не відношення. Якщо виходити з вимоги замкнутості реляційної алгебри щодо поняття відношення, то така операція об'єднання є безглуздою. Це приводить до появи поняття сумісності відношень по об'єднанню; два відношення сумісні по об'єднанню в тому і лише у тому випадку, коли володіють однаковими заголовками. Точніше це означає, що в заголовках обох відношень міститься один і той же набір імен атрибутів, і однойменні атрибути визначені на одному і тому ж домені.

За умови того, що два відношення сумісні по об'єднанню, то при звичайному виконанні над ними операцій об'єднання, перетину і віднімання результатом операції є відношення із коректно певним заголовком, співпадаючим із заголовком кожного з відношень-операндів. Якщо ж два відношення не повністю сумісні по об'єднанню, тобто сумісні у всьому, окрім імен атрибутів, то до виконання операції типу об'єднання ці відношення можна зробити повністю сумісними по об'єднанню шляхом застосування операції перейменування.

Операція прямого добутку двох відношень викликає нові проблеми. У теорії множин прямий добуток може бути одержаний для будь-яких двох множин. Елементами результуючої множини будуть пари, складені із елементів першої і другої множин. Оскільки відношення є множинами, то і для будь-яких двох відношень можливе отримання прямого добутку, проте, результат не буде відношенням. Елементами результату будуть не кортежі, а пари кортежів. Тому в реляційній алгебрі використовується спеціальна форма операції взяття прямого добутку – розширений прямий добуток відношень. При взятті розширеного прямого добутку двох відношень елементом результуючого відношення є кортеж, що формується при злитті одного кортежу першого відношення і одного кортежу другого відношення. Тут же виникає друга проблема, пов'язана з отриманням коректно сформованого заголовка результуючого відношення. Це приводить до необхідності введення поняття сумісності відношень по взяттю розширеного прямого добутку. Два відношення сумісні по взяттю прямого добутку в тому і лише в тому випадку, якщо множина імен атрибутів цих відношень не перетинається. Будь-які два відношення можуть бути перетворені до сумісного вигляду по взяттю прямого добутку шляхом застосування операції перейменування до одного з цих відношень.

Операція вибірки вимагає наявність двох відношень: первинного відношення-операнда і простої умови обмеження. В результаті виконання операції вибірки проводиться відношення, заголовок якого співпадає із заголовком відношення-операнда, а в тіло входять ті кортежі відношення-операнда, які задовольняють значенням умови обмеження.

Введемо ряд операторів. Нехай UNION позначає операцію об'єднання. INTERSECT – операцію перетину, а MINUS – операцию віднімання. Для позначення операції вибірки використовуватимемо конструкцію A WHERE B. де A – відношення-операнд, а B – проста умова порівняння. Нехай C1 і C2 – дві прості умови вибірки. Тоді за означенням:

```
A WHERE Cl AND C2 позначає те ж саме,
     що і (A WHERE C1) INTERSECT (A WHERE C2)
A WHERE Cl OR C2 означає те ж саме, що
          і (A WHERE C1) UNION (A WHERE C2)
A WHERE NOT Cl означає те ж саме, що і
                A MINUS (A WHERE C1).
```

З використанням цих означень можна реалізувати операції вибірки, в яких умовою вибірки є довільний булевий вираз, складений із простих умов з

використанням логічних зв'язок AND (логічне І), OR (логічне АБО), NOT (логічне НЕ) і дужок.

Операція взяття проекції також вимагає наявність двох операндів – проектованого відношення A і списку імен атрибутів; що входять в заголовок відношення A.

Результатом проекції відношення A за списком атрибутів a1, a2,..., an буде відношення, заголовком якого є безліч атрибутів a1, a2,...., an. Тіло результату складатиметься з кортежів, для яких у відношенні A є кортеж, атрибут a1 якого має значення v1, атрибут a2 має значення v2, ..., атрибут an має значення vn. По суті, при виконанні операції проекції визначається "вертикальна" вирізка відношення-операнда із видаленням виникаючих кортежів-дублікатів.

Операція об'єднання, яка називається іноді об'єднанням по умові, вимагає наявність двох операндів-відношень, що сполучаються і третього операнда – простої умови. Нехай об'єднуються відношення A і B. Як і у разі операції вибірки, умова об'єднання C має вигляд або (a comp-op b), або (a comp-op const), де a і b – імена атрибутів відношень A і B, const – літерально задана константа, а comp-op – допустима в даному контексті операція порівняння. Тоді за означенням результатом операції об'єднання є відношення, що отримується шляхом виконання операції обмеження по умові C прямого добутку відношень A і B.

Є важливий окремий випадок об'єднання – природне об'єднання. Операція об'єднання називається операцією природного об'єднання, якщо умова об'єднання має вигляд (a=b), де a і b – атрибути різних операндів об'єднання. Цей випадок важливий тому, що він особливо часто зустрічається на практиці і для нього існують ефективні алгоритми реалізації в СУБД. Операція природного об'єднання застосовується до пари відношень A і B, що володіють спільним атрибутом R, тобто атрибутом із одним і тим же ім'ям і визначеним на одному і тому ж домені. Нехай ab позначає об'єднання заголовків відношень A і B. Тоді природне об'єднання A і B – це спроектований на ab результат об'єднання A і B. Операція природного об'єднання не включається прямо до складу набору операцій реляційної алгебри, але вона має дуже важливе практичне значення.

Операція ділення відношень потребує докладнішого пояснення, оскільки найбільш важка для розуміння. Нехай задані два відношення – A із заголовком {a1, a2,..., an, b1, b2,..., bm} і B із заголовком {b1, b2, ..., bm}. Вважатимемо, що атрибут b1 відношення A і атрибут b1, відношення B володіють одним і тим же ім'ям і визначені на одному і тому ж домені. Назвемо множину атрибутів {aj} складовим атрибутом a, множину атрибутів {bj} – складовим атрибутом b. Після цього говоритимемо 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 таких кортежів v, що у відношенні A є кортежі <v, w> які в множині значень {w} включають множину значень атрибуту b у відношенні B.

Оскільки ділення найбільш важка операція, пояснимо її прикладом. Нехай в БД студентів є два відношення: СТУДЕНТИ(ПІП, НОМЕР) і ІМЕНА(ПІП), причому унарне відношення ІМЕНА містить всі прізвища, якими володіють студенти інституту. Тоді після виконання операції реляційного ділення відношення СТУДЕНТИ на відношення ІМЕНА буде одержане унарне відношення, що містить номери студентських квитків, що належать студентам зі всіма можливими в цьому інституті прізвищами.

1.2.4.Реляційне обчислення

Припустимо, що є БД. що володіє наступною структурою: відношення СТУДЕНТИ (СТУД_НОМ, СТУД_ІМ'Я, СТУД_СТИП, ГР_НОМ) і відношення ГРУПИ (ГР_НОМ, ГР_КІЛ, ГР_СТАР). Припустимо, що необхідно дізнатися імена і

номери студентських квитків у студентів, що є старостами груп, із кількістю студентів більше 25.

Якщо б для формулювання такого запиту використовувалася реляційна алгебра, то ми одержали б вираз алгебри, який читався б. наприклад, таким чином:

- виконати об'єднання відношень СТУДЕНТИ і ГРУПИ по умові СТУД_НОМ=ГР_СТАР;
- обмежити одержане відношення по умові ГР_КІЛ>25;
- спроектувати результат попередньої операції на атрибут СТУД_ІМ'Я, СТУД_НОМ.

Тут покроково сформульована послідовність виконання запиту до БД, кожний з яких відповідає одній реляційній операції. Якщо ж сформулювати той же запит з використанням реляційного обчислення, то ми одержали б формулу, яку можна було б прочитати, наприклад, таким чином. Видати СТУД_ІМ'Я і СТУД_НОМ для таких студентів, щоб існувала група із таким же значенням ГР_СТАР і значенням ГР_КІЛ, більшим 25.

У другому формулюванні ми вказали лише характеристики результуючого відношення, але нічого не сказали про спосіб його формування. В цьому випадку СУБД повинна сама вирішити, що за операції і в якому порядку потрібно виконати над відношеннями СТУДЕНТИ і ГРУПИ. Обидва розглянуті в прикладі способи насправді еквівалентні, і існують не дуже складні правила перетворення одного в інший.

Базисними поняттями реляційного обчислення є поняття змінної із визначеною для неї областю допустимих значень і поняття правильно побудованої формули, що опирається на змінні і спеціальні функції.

Залежно від того, що є областю визначення змінної, розрізняються обчислення кортежів і обчислення доменів. У обчисленні кортежів областями визначення змінних є відношення БД, тобто допустимим значенням кожної змінної є кортеж деякого відношення. У обчисленні доменів областями визначення змінних є домени, на яких визначені атрибути відношень БД, тобто допустимим значенням кожної змінної є значення деякого домена.

Для визначення змінної кортежу, використовується оператор RANGE. Наприклад, для того, щоб визначити змінну СТУДЕНТ, областю визначення якої є відношення СТУДЕНТИ, потрібно використати конструкцію

```
RANGE СТУДЕНТ IS СТУДЕНТИ
```

З цього означення виходить, що у будь-який момент часу змінна СТУДЕНТ представляє деякий кортеж відношення СТУДЕНТИ. При використанні змінних кортежів, у формулах можна посилатися на значення атрибуту змінної. Наприклад, для того, щоб послатися на значення атрибуту СТУД_ІМ'Я змінної СТУДЕНТ, потрібно використати конструкцію СТУДЕНТ.СТУД_ІМ'Я.

Правильно побудовані формули служать для вираження умов, що накладаються на змінні кортежів. У основі таких формул лежать прості порівняння, що є операціями порівняння значень атрибутів змінних або літерально заданих констант. Наприклад, конструкція "СТУДЕНТ.СТУД_НОМ=123456" є простим порівнянням. Складніші варіанти правильно побудованих формул реалізуються за допомогою логічних зв'язок NOT, AND, OR і IF ... THEN. Нарешті, допускається побудова правильно побудованих формул за допомогою кванторів. Якщо F – це правильно побудована формула, в якій бере участь змінна var, то конструкції EXISTS var (F) і FORALL var (F) є правильними. Тут квантор EXISTS позначає "існування", а FORALL – "для всіх кортежів".

Змінні, що входять в правильно побудовані формули, можуть бути вільними або зв'язаними. Всі змінні, що входять в її склад, при побудові якої не використовувалися квантори, є вільними. Фактично, це означає, що якщо для якогось набору значень

вільних змінних кортежів при обчисленні формул набуте значення "істина", то ці значення змінних кортежів, можуть входити в результуюче відношення. Якщо ж ім'я змінної використане відразу після квантора при побудові формул виду EXISTS var (F) або FORALL var (F), то тут, і у всіх формулах де вона використана, var – зв'язана змінна. При обчисленні значення такої правильно побудованої формули використовується не одне значення зв'язаної змінної, а вся її область визначення.

Нехай СТУД1 і СТУД2 – дві змінні кортежів, визначених на відношенні СТУДЕНТИ. Тоді, формула

```
EXISTS СТУД2(СТУД1.СТУД_СТИП>СТУД2.СТУД_СТИП)
```

для поточного кортежу змінної СТУД1 приймає значення "істина" тільки в тому випадку, якщо у всьому відношенні СТУДЕНТИ знайдеться такий кортеж, пов'язаний із змінною СТУД2, що значення його атрибуту СТУД_СТИП задовольняє внутрішній умові порівняння.

Правильно побудована формула

```
FORALL СТУД2(СТУД1.СТУД_СТИП > СТУД2.СТУД_СТИП)
```

для поточного кортежу змінної СТУД1 приймає значення "істина" тільки в тому випадку, якщо для всіх кортежів відношення СТУДЕНТИ, пов'язаних із змінною СТУД2, значення атрибуту СТУД_СТИП задовольняють умові порівняння.

Таким чином, правильно побудовані формули забезпечують засоби вираження умови вибірки із відношень БД. Щоб можна було використовувати реляційне обчислення для реальної роботи із БД, потрібен ще один компонент, який визначає набір і імена стовпців результуючого відношення. Цей компонент називається цільовим списком.

Цільовий список будується із цільових елементів, кожний з яких може мати наступний вигляд:

- var.attr, де var – ім'я вільної змінної, яка відповідає формулі, а attr – ім'я атрибуту відношення, на якому визначена змінна var;
- var, що еквівалентно наявності підсписку var.attr1, var.attr2, ..., var.attrn, де attrl, attr2, ..., attrn включає імена всіх атрибутів визначального відношення;
- new_name=var.attr; new_name – нове ім'я відповідного атрибуту результуючого відношення.

Останній варіант потрібен в тих випадках, коли у формулі використовуються декілька вільних змінних із однаковою областю визначення.

У обчисленні доменів областю визначення змінних є не відношення, а домени. Стосовно БД СТУДЕНТИ-ГРУПИ можна говорити, наприклад, про доменні змінні ІМ'Я (значення домена – допустимі імена) або НОМСТУД (значення домена – допустимі номери студентів).

Основною відмінністю обчислення доменів від обчислення кортежів є наявність додаткового набору предикатів (див. нижче), що дозволяють виражати так звані умови членства. Якщо R – це n-арне відношення з атрибутами a1, a2,..., an, та умова членства має вигляд

```
R (ai1:vi1, ai2:vi2, ..., aim:vim) (m<=n)
```

де vij – це або константа, що літерально задається, або ім'я змінної кортежу. Умова членства приймає значення ІСТИНА тільки в тому випадку, якщо відносно R існує кортеж, що містить відповідні значення вказаних атрибутів. Якщо vij – константа, то на атрибут aij задається жорстка умова, не залежна від поточних значень доменних

змінних, якщо ж vij – ім'я доменної змінної, то умова членства може приймати різні значення при різних значеннях цієї змінної.

Декілька слів про предикати. Предикатом прийнято називати деяку логічну функцію, яка для деякого аргументу повертає значення ІСТИНА або БРЕХНЯ. Відношення може бути розглянуто як предикат із аргументами, які є атрибутами даного відношення. Якщо заданий конкретний набір кортежів присутній у відношенні, то предикат видасть дійсний результат, інакше – помилковий.

У всій решті відношень формули і вирази обчислення доменів виглядають схожими на формули і вирази обчислення кортежів. Реляційне обчислення доменів покладене в основу більшості мов запитів, що базуються на використанні форм.

1.3. Проектування логічної структури бази даних

1.3.1. Концепція функціональної залежності

Розглянемо концепцію функціональної залежності, яка, по суті, є зв'язком типу "багато до одного" між множиною атрибутів всередині відношення. Для прикладу розглянемо БД структури, приведеної на рис. 3.1.

Тут відношення S (студенти) – має чотири атрибути:

- SN – номер студентського квитка;
- NAME – прізвище, ім'я, по батькові студента;
- GROUP – студентська група;
- SPEC – найменування спеціальності.

Відношення P (предмети) має чотири атрибути:

- PN – код предмету;
- PNAME – найменування предмету:
- TEACHER – прізвище, ім'я, по батькові викладача:
- KAFEDRA – найменування кафедри, на якій читається даний предмет.

Відношення SP (оцінки) складається з трьох атрибутів:

- PN - код предмету;
- NAME - прізвище, ім'я, по батькові студента;
- SN - номер студентського квитка;
- OCINKA - оцінка по предмету.

S (Студенти)			
SN	NAME	GROUP	SPEC

P (Предмети)			
PN	PNAME	TEACHER	KAFEDRA

SP (Оцінки)			
PN	NAME	SN	OCINCA

Рис. 3.1. Структура БД.

Тоді можна говорити про те, що є функціональна залежність між множиною атрибутів {SN, PN} і {OCINKA}, тобто множині пар кортежів номерів студентських квитків і кодів предметів відповідає одне значення оцінки студента по предмету.

Необхідно відмітити той факт, що слід чітко розрізняти значення того або іншого відношення і набір всіх можливих значень, які дане відношення може приймати в той або інший момент часу.

Для першого випадку розглянемо означення функціональної залежності. Якщо R – деяке відношення, а X і Y – деякі підмножини множини атрибутів відношення R, то

Y функціонально залежна від X за умови, що кожне значення множини X відношення R пов'язане з одним значенням множини Y відношення R. Це записується в наступному вигляді:

X → Y

Наприклад, відносно SP всі кортежі задовольняють умові {SN, PN} → SP.

При розгляді другого випадку треба відмітити, що інтерес представляють не стільки функціональні залежності для визначеного в даний момент значення, а функціональні залежності, що мають місце для всіх можливих значень даної змінної. У нашому випадку залежність

PN → NAME

виконується для всіх можливих значень SP.

Таким чином, якщо R є змінною відношення, а X і Y – деякі підмножини множини атрибутів відношення R, то Y функціонально залежне від X за умови, що для будь-якого допустимого значення R кожне значення X пов'язане з одним значенням множини Y.

Крім того, якщо X є потенційним ключем відношення R, то всі атрибути Y відношення R повинні бути функціонально залежні від X. Наприклад:

PN → {PN, NAME, SN,OCINKA}

Якщо ж відношення R задовольняє функціональній залежності X → Y, але X не є потенційним ключем, то можна говорити про надмірність R. Так у відношенні PN дані про прізвище студента повторюються кожного разу при введенні оцінки.

Принципово, деякі функціональні залежності можуть означати існування інших функціональних залежностей. Наприклад:

залежність {SN, PN} → SP означає і
залежності {SN, PN} → NAME і
{SN, PN} → OCINKA.

У складнішому випадку, якщо є відношення R із трьома атрибутами X, Y, Z, то за умови виконання функціональних залежностей X → Y і Y → Z, очевидно, має місце залежність X → Z. При цьому набір всіх функціональних залежностей, які визначені даною множиною функціональних залежностей, прийнято називати замиканням.

Таким чином, викладені вище поняття і означення дозволяють перейти до питань нормалізації БД.

1.3.2. Нормалізація бази даних

Розглянемо для початку класичний підхід, при якому весь процес проектування проводиться в термінах реляційної моделі даних методом послідовних наближень до прийнятного набору схем відношень. Початком повинне служити представлення предметної області (тобто тієї реальної інформації, яка зберігатиметься в БД) у вигляді одного або декількох відношень. Рекомендується на кожному кроці проектування проводити деякий набір схем відношень, що володіють кращими властивостями з погляду представлення інформації Взагалі кажучи, процес проектування є у тому числі і нормалізацію схем відношень, причому кожна наступна нормальна форма (НФ) володіє властивостями кращими, ніж попередня.

Так, скажімо, в структурі БД, приведеної на рис. 3.1 інтуїтивно відчувається, що є надмірність. Дійсно, відносно SP присутній атрибут NAME, проте і відносно S цей атрибут має місце. Таким чином, інформація про прізвище і ім'я студента дублюється. При цьому виникають проблеми, наприклад пов'язані з тим. що при зміні прізвища студента виникає необхідність її зміни в обох відношеннях.

З вищесказаного виходить простий висновок –добра структура БД містить по одному елементу інформації і лише в одному місці, що і дозволяє уникнути надмірність.

У теорії реляційних БД звичайно виділяється наступна послідовність нормальних форм:

- перша нормальна форма (1НФ);
- друга нормальна форма (2НФ);
- третя нормальна форма (3НФ);
- нормальна форма Бойса-Кодда (НФБК);
- четверта нормальна форма (4НФ);
- п'ята нормальна форма, або нормальна форма проекції-з'єднання (5НФ або НФПЗ).

Кожна НФ вищого порядку є вигіднішою з погляду концепції реляційних БД, ніж попередня При цьому необхідно відмітити той факт, що якщо деяка БД знаходиться, скажімо, в 2НФ, то не виключено, що її частина вже знаходиться в 3НФ, всередині якої може знаходитися частина в НФБК і т.д.

Основні властивості НФ:

- кожна наступна НФ в деякому розумінні краща за попередню;
- при переході до наступної НФ властивості попередніх нормальних форм зберігаються.

У основі процесу проектування лежить метод нормалізації – декомпозиція відношення, що знаходиться в попередній НФ, у два або більше відношень, що задовольняють вимогам наступної НФ.

Кожній НФ відповідає деякий набір обмежень. Відношення, що знаходиться в деякій НФ, повинно задовольняти властивому цій формі набору обмежень. Оскільки вимога 1НФ є базовою в класичній реляційній моделі даних, почнемо виклад з вимог саме до цієї форми.

Відношення знаходитиметься в 1НФ за умови, що воно містить тільки логічно неподільні значення. Такі значення надалі називатимемо скалярними. При цьому необхідно відмітити, що приведене означення говорить про знаходження будь-яких нормалізованих відношень в 1НФ. В той же час відношення, що знаходиться тільки в 1НФ, володіє структурою не зовсім бажаною, наприклад, внаслідок її можливої надмірності.

Припустимо, що відношення SP (див. рис. 3.1) містить всю інформацію про студентську групу і оцінки. Назвемо його SP1, що приведено на рис. 3.2.

SP1 (Оцінки)				
OCINCA	**SN**	**SPEC**	**GROUP**	**PN**

Рис. 3.2. Структура відношення SP1.

Для відношення SP1 маємо як первинний ключ комбінацію {SN, PN}. Крім того, має місце функціональна залежність номера групи студента і його спеціальності, тобто GROUP→SPEC. Якщо побудувати діаграму функціональних залежностей для відношення SP1, то вона матиме вигляд, представлений на рис. 3.3 (для спрощення деякі функціональні зв'язки не показані).

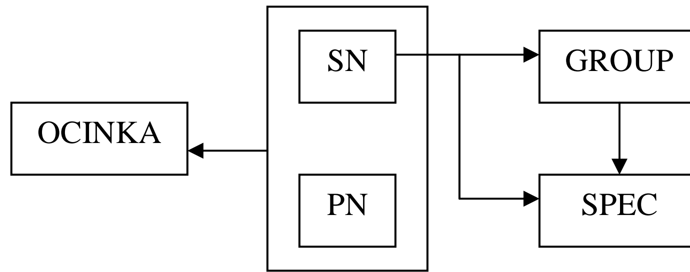


Рис. 3.3. Діаграма функціональних залежностей для відношення SP1.

Небажаність такої структури зв'язана, наприклад, із тим, що неключові атрибути не є взаємно незалежними (GROUP→SPEC), або з тим, що неключові атрибути не всі неприводимо залежні від первинного ключа (GROUP і SPEC залежні від SN кожен окремо).

Крім того, виникають додаткові проблеми, пов'язані з надмірністю SP1. Так, скажімо, при виконанні операції INSERT (вставка) не можна вставити дані про студентів і студентську групу, якщо ще ними не здавався іспит – адже для такого випадку не буде відоме значення первинного ключа.

При виконанні операції DELETE (видалення) буде видалена інформація не тільки про конкретну оцінку студента по конкретному іспиту, але і інформація про те, у якій групі і на якій спеціальності цей студент вчиться.

Нарешті, при виконанні операції UPDATE (оновлення) для якого-небудь студента при переведенні його із однієї групи в іншу виникає ситуація, коли ми будемо змушені або шукати всю інформацію про цього студента, вносячи відповідні корективи, або у відношенні матиме місце несумісний результат – один і той же студент буде як би одночасно знаходитися в різних групах.

Уникнути цих ситуацій можна шляхом заміни відношення SP1 на два – SP2 і SP, діаграма функціональних залежностей для яких приведена на рис. 3.4. Така структура дозволяє виконати без проблем операцію INSERT у відношення SP2 навіть за відсутності інформації про складені іспити; операцію DELETE над оцінками без втрати інформації про студента; операцію UPDATE без необхідності виправлення даних у всіх кортежах, що відносяться до студента, що переводиться в іншу групу.

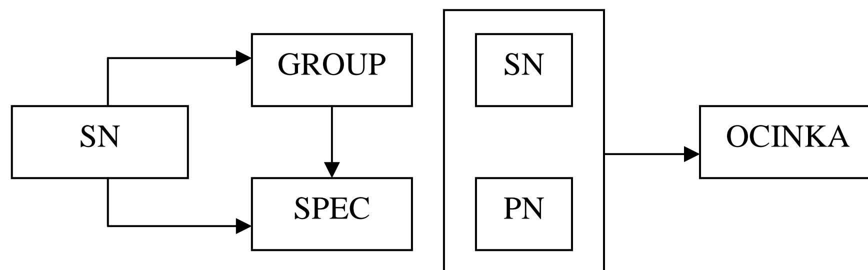


Рис. 3.4. Діаграма функціональних залежностей для відношень SP2 і SP.

Відношення знаходиться в 2НФ у тому випадку, коли воно вже вважається таким, що знаходиться в 1НФ, і кожен неключовий атрибут повністю (неприводимо) залежить від первинного ключа. Крім того, всяке відношення, яке знаходиться в 1НФ і не знаходиться в 2НФ, завжди можна перетворити і звести до еквівалентного набору відношень, що знаходяться в 2НФ. Цей процес полягає у заміні відношення, що знаходиться в 1НФ, набором проекцій, еквівалентних вихідному. Відповідно, об'єднання цих проекцій дасть вихідне відношення.

Відношення SP і SP2 знаходяться в 2НФ з первинними ключами відповідно до SN і {SN, PN}. Дійсно, якщо над відношеннями SP і SP2 провести об'єднання по атрибуту SN, то одержимо відношення SP1.

Проте у відношенні SP2 із-за функціональної залежності GROUP→SPEC все одно можуть виникнути проблеми при маніпуляції із даними. Так, при виконанні операції INSERT не можна вказати, що дана група відноситься до деякої спеціальності, доки в цій групі не з'явиться хоч би один студент. Якщо над відношенням SP2 буде виконана операція DELETE, то разом з інформацією про групу буде втрачена інформація про спеціальність, по якій проходить навчання дана група. В той же час, при виконанні операції UPDATE, при зміні найменування спеціальності виникне необхідність або шукати всю інформацію по коректованій спеціальності у всьому відношенні, вносячи відповідні зміни, або матиме місце несумісний результат.

Для вирішення щойно виниклих проблем замінимо відношення SP2 на дві проекції – SG і GS, структура яких приведена на рис. 3.5.

SG	
SN	**GROUP**

GS	
GROUP	**SPEC**

Рис. 3.5. Структура відношень SG і GS.

Дійсно, шляхом декомпозиції була виключена "подвійна" залежність атрибуту SPEC від первинного ключа SN і атрибуту GROUP. Таким чином, діаграма функціональних залежностей для відношень SG і GS матиме вигляд, показаний на рис. 3.6.

Рис. 3.6. Діаграма функціональних залежностей відношень SG і GS.

Відношення знаходиться в 3НФ в тому випадку, якщо воно знаходиться в 2НФ, неключові атрибути взаємно незалежні і кожен неключовий атрибут неприводимо залежить від первинного ключа (тобто можливо змінювати значення атрибутів без зміни первинного ключа і інших неключових атрибутів). Відношення, що знаходиться в 2НФ, завжди може бути приведено до еквівалентного набору відношень в 3НФ.

Відношення SG і GS знаходяться в 3НФ, причому первинними ключами в них є відповідно атрибути SN і GROUP.

На практиці схема відношень в 3НФ в більшості випадків достатня, тобто приведенням до 3НФ процес проектування реляційної БД звичайно закінчується. Проте іноді корисно продовжити процес нормалізації.

Введемо означення детермінанта – це будь-який атрибут, від якого повністю функціонально залежить деякий інший атрибут. Тоді відношення знаходитиметься в НФБК, якщо кожен детермінант є потенційним ключем.

Для розглянутих вище прикладів можна сказати, що відношення SP1 не знаходиться в НФБК, а відношення SP, SG і GS знаходяться в НФБК.

Дійсно, відношення SP1 (див. рис. 3.3) містить три детермінанти – SN, GROUP і {SN, PN}, із яких тільки {SN, PN} є потенційним ключем. Звідси і зроблений висновок, що SP1 не знаходиться в НФБК. Для відношень SP, SG і GS можна говорити про те, що вони знаходяться в НФБК, оскільки в кожному з них єдиний потенційний ключ є єдиним детермінантом.

Розглянемо схему відношення PR (ПРЕДМЕТИ), приведену на рис. 3.7.

PR (Предмети)		
PN	PNAME	PZAN

Рис. 3.7. Структура відношення PR (ПРЕДМЕТИ).

Відношення ПРЕДМЕТИ містить номери (коди) предметів, для кожного предмету – перелік курсів, включених в цей предмет, і. нарешті, перелік завдань, що передбачаються по курсу. При цьому по курсу їх може бути декілька, а різні курси можуть включати однакові завдання.

Кожен кортеж відношення пов'язує деякий предмет з курсом і завданням, які повинні бути виконані в рамках даного курсу. Внаслідок сформульованих вище умов, єдиним можливим ключем відношення є складений атрибут {PN, PNAME, PZAN} і немає ніяких інших детермінантів. Отже, відношення ПРЕДМЕТИ знаходиться в НФБК. Але при цьому воно володіє недоліками: якщо, наприклад, деяке завдання додається до даного курсу, необхідно вставити у відношення ПРЕДМЕТИ стільки кортежів, скільки завдань в ньому передбачено.

Справа в тому, що в даному відношенні існують багатозначні залежності. Відносно R {A, B, C} існує багатозначна залежність між A і B (A→→B) в тому випадку, якщо множина значень B, що відповідає парі значень A і C, залежить тільки від A і не залежить від C.

У відношенні ПРЕДМЕТИ існують наступні дві багатозначні залежності:

PN→→PNAME
PN→→PZAN

Подальша нормалізація відношень, подібних відношенню ПРЕДМЕТИ, грунтується на проектуванні без втрат. Тут під останнім розуміється такий спосіб декомпозиції відношення, при якому початкове відношення повністю і без надмірності відновлюється шляхом природного об'єднання одержаних відношень.

Таким чином, відношення знаходиться в 4НФ в тому випадку, якщо у разі існування багатозначної залежності A→→B всі решту атрибутів відношення функціонально залежать від A.

У нашому прикладі можна провести декомпозицію відношення ПРЕДМЕТИ в два відношення ПРЕДМЕТИ-КУРСИ і ПРЕДМЕТИ-ЗАВДАННЯ, як показано на рис. 3.8.

PQ (ПРЕДМЕТИ-КУРСИ)	
PN	PNAME

PZ (ПРЕДМЕТИ-ЗАВДАННЯ)	
PN	PZAN

Рис. 3.8. Декомпозиція відношення PR (ПРЕДМЕТИ).

Обидва одержані відношення знаходяться в 4НФ і вільні від відмічених вище проблем.

У всіх розглянутих до цього моменту нормалізація проводилася декомпозиція одного відношення в два. Іноді це зробити не вдається, але можлива декомпозиція в більше число відношень, кожне з яких володіє кращими властивостями. Розглянемо відношення P1 (ВИКЛАДАЧІ), структура якого приведена на рис. 3.9.

P1 (ВИКЛАДАЧІ)		
TEACHER	**KAFEDRA**	**PNAME**

Рис. 3.9. Структура відношення P1 (ВИКЛАДАЧІ).

Припустимо, що один і той же викладач може працювати на різних кафедрах і проводити заняття по декількох навчальних предметах. Первинним ключем цього відношення є повна сукупність його атрибутів, тобто {TEACHER, KAFEDRA, PNAME}. У відношенні відсутні функціональні і багатозначні залежності. Тому відношення знаходиться в 4НФ. Проте в ньому можуть існувати проблеми, які можна усунути шляхом декомпозиції в три відношення.

Введемо поняття залежності об'єднання. Відношення R (X, Y, ..., Z) задовольняє залежності об'єднання * (X, Y, .., Z) тільки у тому випадку, коли R відновлюється без втрат шляхом об'єднання своїх проекцій на X, Y, ..., Z.

Тоді відношення знаходиться в 5НФ (НФПЗ) тільки у тому випадку, коли будь-яка залежність об'єднання у відношенні виходить з існування деякого можливого ключа в даному відношенні.

Введемо наступні імена складених атрибутів:

```
TK = {TEACHER, KAFEDRA}
TP = {TEACHER, PNAME}
KP = {KAFEDRA, PNAME}
```

Припустимо, що у відношенні ВИКЛАДАЧІ існує залежність об'єднання: * (TK, TP, KP).

На прикладах легко показати, що при вставках і видаленні кортежів із ВИКЛАДАЧІ можуть виникнути проблеми. Їх можна усунути шляхом декомпозиції початкового відношення в три нові відношення, представлені на рис. 3.10.

TK	
TEACHER	KAFEDRA

TP	
TEACHER	PNAME

KP	
KAFEDRA	PNAME

Рис.3.10. Декомпозиція відношення P1 (ВИКЛАДАЧІ).

5НФ – це остання нормальна форма, яку можна одержати шляхом декомпозиції. Її умови достатньо нетривіальні, і на практиці 5НФ використовується достатньо рідко. Таким чином, ми дійшли підсумкової схеми процедури нормалізації відношень:

Відношення в 1НФ необхідно розбити на проекції для виключення всіх залежностей, що не є такими, що неприводяться. Як результат у результаті буде отриманий набір відношень в 2НФ.

Відношення, що знаходиться в 2НФ слід розбити на проекції для виключення "подвійних" функціональних зависимостей. В результаті буде отриманий набір відношень в 3НФ.

Отримані відношення в 3НФ слід розбити на проекції для виключення будь-яких функціональних залежностей, в яких детермінанти не є потенційними ключами. В результаті такого приведення буде отриманий набір відношень в НФБК.

Відношення, що перебувають в НФБК, необхідно розбити на проекції для виключення будь-яких багатозначних залежностей, які не є функціональними. У результаті одержимо набір відношень в 4НФ.

Відношення в 4НФ розбивають на проекції з метою виключення будь-яких залежностей об'єднання, які не обумовлені потенційними ключами. Таким чином буде отриманий набір відношень в 5НФ.

1.3.3. Об'єктне моделювання

Тепер необхідно зупинитися на тому факті, що СУБД володіють достатньо обмеженими відомостями про змістовне значення тих даних, які зберігаються в БД. Широке розповсюдження реляційних СУБД і їх використання в найрізноманітніших областях показує, що реляційна модель даних достатня для моделювання наочних областей. Проте проектування реляційної БД в термінах відношень на основі викладеної вище методології нормалізації досить часто є складним і незручний для проектувальника процес.

При цьому виявляється обмеженість реляційної моделі даних в наступних моментах:

- реляційна модель не дає проектувальнику достатніх засобів для представлення змісту даних. Семантика реальної предметної області винна незалежним від моделі способом як би вимальовуватися в голові проектувальника. Наприклад, це відноситься до згадуваної вище проблеми представлення обмежень цілісності;
- для багатьох застосувань важко моделювати предметну область на основі простих таблиць У ряді випадків на початковій стадії проектувальнику доводиться прикладати великі зусилля, щоб описати предметну область у вигляді однієї (можливо, навіть ненормалізованої) таблиці-відношення;
- хоча весь процес проектування БД відбувається на основі врахування залежностей між даними, реляційна модель не дає яких-небудь засобів для представлення цих залежностей;
- не дивлячись на те, що процес проектування починається з виділення деяких суттєвих для застосування об'єктів предметної області ("суті") і виявлення зв'язків між цією суттю, реляційна модель даних не володіє яким-небудь апаратом для розділення суті і зв'язків.

Необхідність в зручніших і могутніших засобах моделювання предметної області поставила перед проектувальниками БД завдання семантичного моделювання даними.

Найчастіше на практиці семантичне моделювання використовується на першій стадії проектування БД. При цьому в термінах семантичної моделі проводиться концептуальна схема БД, яка потім перетвориться до реляційної схеми. Цей процес виконується під управлінням методик, в яких достатньо чітко обумовлені всі етапи такого перетворення. В результаті проводиться реляційна схема БД в 3НФ.

В даний час проводяться дослідницькі розробки над БД в семантичній моделі, тобто СУБД, заснованої на семантичній моделі даних. При цьому розглядаються два варіанти – забезпечення призначеного для користувача інтерфейсу на основі семантичної моделі даних із автоматичним відображенням відношень в реляційну модель даних і пряма реалізація СУБД, заснована на якій-небудь семантичній моделі даних. Найближче до другого підходу знаходяться сучасні об'єктно-орієнтовані СУБД, моделі даних яких по багатьох параметрах близькі до семантичних моделей.

Розглянемо одну з найбільш важливих і поширених семантичних моделей даних – модель «Суть-Зв’язки» (часто її називають стисло ER-моделлю). На використанні різновидів ER-моделей заснована більшість сучасних підходів до проектування реляційних БД.

Моделювання предметної області базується на використанні діаграм, що включають невелике число різнорідних компонентів. У зв'язку з наочністю представлення концептуальних схем БД ER-моделі набули широкого поширення в системах, що підтримують автоматизоване проектування реляційних БД.

Основними поняттями ER-моделі є суть, зв'язок і атрибут.

Суть – це реальний або уявний об'єкт, інформація про який повинна зберігатися і бути доступна. У діаграмах ER-моделі суть представляється у вигляді прямокутника, що містить ім'я суті. При цьому ім'я суті – це ім'я типу, а не деякого конкретного елементу цього типа. Наприклад, на рис. 3.11 приведена суть ІНСТИТУТ із зразковими об'єктами Прикарпатський і Політехніка.

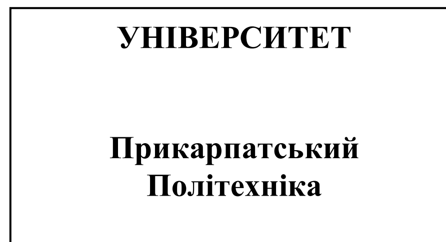


Рис. 3.11. Суть ІНСТИТУТ.

Кожен елемент суті повинен бути відмінний від будь-якого іншого елементу тієї ж суті, що аналогічно вимозі відсутності кортежів-дублікатів у відношеннях.

Зв'язок – це графічна асоціація, що встановлюється між двома суттями. Ця асоціація завжди є бінарною і може існувати між двома різними суттями або між суттю і їй же самій – останній вид зв'язку називають рекурсивним. У будь-якому зв'язку виділяються два кінці, на кожному з яких вказується ім'я кінця зв'язку, степінь кінця зв'язку (скільки елементів даної суті зв'язується) і обов'язковість зв'язку (тобто чи будь-який елемент даної суті повинен брати участь в цьому зв'язку).

Зв'язок представляється у вигляді лінії, що сполучає дві суті або ведучої від суті до неї ж самої. При цьому в місці зіткнення зв'язку із суттю використовується триточковий вхід в прямокутник, якщо для зв'язку можуть використовуватися декілька елементів, і одноточковий вхід, якщо в зв'язку може брати участь тільки один елемент суті. Обов'язковий кінець зв'язку зображається суцільною лінією, а необов'язковий – пунктирною лінією.

На рис. 3.12 приведений приклад зв'язку між суттю ЗАВДАННЯ і СТУДЕНТ. При цьому кінець зв'язку із ім'ям "для" дозволяє з'єднати із одним студентом більше одного завдання, причому кожне завдання повинне бути пов'язане з яким-небудь студентом. Кінець суті із ім'ям "вирішує" означає, що кожне завдання може розв'язуватися тільки одним студентом, причому студент може і не мати завдання.

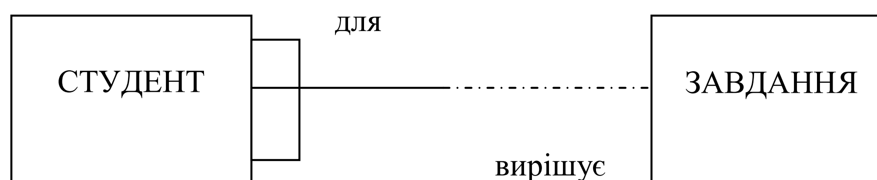


Рис. 3.12. Зв'язок суті ЗАВДАННЯ-СТУДЕНТ.

Трактування зображеної діаграми наступне: кожне завдання призначене для одного студента, проте кожен студент може мати одне або більше завдань.

На рис. 3.13 зображений рекурсивний зв'язок, що сполучає суть ЛЮДИНА із нею ж самою. Кінець зв'язку із ім'ям "учень" встановлює той факт, що людина може виступати один або кілька разів в житті в ролі учня. Кінець зв'язку із ім'ям "студент" означає, що не кожна людина була студентом.

Рис. 3.13. Рекурсивний зв'язок суті.

Атрибутом суті є будь-який елемент, який служить для уточнення, ідентифікації, класифікації, числової характеристики або вираження стану суті. Імена атрибутів заносяться в прямокутник, що позначає суть, і зображаються малими буквами під ім'ям суті, можливо, із прикладами.

Як і в реляційних схемах БД, в ER-схемах вводиться поняття нормальних форм, причому їх зміст дуже близько відповідає змісту реляційних нормальних форм. Відмітимо, що формулювання нормальних форм ER-схем роблять зрозумілішим зміст нормалізації реляційних схем. Приведемо тільки дуже короткі і неформальні означення трьох перших нормальних форм.

У 1НФ ER-схем усуваються атрибути, що повторюються, або групи атрибутів, тобто проводиться виявлення неявної суті, "замаскованої" під атрибути.

У 2НФ усуваються атрибути, залежні тільки від частини унікального ідентифікатора. Ця частина унікального ідентифікатора визначає окрему суть.

У 3НФ усуваються атрибути, залежні від атрибутів, що не входять в унікальний ідентифікатор. Вони є основою окремої суті.

До складніших елементів ER-моделі належать наступні:

- підтипи і супертипи сутностей. Як в мовах об'єктно-орієтованого програмування, вводиться можливість успадкування типу сутності, виходячи з одного або декількох супертипів;
- зв'язки "багато-з-багатьма". Іноді буває необхідно зв'язувати сутності таким чином, що з обох кінців зв'язку можуть бути присутніми декілька екземплярів сутності;
- уточнювані ступені зв'язку. Іноді буває корисно визначити можливу кількість екземплярів сутностей, що беруть участь в даному зв'язку. Для вираження цього обмеження дозволяється вказувати на кінці зв'язку її максимальний або обов'язковий степінь;
- каскадні видалення екземплярів сутностей. Деякі зв'язки бувають настільки сильними, що при видаленні опорного екземпляра сутності (відповідного кінцю зв'язку "один") потрібно видалити і всі екземпляри сутності, що відповідають кінцю зв'язку "багато". Відповідну вимогу "каскадного видалення" можна сформулювати при означенні сутності;
- домени. Як і у разі реляційної моделі даних, корисна можливість визначення потенційно допустимої множини значень атрибуту сутності (домена).

Ці та інші складніші елементи моделі даних «Сутність-Зв’язки» роблять її істотно могутнішою, але одночасно серйозно ускладнюють її використання. Наприклад, сутність може бути розщеплена на два або більше взаємно виключаючих підтипи, кожний з яких включає спільні атрибути. У підтипах можуть визначатися

власні атрибути і зв'язки. Сутність, на основі якої визначаються підтипи, називається супертипом.

Отримання реляційної схеми з ER-схеми здійснюється таким чином:

- кожна проста сутність перетворюється на відношення. Проста сутність – сутність, що не є підтипом і що не має підтипів. Ім'я сутності стає ім'ям відношення;
- кожен атрибут стає можливим стовпцем з тим же ім'ям; може вибиратися точніший формат. Стовпці, що відповідають необов'язковим атрибутам, можуть містити невизначені значення; стовпці, що відповідають обов'язковим атрибутам – не можуть;
- компоненти унікального ідентифікатора сутності перетворюються в первинний ключ відношення. Якщо є декілька можливих унікальних ідентифікаторів, вибирається найбільш зручний. Якщо до складу унікального ідентифікатора входять зв'язки, до стовпців первинного ключа додається копія унікального ідентифікатора сутності, що знаходиться на дальньому кінці зв'язку (цей процес може продовжуватися рекурсивно). Для іменування цих стовпців використовуються імена кінців зв'язків і/або імена сутності;
- зв'язки "багато до одного" і "один до одного" стають зовнішніми ключами. Тобто робиться копія унікального ідентифікатора з кінця зв'язку "один", і відповідні стовпці складають зовнішній ключ. Необов'язкові зв'язки відповідають стовпцям, що допускають невизначені значення; обов'язкові зв'язки – стовпцям, що не допускають невизначені значення;
- індекси створюються для первинного ключа, зовнішніх ключів і тих атрибутів, які передбачається використовувати в запитах;
- якщо в концептуальній схемі були присутні підтипи, то можливі два способи: всі підтипи в одній таблиці або для кожного підтипу – окрема таблиця. При застосуванні першого способу відношення створюється для найбільш зовнішнього супертипа, а для підтипів можуть створюватися уявлення. У відношення додається по крайній мірі один стовпець, що містить код ТИПУ; він стає частиною первинного ключа. При використанні другого методу для кожного підтипу першого рівня (для більш нижніх – уявлення) супертип відтворюється за допомогою уявлення;
- є два способи роботи за наявності звиключаючих зв'язків: загальний домен і явні зовнішні ключі. Якщо решту зовнішні ключі всі в одному домені, тобто мають загальний формат, то створюються два стовпці: ідентифікатор зв'язку і ідентифікатор сутності. Стовпець ідентифікатора зв'язку використовується для розрізнення зв'язків, що покриваються дугою виключення. Стовпець ідентифікатора сутності використовується для зберігання значень унікального ідентифікатора сутності на дальньому кінці відповідного зв'язку. У другому випадку, якщо результуючі зовнішні ключі не відносяться до одного домена, то для кожного зв'язку, що покривається дугою виключення, створюються явні стовпці зовнішніх ключів; всі ці стовпці можуть містити невизначені значення.

Таким чином, розглянуті питання дозволять на початковому етапі правильно спроектувати логічну структуру БД.

1.4. Функції захисту бази даних

1.4.1. Транзакції і паралелізм

При роботі СУБД виникає необхідність захисту БД від можливих випадкових або навмисних ситуацій, коли існує вірогідність втрати даних. Наприклад, при доступі до БД відразу декількох користувачів можливе пошкодження або неправильний запис даних, що в свою чергу може привести до непередбачуваних наслідків. Очевидно, що з

таких ситуацій СУБД повинна вміти коректно виходити. Одним із способів розв'язання цих проблем є механізм транзакцій.

Підтримка механізму транзакцій – показник рівня розвиненості СУБД. Коректна підтримка транзакцій одночасно є основою забезпечення цілісності БД. Транзакції також складають основу ізольованості користувачів в розрахованих на багато користувачів системах.

Під транзакцією розуміється неподільна з погляду дії на БД послідовність операторів маніпулювання даними (читання, видалення, вставки, модифікації) така, що можливі два підсумки:

- результати всіх операторів, що входять в транзакцію, відповідним чином відображаються в БД;
- дія всіх цих операторів повністю відсутня.

У СУБД транзакція починається з оператора BEGIN. При цьому якщо транзакція завершена оператором COMMIT, то результати фіксуються в зовнішній пам'яті; при завершенні транзакції оператором ROLLBACK результати відсутні в зовнішній пам'яті.

Оператор COMMIT встановлює так звану точку фіксації, тобто момент, коли закінчується логічна одиниця роботи. Отже, БД знаходитиметься в узгодженому (цілісному) стані як на початку, так і в кінці транзакції. При завершенні транзакції оператором ROLLBACK БД теж знаходитиметься в узгодженому стані, оскільки ніякого впливу на дані така транзакція не чинитиме. Тут під узгодженістю розумітимемо той факт, що транзакції переводять БД із одного узгодженого стану в інший, тобто в такий, що виконуються всі умови обмеження цілісності і.т.п.

В свою чергу, поняття відновлення СУБД – процес, що має на увазі повернення БД в правильний стан, якщо який-небудь процес викликав збій даних. Відновлення базується на принципі надмірності БД при цьому по частині даних, що зберігаються, є можливість відновлення всієї інформації повністю.

Система повинна забезпечувати відновлення як після невеликих порушень (наприклад, після невдало завершених транзакцій), так і після серйозних збоїв (іноді їх називають глобальними), наприклад, збоїв живлення.

Глобальний збій повністю діє на СУБД і звичайно виділяють два види: збій системи, що порушує всі виконувані в даний момент транзакції, але не порушує БД фізично, і збій носіїв, який є фізичною загрозою для даних.

Відновлення системи після першого виду глобального збою може бути здійснено по журналу транзакцій, в який заноситься інформація про транзакції, що почали своє виконання, і транзакції, що успішно завершилися. Якщо після перезавантаження системи в журналі будуть зустрічатися транзакції, що почалися до збою, але не закінчилися, то для всіх них виконується оператор ROLLBACK, внаслідок чого БД знову знаходитиметься в цілісному стані.

Процес відновлення після збою носіїв принципово інший. Відновлення в цьому випадку здійснюється з резервної копії БД. Зрозуміло, що для реалізації цього процесу необхідно, щоб в СУБД передбачалося резервне копіювання за допомогою відповідної програмної реалізації.

Поняття транзакції має безпосередній зв'язок з поняттям цілісності БД. Дуже часто БД може володіти такими обмеженнями цілісності, які просто неможливо не порушити, виконуючи зміну тільки одним оператором. Наприклад, в базі даних СТУДЕНТИ-ПРЕДМЕТИ-ОЦІНКИ (див. рис. 3.1) очевидним обмеженням цілісності є збіг значення атрибуту SN в кортежі відношення СТУДЕНТИ із аналогічним атрибутом відношення ОЦІНКИ. В цьому випадку виникає проблема із додаванням новому студенту оцінки по іспиту. Незалежно від того, яка операція буде виконана першою, вставка нового кортежу у відношення СТУДЕНТИ або у відношення ОЦІНКИ, після виконання операції БД опиниться в нецілісному стані.

Для підтримки подібних обмежень цілісності допускається їх порушення всередині транзакції із тією умовою, щоб до моменту завершення транзакції умови цілісності повинні бути дотримані. У системах із розвиненими засобами обмеження і контролю цілісності кожна транзакція починається при цілісному стані БД і повинна залишити цей стан цілісним після свого завершення. Недотримання цієї умови призводить до того, що замість фіксації результатів транзакції відбувається її відміна (тобто відкат транзакції, коли вона замість оператора COMMIT завершується оператором ROLLBACK), і БД залишається в такому стані, в якому знаходилася до моменту початку транзакції.

У розрахованих на багато користувачів системах із однією БД, взагалі кажучи, одночасно можуть працювати декілька користувачів або прикладних програм. Одним з основних завдань СУБД є забезпечення ізольованості користувачів, тобто створення такого режиму роботи, щоб кожному з користувачів здавалося, що він працює з БД поодинці. Таке завдання СУБД прийнято іноді називати паралелізмом транзакцій.

При паралельній обробці БД виникає три основні проблеми:

- проблема втрати результатів оновлення – полягає в першу чергу в тому, що транзакція може бути незавершена через те, що дані, які вона обробляє, можуть бути модифіковані іншою транзакцією;
- проблема незафіксованості залежності – полягає в тому, що транзакція може використовувати для роботи дані, які зараз модифікуються іншою транзакцією. Зрозуміло, що перша із них цілком може працювати із даними, які по завершенню другої транзакції в БД просто відсутні;
- проблема несумісного аналізу – пов'язана з тим, що в результаті модифікації БД транзакцією, інша транзакція може внести в БД якусь інформацію, яка не відповідатиме цілісному стану БД.

Для вирішення цих проблем використовують блокування – метод управління паралельними процесами, при якому об'єкт БД, наприклад, кортеж не може бути модифікований без відома транзакції. Тобто результатом блокування є блокування доступу до об'єкту з боку інших транзакцій, чим виключається непередбачувана зміна об'єкту.

Розрізняють два види блокування:

- блокування запису – при цьому транзакція блокує кортеж таким чином, що запит іншої транзакції до цього кортежу буде скасований;
- блокування читання – в цьому випадку транзакція блокує кортеж так, що запит з боку іншої транзакції на блокування запису цього кортежу буде знехтуваний, а на блокування читання – прийнятий.

У СУБД використовують протокол доступу до даних, що дозволяє уникнути проблем паралелізму. Його суть полягає в наступному:

- транзакція, результатом дії якої на кортеж є його видобування, зобов'язана накласти блокування читання на цей кортеж;
- транзакція, призначена для модифікації кортежу, зобов'язана накласти блокування запису на цей кортеж;
- у випадку, якщо запрошуване блокування на кортеж відкидається через те, що на кортеж вже накладене блокування, то транзакція переводиться в режим очікування до тих пір, поки блокування не буде зняте;
- блокування запису зберігається аж до кінця виконання транзакції, тобто до виконання операторів COMMIT або ROLLBACK.

Рішення проблем, паралельної обробки БД, полягає в тому. що кортеж блокується, а подальші транзакції, що модифікують цей кортеж, відкидаються і переводяться в режим очікування.

У зв'язку з властивістю збереження цілісності БД транзакції є відповідними одиницями ізольованості користувачів. Дійсно, якщо кожен сеанс роботи з БД

реалізується транзакцією, то кожен користувач починає роботу із узгодженим станом БД, тобто з таким її станом, в якому вона могла б знаходитися, навіть якщо б користувач працював з нею поодинці.

1.4.2. Безпека і цілісність баз даних

Питання безпеки і цілісності – ще одна з найважливіших сторін роботи СУБД. Під безпекою розуміють захист БД від несанкціонованого руйнування, зміни і модифікації. Систему можна вважати безпечною тільки в тому випадку, якщо користувачу допускається виконувати тільки дозволені дії. Цілісність БД пов'язана з коректним виконанням цих дій.

У СУБД традиційно підтримуються вибірковий або обов'язковий підходи забезпечення безпеки даних. При вибірковому підході до управління безпекою кожен користувач володіє різними правами (повноваженнями) при роботі із тим або іншим об'єктом БД. У разі обов'язкового підходу кожному об'єкту БД привласнюється рівень доступу, а користувачам – рівні допуску. Зрозуміло, для отримання доступу до об'єкту користувач повинен володіти відповідним рівнем допуску.

Обидва підходи реалізуються в СУБД у вигляді особливих правил безпеки, що передбачають розпізнання джерела запиту. З цієї причини на початку роботи користувач вводить свій ідентифікатор (логін) і пароль для підтвердження своїх повноважень.

Вибіркове управління доступом задається правилами, які повинні включати наступне:

- ім'я правила – є структурою, по якій це правило ідентифікується системою;
- власне правила або привілеї – набір директив, що становлять спосіб і можливість доступу, модифікацій і т.п. об'єктів БД;
- діапазон застосування привілеїв;
- ідентифікатори користувачів, що володіють вищепереліченими привілеями;
- дія при порушенні правила – тут вказується поведінка системи у випадку, якщо користувач порушив правило безпеки. Як правило, воно полягає у відмові, від виконання запрошуваної дії.

Для прикладу розглянемо правило безпеки для відношення P структури, приведеної на рис. 3.1.

```
СТВОРИТИ ПРАВИЛО БЕЗПЕКИ RULE1
ДЛЯ МОДИФІКАЦІЇ І ВИДАЛЕННЯ P
            (PN, PNAME, TEACHER, KAFEDRA)
ДЛЯ KAFEDRA="ФИЗИКА"
КОРИСТУВАЧІ: Ivan, Denis, Andrew
НЕ ВИКОНУВАТИ ПРИ ПОРУШЕННІ ПРАВИЛА
```

Приведене вище правило містить всі п'ять обумовлених елементів. Фактично створене правило безпеки з ім'ям RULE1, що дозволяє модифікацію і видалення кортежів (PN, PNAME, TEACHER, KAFEDRA) відношення P, що відносяться до кафедри фізики для користувачів з ідентифікаторами Ivan, Denis і Andrew. Якщо правило буде порушено (наприклад, запрошується обговорювана дія з боку користувача Petr), то в запрошуваній дії буде відмовлено.

Обов'язкове управління доступом до БД реалізується при виконанні наступних правил:

- користувач має можливість роботи (але не модифікації) з об'єктом, якщо рівень його допуску більше або рівний рівню доступу об'єкту;
- користувач має можливість модифікувати об'єкт, якщо рівень його допуску рівний рівню доступу об'єкту.

Правило безпеки в цьому випадку для користувача із ідентифікатором DENIS можна, наприклад, сформулювати таким чином:

```
СТВОРИТИ ПРАВИЛО БЕЗПЕКИ RULE2
ДЛЯ КОРИСТУВАЧА Denis
ВСТАНОВИТИ РІВЕНЬ ДОПУСКУ =5
```

А для відношення P, наприклад, ось так:

```
СТВОРИТИ ПРАВИЛО БЕЗПЕКИ RULE3
ДЛЯ МОДИФІКАЦІЇ І ВИДАЛЕННЯ P
     (PN, PNAME, TEACHER, KAFEDRA)
ВСТАНОВИТИ РІВЕНЬ ДОПУСКУ = 5
НЕ ВИКОНУВАТИ ПРИ ПОРУШЕННІ ПРАВИЛА
```

Тоді користувач DENIS має доступ для модифікації і видалення кортежів відношення P, оскільки рівень його допуску і рівень доступу до відношення відповідають один одному.

Для забезпечення високого рівня безпеки СУБД ведуть журнал виконуваних операцій. По цьому журналу – з одного боку можна здійснити відновлення даних, з іншою – виявити коли, яким чином і ким були здійснені несанкціоновані дії над БД.

Природно у файлі журналу зберігається наступна інформація:

- початковий текст запиту;
- ім'я віддаленого терміналу, звідки був поданий запит;
- ідентифікатор користувача, що подав запит;
- дата і час здійснення запиту;
- використовувані запитом відношення, кортежі і атрибути;
- значення даних, з якими працювали до їх модифікації;
- значення даних, з якими здійснювалася робота після їх модифікації.

Крім вищесказаного, досить часто для захисту даних використовується зберігання і передача шифрованих даних. Відкритий (незашифрований) текст шифрується за допомогою спеціальних алгоритмів.

Іншою стороною проблеми безпеки і цілісності БД є точність і коректність даних, що зберігаються в ній. Зазвичай це питання вирішують за допомогою обмежень цілісності.

Традиційно розрізняють два види обмежень цілісності: що перевіряються негайно і відкладаються. До обмежень цілісності, що негайно перевіряються, відносяться такі обмеження, перевірку яких не має сенсу відкладати на пізніший період. Прикладом такого обмеження, перевірку якого відкладати безглуздо, є обмеження домена – оцінка по іспиту не може бути менше 1 і більше 5 або вік викладача не може перевищувати 100 років. Складнішим обмеженням, перевірку якого неможливо відкласти, є наступне: студент не може складати іспит і одержати оцінку, якщо у нього до цього моменту не здані всі заліки. Обмеження цілісності, що перевіряються негайно, відповідають рівню окремих операторів мовного рівня СУБД. При їх порушеннях не проводиться відкат транзакції, а лише відкидається відповідний оператор.

Обмеження цілісності, що відкладаються – це обмеження на БД, а не на які-небудь окремі операції. Звичайно такі обмеження перевіряються в кінці транзакції, і їх порушення викликає завершення транзакції оператором ROLLBACK. У деяких СУБД підтримується спеціальний оператор перевірки обмежень цілісності всередині транзакції. Якщо після виконання такого оператора виявляється, що умови цілісності не

виконані, користувач може сам виконати оператор ROLLBACK або постаратися усунути причини нецілісного стану БД всередині транзакції.

При дотриманні обов'язкової вимоги підтримки цілісності БД можливі наступні рівні ізольованості транзакцій:

- перший рівень – відсутність втрачених змін. Розглянемо наступний приклад сумісного виконання двох транзакцій. Транзакція 1 змінює об'єкт бази даних S. До завершення транзакції 1 транзакцію 2 також змінює об'єкт S. Транзакція 2 завершується оператором ROLLBACK, наприклад, внаслідок порушення обмежень цілісності. Тоді при повторному читанні об'єкту S транзакція 1 не бачить змін цього об'єкту, проведених раніше. Виникає ситуація втрачених змін. Зрозуміло, вона суперечить вимозі ізольованості користувачів. Щоб уникнути такої ситуації в транзакції 1 потрібно, щоб до завершення транзакції 1 ніяка інша транзакція не могла змінювати об'єкт S;
- другий рівень – відсутність читання даних, що модифікуються іншою транзакцією. Розглянемо такий приклад сумісного виконання транзакцій 1 і 2. Транзакція 1 змінює об'єкт бази даних S. Паралельно із цим транзакція 2 читає об'єкт S. Оскільки операція зміни ще не завершена, транзакція 2 бачить неузгоджені дані. Зокрема, операція транзакції 1 може бути відвернута при контролі обмеження цілісності, що негайно перевіряється. Це теж не відповідає вимозі ізольованості користувачів (кожен користувач починає свою транзакцію при узгодженому стані БД і має право чекати узгоджені дані). Щоб уникнути ситуації читання таких даних, до завершення транзакції 1, що змінила об'єкт S, ніяка інша транзакція не повинна читати об'єкт S;
- третій рівень – відсутність читань, що не повторюються. Розглянемо наступний сценарій. Транзакція 1 читає об'єкт бази даних S. До завершення транзакції 1 транзакцію 2 змінює об'єкт S і успішно завершується оператором COMMIT. Транзакція 1 повторно читає об'єкт S і бачить його змінений стан. Щоб уникнути читань, що не повторюються, до завершення транзакції 1 ніяка інша транзакція не повинна змінювати об'єкт S.

Як вже було сказано вище, існує можливість забезпечення різних рівнів ізольованості для тієї або іншої транзакції шляхом введення блокування. Крім того, може використовуватися серіалізація транзакцій. Спосіб виконання набору транзакцій називається серійним, якщо результат спільного виконання транзакцій еквівалентний результату деякого послідовного виконання цих же транзакцій.

Серіалізація транзакцій – це такий механізм їх виконання по деякому серійному плану, який забезпечується на рівні основних функцій СУБД, відповідальних за управління транзакціями. Система, в якій підтримується серіалізація транзакцій, забезпечує реальну ізольованість користувачів.

Основна проблема в реалізації полягає у виборі методу серіалізації набору транзакцій, який не дуже обмежував би їх паралельність. Простим розв'язанням є дійсно послідовне виконання транзакцій. Але існують ситуації, в яких можна виконувати оператори різних транзакцій у будь-якому порядку із збереженням серійності. Прикладами можуть служити тільки трансакції, що читаються, а також транзакції, що не конфліктують по об'єктах БД.

Звичайно обмеження цілісності застосовують для опису базових відношень – останні містять дані, що відображають реальну дійсність, тому їх обробляють так, щоб дані були коректними.

У загальному випадку обмеження цілісності повинне містити три основні частини:

- ім'я обмеження – є структурою, по якій це обмеження ідентифікується системою;
- власне обмеження – набір директив і команд, що становлять спосіб і можливість контролю, і що представляє зрештою логічний вираз. Обмеження

задовольняється, якщо воно істинне, і порушується – якщо воно помилкове;
- дія при порушенні обмеження – тут пропонується дія системи при порушенні обмеження.

Наприклад, обмеження цілісності для відношення SP, приведеного на рис. 3.1, можна сформулювати таким чином:

```
СТВОРИТИ ОБМЕЖЕННЯ ЦІЛІСНОСТІ RULE4
ДЛЯ ВСІХ SP(SP.OCINKA>0 I SP.OCINKA<6)
НЕ ВИКОНУВАТИ ПРИ ПОРУШЕННІ ПРАВИЛА
```

У даному прикладі обмеження цілісності накладається на атрибут OCINKA відношення SP таким чином, що ігноруються всі спроби встановити оцінку менше 1 і більше 5. Точніше кажучи, в обмеженні обумовлений допустимий інтервал оцінок (більше 0 і менше 6).

При створенні обмеження цілісності система спочатку перевіряє, чи задовольняє поточний стан БД новому обмеженню. Якщо ця умова не виконується, то створюване обмеження може бути знехтуване. Інакше воно приймається і надалі використовується системою.

Розрізняють чотири типи обмежень цілісності:
обмеження цілісності домена – ним визначається безліч значень, із яких складається домен. Особливості обмеження такого роду полягають в тому, що його ім'я повинне співпадати з ім'ям домена. Крім того, оскільки домени самі по собі не обновляються, то відпадає необхідність передбачати реакцію на порушення обмеження. Більше того, ці обмеження можна усунути тільки за рахунок усунення самого домена;
обмеження цілісності атрибуту – це фактично означення домена, із якого беруться значення для даного атрибуту. Ім'я такого обмеження повинне співпадати з ім'ям відповідного обмеження домена, тобто з ім'ям домена. Перевірка здійснюється відразу, і спроба виконати дію, що порушує обмеження, буде знехтувана відразу ж. Нарешті, обмеження цілісності атрибуту знімаються тільки за допомогою усунення самого атрибуту;
обмеження цілісності відношення – правило, що задається тільки для даного відношення БД. Обмеження цілісності відношення завжди перевіряється негайно, тобто при будь-якій спробі модифікації відношення здійснюється контроль всіх заданих умов. Дія такого обмеження відбувається у тому випадку, коли обмеження (задане логічним виразом) стає помилковим;
обмеження цілісності БД – задається для двох або більше зв'язаних між собою відношень. На відміну від інших обмежень, цей їх різновид крім традиційних частин обов'язково повинен містити, принаймні, одну умову об'єднання відношень: зазвичай ця умова містить дві зв'язані змінні, визначені в двох різних відношеннях. Отже, обмеження призводить до того, що відношення будуть зв'язані між собою. Обмеження цілісності БД не перевіряються негайно, а їх виконання відкладається до кінця виконання транзакції. Як дія на порушення умови цілісності в переважній більшості випадків використовується завершення транзакції оператором ROLLBACK.

Різновидом традиційних обмежень цілісності є обмеження стану і переходу. Зміст їх полягає в тому, що достатньо часто виникає потреба розглянути не тільки область допустимих значень кортежу, але і перехід значень з одного стану в інший. Розглянемо приклад такого обмеження для модифікованого відношення S (СТУДЕНТИ), куди доданий атрибут KURS – курс, на якому вчиться студент:

```
СТВОРИТИ ОБМЕЖЕННЯ ЦІЛІСНОСТІ RULE5
ДЛЯ ВСІХ S(S.KURS >= S'.KURS)
НЕ ВИКОНУВАТИ ПРИ ПОРУШЕННІ ПРАВИЛА
```

Таке обмеження накладається на атрибут відповідно S'.KURS – до і S.KURS – після виконання оновлення. Приведений приклад обмежує курс, на якому вчиться студент так, що його можна змінити або у більшу сторону, або залишити без змін. Дійсно – адже курс не повинен зменшуватися. Обмеження стану і переходу використовують тільки для відношення або БД.

Таким чином, розглянуті питання безпеки і цілісності як БД в цілому, так і її елементів дозволяють сформулювати і означити відповідні обмеження цілісності, що, у свою чергу, допомагає вирішити проблему несумісних даних.

1.5. Додаткові аспекти реляційної технології

1.5.1. Уявлення

Механізм уявлень (view) є могутнім засобом СУБД, що дозволяє приховати реальну структуру БД від деяких користувачів за рахунок визначення представлення БД. Реально уявлення є деяким запитом, що зберігається в БД, а для користувача нічим не відрізняється від базового відношення БД. Будь-яка реалізація уявлення повинна гарантувати, що стан відношення, що представляється, точно відповідає стану даних, на яких означене це уявлення. Зазвичай обчислення уявлення проводиться кожного разу при його використанні. Розглянемо ці аспекти докладніше.

Розглянемо приклад створення уявлення на основі відношення SP (Оцінки), приведеного на рис. 3.1.

```
СТВОРИТИ УЯВЛЕННЯ VIEW1
ДЛЯ (SP.OCINKA >= 3) [NAME, SN, OCINKA]
```

При створенні уявлення інформація про нього записується в каталог БД під власним ім'ям (у нашому прикладі – VIEW1), а у користувача створиться повне уявлення того, що в БД реально існує відношення з таким ім'ям. При цьому будь-які зміни в даних адекватно відобразяться в уявленні – в цьому є його відмінність від запиту до БД, на яке уявлення дуже схоже. В той же час запити є як би "миттєвою фотографією" даних і при зміні останніх запит до БД необхідно повторювати.

У прикладі створюється уявлення VIEW1, яке складається з трьох атрибутів – NAME, SN і OCINKA, причому вибиратимуться тільки ті кортежі, для яких оцінка буде більша або рівна 3 – грубо кажучи, інформація по успішних студентах. З таким уявленням користувач може працювати, як із тим, що реально існує.

Наявність уявлень в БД необхідна для забезпечення логічної незалежності даних. Якщо система забезпечує фізичну незалежність даних, то зміни у фізичній структурі БД не впливають на роботу призначених для користувача програм. Логічна незалежність має на увазі той факт, що при зміні логічної структури даних вплив на призначені для користувача програми також не виявляється. Тут система повинна вміти вирішувати проблеми, пов'язані із зростанням і реструктуризацією БД.

SPO	
SN	**OCINKA**

SPP		
SN	**PN**	**NAME**

Рис. 5.1. Структура відношень SPO і SPP.

Очевидно, що із збільшенням кількості даних, що зберігаються в БД, виникає необхідність в розширенні за рахунок додавання нового атрибуту або відношення – це ми і називатимемо зростанням БД.

Реструктуризація даних має на увазі, що інформація залишається та ж, але змінюється її розташування, наприклад, за рахунок перегрупування атрибутів відношень. Припустимо, що відношення SP через які-небудь причини необхідно розділити на два – SPO і SPP, що приведено на рис. 5.1.

Зрозуміло, що об'єднання одержаних відношень відтворить відношення SP. Отже, для подальшої роботи можна створити уявлення SP, а значить у користувача створюється враження, що ніякі реструктуризації просто не проводилися:

```
СТВОРИТИ УЯВЛЕННЯ SP
SPP ОБ'ЄДНАТИ З SPO
```

Будь-яка призначена для користувача програма після цього замість відношення SP використовуватиме створене уявлення для маніпуляції із даними.

Крім розв'язання проблеми реструктуризації БД, уявлення можна використовувати для можливості перегляду одних і тих же даних різними користувачами, та ще в різних варіантах. Крім того, тепер користувачу можна не працювати із великим масивом даних, багато з яких в даний момент йому можуть бути і не потрібні. Шляхом використання уявлень, він має можливість обмежити об'єм даних для зручності роботи. Приклад такого уявлення для успішних студентів приведений вище.

Нарешті, використовуючи механізм уявлень можна приховати службові дані, які не цікаві користувачу. Так, номер студентського квитка, як правило, інтересу для користувача не представляє, проте він служить для забезпечення зв'язку відношень в БД. Можна створити уявлення, де замість номерів студентських квитків будуть підставлені прізвища студентів з іншого відношення. Очевидно, таке уявлення зручніше у використанні.

Часто уявлення використовується для вибірки даних. Про цю операцію ми вже говорили вище, проте має сенс зробити ряд додаткових зауважень. Механізм уявлень реалізується безпосередньо на відношеннях і зберігається у вигляді функції. Матеріалізація уявлення-вибірки здійснюється безпосередньо при реалізації відповідної дії користувача, до цього моменту матеріалізованих даних уявлення в БД не зберігається.

Проблема оновлення даних при роботі з уявленнями полягає в наступному: іноді важко внести в початкові відношення БД оновлені дані так, щоб в результаті одержати необхідне оновлення даних в уявленні.

Оновлюваність (або необновлюваність) уявлення не повинна залежати від його форми запису. Крім того, повинна дотримуватися симетричність оновлення, тобто виключена можливість неоднозначності способу реалізації даного оновлення. Так, скажімо, оновлення уявлення, побудованого на основі двох відношень, повинно проводитися так, щоб маніпуляція з даними здійснювалося в обох відношеннях. При цьому повинні враховуватися правила – тригери контролю цілісності даних. Ці тригери ніколи не запускаються всередині операції оновлення (UPDATE) з тієї причини, що під час оновлення взагалі кажучи може існувати порушення цілісності даних. Проте повністю виконана операція оновлення, як вже було сказано, цілісність даних порушувати не повинна.

Операція оновлення звичайно реалізується за допомогою дрібніших однотипних конструкцій типу ВИДАЛИТИ (DELETE) і ВСТАВИТИ (INSERT) на початкових відношеннях, після чого здійснюється перевірка цілісності. Очевидно, що використання і коректна робота цих операторів повинні бути можливі і в БД, яка не повністю

нормалізована, хоча в деяких випадках це може приводити до несподіваних для користувача результатів. Причому, крім цього, коректна робота повинна забезпечуватися як у разі роботи з одним, так і при роботі відразу із декількома кортежами.

Розглянемо виконання операції ВСТАВИТИ для уявлення вигляду А З'ЄДНАТИ Б, тобто для уявлення, одержаного за допомогою операції об'єднання. Необхідно мати на увазі, що новий (що вставляється) кортеж повинен задовольняти логічному виразу, що визначає наявність або відсутність кортежу хоч би в одному з початкових відношень (предикату). Якщо новий кортеж задовольняє предикату А, то він вставляється в А, якщо він задовольняє предикату Б – то в Б. Можлива ситуація, коли кортеж вставляється і в А і в Б. Вставку можна пояснити наступним прикладом на основі уявлень, приведених на рис. 5.1. Припустимо, що уявлення означене таким чином:

```
СТВОРИТИ УЯВЛЕННЯ VIEW2
SPP (ДЛЯ SPP.PN = "Фізика") З'ЄДНАТИ
                З SPO (ДЛЯ SPO.OCINKA>=3)
```

Внаслідок чого буде одержане уявлення VIEW2, приклад якого із значеннями приведений на рис. 5.2.

VIEW2			
SN	**PN**	**NAME**	**OCINKA**
3412	**Фізика**	**Іванюк**	**5**
3413	**Фізика**	**Василюк**	**4**
3414	**Фізика**	**Семенюк**	**3**

Рис. 5.2. Структура і приклад даних для уявлення VIEW2.

При спробі виконати вставку кортежу (3415, Математика, Сурядний, 5) маємо, що цей кортеж задовольняє предикату SPO.OCINKA>=3, але не задовольняє предикату SPP.PN = "Фізика". Значить, новий кортеж буде вставлений у відношення, що задовольняє першому предикату, тобто у відношення, що містить інформацію про оцінки.

Аналогічним чином можемо розглянути процес видалення кортежу з уявлення, одержаного об'єднанням: якщо кортеж, що видаляється, належить А, то він видаляється з А. Якщо після видалення кортежу з А цей кортеж має місце в Б, то він також віддаляється і з Б. Наприклад, при спробі видалити з VIEW2 кортеж (3413, Фізика, Семенюк, 3), видаляється кортеж з відношення, що задовольняє предикату SPP.PN = "Фізика", а потім – з відношення, що задовольняє предикату SPO.OCENKA>=3.

Процес оновлення відбувається так, щоб поточний кортеж задовольняв одному або обом предикатам. Наприклад, якщо оновлюваний кортеж належить А, то він видаляється з А. Якщо змінена версія кортежу задовольняє предикату А, то він вставляється в А. При цьому, якщо змінена версія кортежу задовольняє предикату Б, то він вставляється і в Б. Наприклад, при спробі оновлення кортежу (3413, Фізика, Василюк, 4) на (3413, Фізика, Семенюк, 5) оновлення буде зроблено в обох відношеннях.

Тепер більш стисло розглянемо виконання аналогічних операцій для уявлення вигляду А ПЕРЕТИНАЄ Б, тобто для уявлення, одержаного як результат виконання операції перетину. При вставці новий кортеж повинен задовольняти предикату А і предикату Б. Якщо новий кортеж не входить в А, то той кортеж вставляється в А. Якщо новий кортеж не входить в Б, то він буде вставлений і в Б.

При видаленні кортежу з уявлення, він видаляється з А, якщо він не видалений з Б, то він видаляється і з Б. При виконанні операції оновлення, змінений кортеж повинен задовольняти предикату А і предикату Б. Кортеж видаляється з А і з Б. Потім, якщо оновлений кортеж відсутній в А, він вставляється в А, якщо він відсутній в Б, то вставляється і в Б.

Якщо ж уявлення одержане за допомогою операції віднімання, тобто А ВІДНІМАЄТЬСЯ ІЗ Би, то виконання операцій відбувається таким чином. При вставці новий кортеж повинен задовольняти предикату А і не повинен задовольняти предикату Б. Отже, новий кортеж вставляється в А. При видаленні кортежу, останній видаляється із А. У випадку оновлення кортеж повинен задовольняти предикату А і не задовольняти предикату Б. Кортеж видаляється із А, а потім його оновлений варіант вставляється в А.

Уявлення, отримане як вибірка, може бути реалізоване шляхом виконання операції А ДЛЯ <умова>. Таким чином, дані, що знаходяться в такому уявленні, повинні задовольняти як предикату А, так і умові. При виконанні операції вставки новий кортеж вставляється в А. При виконанні видалення кортеж видаляється із А. При виконанні оновлення кортеж видаляється із А. Обновлена версія, що задовольняє предикату А і умові, вставляється в А.

Припустимо, що уявлення означене таким чином:

```
СТВОРИТИ УЯВЛЕННЯ VIEW3
SPP ДЛЯ (SPP.PN = "Фізика")
```

В результаті виконання отримаємо уявлення VIEW3, приклад якого із значеннями приведений на рис. 5.3.

VIEW3		
SN	**PN**	**NAME**
3412	**Фізика**	**Іванюк**
3413	**Фізика**	**Василюк**
3414	**Фізика**	**Семенюк**

Рис. 5.3. Структура і приклад даних для уявлення VIEW3.

Спроба здійснити вставку в таке представлення кортежу (3416, Фізика, Бичкова) закінчиться успішно, а кортежу (3417, Математика, Котенко) – невдачею, оскільки останній кортеж не задовольняє умові. Спроба відновити кортеж (3412, Фізика, Іванюк) (3412, Хімія, Іванюк) закінчиться невдачею, оскільки оновлений кортеж не задовольняє умові.

Для уявлення, одержаного за допомогою проекції, слід мати на увазі, що кортеж буде включений тільки в тому випадку, якщо у відповідного атрибуту значення кортежу входить в задану множину А. Ця умова буде предикатом. При спробі вставити кортеж оновлення здійснюватиметься для відношення А в тому випадку, якщо кортеж задовольняє предикату. При спробі видалення з А видаляються всі кортежі, в яких значення співпадає із значенням, що видаляється. Нарешті, при виконанні оновлення кортежу спочатку з відношення А видаляються кортежі, подібні оновлюваним, а потім в А будуть вставлені оновлені варіанти кортежу, що задовольняють предикату А.

Як приклад розглянемо уявлення, отримане як

```
СТВОРИТИ УЯВЛЕННЯ VIEW4 (SPP.SN, SPP.PN)
```

В результаті виконання отримаємо дані, приклад яких приведений на рис. 5.4.

VIEW4	
SN	**PN**
3412	**Фізика**
3413	**Хімія**
3414	**Фізика**

Рис. 5.4. Структура і приклад даних для уявлення VIEW4.

Вставка в VIEW4 кортеж (3419, Математика) закінчиться успішно. В результаті цього у відношення SPP буде вставлений кортеж (3419, Математика, X), де X – значення атрибуту NAME, використовуване за замовчуванням. В той же час, спроба вставити кортеж (3412, Хімія) закінчиться невдачею, оскільки порушується умова унікальності ключа SN. Спроба обновити кортеж (3412, Фізика) на (3412, Математика) приведе до того, що успішно відбудеться зміна кортежу (3412, Фізика, Іванюк) на (3412, Математика, Іванюк) відносно SPP (дані узяті згідно рис. 5.3).

Розглянемо тепер уявлення, отримане як розширення відношення А. Таке уявлення можна отримати як

```
СТВОРИТИ ПРЕДСТАВЛЕННЯ VIEW ДОДАТИ ДО А
                              <вираз> ЯК X
```

Предикат має такий вигляд, що в розширеному варіанті кожен кортеж уявлення а є породженням кортежу відношення за допомогою проекції, що виключає значення X від кортежу а з кортежу відношення, і, крім того, задовольняє предикату А. Правило оновлення такого уявлення при вставці полягає в тому, що кортеж, що вставляється, повинен задовольняти предикату уявлення. При цьому у відношення А вставляється кортеж, що виключає значення X від кортежу відношення А. При видаленні з А видаляється кортеж, що виключає значення X від кортежу відношення А. Нарешті, при оновленні спочатку з А видаляється кортеж за правилом, приведеним вище. Потім в А буде вставлений кортеж, що задовольняє предикату уявлення, причому з модифікованого кортежу виключається значення X від кортежу відношення А.

Розглянемо тепер уявлення, отримане як розширення відношення А. Таке уявлення можна отримати як

```
СТВОРИТИ УЯВЛЕННЯ VIEW5
ДОДАТИ ДО SPO (ЯКЩО OCINKA>=3 TO
          "Залік" ІНАКШЕ "Не зараховано") ЯК ZALIK
```

В результаті отримаємо уявлення, приведене на рис. 5.5.

VIEW5		
SN	**OCINKA**	**ZALIK**
3412	**5**	**Залік**
3413	**2**	**Не зараховано**
3414	**3**	**Залік**

Рис. 5.5. Структура і приклад даних для уявлення VIEW5.

Вставка в V1EW5 кортежу (3419, 4, Залік) буде завершена успішно і приведе до додавання у відношення SPO кортежу (3419, 4). Невдачу потерпить спроба вставки кортежу (3418, 2, Залік), як та, що не задовольняє предикату. Оновлення кортежу (3414,

3, Залік) на (3414, 4, Залік) завершиться успішно і відносно SPO оновиться кортеж (3414, 3) на (3414, 4).

І, нарешті, розглянемо уявлення, одержане як результат операції з'єднання, тобто типу

```
СТВОРИТИ УЯВЛЕННЯ VIEW А ПРИЄДНАТИ Б
```

Предикатом такого уявлення буде як би складений предикат А, для частини уявлення, отриманого від відношення А, і предикат Б, для частини уявлення, отриманого із Б. При спробі вставки новий кортеж повинен задовольняти предикату уявлення. Якщо частина уявлення, отримана від відношення А, в А відсутня, то ця частина вставляється в А. Якщо частина уявлення, отримана від відношення Б, у Б відсутня, то ця частина вставляється в Б. При видаленні кортежу видаляються відповідні частини уявлення з відношень А і Б. І, нарешті, при оновленні кортежу, його оновлена версія повинна задовольняти предикату уявлення. З відношення А видаляється частина кортежу, одержана з А, з відношення Б – одержана з Б. Після цього здійснюється вставка оновленого кортежу за правилом, викладеним вище.

Розглянемо приклад уявлення, отриманого з'єднанням відношень SPP і SPO:

```
СТВОРИТИ УЯВЛЕННЯ VIEW6 SPP ПРИЄДНАТИ SPO
```

Як результат отримаємо уявлення, приведене на рис. 5.6.

VIEW6			
SN	**PN**	**NAME**	**OCINKA**
3412	**Фізика**	**Іванюк**	**5**
3413	**Хімія**	**Василюк**	**4**
3414	**Фізика**	**Семенюк**	**3**
3412	**Математика**	**Іванюк**	**4**

Рис. 5.6. Структура і приклад даних для уявлення VIEW6.

Приведемо декілька прикладів роботи з цим уявленням. Вставка в VIEW6 кортежу (3412, Математика, Іванюк, 5) буде завершена успішно, при цьому у відношення SPO буде доданий кортеж (3412, 5). Видалення кортежу (3413, Хімія, Василюк, 4) буде завершено успішно, з відношення SPP буде видалений кортеж (3413, Хімія, Василюк) і із відношення SPO – (3413, 4). Видалення кортежу (3412, Математика, Іванюк, 4) завершиться успішно, при цьому із відношення SPO буде видалений кортеж (3412, 4). Спроба обновити кортеж (3414, Фізика, Семенюк, 3) на (3414, Фізика, Семенюк, 5) завершиться успішно, а відносно SPO кортеж (3414, 3) буде замінений кортежем (3414, 5).

Таким чином, розглянутий механізм використання уявлень, отриманих різними способами, і правил їх оновлення дозволяє розширити можливості СУБД при роботі в першу чергу в розрахованих на багато користувачів системах. Близькі до уявлень так звані знімки, які фіксують стан БД в певний момент часу – їх часто використовують в системах, де потрібні дані на заданий момент часу.

1.5.2. Підвищення продуктивності за допомогою оптимізації

Проблема оптимізації структури БД пов'язана з проектуванням концептуальної логічної схеми даних, проте грамотно побудована БД дозволяє підвищити продуктивність всієї системи. Інша сторона цієї проблеми пов'язана з тим, що деякі СУБД для нормального функціонування вимагають оптимізації.

Багато сучасних СУБД володіють вбудованими системами автоматичної оптимізації: зокрема, це виражається в тому, що кінцевому користувачу вже не потрібно замислюватися над оптимальним виразом запитів – в цьому випадку сама СУБД побудує запит найкращим чином.

Кажучи про оптимізацію запитів в реляційних СУБД, звичайно мають на увазі такий спосіб обробки запитів, коли за початковим уявленням запиту, шляхом його перетворення, виробляється якийсь план його виконання, найбільш прийнятний при існуючих в БД структурах, що управляють. Відповідні перетворення початкового представлення запиту виконуються спеціальним компонентом СУБД – оптимізатором, і оптимальність вироблюваного ним плану запиту носить умовний характер: план оптимальний відповідно до критеріїв, закладених в оптимізатор. Можна уявити, що обробка запиту, що поступив в систему, складається з фаз, зображених на рис. 5.7.

На першому етапі запит, заданий на мові запитів, піддається лексичному і синтаксичному аналізу. При цьому виробляється його внутрішнє уявлення, що відображає структуру запиту і що містить інформацію, яка характеризує об'єкти БД, згадані в запиті (відношення, атрибути і константи). Інформація про об'єкти, що зберігаються в БД, вибирається з відповідних каталогів. Внутрішнє представлення запиту використовується і перетворюється на наступних стадіях обробки запиту, оскільки форма внутрішнього уявлення повинна бути достатньо зручною для подальших оптимізаційних перетворень.

На другому етапі запит у внутрішньому уявленні піддається логічній оптимізації. Можуть застосовуватися різні перетворення, "поліпшуючі" початкове представлення запиту. Серед перетворень можуть бути еквівалентні, після проведення яких виходить внутрішнє уявлення, семантично еквівалентне початковому, наприклад, приведення запиту до деякої канонічної форми. Перетворення можуть бути і семантичними: одержуване уявлення не є семантично еквівалентним початковому, але гарантується, що результат виконання перетвореного запиту співпадає з результатом запиту в початковій формі при дотриманні обмежень цілісності, що існують в БД. Після виконання другого етапу обробки запиту його внутрішнє уявлення стає ефективнішим, ніж початкове.

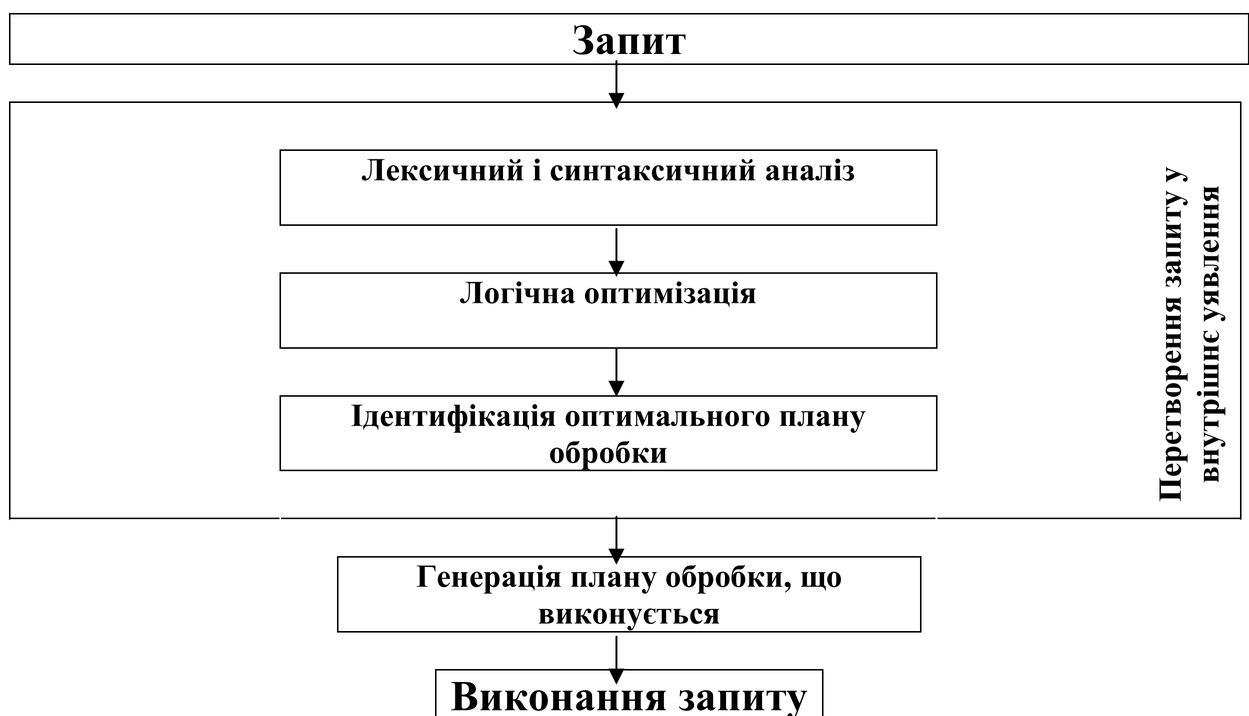


Рис. 5.7. Структура обробки запиту в СУБД.

Третій етап обробки запиту полягає в тому, що реалізується вибір альтернативних процедурних планів виконання даного запиту відповідно до його внутрішнього уявленням на основі інформації, яку має в своєму розпорядженні оптимізатор. При цьому для кожного плану оцінюється передбачувана тривалість і швидкість виконання запиту на основі статистичної інформації про стан БД. З одержаних альтернативних планів вибирається найбільш вигідний, і його внутрішнє уявлення тепер відповідає оброблюваному запиту.

На четвертому етапі за внутрішнім уявленням найбільш оптимального плану виконання запиту формується виконуване представлення плану. Виконуване представлення плану може бути, наприклад, програмою в машинних кодах, або бути машинно-незалежним. Проте, варто відмітити, що четвертий етап обробки запиту вже не пов'язаний з оптимізацією.

На останньому етапі обробки запиту відбувається його реальне виконання. Це або виконання відповідної підпрограми, або виклик інтерпретатора з передачею йому для інтерпретації виробленого оптимального плану.

У традиційному підході до організації оптимізаторів запитів на етапі логічної оптимізації проводяться еквівалентні перетворення внутрішнього уявлення запиту, які "покращують" початкове внутрішнє уявлення відповідно до стратегій оптимізатора. Характер "покращань" пов'язаний із специфікою загальної організації оптимізатора, зокрема, із особливостями процедури пошуку можливих процедурних планів запитів, виконуваної на третьому етапі обробки запиту. Саме з цієї причини важко привести повну характеристику і класифікацію методів логічної оптимізації, тому обмежимося декількома прикладами.

Очевидний клас логічних перетворень запиту складають перетворення предикатів, що входять в умову вибірки, до канонічного уявлення. Маються на увазі предикати, що містять операції порівняння простих значень. Такий предикат має вигляд

```
арифметичний_вираз1 OPERATOR
                  арифметичний_вираз2,
```

де OPERATOR – якась операція порівняння; арифметичний_вираз1 і арифметичний_вираз2 – вирази, що містять імена атрибутів відношень і константи.

Канонічні уявлення можуть бути різними для предикатів різних типів. Якщо предикат включає тільки одне ім'я атрибуту, то його канонічне уявлення може, наприклад, мати вигляд

```
ім’я_атрибуту OPERATOR константа
```

Ця форма предиката дуже корисна при виконанні наступного етапу оптимізації.

Якщо предикат включає два імена атрибутів різних відношень, то його канонічне уявлення має вигляд

```
ім’я_атрибута OPERATOR
            арифметичний_вираз
```

де арифметичний_вираз – включає тільки константи і ім'я другого атрибуту.

І, нарешті, для даних предикатів більш загального вигляду має сенс приведення предиката до канонічного представлення вигляду

```
арифметичний_вираз OPERATOR
      константний_арифметичний_вираз
```

Тут вирази в правій і лівій частині також повинні бути приведені до канонічного уявлення, наприклад, у виразах повинні бути повністю розкриті дужки. Надалі можна провести пошук загальних арифметичних виразів в різних предикатах запиту. Це виправдано, оскільки при виконанні запиту обчислення арифметичних виразів проводитиметься при вибірці кожного чергового кортежу, тобто потенційно велике число раз. Крім того, при приведенні предикатів до канонічного уявлення бажано обчислювати константні вирази і позбавлятися від логічних заперечень.

Наступний клас логічних перетворень пов'язаний з приведенням до канонічного вигляду логічного виразу, що задає умову вибірки запиту. Як правило, використовуються або диз'юнктивна, або кон'юнктивна нормальні форми. Вибір канонічної форми залежить від загальної організації оптимізатора.

При приведенні логічної умови до канонічного уявлення проводять пошук загальних предикатів і спрощують логічний вираз за рахунок, наприклад, виявлення кон'юнкції суперечних предикатів. Такі спрощення можуть виявитися істотними для подальшої обробки запиту, що прискорює його виконання.

Необхідно звернути увагу і на той факт, що в традиційних оптимізаторах поширені логічні перетворення, пов'язані із зміною порядку виконання реляційних операцій. Хоча небагато реляційних систем мають мови запитів, що базуються в чистому вигляді на реляційній алгебрі, правила перетворень алгебраїчних виразів, можуть бути корисні і в інших випадках. Досить часто реляційна алгебра використовується як основа внутрішнього представлення запиту, як більш універсальний і простий засіб. Крім того, перетворення запиту на SQL до алгебраїчного уявлення, скорочує простір пошуку планів виконання запиту з гарантією того, що оптимальні плани не будуть втрачені.

При цьому можуть виникнути проблеми, пов'язані із приведенням запитів з вкладеними запитами. Основною відмінністю мови SQL від мови реляційної алгебри є можливість використовувати в логічній умові вибірки предикатів, що містять вкладені підзапити. Глибина вкладеності не обмежується мовою, тобто, взагалі кажучи, може бути довільною. В цьому випадку прагнуть до такого перетворення запиту, що містить предикати із вкладеними підзапитами, яке зробить структуру підзапиту більш явною, надавши тим самим надалі оптимізатору можливість вибрати найбільш швидкий спосіб виконання запиту.

Тому в загальному випадку канонічним уявленням запиту на n відношеннях називається запит, що містить n-1 предикат з'єднання і що не містить предикатів із вкладеними підзапитами. Для пояснення цього пропонуємо наступний приклад, заснований на відношеннях, приведених на рис. 5.1:

```
ВИБРАТИ SPO.OCINKA IЗ SPO ДЛЯ SPO.SN(ВИБРАТИ
                     SPP.PN IЗ SPP ДЛЯ SPP.PN="Физика")
```

перетвориться в еквівалент

```
ВИБРАТИ SPO.OCINKA IЗ SPO, SPP ДЛЯ SPO.SN =
                     SPP.PN I SPP.PN="Физика"
```

Важливість таких перетворень обґрунтовується тим, що оптимізатор дістає можливість вибору із більшого числа способів виконання запитів і способи їх виконання ефективніші. При використанні в оптимізаторі запитів подібного підходу не обов'язково проводити формальні перетворення запитів – оптимізатор повинен більшою мірою використовувати семантику оброблюваного запиту, а яким чином вона розпізнаватиметься – це питання самої СУБД.

Розглянуті перетворення запитів ґрунтувалися на семантиці мови запитів, але в них не використовувалася семантика БД, до якої адресується запит. Будь-яке перетворення може бути проведене незалежно від того, яка конкретна БД використовується. Насправді ж, при кожній істинно реляційній БД зберігається і деяка семантична інформація, наприклад, що визначає цілісність даних, що було розглянуто вище.

Семантична оптимізація запитів заснована на наявності в БД семантичної інформації, яку не обов'язково використовувати при обробці запиту, але використання якої може привести до його оптимального виконання. Якщо семантична оптимізація має справу тільки із знаннями, представленими у вигляді обмежень цілісності БД, то при семантичній оптимізації проводяться перетворення внутрішніх уявлень запиту, і при кожному перетворенні використовується деякий набір обмежень цілісності. Пояснимо це прикладом.

Нагадаємо вже розглянуті вище обмеження цілісності для відношення SP, приведеного на рис. 3.1, яке сформульоване як

```
СТВОРИТИ ОБМЕЖЕННЯ ЦІЛІСНОСТІ RULE
ДЛЯ ВСІХ SP(SP.OCINKA>0 I SP.OCINKA<6)
```

Якщо, наприклад, відносно SP визначені ці обмеження цілісності і обробляється запит з умовою вибірки PN=2101, тобто

```
ВИБРАТИ SP.OCINKA З SP ДЛЯ SP.PN=2101
```

то в ході семантичної оптимізації будуть отримані внутрішні уявлення, еквівалентні запитам із умовами, і запит прийме вигляд:

```
ВИБРАТИ SP.OCINKA З SP ДЛЯ SP.PN=2101 I
(SP.OCINKA>0 I SP.OCINKA<6)
```

Одержаний новий вираз запиту піддається повній подальшій обробці, включаючи логічну оптимізацію і вибір оптимального плану виконання; із одержаних планів вибирається найбільш дешевий, який і стає реальним планом виконання початкового запиту. Слід мати на увазі, що після перетворення запиту відповідно до деякого правила до одержаного уявлення може виявитися застосовним інше правило, тобто можливо поява циклів перетворень. Проблема побудови повного набору семантично еквівалентних уявлень запиту на основі заданого набору правил в загальному випадку є вельми складною. Точне розв'язання цієї проблеми може зажадати обчислювальних затрат, сумірних із затратами на виконання запиту по найбільш оптимальному плану. Тому часто необхідне застосування евристичних алгоритмів, що скорочують простір пошуку, тобто задаючих умову припинення генерації нових уявлень.

Таким чином, однією із сторін підвищення продуктивності БД є оптимізація запитів, що виконується самій СУБД шляхом їх декомпозиції і перетворення у внутрішнє уявлення у декілька етапів із оцінкою швидкості виконання.

1.5.3. Домени, відношення і типи даних

Вище вже йшла розмова про те, що домени і відношення не просто набір деяких даних, а структури, що мають строго певний характер (тип) даних. Це виражається, наприклад, в тому, що для виконання таких операцій над відношеннями, як порівняння або об'єднання, дані повинні не тільки належать одному домену, але і бути сумісні по типу.

Для пояснення розглянемо наступний приклад, заснований на відношеннях, приведених на рис. 5.8: користувача може здійснити запит, заснований на операції порівняння SPO.SN = SPP.SN, оскільки атрибут SN у відношеннях належить одному домену і має один тип. Проте запит, що припускає порівняння SPO.OCINKA = SPP.PN, не є коректним – атрибути належать різним доменам і СУБД не повинна його виконувати.

Іноді перевірку відповідності доменів необхідно відключати – це може бути пов'язано з особливостями предметної області, про які в БД інформація відсутня. Прикладом такого цілком коректного запиту (з предметної точки зору) може служити вибірка даних про студентів по номеру студентського квитка у відношенні, в якому зберігаються дані про заробітну платню співробітників: в принципі, студент може вчитися і працювати у Внзі. Проте в СУБД, для виконання такого запиту, система перевірки доменів повинна бути неактивна – адже табельний номер співробітника і номер студентського квитка належать різним доменам.

Більше того, цікаві моменти, пов'язані з перевіркою приналежності тому або іншому домену, що виникають і в простіших випадках, наприклад, при вибірці даних по умові SPO.OCINKA =5. Атрибут SPO.OCINKA належить домену оцінок, а 5 – це ціле число, тобто значення, взагалі кажучи, належать різним доменам. З цієї причини в СУБД в неявному вигляді відбувається перетворення даних одного типа до одного домену, а значить, в нашому прикладі ціле число перетвориться до домену оцінок і лише після цього відбуваються подальші дії. Для цих цілей в СУБД використовуються спеціальні функції приведення типів і при роботі з аналогічними виразами вони неявно використовуються системою для відміни перевірки доменів.

Таким чином, для спрощення роботи СУБД на дані накладають обмеження не на приналежність тому або іншому домену, а на приналежність відповідному типу даних. При цьому система або перевіряє відповідність типів, або шукає таку функцію перетворення, щоб дані стали такими, що належать одному типу.

При цьому виникає ще одна сторона проблеми використання типів даних і підтримки в повному розумінні поняття домену: при маніпуляціях із доменами ряд операцій мають сенс в предметній області, а деякі операції безглузді. Наприклад, вибірки даних за умовами SPP.PN = "Фізика" і SPP.PN > "Фізика" з урахуванням вищесказаного коректні, але другий випадок сенсу в предметній області не має.

Тому СУБД повинно бути відомо, які вирази і операції є коректними, а які ні, а множина всіх доменів повинна бути замкнутою, щоб система знала до якого домена віднести ті або інші дані. З цього виходить, що розглянута вище вимога скалярності даних в домені полягає в тому, що вони можуть мати достатньо складну структуру, проте система не повинна останню бачити.

З аналогічної точки зору розглянемо тепер відношення. Відмітимо, що в реляційній моделі відношення є як би конструкторами типів даних: адже при створенні відношення в першу чергу створюється складений тип даних, що є множиною кортежів, що визначається заголовком. Тому, наприклад, для порівняння або присвоєння необхідно, щоб відношення мали однакові набори імен атрибутів і, крім того, атрибути із ідентичними назвами були сумісні один з одним по типу.

Очевидно, що в результаті виконання маніпуляцій із відношеннями виникає проблема типу відношення-результату, тобто структури його заголовка. Вище були розглянуті питання, пов'язані з виконанням основних операцій над відношеннями, проте при викладі не враховувався той факт, що заголовок включає крім імені атрибуту ще і ім'я домена. З вищевикладеного виходить, що, скажімо, для операції порівняння атрибути повинні бути сумісні по типу, а приналежність одному домену не є обов'язковою, оскільки СУБД проводить неявне перетворення типів доменів.

Отже, операція з'єднання відношень A{A.X,A.Y} і B{B.Y,B.Z} є коректною, не дивлячись на те, що атрибути Y мають однакове ім'я і тип в обох відношеннях, але

визначені на різних доменах. При цьому виникає цікавий момент – в одержаному в результаті виконання операції у відношенні C{ C.X, C.Y, C.Z} неясно, на якому домені визначений атрибут Y. З сумісності по типу доменів відношень A і B, на яких визначений атрибут Y, випливає, що їх можна перетворити один до іншого, а в СУБД передбачають домени з великим пріоритетом.

Атрибути відношення, як уже згадувалося, складаються з кортежів – впорядкованих пар складеного типа, що включають ім'я атрибуту і його значення, а отже, що мають свій заголовок. Звідси випливає, що питання сумісності по типу торкаються і кортежів при виконанні таких операцій, як порівняння, присвоєння і т.д.

Тут необхідно ще раз зупинитися на питаннях, пов'язаних із спадкоємством типів, які вже були розглянуті вище. Ці питання тісно пов'язані з поняттями підтипу і супертипу. Тип X є підтипом типу Y, якщо кожен екземпляр типу X обов'язково належить типу Y. Відповідно, тип Y буде супертипом для типу X. Наприклад, тип СТУДЕНТ є підтипом типу ВЧИТЬСЯ, а тип УЧЕНЬ є супертипом для типу СТУДЕНТ.

Ці поняття важливі з наступної точки зору: якщо СУБД знає, що екземпляр типу X є підтипом типу Y, то властивості Y властиві X. У такому разі говорять, що тип X успадковує властивості типу Y, а користувач має можливість застосовувати дані типу X там, де можна застосовувати дані типу Y – це називають принципом підстановки. Зручність спадкоємства полягає і в тому, що для типу можна багато разів виконувати якусь дію, яка застосовна і для його підтипів без будь-якихдоопрацювань.

Використання цих принципів для доменів пов'язане з тим, що, не дивлячись на спадкоємство властивостей, внутрішня структура кожного типа, взагалі кажучи, може бути різна, але один тип можна привести до іншого. Наприклад, тип СТУДЕНТ можна привести до типу ВЧИТЬСЯ, але при зворотному перетворенні може виникнути втрата інформації – у типу СТУДЕНТ може існувати ряд специфічних властивостей (наприклад, НАМЕНУНЯ_ВНЗА), яких в супертипі може і не бути.

Крім того, дані домени можна представити у вигляді простої ієрархії типів, оскільки супертип є узагальнення своїх підтипів. Будь-який екземпляр ієрархії типів автоматично при створенні успадковує всі властивості свого супертипа, але завжди є можливість деякі з них на заданому рівні ієрархії перевизначити або додати нові, специфічні.

Можливість застосування якоїсь однієї функції до даних різного типу часто називають поліморфізмом. І, нарешті, оскільки спадкоємство застосовується до властивостей типів, то у випадку доменів в їх якості виступають оператори і функції –, порівняння і т.д. на відміну від відношень, де успадкованими властивостями є атрибути.

Для пояснення концепції спадкоємства типів стосовно відношень розглянемо приклад, приведений на рис. 5.8.

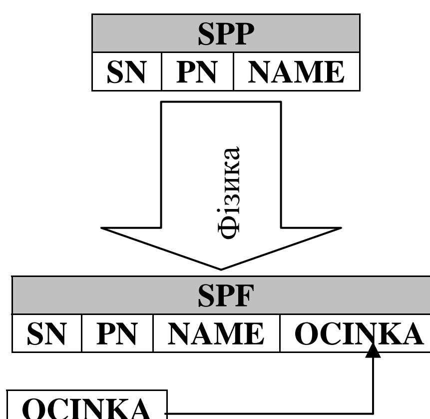


Рис. 5.8. Спадкоємство типів у відношеннях.

У прикладі розглянуте відношення SPP{SN, PN, NAME}, на основі якого створене відношення SPF{SN, PN, NAME, OCINKA}, що містить інформацію про успішність по фізиці. При цьому SPF від свого батька успадкувало атрибути SN, PN, NAME, а атрибут OCINKA доданий при створенні. Таким чином, одержане відношення SPF можна розглядати як підтип відношення SPP, власною властивістю (атрибутом) OCENKA, що володіє, причому над SPF можна виконувати ті ж операції і функції, що і над його супертипом.

1.5.4. Невизначені значення і тризначна логіка

При роботі з даними достатньо часто стикаються з проблемою недоліку інформації, для вирішення якої в СУБД часто застосовують невизначені (NULL) значення і тризначну логіку. Така ситуація, наприклад, може виникнути, якщо по предмету оцінка невідома або не виставлена – в цьому випадку точне обґрунтоване значення відповідного кортежу не вказується, хоча студент і предмет, який він здає, існують.

У зв'язку з цим може виникнути питання, пов'язане з тим, що в результаті виконання маніпуляції над даними один або декілька операндів мають значення NULL, а, отже, набуде невідомого значення результат. Тому, наприклад, при порівнянні X і Y, якщо значення X невизначене, буде набуте значення НЕВІДОМЕ, а не ІСТИНА або БРЕХНЯ. Більш того, якщо X і Y прийняли невизначене значення, то виконання порівняння X і Y також дасть як результат значення НЕВІДОМЕ, оскільки обидва значення не можна розглядати як рівні один одному.

Для ідентифікації подібних значень потрібні спеціальні засоби СУБД, найважливішим з яких є оператор ЯКЩО_НЕВИЗНАЧЕНЕ, що визначає, чи є значення-операнд невизначеним. До речі кажучи, тільки виконання цього оператора над невизначеними значеннями дає значення ІСТИНА або БРЕХНЯ; як результат виконання інших операторів буде набуте значення НЕВІДОМЕ.

Цю властивість можна використовувати, скажімо, для контролю цілісності даних, якщо значення атрибуту не повинне приймати невизначених значень, наприклад

```
СТВОРИТИ ОБМЕЖЕННЯ ЦІЛІСНОСТІ RULE6
ДЛЯ ВСІХ SPP(ЯКЩО_НЕВИЗНАЧЕНЕ(SPP.PN))
НЕ ВИКОНУВАТИ ПРИ ПОРУШЕННІ ПРАВИЛА
```

що дозволяє уникнути невизначених значень найменування предмету відносно SPP. Приведемо таблиці, що містять можливі значення найбільш важливих логічних функцій з урахуванням тризначної логіки (тут невідоме значення позначене як "Н" – очевидно, що невизначене значення NULL і невідоме значення, як результат виконання операції – це різні речі).

Таблиця 5.1. Можливі значення логічної функцій І.

І	ІСТИНА	БРЕХНЯ	NULL
ІСТИНА	ІСТИНА	БРЕХНЯ	Н
БРЕХНЯ	БРЕХНЯ	БРЕХНЯ	БРЕХНЯ
NULL	Н	БРЕХНЯ	Н

Таблиця 5.2. Можливі значення логічної функцій АБО.

АБО	ІСТИНА	БРЕХНЯ	Н
ІСТИНА	ІСТИНА	ІСТИНА	ІСТИНА
БРЕХНЯ	ІСТИНА	БРЕХНЯ	Н
NULL	ІСТИНА	Н	Н

Таблиця 5.3. Можливі значення логічної функцій НЕ.

НЕ	
ІСТИНА	БРЕХНЯ
БРЕХНЯ	ІСТИНА
NULL	Н

Необхідно звернути увагу на те, що домени невизначених значень містити не можуть, наприклад унаслідок того, що в результаті перевірки входження виразу в домен завжди виходитиме значення ІСТИНА або НЕВІДОМЕ.

Введення невизначеного значення, природно, впливає на реляційні вирази. Наприклад, операція вибірки модифікується так, щоб як результат поверталися тільки ті кортежі, для яких умова вибірки приймає тільки значення ІСТИНА, а для значень БРЕХНЯ або НЕВІДОМЕ не включаються.

1.5.5. Розподілені бази даних

Основне завдання систем управління розподіленими базами даних полягає в забезпеченні засобу інтеграції локальних БД, розташованих у вузлах обчислювальної мережі, з тим, щоб користувач, що працює в будь-якому вузлі мережі, мав доступ до всіх частин, як до єдиної БД. При цьому повинні забезпечуватися:

- простота використання системи;
- можливості автономного функціонування при на порушення цілісності мережі або при адміністративних потребах;
- високий ступінь ефективності.

Традиційно виділяють однорідні і неоднорідні розподілені БД. У однорідних розподілених БД кожна локальна БД управляється однією і тією ж СУБД. На відміну від цього, в неоднорідній системі локальні БД можуть відноситися до різних моделей даних, тому мережева інтеграція неоднорідних БД – це актуальна, але дуже складна проблема. Багато рішень відомі на теоретичному рівні, але поки не вдається справитися з головною проблемою - недостатньою ефективністю інтегрованих систем. Успішніше практично розв'язується проміжне завдання - інтеграція неоднорідних SQL-орієнтованих систем. Зрозуміло, що цьому у великій мірі сприяє стандартизація мови SQL і загальне проходження виробників СУБД принципам відкритих систем.

Необхідність в розподілених БД виникла у зв'язку з тим. що сучасні підприємства можуть географічно охоплювати великі території, причому робота з даними здійснюється в кожному підрозділі і на певному рівні. Розподіл даних дозволяє ефективно використовувати наявне устаткування, раціонально завантажити обчислювальну мережу і підвищити доступність БД.

Реалізація розподілених БД переслідує ряд цілей, перерахованих нижче. В першу чергу при проектуванні розподіленої СУБД вузли системи слід робити в максимальному ступені автономними, тобто робота кожного вузла як можна більшою мірою не повинна залежати від успішного або неуспішного виконання якої-небудь операції в іншому вузлі.

З іншого боку, всі вузли розподіленої БД повинні розглядатися системою як рівноправні. Залежність всієї системи від якого-небудь її вузла може привести до того, що він стане найуразливішим її місцем і, крім того, в цьому випадку навряд чи можна буде реалізувати умову автономності. Крім того, розподілена БД повинна забезпечувати високу надійність і доступність.

Надійність даної системи полягає в тому, що система справна і працює в заданий момент. Це реалізується за рахунок того, що система продовжує свою роботу (іноді і з обмеженими можливостями) навіть якщо має місце збій окремого вузла. Доступність

системи визначається справністю системи протягом тривалого проміжку часу і можливістю швидкого виконання маніпуляцій із даними.

Розподілена БД повинна задовольняти умові незалежності від розташування даних – користувачу необхідно забезпечити такий режим роботи з БД, щоб в нього складалося враження про нерозподілене розташування даних, що дозволяє, наприклад, спростити реалізацію призначених для користувача програм.

При використанні в розподіленій БД розділення відношень для зберігання, СУБД повинна забезпечити незалежність роботи з даними, не дивлячись на те, що фрагменти можуть бути розташовані в різних вузлах мережі. Крім цього, в системі повинна підтримуватися незалежність від реплікації, якщо задане відношення або його частина можуть бути представлені в БД декількома копіями (репліками), що зберігаються на різних вузлах.

І, нарешті, розподілена БД не повинна бути залежна від апаратного забезпечення, операційної системи, обчислювальної мережі і від самої СУБД.

Основною метою побудови розподіленої БД можна вважати забезпечення інтеграції локальних БД. Для вирішення цієї проблеми необхідно прийняти ряд проектних рішень, що стосуються декомпозиції початкового запиту, оптимального вибору способу виконання запиту, узгодженого виконання транзакцій, забезпечення синхронізації, виявлення і розв'язання розподіленої безвиході, відновлення стану БД після різного роду збоїв вузлів мережі.

Більшість сучасних СУБД автоматично виявляють поточне місцеположення згадуваних в запиті користувача об'єктів даних: одна і та ж прикладна програма, що включає пропозиції SQL, може бути виконана в різних вузлах мережі. Звичайно в кожному вузлі мережі на етапі компіляції запиту вибирається найбільш оптимальний план виконання запиту відповідно до розташування даних в розподіленій системі. Традиційно кожна локальна БД адмініструється незалежно від інших. При цьому можливі автономне підключення нових користувачів, зміна версії автономної частини системи і т.д.

Добре спроектована розподілена система функціонує таким чином, що в ній не потрібні централізовані служби іменування об'єктів або виявлення безвиході, а в окремих вузлах не потрібна наявність загальних знань про операції, що виконуються в інших вузлах мережі. Крім того, робота із доступними БД повинна продовжуватися при виході з ладу окремих вузлів мережі або ліній зв'язку.

Висока міра ефективності системи є однією з найбільш ключових вимог до розподілених СУБД, для досягнення чого можуть бути використані два основні прийоми.

По-перше, в СУБД, орієнтованих на розподілені дані, як правило, виконанню запиту передує його компіляція. В ході цього процесу проводиться пошук імен об'єктів БД, що використовуються в запиті, в розподіленому каталозі і заміна імен на внутрішні ідентифікатори; перевірка прав доступу користувача, від імені якого проводиться компіляція, на виконання відповідних операцій над даними і вибір найбільш оптимального глобального плану виконання запиту, який потім піддається декомпозиції і по частинах розсилається у відповідні вузли мережі. Після цього проводиться вибір оптимальних локальних планів виконання компонентів запиту, тобто більшість дій проводяться на стадії компіляції до реального виконання запиту. Оброблений таким чином запит може надалі виконуватися багато раз без додаткових затрат обчислювальної потужності і завантаження мережі.

Другим засобом підвищення ефективності системи є можливість переміщення віддалених відношень до локальної БД. Якщо цей засіб передбачений, то у ряді випадків він може допомогти добитися ефективнішого проходження транзакцій.

Крім того, добра система роботи із розподіленими БД повинна мати засоби дублювання відношень в декількох вузлах з підтримкою узгодженості копій і засобу підтримки миттєвих знімків стану БД відповідно до заданого запиту.

Іноді в розподілених БД використовують горизонтальне і вертикальне розділення відношень. У першому випадку відношення ділиться на частини по атрибутах, і одержані фрагменти зберігаються в різних вузлах мережі. Другий випадок пов'язаний з розділенням відношення не по атрибутах, а по групах кортежів, які теж зберігаються розподіленими по вузлах мережі.

Такий підхід дозволяє знизити апаратні вимоги до всієї системи, у ряді випадків збільшити швидкість виконання запитів і підвищити надійність зберігання даних, особливо при збої. Труднощі при цьому виникають в забезпеченні узгодженості отриманих розділів, а власне розділені відношення важко використовувати при виконанні запитів обліку наявності розділів відношення в різних вузлах мережі звичайно проводить оптимізатор, а отже, кількість потенційно можливих планів виконання запитів, які повинні оцінюватися оптимізатором, ще більш зростає.

Дня реалізації вимоги підтримки копій відношення в декількох вузлах мережі повинна проводитися розсилка копій вказаного відношення для зберігання в іменованих сегментах вказаних вузлів мережі. Система повинна автоматично підтримувати узгодженість копій. Як і у разі розділених відношень, крім суттєвих проблем підтримки узгодженості копій, проблемою є і розумне використання копій, наявність яких повинна була б враховуватися оптимізатором.

Створення миттєвого знімка стану БД відповідно до заданого запиту звичайно проводиться так, що результуюче відношення зберігається під вказаним користувачем ім'ям в локальній БД в тому вузлі, в якому виконується запит. Після цього миттєвий знімок періодично обновляється відповідно до запиту, що запам'ятований.

Основне використання миттєвих знімків пов'язане з тим, що їх можна в деякому розумінні розглядати як матеріалізовані представлення БД. Великі проблеми пов'язані з оновленням відношень через їх миттєві знімки, оскільки у момент оновлення вміст миттєвого знімка може розходитися з поточним вмістом базового відношення.

По відношенню до миттєвих знімків проблем підтримки узгодженого стану миттєвого знімка і базових відношень не існує, оскільки автоматичне узгодження не потрібне. Що ж до розділених відношень і копій відношень, то для них ця проблема спільна і достатньо важка. По-перше, узгодження розділів і копій викликає істотні обчислювальні затрати і завантаження мережі при виконанні операцій модифікації відношень, що зберігаються. Для цього потрібно вироблення і дотримання спеціальних протоколів модифікації. По-друге, введення копій відношень звичайно проводиться не стільки для збільшення ефективності системи, скільки для збільшення доступності даних при порушенні зв'язку в мережі. У системах, в яких застосовується цей підхід, при порушенні зв'язку в мережі робота з розподіленою БД звичайно продовжується тільки в одній з підмереж, при цьому для вибору підмережі використовуються алгоритми, засновані на обліку кількості зв'язаних вузлів мережі.

Як вже було сказано, для підвищення продуктивності в розподілених БД використовується попередня компіляція. Називатимемо головним вузлом той вузол мережі, в якому ініційований процес компіляції запиту, і додатковими вузлами – ті вузли, які залучаються до цього процесу в ході його виконання. Тоді процес компіляції можна розбити на ряд фаз.

У головному вузлі проводиться граматичний розбір запиту з побудовою внутрішнього уявлення у вигляді дерева. На основі інформації з локального каталогу головного вузла і видалених каталогів додаткових вузлів проводиться заміна імен об'єктів, що фігурують в запиті, на їх системні ідентифікатори. У головному вузлі генерується глобальний план виконання запиту, в якому враховується лише порядок взаємодій вузлів при реальному виконанні запиту. Якщо в глобальному плані

виконання запиту беруть участь додаткові вузли, проводиться його декомпозиція на частини, кожну з яких можна виконати в одному вузлі, після чого відповідні частини запиту розсилаються в додаткові вузли. У кожному вузлі, що бере участь у виконанні запиту, виконується завершальна стадія виконання компіляції, яка включає оптимізацію і генерацію машинних кодів. Потім проводиться перевірка прав користувача, від імені якого проводиться компіляція, на виконання відповідних дій і лише після цього відбувається обробка представлень БД.

Виконання транзакції в розподіленій СУБД починається в головному вузлі, проте виконання однієї транзакції, взагалі кажучи, ініціює транзакції і в додаткових вузлах. Основною тут є проблема узгодженого завершення розподіленої транзакції, щоб результати її виконання у всіх вузлах, що торкнулися нею, були або відображені в станах локальних БД, або повністю відсутні.

Для досягнення цієї мети звичайно використовується спеціальний механізм завершення розподіленої транзакції, що полягає в наступному: ряд незалежних транзакцій-учасників розподіленої транзакції виконуються під управлінням транзакції-координатора, якій ухвалюється рішення про закінчення розподіленої транзакції. Після цього виконується перша фаза завершення транзакції, коли координатор передає кожному з учасників повідомлення про підготовку до завершення. Одержавши таке повідомлення, кожен учасник переходить в стан готовності до негайного завершення транзакції або до її відкату. Після цього кожен учасник, що успішно виконав підготовчі дії, посилає координатору повідомлення про готовність до завершення. Якщо координатор одержує такі повідомлення від всіх учасників, то він починає другу фазу завершення, розсилаючи всім учасникам команду про завершення транзакції, і це вважається завершенням розподіленої транзакції. Якщо не всі учасники успішно виконали першу фазу, то координатор розсилає всім учасникам команду про відкат транзакції, і тоді ефект дії розподіленої транзакції на стан БД відсутній.

Тут може виникнути проблема розподіленої безвиході, яка може виникнути між декількома розподіленими транзакціями, що виконуються паралельно. Для виявлення розподіленої безвиході синхронізацій звичайно застосовуються методи, основна ідея яких полягає в тому, що в кожному вузлі періодично проводиться аналіз на предмет існування безвиході з використанням інформації про зв'язки транзакцій по очікуванню ресурсів, локальної в даному вузлі і одержаної від інших вузлів. При проведенні цього аналізу виявляються або цикли очікувань, що означає наявність безвиході, або потенційні цикли, які необхідно уточнити в інших вузлах. Якщо виявляється наявність синхронізаційної безвиході, він руйнується за рахунок відкату однієї з транзакцій, що входить в цикл, причому як остання, вибирається транзакція, що виконала до цього моменту найменший об'єм роботи.

Як вже було сказано вище, однією з вимог до розподіленої БД є незалежність від СУБД, проте деякі системи, з одного боку, не є повністю або частково сумісними одина з одною, а з іншою – достатньо часто виникає об'єктивна потреба в об'єднанні різних систем. У такому разі вдаються до допомоги спеціальних прикладних програм, які називаються шлюзами. Робота шлюзів полягає в тому, що за рахунок їх використання одна СУБД бачить роботу іншої в зрозумілому для неї вигляді, для чого використовуються спільні протоколи обміну інформацією, типи даних (або здійснюється перетворення одних типів даних в інші) і забезпечення спільної реалізації блокування, виконання транзакцій і т.д.

Стосовно СУБД архітектура клієнт/сервер цікава і актуальна головним чином тому, що забезпечує просте і відносно дешеве розв'язання проблеми колективного доступу до БД в локальній мережі. У деякому роді системи, засновані на архітектурі клієнт/сервер, є спрощеним наближенням до розподілених систем, що не вимагають розв'язання основного набору проблем дійсно розподілених БД. Реальне поширення

архітектури клієнт/сервер стало можливим завдяки розвитку і широкому впровадженню в практику концепції відкритих систем.

Основною ідеєю відкритих систем є спрощення сполучення обчислювальних систем за рахунок стандартизації апаратної і програмної частини. Головною причиною розвитку концепції відкритих систем з'явився перехід до використання локальних обчислювальних мереж і проблеми сполучення апаратно-програмних засобів, які викликав цей перехід. У зв'язку з бурхливим розвитком технологій глобальних комунікацій відкриті системи набувають ще більшого значення і масштабності.

Найважливішим моментом відкритих систем, направлених у бік користувачів, є незалежність від конкретного постачальника. Орієнтуючись на продукцію компаній, що дотримуються стандартів відкритих систем, споживач, що придбав будь-який продукт такої компанії, не потрапляє в залежність від неї. Він може продовжити нарощування потужності своєї системи шляхом придбання продуктів будь-якої іншої компанії, що дотримується стандартів, причому це стосується як апаратних, так і програмних засобів.

Практичною опорою системних і прикладних програмних засобів відкритих систем є стандартизована операційна система: в даний час такою системою є UNIX. Фірмам-постачальникам різних варіантів ОС UNIX в результаті тривалої роботи вдалося дійти до згоди про основні стандарти цієї операційної системи, і зараз всі поширені версії UNIX в основному сумісні по частині інтерфейсів, що надаються користувачам. Технології і стандарти відкритих систем забезпечують реальну і перевірену практикою можливість виробництва системних і прикладних програмних засобів при простоті перенесення програмної системи незалежно від апаратно-програмних засобів. Перш за все відкриті системи забезпечують природне розв'язання проблеми поколінь апаратних і програмних засобів – виробники таких засобів не повинні вирішувати всі проблеми заново; вони можуть принаймні тимчасово продовжувати концепцію сумісності системи, використовуючи існуючі компоненти.

Перевагою для користувачів таких систем є і те, що вони можуть поступово замінити компоненти системи на більш досконалі, не втрачаючи працездатності системи. Зокрема, в цьому криється розв'язання проблеми поступового нарощування обчислювальних, інформаційних і інших потужностей СУБД.

У основі широкого розповсюдження локальних мереж комп'ютерів лежить відома ідея розділення ресурсів. Висока пропускна спроможність локальних мереж забезпечує ефективний доступ з одного вузла локальної мережі до ресурсів, що знаходяться в інших вузлах. Розвиток цієї ідеї приводить до функціонального розділення компонентів мережі: користувач повинен мати не тільки доступ до ресурсів віддаленого комп'ютера, але також одержувати від цього комп'ютера деякий сервіс, який специфічний для ресурсів даного роду і для забезпечення якого недоцільно дублювати в декількох вузлах відповідні програмні засоби. Таким чином, приходять до диференціації робочих станцій і серверів локальної мережі.

Робоча станція призначена для безпосередньої роботи користувача або категорії користувачів і володіє ресурсами, які відповідають локальним потребам даного користувача. Специфічними особливостями робочої станції можуть бути невеликий об'єм оперативної пам'яті, наявність і об'єм дискової пам'яті, характеристики процесора і монітора і т.д. При необхідності можна використовувати ресурси і послуги, що надаються сервером.

Сервер локальної мережі повинен володіти ресурсами, які відповідають його функціональному призначенню і потребам мережі. Відмітимо, що у зв'язку з орієнтацією на підхід відкритих систем, правильніше говорити про логічні сервери, маючи на увазі набір ресурсів і програмних засобів, що забезпечують послуги над цими ресурсами, які розташовуються не обов'язково на різних комп'ютерах. Особливістю логічного сервера у відкритій системі є те, що якщо з міркувань ефективності сервер

доцільно розмістити на окремому комп'ютері, не потрібне яке-небудь доопрацювання використовуючих його прикладних програм і апаратного забезпечення.

Серверами можуть служити:

- обчислювальний сервер, що надає можливість проводити обчислення, які неможливо виконати на робочих станціях;
- сервер телекомунікацій, що забезпечує послуги із зв'язку даної локальної мережі із зовнішнім світом;
- дисковий сервер, що володіє могутніми ресурсами зовнішньої пам'яті і що надає їх у використання робочим станціями або іншим серверам;
- файловий сервер, що підтримує загальне сховище файлів для всіх робочих станцій;
- сервер БД, фактично звичайна СУБД, що приймає запити по локальній мережі і що повертає результати.

Очевидно, що для забезпечення взаємодії прикладної програми, що виконується на робочій станції, з сервером буде потрібно спеціальний програмний продукт, що виконує такого роду функції. З цього, власне, і випливають основні принципи системної архітектури клієнт/сервер.

При цьому вся система розбивається на дві частини, які можуть виконуватися в різних вузлах мережі – клієнтську і серверну частини. Прикладна програма або кінцевий користувач взаємодіють з клієнтською частиною системи, яка при необхідності звертається по мережі до серверної частини. Відмітимо, що в розвинених системах мережеве звернення до серверної частини може і не знадобитися, якщо система може передбачати потреби користувача і в клієнтській частині містяться дані, здатні задовольнити його наступний запит. Інтерфейс серверної частини визначений і фіксований, тому можливе створення нових клієнтських частин вже існуючої системи.

Основною проблемою систем, заснованих на архітектурі клієнт/сервер, є те, що відповідно до концепції відкритих систем від них потрібна мобільність в якомога ширшому класі апаратно-програмних рішень відкритих систем, проте спроби створення систем, що підтримують всі можливі протоколи, приводять до їх перевантаження мережевими деталями в шкоду функціональності. Ще складнішим моментом є можливість використання різних представлень даних в різних вузлах неоднорідної локальної мережі. оскільки в різних комп'ютерах може існувати різна адресація, представлення чисел, кодування символів і т.д.

Найбільш загальним розв'язанням проблеми мобільності систем, заснованих на архітектурі клієнт/сервер, є опора на програмні пакети, що реалізовують протоколи віддаленого виклику процедур. Використання таких засобів для звернення до сервісу у віддаленому вузлі виглядає як звичайний виклик процедури, тим самим приховуючи від користувача специфіку мережевого середовища і протоколів. При виклику віддаленої процедури такі програми проводять перетворення форматів даних клієнта в проміжні машинно-незалежні формати і потім перетворення у формати даних сервера, а при передачі параметрів відповіді проводяться аналогічні перетворення.

Термін сервер БД традиційно використовують для позначення всієї СУБД, що базується на архітектурі клієнт/сервер, включаючи як серверну, так і клієнтську частини. Такі системи призначені для зберігання і забезпечення доступу до БД. Звичайно одна БД цілком зберігається в одному вузлі мережі і підтримується одним сервером, а сервери БД є деяким наближенням до розподілених БД, оскільки спільні дані доступні для всіх користувачів локальної мережі.

Доступ до БД від прикладної програми або користувача проводиться шляхом звернення до клієнтської частини системи. Як основний інтерфейс між клієнтською і серверною частинами виступає мова баз даних SQL, яка по суті є стандартом інтерфейсу СУБД у відкритих системах.

Назва SQL-сервер відноситься до всіх серверів БД, заснованих на SQL. Сервери БД, інтерфейс яких заснований на мові SQL, володіють своїми перевагами і своїми недоліками.

Важливою перевагою є стандартність інтерфейсу. Взагалі кажучи, клієнтські частини будь-якої SQL-орієнтованої СУБД могли б працювати з будь-яким SQL-сервером незалежно від того, хто його виробив. Недолік полягає в наступному: при такому рівні інтерфейсу між клієнтською і серверною частинами системи на стороні клієнта працює дуже мало програм СУБД. Це нормально, якщо на стороні клієнта використовується малопотужна робоча станція, але якщо клієнтський комп'ютер володіє достатньою потужністю, то часто виникає бажання покласти на нього більше функцій управління БД, розвантаживши сервер, який, у свою чергу, є вузьким місцем всієї системи.

Одним з перспективних напрямів СУБД є гнучка конфігурація системи, при якій розподіл функцій між клієнтською і призначеною для користувача частинами СУБД визначається при установці системи.

Згадувані вище протоколи віддаленого виклику процедур особливо важливі в СУБД, заснованих на архітектурі клієнт/сервер. Це пов'язано з тим, що використання механізму віддалених процедур дозволяє дійсно перерозподіляти функції між клієнтською і серверною частинами системи, оскільки в тексті програми віддалений виклик процедури нічим не відрізняється від віддаленого виклику, і, отже, будь-який компонент системи може розташовуватися як на стороні сервера, так і на стороні клієнта.

Далі, механізм віддаленого виклику приховує відмінності між взаємодіючими комп'ютерами. Фізично неоднорідна локальна мережа комп'ютерів приводиться до логічно однорідної мережі взаємодіючих програмних компонентів, внаслідок чого користувачі не зобов'язані серйозно піклуватися про разову закупівлю сумісних серверів і робочих станцій.

У типовому на сьогодні випадку на стороні клієнта СУБД працює тільки таке програмне забезпечення, яке не має безпосереднього доступу до БД, а звертається до цього до сервера з використанням мови SQL. В деяких випадках хотілося б включити до складу клієнтської частини системи деякі функції для роботи з локальною частиною БД, тобто з тим її фрагментом, який інтенсивно використовується клієнтською прикладною програмою. У сучасній технології це можна зробити тільки шляхом формального створення на стороні клієнта локальної копії сервера БД і розгляду всієї системи як набору взаємодіючих серверів.

З іншого боку, іноді виникає необхідність в перенесенні більшої частини прикладної системи на сторону сервера, якщо різниця в потужності клієнтських робочих станцій і сервера занадто велика. Зробити це неважко, проте потрібно, щоб базове програмне забезпечення сервера дійсно дозволяло виконати такий перерозподіл. Зі всього цього виходить, що вимоги до апаратури і програмного забезпечення клієнтських і серверних комп'ютерів розрізняються залежно від виду використання системи. Якщо розділення між клієнтом і сервером досить жорстке, то користувачам, що працюють на робочих станціях, абсолютно все одно, яка апаратура і операційна система встановлені на сервері – лише б він справлявся із потоком запитів. Якщо ж виникають потреби в перерозподілі функцій між клієнтом і сервером, то виникає проблема, пов'язана з вибором апаратури і операційної системи, використовуваної в клієнтській частині.

Таким чином, розглянувши такий напрям розвитку СУБД, як розподілених систем, можна назвати його одним з найперспективніших.

1.6. Технологія фізичного зберігання і доступу до даних

1.6.1. Основні етапи доступу до бази даних

Кажучи про процес роботи із БД, не можна не зупинитися на питаннях зберігання і методів доступу до даних. Взагалі кажучи, основні проблеми, пов'язані з фізичним зберіганням даних, викликані повільністю доступу і пошуку, а також низькою швидкістю передачі, тому основною метою підвищення продуктивності системи з цієї точки зору є мінімізація числа дискових операцій вводу-виводу даних.

Для зберігання даних можуть бути використані різні структури, що володіють різною продуктивністю, проте ідеального способу зберігання даних не існує. З цієї причини СУБД повинна містити декілька структур зберігання даних для різних завдань і частин системи, а також передбачати можливість зміни способів зберігання залежно від вимог, що змінюються, до продуктивності системи.

Почнемо з опису основних етапів процесу доступу до БД. Спочатку СУБД визначає шуканий запис в БД, для чого в оперативну пам'ять поміщається набір записів, в якому шукається запрошений, а для видобування запису запрошується так званий диспетчер файлів. Диспетчер файлів визначає сторінку, на якій знаходиться шуканий запис, а потім для видобування цієї сторінки запрошується диспетчер дисків. Диспетчер дисків визначає фізичне розташування сторінки на пристрої зберігання інформації і посилає запит на ввід-вивід даних.

Таким чином, СУБД розглядає БД як безліч записів, що переглядаються за допомогою диспетчера файлів. Останній розглядає БД як набір сторінок, що переглядаються за допомогою диспетчера дисків, який вже безпосередньо працює із пристроями зберігання інформації.

Відмітимо, що диспетчер дисків є частиною операційної системи, за допомогою якого виконуються всі дискові операції вводу-виводу. Для того, щоб виконувати ці операції, диспетчеру необхідно володіти інформацією про значення фізичних адрес на диску, де розташовуються ті або інші дані. Проте диспетчеру файлів така інформація зовсім не потрібна – замість цього йому досить розглядати диск як набір сторінок строго фіксованого розміру з унікальним ідентифікаційним номером набору сторінок. У свою чергу, кожна сторінка, володіє унікальним всередині даного набору ідентифікаційним номером сторінки, причому набори не мають спільних сторінок. При цьому відповідність фізичних адрес на диску і номерів сторінок досягається за допомогою диспетчера дисків. Найважливішою перевагою такої організації зберігання даних є ізоляція програмного коду всередині диспетчера дисків, залежного від конкретного пристрою диска, за рахунок чого багато компонентів системи можуть бути апаратно незалежними.

Всі сторінки диска діляться на незв'язані набори, а один з таких наборів, що містить пусті сторінки, відповідно містить всі наявні вільні сторінки, що не використовувались для розміщення даних. Цей набір іноді називають вільним простором на диску. При цьому використання або звільнення сторінок з наборів сторінок здійснюється диспетчером позовів по запиту диспетчера файлів.

Основні операції, що виконуються диспетчером дисків із наборами сторінок за запитом з боку диспетчера файлів, наступні:

- витягнути сторінку s із набору сторінок n;
- замінити сторінку s із набору сторінок n;
- додати нову сторінку в набір сторінок n;
- видалити сторінку s із набору сторінок n.

При роботі із диском, як із набором файлів, що зберігаються, диспетчер файлів використовує всі наявні засоби диспетчера дисків, при цьому кожен набір сторінок може містити один або декілька файлів, що зберігаються.

Кожен файл, що зберігається, має унікальні в даному наборі сторінок ім'я або ідентифікаційний номер, а кожен запис, що зберігається, володіє ідентифікаційним номером запису, унікальним в межах даного файлу, що зберігається.

За допомогою операцій із файлами в СУБД можна створювати структури зберігання і керування, але слід мати на увазі, що в одних системах диспетчер файлів є компонентом операційної системи, а в інших є частиною СУБД, проте принципи його роботи суттєво від цього не відрізняються. Основні операції із файлами, що виконуються диспетчером файлів, за запитом з боку СУБД, наступні:

- витягнути запис z, що зберігається, із файлу f, що зберігається;
- замінити запис z, що зберігається, у файлі f, що зберігається;
- додати новий запис z, що зберігається, у файл f, що зберігається;
- видалити запис z, що зберігається, із файлу f, що зберігається;
- створити новий файл f, що зберігається;
- видалити файл f, що зберігається.

При зберіганні даних використовують принцип кластеризації даних, в основі якого знаходиться підхід якомога ближчого фізичного розміщення на диску логічно зв'язаних між собою і часто використовуваних даних. Фізична кластеризація даних достатньо важлива умова високої продуктивності, при цьому розрізняють внутрішньофайлову кластеризацію, коли вона здійснюється в рамках одного файлу, що зберігається. Наприклад, якщо в системі часто потрібно здійснювати доступ до даних згідно порядкового номеру, то всі записи слід фізично розміщувати так, щоб перший запис був біля другого запису, другий запис – біля третьої і т.д.

Інший варіант кластеризації – міжфайлова, коли нею охоплюється відразу декілька файлів. Це використовують, якщо в системі часто потрібно здійснювати доступ до якихось записів і до даних, пов'язаними з ними, при цьому перші прагнуть розмістити поряд з другими.

Зрозуміло, в кожен момент часу кластеризацію файлу або набору файлів можна здійснювати тільки одним з цих способів. Внутрішньофайлову і міжфайлову кластеризацію СУБД може здійснювати, розміщуючи логічно зв'язані записи на одній сторінці, якщо це можливо, або на сусідніх сторінках. З цієї причини СУБД важливо мати відомості не тільки про збережені файли, але і про сторінки: при створенні в СУБД нового запису необхідно, щоб вона за допомогою диспетчера файлів була розміщена біля деякої поточної сторінки, тобто на тій же або, принаймні, на логічно близькій сторінці. У свою чергу, диспетчер дисків повинен забезпечити, щоб логічно близькі сторінки були фізично близько розташовані на диску.

Звичайно, кластеризація всередині СУБД можлива тільки в тому випадку, якщо адміністратор БД організує її, при цьому часто передбачається використання декількох різних типів кластеризації даних із різних файлів, при необхідності вибираючи той або інший її тип.

1.6.2. Управління сторінками

Основною функцією диспетчера дисків є приховання від диспетчера файлів всіх деталей фізичних дискових операцій вводу-виводу і заміна їх логічними сторінковими операціями вводу-виводу. Ця функція диспетчера дисків називається управлінням сторінками. Прикладом такої операції може бути упорядкування в кожній використовуваній таблиці логічних записів згідно ключового поля.

На початковому етапі БД зовсім не містить даних, але в ній є набір порожніх сторінок, послідовно пронумерованих починаючи з номера один, в якому містяться всі сторінки диска, за винятком сторінки з нульовим номером, якій відводиться особлива роль. Для розміщення записів з даними X диспетчер файлів створить набір сторінок і розмістить на них дані X. З цією метою диспетчер дисків перемістить відповідну кількість сторінок з набору пустих сторінок і позначить їх як набір сторінок даних X.

Аналогічні дії будуть виконані для розміщення даних Y, Z і т. д., але для простоти обмежимося трьома наборами. В результаті буде створено чотири набори сторінок: дані X, дані Y, дані Z і набір порожніх сторінок.

Припустимо, що необхідно додати новий запис з даними X, для цього диспетчер файлів вставляє новий запис, що зберігається, а диспетчер дисків здійснює пошук першої порожньої сторінки, а потім додає її до набору сторінок даних X.

Якщо необхідно видалити запис, що зберігається, з даними X, диспетчер файлів видаляє її, а диспетчер дисків повертає сторінку даних X, що звільнилася, в набір порожніх сторінок.

Аналогічним способом вставляються або видаляються записи із наборів сторінок з даними Y або Z, з чого можна зробити висновок про те, що після виконання декількох найзвичайніших дій не можна гарантувати, що логічно близькі сторінки фізично розташовуватимуться одна біля іншої. Тому логічну послідовність сторінок в даному наборі слід задавати за допомогою покажчиків, а не на основі їх фізично близького розміщення на диску. Для цього кожна сторінка містить заголовок сторінки з інформацією про фізичну дискову адресу сторінки, яка логічно слідує за даною сторінкою. Особливості використання покажчиків наступні:

- заголовки сторінок, зокрема покажчики наступної сторінки, обробляються диспетчером дисків і повинні бути приховані для диспетчера файлів;
- бажано зберігати логічно зв'язані сторінки у фізично близьких фрагментах диска, тому диспетчер дисків звичайно розміщує або видаляє сторінки в наборах не по одній, а цілими блоками фізично зв'язаних сторінок;
- для отримання інформації про розміщення різних наборів сторінок диспетчером дисків, на диску організовується окрема сторінка, в якій зберігається вся необхідна для цього інформація. Ця сторінка часто називається таблицею розміщення або просто сторінкою 0, де перераховані всі наявні набори сторінок, що є на даному диску, разом з покажчиками на перші сторінки кожного з наборів.

Диспетчер дисків приховує особливості фізичної організації вводу-виводу від диспетчера файлів і надає йому можливість вести роботу тільки на логічному рівні. Аналогічно диспетчер файлів приховує всі подробиці операцій вводу-виводу на основі сторінок від СУБД і надає їй можливість вести роботу тільки із записами, що зберігаються, і файлами. Така робота, що виконується диспетчером файлів, називається управлінням записами, що зберігаються.

Припустимо, що на одній сторінці можуть бути розміщені не один, а декілька записів, що зберігаються. При вставці невеликої кількості записів, що зберігаються, наприклад, даних X, на відповідній сторінці тепер може залишатися досить вільного простору. Тому, якщо виникне необхідність вставити новий запис даних X, то диспетчер файлів збереже цей запис на тій же сторінці вслід за останнім збереженим записом.

Тепер, при виникненні необхідності видалити існуючий запис даних X, це виконає диспетчер файлів і, якщо утворився порожній проміжок між записами на сторінці, він пересуне записи до початку сторінки. Це говорить про те, що логічна послідовність записів, що зберігаються, для даної сторінки може відповідати фізичній послідовності, заданій всередині цієї сторінки. Для цього диспетчер файлів може пересувати окремі записи вгору або вниз, розміщуючи всі записи у верхній частині сторінки і залишаючи вільний простір в нижній частині сторінки. Більше того, якщо СУБД необхідно вставити новий запис на дану сторінку, то диспетчер файлів розмістить запис відповідно до її ідентифікаційного номера.

Отже, записи, що зберігаються, ідентифікуються за допомогою ідентифікаційного номера запису z, який складається з двох частин: номера сторінки s, на якій даний запис знаходиться, і інформації про зсув запису від кінця сторінки s.

Остання, у свою чергу, містить інформацію про зміщення запису z від початку сторінки s. Ця схема в деякій мірі поєднує швидкість безпосередньої адресації і гнучкість непрямої адресації, оскільки записи всередині сторінки можуть зсовуватися без зміни ідентифікаційних номерів записів, а коректуються тільки значення локальних зміщень в кінці сторінки. До того ж, якщо відомий ідентифікаційний номер записів, доступ до необхідного запису здійснюється достатньо швидко, оскільки використовується тільки доступ до даної сторінки.

Іноді для роботи з деяким записом може бути потрібно доступ до двох сторінок. Така ситуація може виникнути, наприклад, якщо довжина запису перевищує розмір сторінки. Тоді такий запис буде розміщений на спеціальній сторінці переповнювання, а початковий покажчик на ідентифікаційний номер колишнього запису буде замінений новим. При наступному переповнюванні сторінки запис знову переміщається на нову сторінку переповнювання з відповідною зміною покажчика.

Слід звернути увагу на те, що для деякого файлу, що зберігається, завжди можна здійснити послідовний доступ до всіх записів, що зберігаються, тобто доступ згідно послідовності записів всередині сторінки і послідовності сторінок всередині набору сторінок, найчастіше, у порядку зростання ідентифікаційних номерів записів. Така послідовність називається фізичною, хоча треба розуміти, що вона не обов'язково відповідає фізичному розташуванню даних на диску. Доступ до файлу, що зберігається, можна здійснити згідно фізичної послідовності, навіть якщо декілька файлів знаходяться на одній і тій же сторінці, при цьому записи, які не відносяться до шуканого файлу, будуть пропущені при послідовному перегляді змісту даної сторінки.
Крім того, поля запису, що зберігається, використовуються в СУБД для складання індексів і т.д., проте диспетчери файлів цю інформацію не використовують. Таким чином, важливою відмінністю між диспетчером файлів і СУБД є представлення запису, що зберігається: з погляду СУБД запис, що зберігається, володіє внутрішньою структурою, а з погляду диспетчера файлів це всього лише рядок байтів.

Проте існують більш досконалі способи впорядкування записів і способи доступу в порівнянні з фізичною послідовністю: використання індексування, хешування, ланцюжків покажчиків, а також технології стиснення. При цьому часто ці способи використовуються спільно один на основі іншого.

1.6.3. Процедура індексування і хешування

Для того, щоб детальніше розібратися з індексами, розглянемо як приклад таблицю з даними про оцінки і запит на пошук студентів, що здавали той або інший навчальний предмет. За таких умов в БД можна вибрати спосіб зберігання даних, схематично показаний на рис. 6.1.

Він заснований на двох файлах, що зберігаються: файлі з даними про успішність студентів USP і файлі з даними про навчальні предмети. Ці файли можуть розміщуватися в різних наборах сторінок, при цьому передбачається, що у файлі предметів використовується впорядкування по алфавітному переліку їх назв, тобто по ключовому полю PN з покажчиками на відповідні записи у файлі постачальників.

PN(індекс)
Алгебра
Фізика
Фізика
Хімія

USP			
SN	PN	NAME	OCINKA
3412	Фізика	Поляченко	5
3413	Хімія	Старченко	4
3414	Фізика	Грицюк	3
3415	Алгебра	Костенко	5

Рис. 6.1. Індексування файлу оцінок по полю PN.

Можливі наступні стратегії, які можна застосувати для пошуку всіх студентів, що здавали фізику:

- знайти весь файл успішності, знайти всі записи, для яких назвою дисципліни є рядок Фізика.
- знайти файл предметів з рядком Фізика, а потім згідно покажчика витягнути всі відповідні записи з файлу успішності.

Якщо доля всіх студентів, що склали фізику, по відношенню до загальної кількості студентів невелика, то друга стратегія буде набагато ефективніше першої. Річ у тому, що СУБД відома фізична послідовність записів у файлі предметів, а пошук буде припинений після видобування наступної за фізикою в алфавітному порядку назви предмету. Крім того, навіть якщо доведеться проглянути файл предметів повністю, для такого пошуку буде потрібно значно менше операцій вводу-виводу, оскільки фізичний розмір файлу предметів менший, ніж розмір файлу успішності через менший розмір записів.

У розглянутому прикладі файл предметів називають індексним файлом або індексом по відношенню до файлу успішності, або навпаки – файл успішності індексований по відношенню до файлу предметів. Індексний файл є файлом особливого типу, в якому кожен запис складається з двох значень, що зберігається: даних і покажчика номера запису. При цьому дані необхідні для індексного поля з індексованого файлу, а покажчик – для скріплення з відповідним записом індексованого файлу.

Якщо індексування організоване на основі ключового поля, наприклад, на основі поля SN файлу успішності, то індекс називається первинним. А якщо індекс організований на основі іншого поля, наприклад, поля PN, то він називається вторинним.

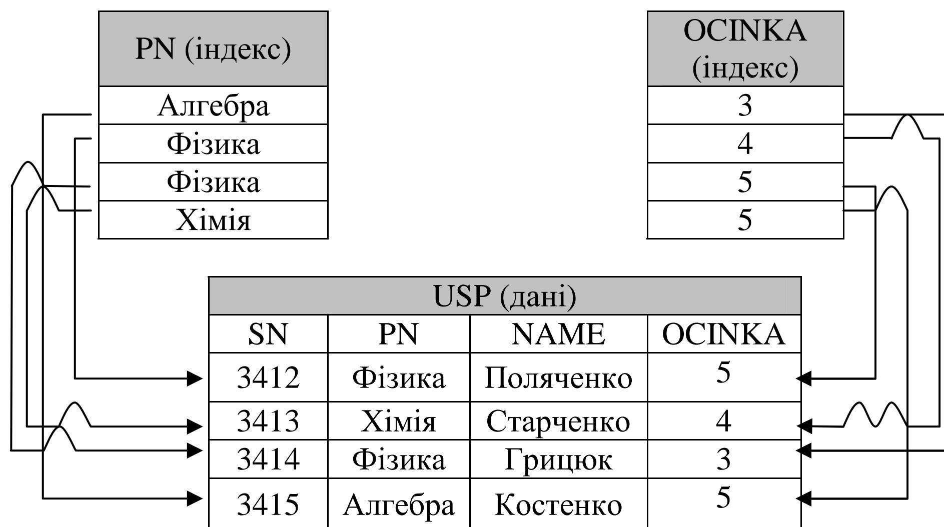


Рис. 6.2. Індексування файлу оцінок по полю PN і OCINKA.

Крім того, індекс, організований на основі ключового поля або іншого ключа, називається унікальним.

Основною перевагою використання індексів є значне прискорення процесу вибірки або видобування даних, а основним недоліком – уповільнення процесу оновлення даних, оскільки при кожному додаванні нового запису в індексований файл потрібно буде також додати новий індекс в індексний файл. Індекси можна використовувати двома різними способами:

- для послідовного доступу до індексованого файлу, тобто в послідовності, заданій значеннями індексного поля. Наприклад, індекс PN визначатиме доступ до

записів файлу успішності згідно алфавітного переліку предметів;

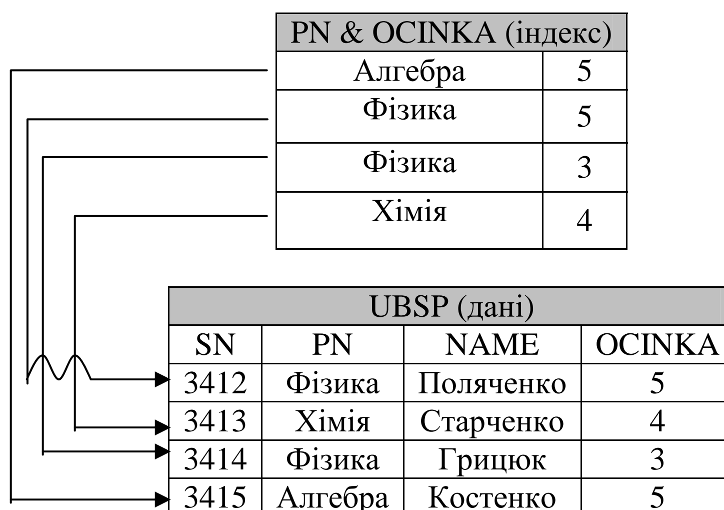
- індекси можуть використовуватися і для прямого доступу до окремих записів індексованого файлу на основі заданого значення індексного поля, як це було зроблено в приведеному прикладі.

Файл, що зберігається, може мати декілька індексів: наприклад, файл успішності, що зберігається, може мати індекс PN і індекс OCINKA(рис. 6.2).

Індекси можуть як роздільно, так і спільно використовуватися для ефективнішого доступу до даних про успішність, наприклад, при запиті на пошук студентів, що здали фізику на 5.

Тоді згідно індексу PN для студентів будуть знайдені записи з ідентифікаційними покажчиками z3412 і z3414, а згідно індексу OCINKA – записи з покажчиками z3412 і г3415. Зрозуміло, що на основі порівняння цих двох наборів записів умовам запиту задовольняє тільки запис з даними про студента z3412 і лише після цього в СУБД буде організований доступ до файлу успішності і буде витягнутий даний запис.

Часто індекс створюють на основі комбінації двох або більш полів. Наприклад, на рис. 6.3 показана схема індексування файлу успішності на основі комбінації полів PN і OCINKA. При такій організації індексів в СУБД можна виконати запит на пошук студентів, що здали фізику на 5 на основі одноразового перегляду за допомогою одного індексу, а, як було показано вище, при використанні пари індексів потрібно два окремі перегляди, тим паче, що швидкість виконання запиту може сильно залежати від послідовності виконання окремих переглядів по індексах.



PN & OCINKA (індекс)	
Алгебра	5
Фізика	5
Фізика	3
Хімія	4

UBSP (дані)			
SN	PN	NAME	OCINKA
3412	Фізика	Поляченко	5
3413	Хімія	Старченко	4
3414	Фізика	Грицюк	3
3415	Алгебра	Костенко	5

Рис. 6.3. Індексування файлу оцінок по комбінації полів PN і OCINKA.

Варто звернути увагу на той факт, що комбінований індекс PN&OCENKA може також служити індексом по одному полю PN, оскільки всі записи в комбінованому індексі розташовані послідовно. Взагалі кажучи, індекс на основі комбінації полів може використовуватися або для окремого індексування по конкретному полю, або для індексування на основі їх комбінації, отже, для створення досить складного індексування може бути потрібно зовсім небагато індексних полів.

Отже, основною метою використання індексу є прискорення процесу видобування даних, за рахунок зменшення числа дискових операцій вводу-виводу, для чого використовуються покажчики. Припустимо, що фізична послідовність файлу успішності відповідає логічній послідовності, заданій на основі номера студентського квитка, тобто в цьому файлі виконана внутрішньофайлова кластеризація по даному полю. Якщо по цьому ж полю здійснюється індексування, то немає необхідності в індексі зберігати покажчики для кожного запису індексованого файлу. Досить вказати

для кожної сторінки максимальний номер постачальника на ній і відповідний номер сторінки.

Індекс із описаною структурою називається нещільним, оскільки в ньому не містяться покажчики на всі записи індексованого файлу. Однією з переваг нещільних індексів є їх малий розмір в порівнянні з щільними індексами, описаними вище, оскільки вони містять менше число записів. Це часто дозволяє проглядати вміст БД з більшою швидкістю, та зате не можна виконати перевірку наявності деякого значення.

Іншим не менш цікавим способом впорядкування записів і доступу є хешування. Хешуванням або хеш-індексуванням називається технологія швидкого прямого доступу до запису, що зберігається, на основі заданого значення деякого поля, при цьому зовсім не обов'язково, щоб поле було ключовим.

Основні особливості цієї технології наступні:

- кожен запис БД, що зберігається, розміщується за адресою, яка обчислюється за допомогою спеціальної хеш-функції на основі значення деякого поля даного запису, тобто хеш-поля, а обчислена адреса називається хеш-адресою.
- для збереження запису в СУБД спочатку обчислюється хеш-адреса нового запису, після чого диспетчер файлів поміщає цей запис за обчисленою адресою;
- для видобування потрібного запису по заданому значенню хеш-поля в СУБД спочатку обчислюється хеш-адреса, а за тим диспетчеру файлів посилається запит на видобування запису за обчисленою адресою.

Простим прикладом хеш-функції може служити

```
хеш-адреса = остача від ділення на
просте натуральне число значення,
          що міститься в хеш-полі
```

З цього ясно, що хешування відрізняється від індексування тим, що у файлі може бути будь-яка кількість індексів, але тільки одне хеш-поле. Теоретично можна було б для визначення адреси замість функції використовувати безпосередньо значення ключового числового поля, проте практично такий спосіб непридатний, оскільки діапазон можливих значень ключового поля може бути набагато ширше за діапазон наявних адрес. Таким чином, щоб уникнути неефективного використання дискового простору слід знайти таку хеш-функцію, щоб можна було звузити діапазон до оптимальної величини з врахуванням можливості резервування додаткового простору.

Недолік хешування полягає в тому, що фізична послідовність записів всередині файлу, що зберігається, майже завжди відрізняється від послідовності ключового поля, а також будь-якої іншої логічно заданої послідовності, а між послідовно розміщеними записами можуть бути проміжки невизначеної протяжності. Практично завжди фізична послідовність записів у файлі, що зберігається хешуванням, принципово інша в порівнянні із заданою в ньому логічною послідовністю.

Іншим недоліком хешування є можливість виникнення ситуацій, коли два або більше різних записів мають однакові адреси, тому іноді виникає необхідність функцію виправляти. Якщо на одній сторінці розташовується декілька записів, то для виправлення можна скористатися методом прямого перебору. Суть методу полягає в наступному: припустимо, що на порожній сторінці s розміщується n записів. Тоді, при розміщенні записів і виникненні перших n збігів по деякій хеш-адресі s, всі такі записи буде розміщено на цій сторінці і знайдено при необхідності за допомогою прямого перебору. Проте при розміщенні наступного n+1 запису і виникненні чергового збігу, запис доведеться розмістити на додатковій сторінці переповнювання, для чого знадобиться додаткова дискова операція вводу-виводу.

Необхідно мати на увазі, що із збільшенням розміру файлу, що зберігається, кількість збігів адрес збільшується, що приводить до значного збільшення середнього

часу доступу – адже все більше часу доведеться витрачати на пошук інформації в наборах конфліктуючих записів. Це можна усунути, якщо реорганізувати файл, тобто завантажити даний файл, використовуючи нову хеш-функцію, що розв'язується за допомогою розширюваного хешування.

При використанні розширюваного хешування необхідно, щоб всі значення хеш-поля були унікальні, а це може бути реалізовано тільки в тому випадку, якщо хеш-поле є ключовим. Основні принципи роботи методу розширюваного хешування наступні:

- якщо як хеш-функція використовується функція φ, а значення ключового поля для деякого запису z рівне p, то як результат хешування значення ключового поля буде набуте значення псевдоключа для запису z у вигляді φ(p). Тут псевдоключ використовується не як адреса запису, а лише як непрямий покажчик на місце їх зберігання;
- файл, що зберігається, має пов'язаний з ним каталог, який також зберігається на диску. Він складається із заголовка, що має значення g, і називається глибиною каталога, а також 2g покажчиків на сторінки з декількома записами даних на кожній.

Таким чином, за допомогою каталога глибиною g можна організувати доступ до файлу, що містить 2g різних сторінок з даними.

Якщо розглядати перші g біт псевдоключа як ціле беззнакове двійкове число b, то i-й покажчик в каталозі відноситиметься до сторінки, що містить всі записи, для яких величина b рівна i-1. Для того, щоб знайти запис із значенням ключового поля, рівним p, слід за допомогою хеш-функції обчислити значення псевдоключа, а потім по перших g біт псевдоключа визначити чисельне значення i-1 і знайти в каталозі відповідний йому i-й покажчик на сторінку, що містить шуканий запис, який реалізується за два доступи до диска.

Тут був описаний лише один з варіантів хешування, проте існує безліч інших способів реалізації цієї основної ідеї.

Для виконання запитів можна застосовувати інший, не менш ефективний, спосіб зберігання даних з використанням ланцюжків покажчиків. При цьому, на відміну від індексування, обидва файли можуть знаходитися в одному наборі файлів, причому файл, аналогічний індексному, в даному випадку прийнято називати батьківським, а файл з основними даними відповідно дочірнім.

Розглянемо таку структуру (її часто називають батьківсько-дочірною) на прикладі вже розглянутого вище файлу, що містить дані про успішність студентів. Тоді структура складатиметься з файлів успішності і навчальних предметів, при цьому батьківський файл міститиме один запис, що зберігається, для кожного предмету, надаючи при необхідності назву предмету як заголовок ланцюжка або кільця покажчиків, що зв'язують разом всі дочірні записи про успішність студентів по цьому предмету (рис. 6.4).

Варто звернути увагу на те, що поле із назвами предметів PN відсутнє у файлі успішності, оскільки СУБД для пошуку студентів залежно від зданого предмету шукатиме рядок з назвою предмету в батьківському файлі, а потім витягується весь пов'язаний з даним рядком ланцюжок покажчиків.

Основною перевагою батьківсько-дочірньої структури є значно простіше виконання операцій вставки або видалення записів в порівнянні з індексною структурою. Крім того, батьківсько-дочірня структура займає менше місця на диску, ніж відповідна індексна структура, оскільки в ній не повторюється інформація в батьківському файлі. Проте така структура має і деякі недоліки. Наприклад, для даного предмету єдиним шляхом доступу до n-го студента є ланцюжок з послідовним перебором всіх попередніх записів студентів, а якщо для цих записів не виконана кластеризація, то для кожного доступу до даних може бути потрібна окрема операція пошуку і час доступу до запису може виявитися дуже великим.

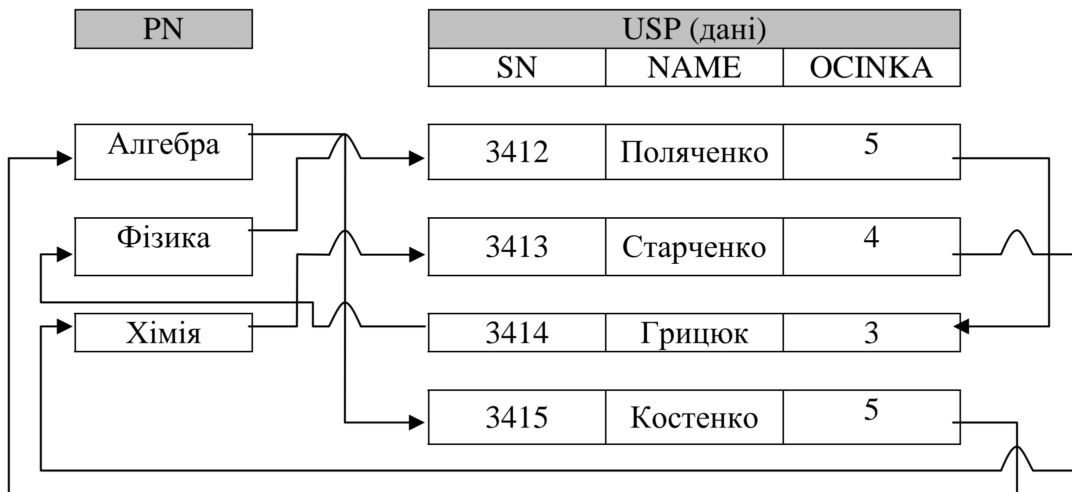


Рис. 6.4. Батьківсько-дочірня структура для файлу з даними про успішність.

З іншого боку, така структура може бути оптимальною для виконання запиту на пошук студентів, що здали той або інший предмет, проте вона неефективна для зворотного запиту на пошук предмету, складеного студентом, тому для зворотного запиту більше підійшла б індексна або хеширована структура. І. нарешті, якщо батьківський файл має дуже великий розмір, то для нього також потрібно буде застосувати індексування або хешування, а значить, ланцюжки покажчиків не є способом створення якісної структури зберігання даних і з ними потрібно спільно використовувати інші методи.

Удосконалення батьківсько-дочірньої структури можна провести за рахунок додавання в дочірніх записах ще одного покажчика на відповідний батьківський запис – при такій організації ланцюжків можна було б уникнути проглядання всього ланцюжка при виконанні запитів.

1.6.4. Стиснення даних

З метою скорочення простору, необхідного для зберігання деякого набору даних, часто використовують технології стиснення. При цьому в результаті економиться не тільки простір на диску, але і кількість дискових операцій вводу-виводу, оскільки доступ до даних меншого розміру вимагає менше дискових операцій вводу-виводу. З іншого боку, для розпаковування і видобування стислих даних потрібні деякі додаткові маніпуляції, але в цілому переваги скорочення операцій вводу-виводу можуть компенсувати недоліки, пов'язані з додатковою обробкою даних.

Технології стиснення засновані на малій ймовірності того, що дані мають абсолютно безладну структуру. Найбільш поширеною є технологія стиснення на основі відмінностей, при якій деяке значення замінюється відомостями про його відмінності від попереднього значення. Слід зазначити, що для реалізації такої технології потрібно розміщувати дані послідовно, оскільки для їх розпаковування необхідно мати значення попередньої величини. Таке стиснення дуже ефективне для даних, до яких необхідний послідовний доступ, наприклад для записів в однорівневому списку. Більше того, в таких випадках разом з даними допускається також стиснути і покажчики. Річ у тому, що якщо логічна послідовність у файлі відповідає фізичній послідовності розміщення даних на диску, то сусідні покажчики трохи відрізнятимуться один від одного, а значить, стиснення покажчиків може виявитися дуже корисним і ефективним.

Як приклад розглянемо декілька записів з наступними даними:

```
Студент
Студентка
```

```
Студентський
```

Припустимо, що довжина поля складає 15 символів і праворуч від даних в нестислому вигляді міститься відповідна кількість пропусків. Один із способів застосування стиснення на основі відмінностей – це видалення символів, що повторюються, на початку кожного запису з вказівкою їх кількості, тобто переднє стиснення. В результаті буде отримано (числа відповідають кількості символів, що повторюються, на початку імені, пропуски справа до повної довжини поля – 15 символів – не показані):

```
0 - Студент
7 - ка
7 - ський
```

При необхідності можна здійснити додаткове стиснення, видаляючи пропуски із вказівкою їх кількості, тобто виконати так зване заднє стиснення. Іноді ще допускається видалення з правого кінця кожного значення всіх символів, що повторюються, в двох найближчих сусідніх значеннях.

Іншим способом зменшення об'єму займаного даними місця є ієрархічне стиснення. Припустимо, що у файлі, що зберігається, задана деяка фізична послідовність згідно вибраному полю, різні значення якого розташовуються в декількох послідовних записах цього файлу.

Наприклад, у файлі успішності, що зберігається, завдяки кластеризації, виконаній згідно поля PN із назвами предметів, окремо містяться всі записи про студентів, що склали той або інший предмет. У такому разі набір всіх записів із даними про студентів, що склали той або інший предмет, може бути успішно стиснутий в один ієрархічний запис, що зберігається, при цьому назва предмету буде згадана тільки один раз, а вслід за ним будуть розташовані дані про студентів (рис. 6.3).

Ієрархічний запис, що зберігається, складається з двох частин: сталої, в нашому прикладі це поля із назвами предметів, і змінної – записи із інформацією про студентів. Такий набір значень змінної кількості всередині одного запису звичайно називається групою повторення. Таким чином, можна сказати, що ієрархічний запис, представлений на рис. 6.5, складається із значення поля предметів, і групи повторення із інформацією про студентів, що складається з полів із номерами студентських квитків, прізвищ студентів і оцінок по предмету.

USP (дані)			
	SN	NAME	OCINKA
Алгебра	3415	Костенко	5
Фізика	3412	Поляченко	5
	3414	Грицюк	3
Хімія	3413	Старченко	4

Рис. 6.5. Внутрішньофайлове ієрархічне стиснення файлу із даними про успішність.

Ієрархічне стиснення такого типа застосовують і для індексу, в якому декілька послідовно розташованих значень містять одні і ті ж значення даних, що повторюються, але різні значення покажчиків.

Аналогічним чином можна застосовувати ієрархічне стиснення на основі міжфайлової кластеризації. Наприклад, записи з даними про студентів як би об'єднуються з даними про всі оцінки по всіх предметах для даного студента.

На закінчення відзначимо, що структуру на основі ланцюжка покажчиків, взагалі кажучи, можна розглядати як міжфайлове стиснення, яке не вимагає міжфайлової кластеризації, оскільки покажчики дозволяють логічно досягти ефекту кластеризації.

Кодування Хафмана – це ще одна технологія кодування символів, яка може бути дуже ефективною для стиснення різних символів, що зустрічаються з різною частотою. Основна ідея цього методу полягає в кодуванні окремих символів бітовими рядками різної довжини, причому символи, що найбільш часто зустрічаються, кодуються рядками найменшої довжини. Крім того, код будь-якого символу завдовжки n не повинен співпадати з першими n символами коду якого-небудь іншого символу.

Припустимо, що деякі дані записані за допомогою символів А, Б, В, Г, Д, тоді з врахуванням відносної частоти, з якою вони зустрічаються, їх коди приведені в таблиці.

Таблиця 6.1 Коди символів.

Символ	Частота, %	Код
А	35	1
В	30	01
Г	20	001
Д	10	0001
Б	5	0000

Символ А зустрічається частіше за інших, і тому має найкоротший код, що складається з одного біта. Вся решта кодів повинна бути довшою, проте не можна використовувати код на основі одного нуля, оскільки він співпадатиме з початковою частиною інших, довших кодів. Оцінивши, можна сказати, що в середньому загальна довжина закодованого тексту на 40% менше, ніж за відсутності кодування.

Таким чином, розглянуті найважливіші сучасні структури зберігання даних, описана загальна схема функціонування програмного забезпечення, призначеного для доступу до даних, а також розподіл виконання цих завдань між СУБД, диспетчером файлів і диспетчером дисків. Основною метою викладу було роз'яснення загальних ідей і принципів без опису окремих подробиць їх реалізації в різних практично використовуваних системах і структурах зберігання.