

Лекція 17

Вступ до SQL

У 2006 році Міжнародна організація зі стандартизації (ISO) та Американський національний інститут стандартів (ANSI) опублікували зміни до свого стандарту SQL, так званий SQL:2006. Стандарт ділиться на частини, і кожна частина затверджується і публікується за певним часовим графіком, тому різні частини мають різні роки публікації; зазвичай, у назві стандарту використовується останній рік створення набору всіх частин, опублікованих протягом цього року. Стандарт SQL:2006, як і його попередники SQL:2003, SQL:1999 (також відомий як SQL3) та SQL-92, заснований на реляційній моделі даних, яка визначає, спосіб збереження та обробки даних. Реляційні системи управління базами даних (RDBMS), такі як Oracle, Sybase, DB2, MySQL та Microsoft SQL Server (або просто SQL Server) використовують стандарт SQL як основу для своєї технології, забезпечуючи середовища баз даних, які підтримують як SQL, так і реляційну модель даних.

Структурування реляційних баз даних

Мова запитів (SQL) підтримує створення та впровадження реляційної бази даних, а також управління даними в межах цієї бази даних.

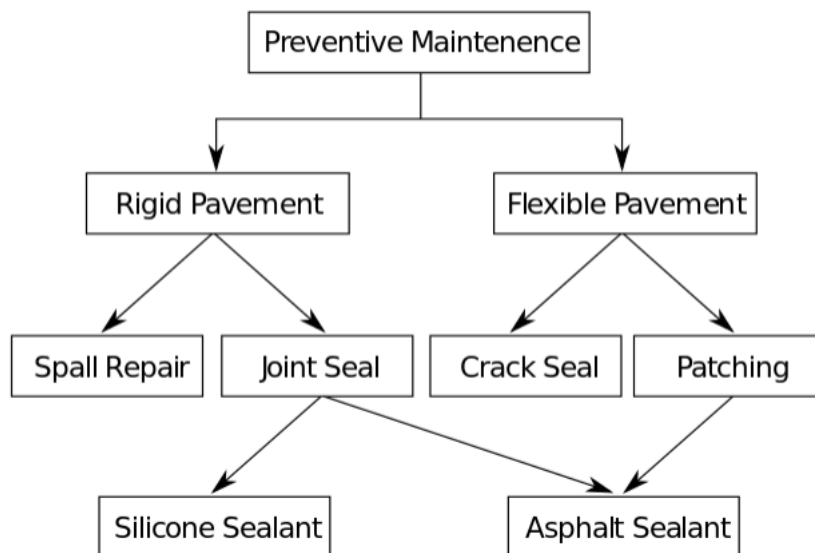
Сам термін РБД використовувався для позначення від будь-якого набору імен та адрес до складних систему пошуку і зберігання даних, які спираються на інтерфейси користувачів та мережу клієнтських комп'ютерів і серверів. Існує стільки визначень для бази даних, скільки є книг про них. Більше того, різні постачальники СУБД розробили різні архітектури, тому не всі бази даних спроектовані однаково. Незважаючи на відсутність абсолютного визначення, більшість джерел погоджуються, що база даних, принаймні, є набором даних, організованих в структурованому форматі, що визначається метаданими, які описують цю структуру. Можна розглядати метадані як дані про дані, що зберігаються; вони визначають спосіб збереження даних в базі даних.

Протягом років було реалізовано ряд моделей баз даних для зберігання та управління даними. Декілька з більш поширених моделей включають наступні:

- *Ієрархічна модель* має батьківсько-дочірню структуру, яка схожа на перевернуте дерево, що формує ієрархію. Дані організовані у вузлах – це логічний еквівалент таблиць в реляційній базі даних. Батьківський вузол може мати багато дочірніх вузлів, але дочірній вузол може мати тільки один батьківський вузол. Хоча модель набула значного поширення, вона часто вважається непридатною для багатьох застосувань через її негнучку структуру та відсутність підтримки складних відносин. Все ж, деякі реалізації, такі як IMS від IBM, ввели функції, які працюють за цих обмежень.

- *Мережева модель* враховує до деякі обмеження ієрархічної моделі. Дані організовані у вигляді записів – це логічний еквівалент таблиць в реляційній базі даних. Як і ієрархічна модель, мережева модель використовує інвертовану структуру дерева, але типи запису організовані за іншою структурою: у запису-нащадка може бути будь-яке число записів-предків. Мережева БД складається з набору екземплярів певного типу запису і набору екземплярів певного типу зв'язків між цими записами. Тип зв'язку визначається для двох типів запису: предка і нащадка. Екземпляр типу зв'язку складається з одного екземпляру типу запису предка і впорядкованого набору екземплярів типу запису-нащадка. Для даного типу зв'язку L з типом запису-предка P і типом запису-нащадка C повинні виконуватися наступні дві умови:

- кожен екземпляр типу запису P є предком тільки в одному екземплярі типу зв'язку L;
- кожен екземпляр типу запису C є нащадком не більше, ніж в одному екземплярі типу зв'язку L.



- Реляційна модель враховує багато обмежень як ієрархічної, так і мережевої моделей. В ієрархічній або мережевій базі даних програма спирається на певну реалізацію цієї бази даних, яка потім жорстко закодована в застосунку. Якщо додати новий атрибут (елемент даних) до бази даних, потрібно змінювати програму, навіть якщо вона не використовує цей атрибут. Однак реляційна база даних не залежить від програми; можна вносити зміни до структури, не впливаючи на застосунок. Крім того, структура реляційної бази даних базується на відношеннях, або таблицях, та надає змогу визначати складні взаємозв'язки між цими відношеннями. Кожне відношення може бути доступним безпосередньо, без обтяжуючих обмежень ієрархічної моделі, яка вимагає навігації по складній структурі даних.

Хоча все ще використовується в багатьох організаціях, ієрархічні та мережеві бази даних зараз вважаються спадковими рішеннями. Реляційна модель є найбільш широко реалізованою моделлю в сучасних бізнес-системах, і саме реляційна модель забезпечує основу для SQL.

Реляційна модель даних

У 1970 році Кодд опублікував свою фундаментальну працю «Реляційна модель даних для великих банків спільних даних» в журналі Communications of the ACM, Volume 13, Number 6 (June 1970). Кодд визначає реляційну структуру даних, яка захищає дані і дозволяє

маніпулювати даними таким чином, що є передбачуваним і стійким до помилок. Реляційна модель, яка базується в основному на математичних принципах теорії множин і логіки предикатів, підтримує легке отримання даних, забезпечує цілісність даних (точність і послідовність даних), і забезпечує структуру бази даних, незалежну від програм, які отримують доступ до збережених даних.

Нагадаємо, що в основі реляційної моделі лежить відношення. Відношення — це набір стовпців і рядків, зібраних в таблично-подібну структуру, яка представляє єдиний об'єкт, що складається з пов'язаних даних.

Сутність — це особа, місце, річ, подія або поняття, які визначають які саме дані збираються, наприклад, художник-записник, книга або транзакція з продажу. Кожне відношення включає один або кілька атрибутів (стовпців).

Атрибут — це одиничний факт, який певним чином описує або характеризує сутність. Наприклад, на рисунку 1-1, сутність є компактдиском (CD) з атрибутами CD_NAME (назва CD), ARTIST_NAME (ім'я виконавця запису), і COPYRIGHT_YEAR (рік запису був авторським правом).

Як видно на рисунку 1-1, кожен атрибут має асоційований домен. Домен визначає тип даних, які можуть бути збережені в певному атрибуті; однак, домен не є тим самим, що і тип даних. Тип даних є специфічним видом обмеження (управління, яке використовується для забезпечення цілісності даних), пов'язаного зі стовпчиком, в той час як домен, як він використовується в реляційній моделі, має набагато ширше значення і описує саме те, які дані можуть бути включені в атрибут, пов'язаний з цим доменом, наприклад, атрибут COPYRIGHT_YEAR пов'язаний з доменом YEAR. В прикладі, поширеною практикою є включення назви класу, яка описує домен в імена атрибутів, але це зовсім не обов'язково. Домен може бути визначений так, щоб атрибут містив тільки дані, значення і формат яких обмежені роками, а не днями або місяцями. Домен також може обмежувати дані до певного діапазону років. Тип даних, з іншого боку, обмежує формат даних, наприклад, дозволяє

використовувати лише числові дані, але не строкові значення, бо ці значення порушують формат.

ARTIST_NAME:FullName	CD_NAME:Title	COPYRIGHT_YEAR:Year
Jennifer Warnes	Famous Blue Raincoat	1991
Joni Mitchell	Blue	1971
William Ackerman	Past Light	1983
Kitaro	Kojiki	1990
Bing Crosby	That Christmas Feeling	1993
Patsy Cline	Patsy Cline: 12 Greatest Hits	1988

Figure 1-1 Relation containing CD_NAME, ARTIST_NAME, and COPYRIGHT_YEAR attributes

Дані зберігаються у відношеннях в кортежах (рядках). Кортєж — це набір даних, значення яких складають один екземпляр кожного атрибута, визначеного для цього відношення. Кожен набір представляє запис пов'язаних даних (такий набір даних ще називають записом.) Наприклад, на рисунку 1-1, другий кортеж зверху містить значення “Джоні Мітчелл” для атрибута ARTIST_NAME, значення “Blue” для атрибута CD_NAME, а також значення “1971” для атрибута COPYRIGHT_YEAR. Разом ці три значення утворюють кортеж.

Зауваження: логічні терміни відношення, атрибут і кортеж використовуються в першу чергу під час роботи з реляційними моделями. SQL використовує терміни таблиці, стовпці та рядки для опису цих елементів. Через те, що реляційна модель базується на математичних принципах (логічна модель), а SQL більше стосується фізичної реалізації моделі, значення термінів моделі та термінів мови SQL дещо відрізняються, але основні принципи однакові.

Реляційна модель, звичайно, складніша, ніж просто атрибути і кортежі, які складають відношення. Двома дуже важливими питаннями в розробці та реалізації будь-якої реляційної бази даних є нормалізація даних та відносин між різними типами даних.

Нормалізація даних

Головним принципом побудови реляційної моделі є поняття нормалізації, техніка створення набору відношень, що має певний набір властивостей, завдяки яким мінімізуються надлишкові дані та зберігається цілісність даних під час їх збереження (додавання, оновлення та видалення).

Процес нормалізації даних був розроблений Е.Ф.Коддом в 1972 році. Нормалізація визначає набори правил, які називаються нормальними формами, які визначають конкретні дії щодо того, як дані повинні бути організовані для того, щоб уникнути аномалій, які призводять до невідповідностей і втрати даних під час їх збереження.

Коли Кодд вперше представив нормалізацію, вона включала три нормальні форми. Хоча з тих пір були додані додаткові нормальні форми, перші три все ще охоплюють більшість ситуацій, з якими можна зіткнутися як в базах даних для особистих потреб, так і в базах даних призначених для потреб бізнесу баз даних.

Вибір унікального ідентифікатора

Унікальний ідентифікатор — атрибут або набір атрибутів, що однозначно ідентифікує кожен рядок даних у відношенні. Унікальний ідентифікатор в решті решт стає первинним ключем таблиці, створеної у фізичній базі даних з нормалізованого відношення, але багато хто використовує терміни «унікальний ідентифікатор» та «первинний ключ» як взаємозамінні.

Кожен потенційний унікальний ідентифікатор називається потенційним ключем, за наявності кількох потенційних ключів, розробник має обрати найкращий в певному сенсі: або той, за значеннями якого рідко відбуваються зміни значень, або той, що є найпростішим та/або той, що є найкоротшим. У багатьох випадках можна знайти один атрибут, який

однозначно ідентифікує дані в кожному кортежу відношення. Однак, коли не може бути знайдено жодного такого єдиного атрибута, значення якого є унікальними для кожного кортежу, розробник шукає кілька атрибутів, які можуть бути застосовані разом для формування унікального ідентифікатора.

У поодиноких випадках, коли не можна визначити адекватних потенційних ключів, розробник має штучно створити унікальний ідентифікатор, який називається сурогатний ключ. Хоча це не зовсім обов'язково до другої нормальної форми, прийнято вибирати унікальний ідентифікатор як перший крок в нормалізації. Так простіше.

Перша нормальна форма

Перша нормальна форма забезпечує основу для другої та третьої нормальних форм. Перша нормальна форма включає наступні принципи:

- кожен атрибут кортежу повинен містити значення лише одного типу (вмісту).
- кожен кортеж у відношенні повинен містити однакову кількість атрибутів.
- всі кортежи повинні бути різними, тобто поєднання всіх значень атрибутів для заданого кортежу не може повторюватися у будь-якому іншому кортежі в даному відношенні.

Як видно на рисунку 1-2, другий та останній кортежі порушують першу нормальну форму. У другому рядку атрибут CD_NAME та атрибут COPYRIGHT_YEAR містять по два значення. В останньому кортежі атрибут ARTIST_NAME містить три значення. Не слід розподіляти значення у стовпчики, що повторюються. Наприклад, розбиття атрибута ARTIST_NAME на три атрибути, що називаються ARTIST_NAME_1, ARTIST_NAME_2, і ARTIST_NAME_3 не є адекватним рішенням, оскільки в решті решт з'явиться потреба в четвертому імені, потім п'ятому імені, і так далі. Більше того, повторення стовпчиків робить запити складнішими, оскільки потрібно пам'ятати про пошук всіх стовпців під час пошуку конкретного значення.

Щоб нормалізувати відношення, представлене на рисунку 1-2, слід створити додаткові відношення, які розділяють дані так, щоб кожен атрибут містив лише одне значення, кожен кортеж містив однакову кількість атрибутів, а кожен кортеж відрізнявся від будь-якого іншого кортежу, як показано на рисунку 1-3. Дані тепер відповідають першій нормальній формі.

Зауважте, що у другому відношенні є значення що повторюються; значення 10002 для атрибута ARTIST_ID повторюється і значення 99308 для атрибута CD_ID також повторюється. Однак, кожен з відповідних кортежів взятих цілком утворюють унікальну комбінацію, що означає, що, незважаючи на наявність дублювання значень, кожен кортеж у відношенні є унікальним.

ARTIST_NAME	CD_NAME	COPYRIGHT_YEAR
Jennifer Warnes	Famous Blue Raincoat	1991
Joni Mitchell	Blue; Court and Spark	1971; 1974
William Ackerman	Past Light	1983
Kitaro	Kojiki	1990
Bing Crosby	That Christmas Feeling	1993
Patsy Cline	Patsy Cline: 12 Greatest Hits	1988
Jose Carreras; Placido Domingo; Luciano Pavarotti	Carreras Domingo Pavarotti in Concert	1990

Рисунок 1-2 Відношення, яке не відповідає першій нормальній формі

ARTIST_ID	ARTIST_NAME	ARTIST_ID	CD_ID	CD_ID	CD_NAME	COPYRIGHT_YEAR
10001	Jennifer Warnes	10001	99301	99301	Famous Blue Raincoat	1991
10002	Joni Mitchell	10002	99302	99302	Blue	1971
10003	William Ackerman	10002	99303	99303	Court and Spark	1974
10004	Kitaro	10003	99304	99304	Past Light	1983
10005	Bing Crosby	10004	99305	99305	Kojiki	1990
10006	Patsy Cline	10005	99306	99306	That Christmas Feeling	1993
10007	Jose Carreras	10006	99307	99307	Patsy Cline: 12 Greatest Hits	1988
10008	Placido Domingo	10007	99308	99308	Carreras Domingo Pavarotti in Concert	1990
10009	Luciano Pavarotti	10008	99308			
		10009	99308			

Рисунок 1-3 Відношення, які відповідають першій нормальній формі

Зауважте, що були додані атрибути ARTIST_ID та CD_ID. Це було зроблено тому, що не було інших можливих ключових полів. ARTIST_NAME не є унікальним (дві людини з однаковими іменами могли бути артистами, що записали певні CD), атрибут CD_NAME так само не є унікальним (два компакт-диски могли мати одну назву, хоча вони, ймовірно, були б записані на різних студіях звукозапису). Тому додані поля: ARTIST_ID – первинний ключ першого відношення, та CD_ID – первинний ключ третього відношення. Первинним ключем другого відношення є комбінація атрибутів ARTIST_ID та CD_ID.

Друга нормальна форма

Щоб зрозуміти принципи побудови другої нормальної форми, слід спочатку ввести поняття функціональної залежності.

Для визначення функціональної залежності, використаємо два довільних атрибути, які позначимо А та В.

Атрибут В функціонально залежить від атрибута А тоді і тільки тоді, коли кожне значення атрибута А зв'язане лише з одним значенням атрибута В. Іншими словами, якщо два кортежі збігаються за значеннями атрибута А, то вони збігаються і за значеннями атрибута В. У цьому випадку атрибут А – детермінант (унікальний ідентифікатор атрибута В), атрибут В – залежна частина.

На рисунку 1-4, атрибут COPYRIGHT_YEAR залежить від атрибута CD_ID, оскільки може бути тільки одне значення COPYRIGHT_YEAR для будь-якого даного CD. Тобто, CD_ID є детермінантом COPYRIGHT_YEAR.

Друга нормальна форма визначає, що відношення має бути в першій нормальній формі і щоб всі атрибути в відношенні функціонально залежали одночасно від всіх атрибутів, які складають унікальний ідентифікатор (або первинний ключ). На рис.1-4, якщо комбінація атрибутів ARTIST_ID та CD_ID обрано за унікальний ідентифікатор, то COPYRIGHT_YEAR порушує другу нормальну форму, оскільки цей атрибут залежить лише від CD_ID, а не від комбінації CD_ID та ARTIST_ID. Незважаючи на те, що відношення відповідає першій нормальній формі, воно порушує другу нормальну форму. Знову ж таки, рішення полягає в тому, щоб розділити дані на різні відношення, аналогічно тому як це показано на рисунку 1-3.

← Unique identifier →		
ARTIST_ID	CD_ID	COPYRIGHT_YEAR
10001	99301	1991
10002	99302	1971
10002	99303	1974
10003	99304	1983
10004	99305	1990
10005	99306	1993
10006	99307	1988

Рисунок 1-4 Відношення зі складеним ключем

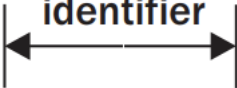
Третя нормальна форма

Щоб задовольняти вимогам третьої нормальної форми, відношення повинно бути в другій нормальній формі та у ньому немає бути транзитивних залежностей між неключовими атрибутами, тобто значення будь-якого атрибута відношення, що не входить до первинного ключа, не залежить від значення іншого атрибута, що не входить до первинного ключа.

Наприклад, унікальним ідентифікатором у відношенні, наведеному на рис.1-5, є атрибут ARTIST_ID. Атрибути ARTIST_NAME і AGENCY_ID залежать від унікального ідентифікатора і не залежать один від одного. Однак атрибут AGENCY_STATE залежить від атрибута AGENCY_ID, і

тому він порушує умови третьої нормальної форми. Цей атрибут краще підходить до відношенні, яке включає дані про агентства.

Unique
identifier



ARTIST_ID	ARTIST_NAME	AGENCY_ID	AGENCY_STATE
10001	Jennifer Warnes	2305	NY
10002	Joni Mitchell	2306	CA
10003	William Ackerman	2306	CA
10004	Kitaro	2345	NY
10005	Bing Crosby	2367	VT
10006	Patsy Cline	2049	TN
10007	Jose Carreras	2876	CA
10008	Placido Domingo	2305	NY
10009	Luciano Pavarotti	2345	NY

Рисунок 1-5 Відношення, яке не задовольняє третій нормальній формі

Зауваження. В теорії реляційного проектування, мета полягає в зберіганні даних за правилами нормалізації. Проте в реальних задачах реалізації та впровадження баз даних іноді доводиться денормалізувати дані, що означає порушення певних правил нормалізації, зокрема другої та третьої нормальних форм. Денормалізація використовується в першу чергу для поліпшення продуктивності або зниження складності баз даних в тих випадках, коли нормалізована структура ускладнює реалізацію. Все ж, метою нормалізації є забезпечення цілісності даних, тому денормізація

повинна виконуватися з великою обережністю в якості крайнього заходу.

Корисне посилання

https://rdb.dp.ua/uk/chapter_03

Відношення

Важливим питанням за проектування будь-якої реляційної бази даних є відносини між таблицями. Асоціації, або відносини, зв'язують відношення між собою змістовним чином, що допомагає забезпечити цілісність даних так, щоб операції над даними в одному відношенні, не впливали негативно на дані в іншому відношенні.

Нагадаємо, що існує три типи відношень:

- Один-до-одного: один кортеж у відношенні A відповідає одному кортежу у відношенні B, і навпаки, один кортеж у відношенні B відповідає одному кортежу у відношенні A.
- Один-до-багатьох: один кортеж у відношенні A відповідає багатьом кортежам у відношенні B, а один кортеж у відношенні B відповідає лише одному кортежу у відношенні A.
- Багато-до-багатьох: один кортеж у відношенні A відповідає багатьом кортежам у відношенні B, і навпаки, один кортеж у відношенні B відповідає багатьом кортежам у відношенні A.

Наведені типи зв'язків проілюстровані на рисунку 1-6.

- Зв'язок типу один-до-одного існує між відношеннями ARTIST_AGENCIES та ARTIST_NAMES. Кожному кортежу відношення ARTIST_AGENCIES відповідає лише один кортеж відношення ARTIST_NAMES і навпаки. Така ситуація відповідає бізнес процесам, за якими один виконавець може працювати лише з одним агентством звукозапису одночасно.
- Зв'язок типу один-до-багатьох існує між відношеннями ARTIST_NAMES та ARTIST_CDS. Кожному кортежу відношення ARTIST_NAMES може відповідати нуль або багато кортежів відношення

ARTIST_CDS. Іншими словами кожен виконавець може або не записати жодного диску, або багато дисків. Кожному кортежу відношення ARTIST_CDS відповідає лише один кортеж відношення ARTIST_NAMES. Така ситуація виникає за умови, що кожен диск може бути записаний лише одним виконавцем.

- Зв'язок типу багато-до-багатьох існує між відношеннями ARTIST_NAMES та ARTIST_CDS, якщо кожному кортежу відношення ARTIST_NAMES може відповідати нуль або багато кортежів відношення ARTIST_CDS. Іншими словами кожен виконавець може або не записати жодного диску, або багато дисків. Кожному кортежу відношення ARTIST_CDS відповідає багато кортежів відношення ARTIST_NAMES. Така ситуація виникає за умови, що кожен диск може бути записаний багатьма виконавцями одночасно.

Зауваження. Реляційні бази даних підтримують тільки зв'язки типу один-до-багатьох. Зв'язки типу багато-до-багатьох фізично реалізуються шляхом додавання третього відношення між першим і другим відношенням для створення двох співвідношень один-до-багатьох. На рис.1-6, відношення ARTIST_CDS було додано між відношеннями ARTIST_NAMES та COMPACT_DISCS. Відношення один-до-одного фізично реалізується так само, як відношення один-до-багатьох, за винятком того, що додається обмеження для запобігання дублювання відповідних рядків на стороні відносин «багато». На рис. 1-6, на атрибуті ARTIST_ID буде додано унікальне обмеження, щоб запобігти появі виконавця, який може працювати більш ніж в одній агенції звукозапису.

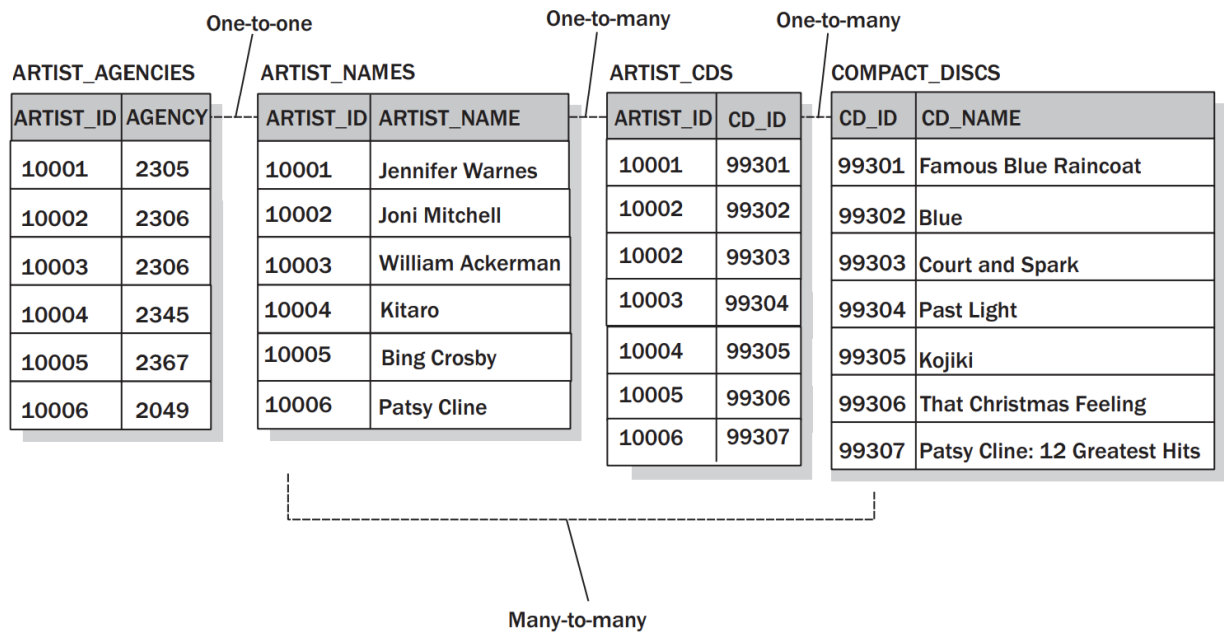


Рисунок 1-6 Типи зв'язків між відношеннями

Відношення також класифікуються за мінімальною потужністю (мінімальною кількістю кортежів, які повинні брати участь у зв'язку). Якщо кожен кортеж в одному відношенні повинен мати один відповідний кортеж в іншому, то відносини є обов'язковими в цьому напрямку. Аналогічно, якщо кожен кортеж в одному відношенні не вимагає відповідного кортежу в іншому, то відношення називається необов'язковим в цьому напрямку. Наприклад, зв'язок між ARTIST_NAMES і ARTIST_AGENCIES є обов'язковим-, оскільки кожен виконавець повинен мати одне агентство і кожен кортеж відношення ARTIST_AGENCIES повинен посилатися на одного і тільки одного виконавця. Слід чітко розуміти бізнес-логіку процесів для визначення мінімальної потужності. Наприклад, чи може існувати виконавець, який в певний момент часу не має компакт-дисків в базі даних (тобто, немає відповідних кортежів в ARTIST_CDS)? Якщо так, то зв'язок між ARTIST_NAMES і ARTIST_CDS необов'язковий; інакше він є обов'язковим.

Ознайомлення з SQL

Розглянемо основні принципи роботи SQL та її основні характеристики. SQL базується на реляційній моделі, хоча вона не є точною реалізацією

цієї моделі. У той час як реляційна модель забезпечує теоретичне підґрунтя реляційної бази даних, саме мова SQL підтримує фізичну реалізацію цієї бази даних.

SQL, майже універсально реалізована реляційна мова, відрізняється від інших комп'ютерних мов, таких як C, COBOL і Java, які є процедурними. Процедурна мова визначає, як повинні виконуватися операції застосунку і порядок, в якому вони виконуються. Непроцедурна мова, з іншого боку, більше визначає результати операції; базове програмне середовище визначає, як будуть оброблятися операції. Це не означає, що SQL підтримує відсутність процедурної функціональності. Наприклад, збережені процедури, додані до багатьох продуктів РСУБД кілька років тому, є частиною стандарту SQL:2006 і забезпечують процедурні можливості. Багато постачальників СКБД додали розширення до SQL, щоб забезпечити ці процедурні можливості, такі як Transact-SQL, створені в Sybase і Microsoft SQL Server і PL/SQL, створені в Oracle.

В SQL все ще не вистачає багатьох основних можливостей програмування більшості інших комп'ютерних мов. З цієї причини, SQL часто називають підмовою даних, тому що вона найчастіше використовується разом з мовами прикладного програмування, такими як C і Java, мовами, які не призначені для маніпулювання даними, що зберігаються в базі даних. Як наслідок, SQL використовується спільно з мовою застосунку для забезпечення ефективного засобу доступу до цих даних.

Розвиток SQL

На початку 1970-х років, після того, як була опублікована новаторська робота Е. Ф. Кодда, IBM почала розробку мови і системи баз даних, які могли б бути використані для реалізації реляційної моделі даних. Коли вперше було визначено рішення цієї задачі, мову, яка була створена назвали Structured English Query Language (SEQUEL). Коли було виявлено, що SEQUEL був торговою маркою, що належить компанії Hawker-Siddeley Aircraft Company Великої Британії, назва була змінена на SQL.

Щойно стало відомо, що IBM розробляє реляційну систему баз даних, засновану на SQL, інші компанії почали розробляти свої власні продукти на базі SQL. Насправді, Relational Software, Inc., тепер корпорація Oracle, випустила свою систему баз даних, перш ніж IBM виклала свій продукт на ринок. В мірі того, як більшість компаній ситали створювати свої продукти на базі цієї мови, SQL почала з'являтися як стандартна мова реляційної бази даних.

У 1986 році Американський національний інститут стандартів (ANSI) випустив перший опублікований стандарт для мови (SQL-86), який був прийнятий Міжнародною організацією зі стандартизації (ISO) в 1987 році. Стандарт був оновлений в 1989, 1992, 2003, 2006 роках, роботи над ним ще тривають. З часом стандарт значно виріс — початковий варіант нараховував менше за 1,000 сторінок, в той час як версія SQL:2006 вже нараховувала більше ніж 3,700 сторінок. Стандарт був написаний частинами, щоб забезпечити більш своєчасну публікацію редакцій окремих частин і полегшити паралельну роботу різних комітетів. У таблиці 1-1 наведено огляд частин і поточний стан кожної з них.

Таблиця 1-1 Частини стандарту SQL

Part	Topic	Status
1	SQL/Framework	Completed in 1999, revised in 2003, corrections published in 2007
2	SQL/Foundation	Completed in 1986, revised in 1999 and 2003, corrections published in 2007
3	SQL/CLI	Completed in 1995, revised in 1999 and 2003, corrections published in 2005
4	SQL/PSM	Completed in 1996, revised in 1999 and 2003, corrections published in 2007
5	SQL/Bindings	Established as a separate part in 1999, but merged back into Part 2 in 2003; there is currently no Part 5
6	SQL/Transaction	Project canceled; there is currently no Part 6
7	SQL/Temporal	Withdrawn; there is no Part 7
8	SQL/Objects and Extended Objects	Merged into Part 2; there is no Part 8
9	SQL/MED	Started after 1999, completed in 2003, corrections published in 2005
10	SQL/OLB	Completed as ANSI standard in 1998, ISO version completed in 1999, revised in 2003, corrections published in 2007
11	SQL/Schemata	Extracted to a separate part in 2003, corrections published in 2007
12	SQL/Replication	Project started in 2000, but subsequently dropped; there currently is no Part 12
13	SQL/JRT	Completed as ANSI standard in 1999, revision completed in 2003, corrections published in 2005
14	SQL/XML	Completed in 2003, expanded in 2006, corrections published in 2007

Через те, що постачальники СУРБД створювали свої продукти та викладали їх на ринок ще до того, як був розроблений стандарт SQL, деякі особливості цих продуктів були реалізовані досить по-різному, тому стандарт SQL не міг вмістити всі ці особливості, коли він був розроблений. Це може пояснити відсутність єдиного стандарту для баз даних. Коли виходить кожен новий випуск стандарту SQL, постачальники СУБД мають допрацьовувати власні продукти, щоб включити в них новий стандарт. Наприклад, зберігання процедур та тригерів було новою можливістю в стандарті SQL:1999, хоча ця можливість була реалізована в різних СУРБД вже протягом багатьох років. Тому випуск SQL:1999 просто стандартизував мову, яка використовується для реалізації функцій, які вже існували раніше.

Збережена процедура (stored procedure) — це набір операторів SQL, які зберігаються як об'єкт на сервері баз даних, але можуть бути викликані клієнтом просто викликом процедури. Тригер схожий на збережену процедуру в тому, що це набір операторів SQL, що зберігаються як

об'єкт в базі даних на сервері. Проте, замість того, щоб викликатися користувачем, запускається автоматично, коли відбувається якась наперед визначена подія, наприклад вставка або оновлення даних.

Об'єктно реляційна модель

Мова SQL заснована на реляційній моделі до версії SQL-92 включно. Однак, починаючи з SQL:1999, стандарт SQL розширився за межі суто реляційної моделі та включив в себе об'єктно-орієнтовані конструкції. Ці конструкції засновані на концепціях, притаманних об'єктно-орієнтованому програмуванню, методології програмування, яка визначає автономні набори структур даних та процедур (які об'єднуються в об'єкти). В об'єктно-орієнтованих мовах, таких як Java і C++, об'єкти взаємодіють один з одним і таким чином дозволяють вирішувати складні проблеми, які значно складніше були б вирішені за використання традиційних мов.

З появою об'єктно-орієнтованого програмування, поряд з досягненнями в області апаратних і програмних технологій, а також зростаючої складності застосунків, стало все більш очевидним, що суто реляційна мова не відповідає вимогам сучасного світу. Специфічною проблемою був той факт, що SQL не міг підтримувати складні і визначені користувачем типи даних або розширення, необхідні для створення складніших застосунків.

Завдяки конкурентному характеру ІТ галузі, постачальники СУБД взяли на себе вирішення проблеми модифікації власної продукції та включення об'єктно-орієнтованих функціональних можливостей в свої системи. Стандарт SQL:2006 відповідає тенденції рішення цієї задачі та розширює реляційну модель об'єктно-орієнтованими можливостями, такими як методи, інкапсуляція, і складні користувацькі типи даних, таким чином SQL перетворюється на об'єктно-реляційну мову баз даних. Як показано в таблиці 1-1, частина 14 (SQL/XML) була значно розширена і перевидана за допомогою SQL:2006, а всі інші частини переносяться з SQL:2003.

Відповідність стандарту SQL:2006

Після того, як SQL була стандартизована, було визначено також вимоги, які визначають відповідність СУБД даному стандарту. Наприклад, стандарт SQL-92 передбачав три рівні відповідності: початковий (entry), середній (intermediate), повний (full). Більшість популярних СУБД досягали тільки відповідності початкового рівня. Через це, SQL:2006 приймає інший підхід до встановлення стандартів відповідності. Щоб продукт був у відповідності з SQL:2006, він повинен підтримувати рівень відповідності базовий (core) SQL, який визначається як відповідність частині 2 (SQL/Foundation) і частині 11 (SQL/Schemata) стандарту SQL.

На додаток до базового рівня відповідності SQL, постачальники можуть претендувати на відповідність будь-якій іншій частині, за умови що їх продукт задовольняє мінімальним вимогам відповідності цієї частини.

Зауваження. Можна переглянути інформацію про стандарт SQL:2006, придбавши копію відповідних стандартних документів, опублікованих ANSI та ISO. Стандарт поділяється на дев'ять документів (по одній частині на документ). Перший документ (ANSI/ISO/IEC 9075-1:2003) включає огляд всіх дев'яти частин. Закінчення назви кожного документа містить рік публікації, а різні частини мають різні роки публікації, тому що частини оновлюються і публікуються незалежно різними комітетами. Як видно з таблиці 1-1, частина 1 була востаннє опублікована в 2003 році, а фактично лише частина 14 містить дату публікації 2006 року — всі інші частини були востаннє опубліковані в 2003 році. Ці документи можна придбати онлайн в Інтернет-магазині ANSI Electronic Standards Store (<http://webstore.ansi.org/>), магазині стандартів NCITS (<http://www.techstreet.com/ncitsgate.html>), або в магазині ISO (<http://www.iso.org/iso/store.htm>). На сайті ANSI існує два варіанти кожного документа з майже ідентичним змістом, які мають назви INCITS/ISO/IEC 9075 і ISO/IEC 9075. Також доступними без оплати є поправки, що називаються Technical Corrigenda. Як показано в таблиці 1-1, три частини мали поправки, опубліковані в 2005 році, а шість інших частин мали поправки, опубліковані в 2007 році.

Типи операторів SQL

Хоча SQL вважається підмовою через її непроцедурний характер, тим не менш, SQL є повноцінною мовою, яка дозволяє створювати та підтримувати об'єкти баз даних, забезпечувати безпеку цих об'єктів і маніпулювати даними всередині об'єктів. Один загальний метод, який використовується для категоризації операторів SQL, полягає в тому, щоб розділити їх відповідно до функцій, які вони виконують. Виходячи з цього методу, SQL можна розділити на три типи інструкцій:

- **Language Definition Data (DDL)** DDL-оператори використовуються для створення, модифікації або видалення об'єктів бази даних, таких як таблиці, відображення, схеми, домени, тригери, і збережені процедури. Ключовими словами SQL, які найчастіше пов'язані з DDL-заявами, є CREATE, ALTER, DROP. Наприклад, для створення таблиці використовують оператор CREATE TABLE, для зміни властивостей таблиці – оператор ALTER TABLE, а для видалення визначення таблиці з бази даних оператор DROP TABLE.

- **Data Control Language (DCL)** DCL – оператори, які дозволяють контролювати, хто або що (користувачем бази даних може бути людина або прикладна програма) має доступ до конкретних об'єктів у певній базі даних. За допомогою DCL можна надавати або обмежувати доступ за допомогою операторів GRANT або REVOKE, двох основних операторів DCL. Оператори DCL також дозволяють контролювати тип доступу кожного користувача до об'єктів баз даних. Наприклад, можна визначити, які користувачі можуть переглядати конкретний набір даних, а які маніпулювати цими даними та змінювати їх.

- **Data Manipulation Language (DML)** DML ці оператори використовуються для отримання, додавання, зміни або видалення даних, що зберігаються в об'єктах бази даних. Основними ключовими словами, пов'язаними з DML-операторами, є SELECT, INSERT, UPDATE та DELETE. Оператори цієї групи використовуються найчастіше. Наприклад, можна використовувати оператор SELECT для отримання даних з таблиці та оператор INSERT для додавання даних до таблиці.

Зауваження. Крім наведеної вище існує декілька способів класифікації операторів SQL. Наприклад, їх можна класифікувати відповідно до того, як вони виконуються, або чи можуть вони бути вбудовані в стандартну мову програмування. Стандарт SQL:2006 передбачає десять категорій на основі функцій, що виконують оператори. Наведена класифікація використовується в документації SQL, та забезпечує повний огляд функціональності, притаманної SQL.

Види виконання

Крім визначення правил використання операторів, стандарт SQL надає деталі того, яким саме чином можуть бути виконані оператори. Ці методи виконання, відомі як стилі зв'язування, не тільки впливають на характер виконання, але і визначають, які твердження щонайменше повинні підтримуватися певним стилем зв'язування. Стандарт визначає чотири методи виконання:

- Прямий виклик (Direct invocation) за допомогою цього методу можна безпосередньо взаємодіяти з базою даних із зовнішнього (Front-end) застосунку, наприклад, з iSQL*Plus в Oracle або з Management Studio в Microsoft SQL Server. Зовнішній застосунок та база даних можуть знаходитися при цьому на одному комп'ютері, але найчастіше на різних. Слід лише ввести запит у вікно програми та виконати інструкції SQL. Результати запиту повертаються так швидко, як це дозволяє потужність процесора та обмеження бази даних. Це швидкий спосіб перевірки даних, перевірки з'єднань і перегляду об'єктів бази даних. Однак, принципи стандарту SQL щодо прямого виклику досить обмежені, тому методи та оператори SQL, які використовуються та підтримуються під час створення застосунків, можуть значно варіюватися від одного продукту до іншого.

- Вбудовані оператори (Embedded SQL) За цього метода SQL-оператори заковані (вбудовуються) безпосередньо в мову програмування хоста. Наприклад, можна вбудовувати оператори SQL в код застосунку на C. Перед компілюванням коду препроцесор аналізує оператори SQL і відділяє їх від коду C. Код SQL конвертується в форму,

яку РСУБД може зрозуміти, а решта коду С компілюється так, як звичайний код С.

- Зв'язування модулів (Module binding) Цей метод дозволяє створювати блоки SQL-операторів (модулі), які є відокремленими від мови програмування хоста. Після створення модуля його об'єднують у застосунок за допомогою компонувальника (linker). Модуль містить, серед іншого, процедури, і саме процедури містять фактичні оператори SQL.

- Інтерфейс рівня викликів (Call-level interface (CLI)) CLI дозволяє викликати оператори SQL через інтерфейс шляхом передачі SQL-операторів як значень аргументів до підпрограм. Оператори попередньо не компілюються, оскільки вони знаходяться у прив'язці до вбудованої SQL та модулів. Замість цього вони виконуються безпосередньо РСУБД.

Прямий виклик, хоча і не найпоширенішим методом, підтримує представлення спеціальних запитів до бази даних і генерує результати одразу. Проте, вбудований SQL в даний час є методом, який найчастіше використовуванням в бізнес-застосунках.

Стандарт SQL у впровадженні застосунків

В основі будь-якої СУБД на базі SQL, звичайно, знаходиться сама мова SQL. Проте, використовувана мова не є виключно SQL. Кожен продукт розширює мову, щоб реалізувати визначені замовником функції та покращену функціональність на основі SQL. Більше того, ряд продуктів РСУБД розробили та вивели на ринок ще до того, як з'явилася стандарт SQL. Таким чином, кожен постачальник підтримує дещо інший варіант SQL, тобто мова, яка використовується в кожному окремому продукті, є специфічною саме для його реалізації. Наприклад, SQL Server використовує Transact-SQL, яка охоплює як розширення SQL, так і розширення постачальника, щоб забезпечити процедурні оператори, необхідні для тригерів і збережених процедур. З іншого боку, Oracle виділяє процедурні інструкції в окремий компонент продукту, який

називається PL/SQL. В результаті, інструкції SQL можуть дещо відрізнятися в реалізації продукту, який використовується.

Однією з переваг використання таких продуктів як Oracle або SQL Server, є те, що вони підтримують прямий виклик через графічний інтерфейс. SQL Server використовує інтерфейс Management Studio, показаний на рисунку 1-7. Інтерфейс GUI дозволяє створювати спеціальні SQL запити, передавати їх до СУБД для обробки, і переглядати результати. Oracle має кілька рішень для графічного інтерфейсу Front-end, включаючи веб-інтерфейс і SQL*Plus, показаний на рис.1-8.

На додаток до інтерфейсів GUI, більшість продуктів включають інтерфейс командного рядка, який можна використовувати на старих терміналах, які не мають графічної можливості. Ці інтерфейси також корисні для виконання скриптів, що містять оператори SQL і для комутованого з'єднання, для якого графічні інтерфейси занадто повільні. На рисунку 1-9 показано інтерфейс командного рядка для MySQL, що працює на Microsoft Windows.

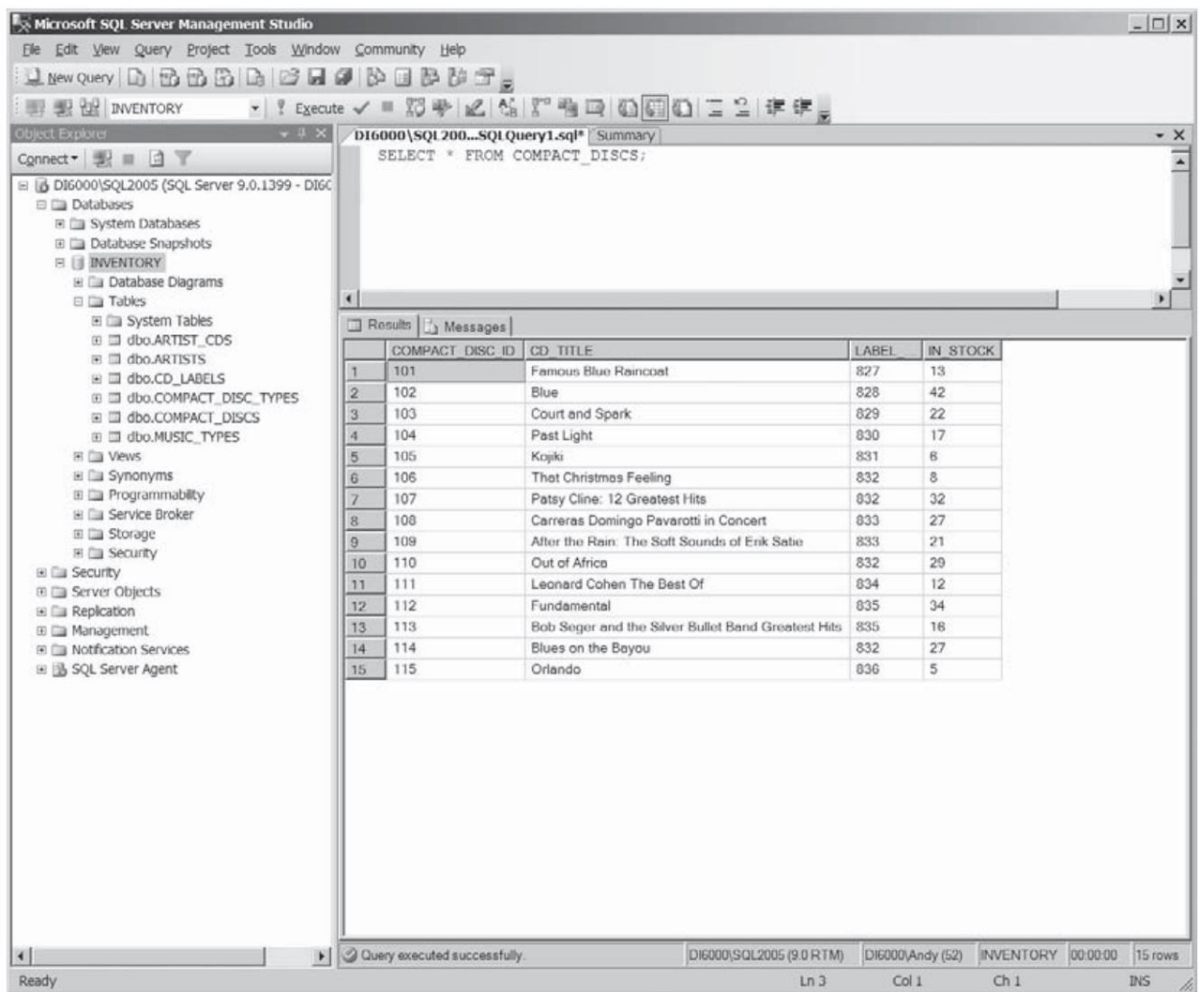


Рисунок 1-7 Використання інтерфейсу SQL Server Management Studio

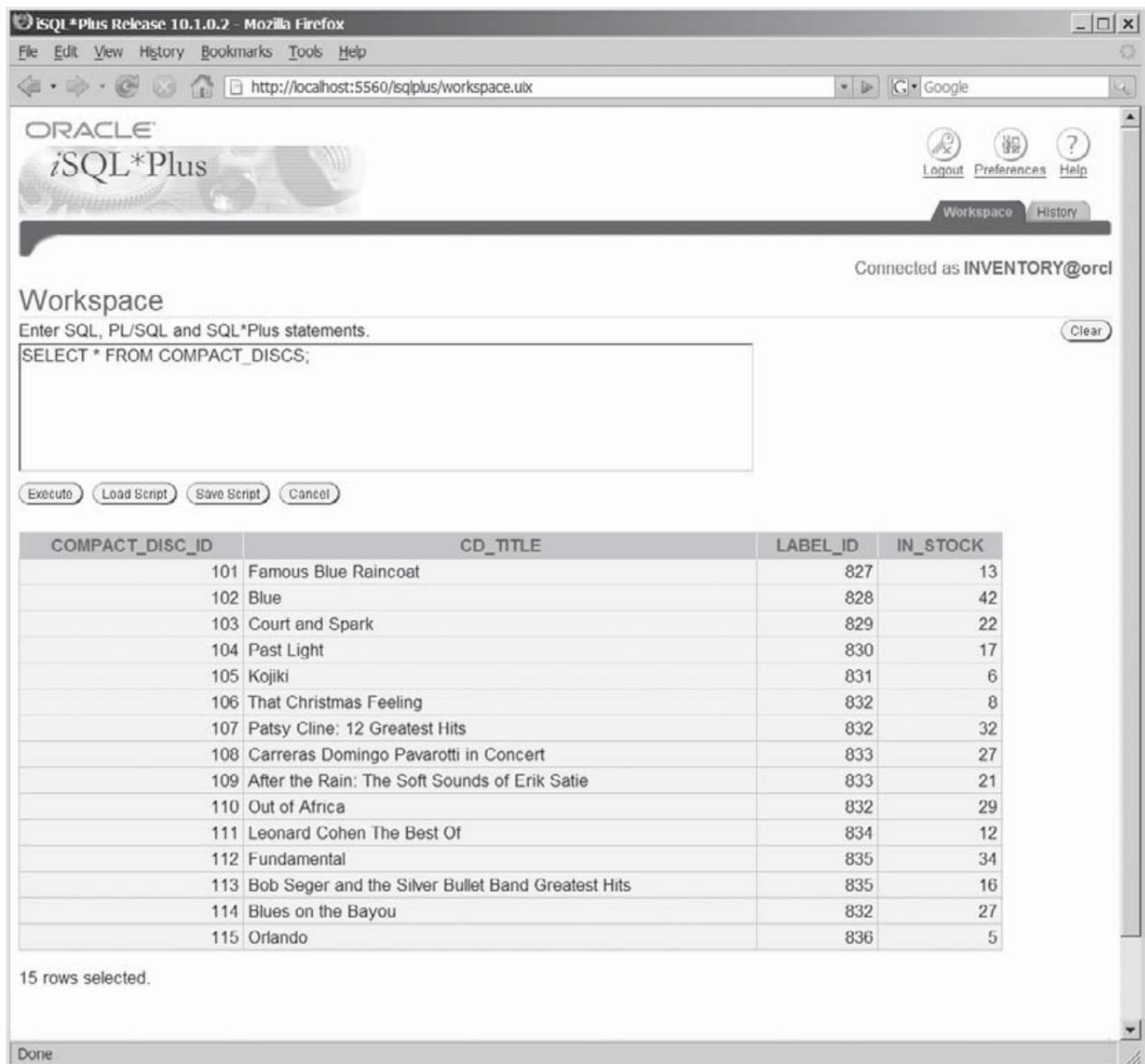


Рисунок 1-8 Використання інтерфейсу Oracle's iSQL*Plus

```
MySQL Command Line Client

mysql> use inventory
Database changed
mysql> SELECT * FROM COMPACT_DISCS;
+-----+-----+-----+-----+
| COMPACT_DISC_ID | CD_TITLE                                     | LABEL_I |
| IN_STOCK |
+-----+-----+-----+-----+
| 101 | Famous Blue Raincoat | 82 |
| 13 | Blue | 82 |
| 42 | Court and Spark | 82 |
| 22 | Past Light | 83 |
| 17 | Kojiki | 83 |
| 6 | That Christmas Feeling | 83 |
| 8 | Patsy Cline: 12 Greatest Hits | 83 |
| 32 | Carreras Domingo Pavarotti in Concert | 83 |
| 27 | After the Rain: The Soft Sounds of Erik Satie | 83 |
| 21 | Out of Africa | 83 |
| 29 | Leonard Cohen The Best Of | 83 |
| 12 | Fundamental | 83 |
| 34 | Bob Seger and the Silver Bullet Band Greatest Hits | 83 |
| 16 | Blues on the Bayou | 83 |
| 27 | Orlando | 83 |
| 5 |  |  |
+-----+-----+-----+-----+
15 rows in set (0.00 sec)

mysql>
mysql>
mysql>
mysql>
mysql>
```

Рисунок 1-9 Використання інтерфейсу командного рядка MySQL