Лекція 26. Захист даних у SQL

Під час збереження даних в будь-якій СУБД одним з головних завдань користувача є забезпечення безпеки цих даних. Проблема безпеки особливо гостро стоїть у реляційних базах даних, оскільки завдяки інтерактивності SQL дозволяє легко отримати до них доступ. Вимоги до системи безпеки в типовій реляційній базі даних досить різноманітні.

• Доступ до даних окремої таблиці має бути дозволено одним користувачам і заборонено іншим.

• Одним користувачам має бути дозволено змінювати дані у деякій таблиці, а іншим – лише отримувати вибірку даних з неї.

• В деяких таблицях доступ має здійснюватися лише до окремих стовпців.

• Певним користувачам має бути заборонено звертатися до деяких таблиць за допомогою інтерактивного SQL, але дозволено користуватися застосунками, що змінюють цю таблицю.

26.1 Принципи захисту даних в SQL

За реалізацію системи безпеки відповідає програмне забезпечення СУБД. Мова SQL є фундаментом системи безпеки реляційної СУБД: вимоги до захисту інформації в базі даних формулюються за допомогою інструкцій SQL.

У SQL існують три основні концепції, пов'язані з захистом даних.

• *Користувачі* є дійовими особами, які працюють з базою даних. Кожного разу, коли СУБД отримує, додає, видаляє або оновлює дані, вона робить це від імені певного користувача. СУБД виконає потрібну дію або відмовиться від її виконання в залежності від того, який користувач запитує цю дію.

• *Об'єкти* бази даних – це елементи, захист яких може здійснюватися за допомогою SQL. Зазвичай захистом об'єктів бази даних є захист таблиць та представлень, але інші об'єкти, такі як форми, прикладні програми та бази даних цілком, також можуть бути захищені.

Більшості користувачів дозволяється використовувати деякі об'єкти бази даних, а інші заборонені.

• *Привілеї* – це права користувача на проведення тих або інших дій над певним об'єктом бази даних. Наприклад, користувачу може бути дозволено отримувати дані з рядків деякої таблиці та додавати їх до неї, але заборонено видаляти або оновлювати рядки цієї таблиці. У іншого користувача може бути інший набір привілеїв.

На рисунку 26.1 представлено схему того, як наведені принципи можуть бути застосовані до БД.

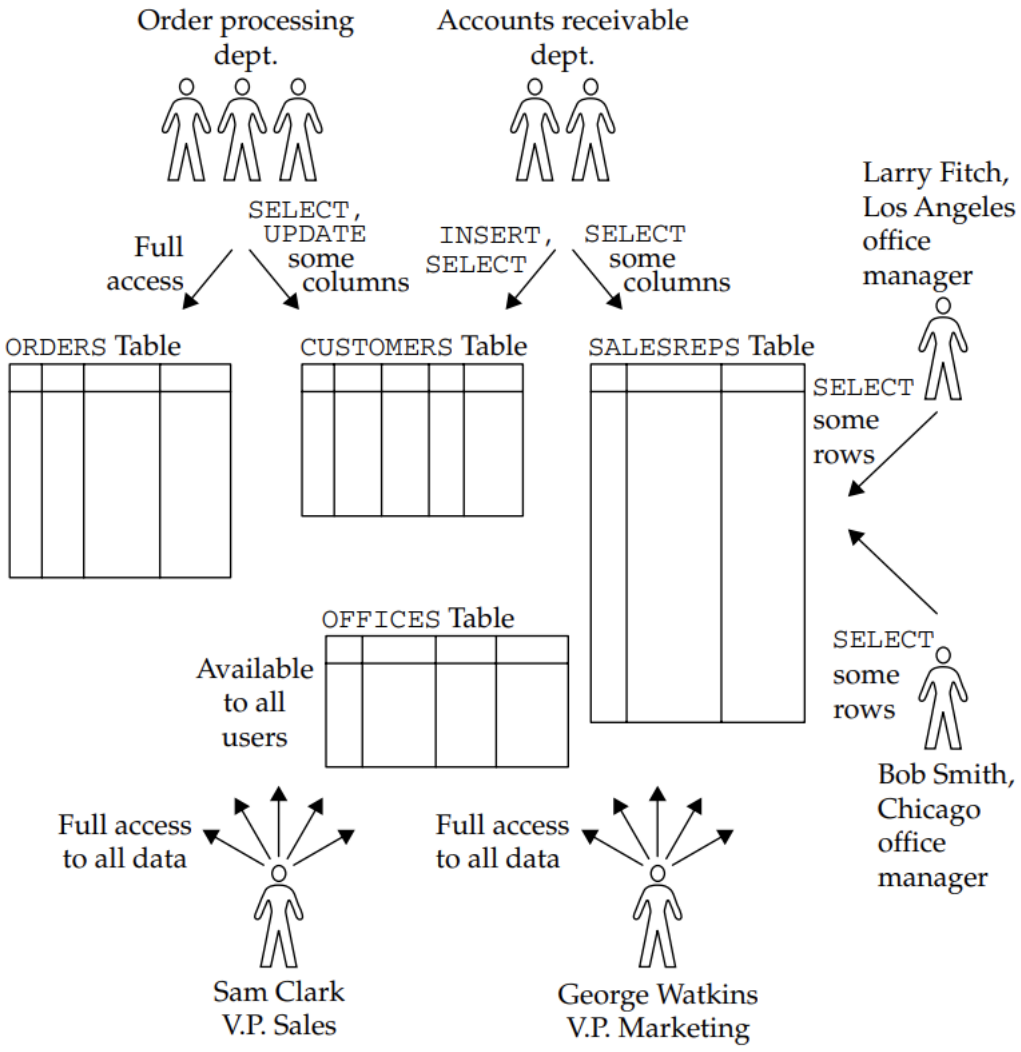

Рисунок 26.1 – Схема безпеки БД

Для введення елементів системи безпеки використовується інструкція GRANT, за допомогою якої певним користувачам надаються

певні привілеї на використання тих або інших об'єктів бази даних. Ось, наприклад, інструкція GRANT, яка дозволяє Сему Кларку (Sam Clark) отримувати дані з таблиці OFFICES та додавати дані до цієї таблиці.

```
GRANT SELECT , INSERT
ON OFFICES
TO SAM;
```

В інструкції GRANT встановлюється комбінація ідентифікатора користувача (SAM), об'єкта (таблиця OFFICES) та привілеїв (SELECT та INSERT). Надані привілеї можна відкликати пізніше за допомогою інструкції REVOKE.

```
REVOKE SELECT , INSERT
ON OFFICES
FROM SAM ;
```

26.2 Ідентифікатори користувачів

Кожному користувачеві в реляційній базі даних присвоюється ідентифікатор – коротке ім'я, яке однозначно визначає користувача для програмного забезпечення СУБД. Такі ідентифікатори є основою системи безпеки. Кожна інструкція SQL виконується в СУБД від імені конкретного користувача. Від його ідентифікатора залежить, чи буде дозволено чи заборонено виконання конкретної інструкції. У промисловій базі даних ідентифікатор користувача призначається її адміністратором. У базі даних на персональному комп'ютері може бути тільки один ідентифікатор користувача, який позначає користувача, який створив базу даних та є її власником.

На практиці обмеження на імена ідентифікаторів залежать від реалізації СУБД. Стандарт SQL1 дозволяв ідентифікатору користувача мати довжину до 18 символів та вимагав, щоб він був допустимим SQL-ім'ям. У деяких СУБД для мейнфреймів довжина ідентифікаторів обмежувалась восьмома символами. У Sybase та SQL Server ідентифікатор користувача може мати до 30 символів. Якщо важлива

переносимість, то краще, щоб у ідентифікатора користувача було не більше восьми символів. На рис. 26.2 показано різні користувачі, які працюють з базою даних, та типові ідентифікатори, які їм присвоєно. Зверніть увагу на те, що всім користувачам в відділі обробки замовлень можна присвоїти один і той же ідентифікатор, оскільки всі вони мають ідентичні привілеї.

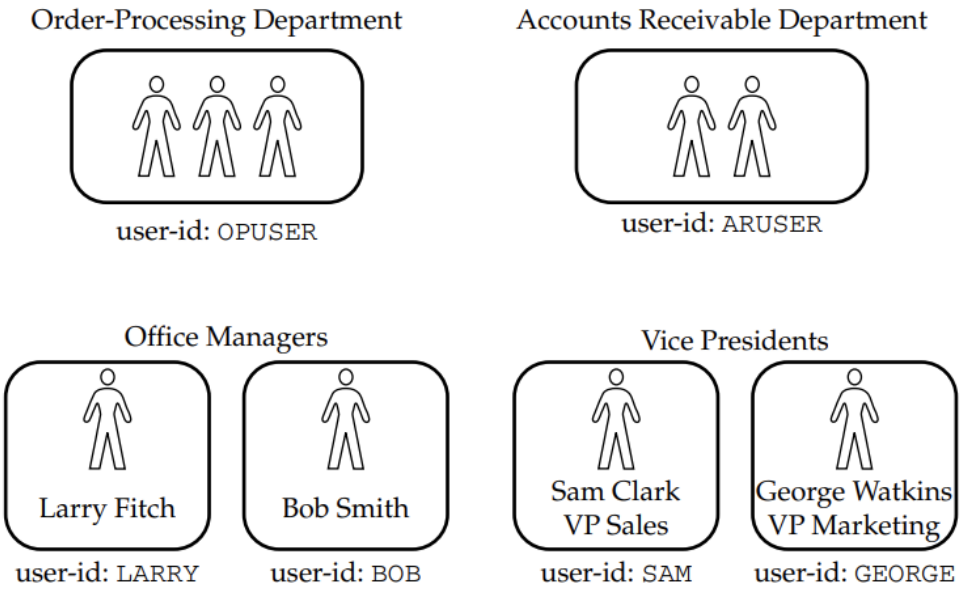

Рисунок 26.2 – Ідентифікатори користувачів

У стандарті ANSI/ISO замість терміна "ідентифікатор користувача" (user-id) використовується термін "ідентифікатор авторизації" (authorization-id), цей термін іноді зустрічається в документації з SQL. З технічної точки зору другий термін є більш правильним, оскільки роль ідентифікатора полягає в визначенні прав або привілеїв. Бувають ситуації (як на рис. 26.2), коли має сенс присвоїти декільком користувачам однаковий ідентифікатор. У інших ситуаціях один користувач може користуватися двома або трьома різними ідентифікаторами. У промисловій базі даних ідентифікатор прав доступу може відноситися до програм і груп програм, а не до окремих людей. У кожній з цих ситуацій термін "ідентифікатор авторизації" є більш точним і зрозумілим, ніж термін "ідентифікатор користувача". Однак найбільш поширена ситуація, коли кожному користувачу

присвоюється свій ідентифікатор, тому в документації більшості реляційних СУБД використовується термін "ідентифікатор користувача".

26.3 Аутентифікація користувачів

Згідно зі стандартом SQL, безпека бази даних забезпечується за допомогою ідентифікаторів користувачів. Однак у стандарті нічого не говориться про механізм зв'язку ідентифікатора користувача з інструкціями SQL. Наприклад, якщо інструкції SQL вводяться в інтерактивному режимі, яким чином СУБД визначить, з яким ідентифікатором користувача пов'язані ці інструкції? Інший приклад, на сервері баз даних може існувати програма, яка щоденно генерує звіти, тоді який в цьому випадку ідентифікатор користувача, якщо немає жодного користувача? Нарешті, як обробляються запити до бази даних по мережі, якщо на локальному комп'ютері користувач може мати один ідентифікатор, а на віддаленому комп'ютері - інший?

У більшості комерційних реляційних СУБД ідентифікатор користувача створюється для кожного *сеансу* (сесії) зв'язку з базою даних. У інтерактивному режимі сеанс починається, коли користувач запускає інтерактивну програму формування запитів, і триває до тих пір, поки користувач не вийде з програми або не введе команду зміни користувача. У застосунку, який використовує програмний SQL, сеанс починається, коли додаток підключається до СУБД, і закінчується при завершенні додатку. Протягом сеансу всі інструкції SQL асоціюються з ідентифікатором користувача, встановленим для цього сеансу. Однак у сучасних прикладних системах можливо також, що програма встановлює декілька з'єднань з базою даних і вибирає одне з них для передачі інструкцій SQL.

Зазвичай на початку сеансу необхідно ввести як ідентифікатор користувача, так і пов'язаний з ним пароль. Пароль слугує для підтвердження того, що користувач дійсно має право працювати під введеним ідентифікатором. Однак деякі СУБД підтримують аутентифікацію операційної системи, тобто СУБД отримує

ідентифікатор користувача від операційної системи, без необхідності введення пароля або виконання іншої аутентифікації.

Хоча ідентифікатори і паролі застосовуються в більшості реляційних СУБД, спосіб, яким користувач вводить свій ідентифікатор і пароль, залежить від конкретної СУБД. У деяких СУБД, особливо тих, що доступні для різних операційних систем, реалізована власна система безпеки з ідентифікаторами користувачів і паролями. Наприклад, коли в СУБД Oracle ви працюєте з інтерактивною SQL-програмою SQLPLUS, можете вказати ім'я користувача та відповідний пароль в командному рядку.

```
SQLPLUS SCOTT/TIGER
```

Зазначимо, що використання такого введення пароля в командному рядку не рекомендується, оскільки в такому випадку пароль не шифрується і може бути перехоплений будь-ким в системі. Набагато краще не вводити пароль (і роздільник "/") і дозволити SQLPLUS самому запитати пароль у вас. Інтерактивний SQL-модуль СУБД Sybase, який називається ISQL, також може приймати ім'я користувача та пароль у командному рядку.

```
ISQL -U SCOTT -P TIGER
```

У будь-якому випадку, перед тим, як розпочати інтерактивну сесію з'єднання, СУБД перевіряє достовірність ідентифікатора користувача (наприклад, SCOTT) і пароля (наприклад, TIGER). Так само, краще опустити пароль і дозволити ISQL запитати його вас самостійно. Старі версії SQL Server також підтримують ISQL, але новіші використовують інший інструмент OSQL для доступу до командного рядка СУБД.

У багатьох інших СУБД, таких як Ingres та Informix, ідентифікатори користувачів є іменами користувачів, зареєстрованими в операційній системі. Наприклад, коли ви реєструєтесь в системі на основі UNIX, ви вводите ім'я користувача та пароль. Для запуску інтерактивного модуля в СУБД Ingres ви просто вводите команду.

ISQL SALESDB

У створеному вами сеансі роботи з базою даних Ingres, ім'я якої SALESDB, Ingres автоматично отримує ваше ім'я користувача UNIX і робить його ідентифікатором користувача в поточному сеансі роботи з СУБД. Таким чином, вам не потрібно вводити окремий ідентифікатор користувача та пароль для бази даних. Аналогічний підхід використовується і в інтерактивному SQL-модулі СУБД DB2, який працює в операційній системі MVS/TSO. Обліковий запис TSO автоматично стає ідентифікатором користувача інтерактивного SQL-сеансу.

Більшість сучасних СУБД мають програми з графічним інтерфейсом користувача для доступу до бази даних, такі як SQL Server Management Studio, DB2 UDB Command Editor або Oracle SQL Developer. Ці інструменти запитують ідентифікатор користувача та пароль під час підключення до бази даних. Ті ж самі заходи захисту використовуються за програмного доступу до бази даних, так що СУБД повинна визначати та аутентифікувати ідентифікатор користувача для кожної прикладної програми, яка намагається отримати доступ до бази даних. В цьому випадку також способи та правила використання ідентифікаторів користувачів змінюються залежно від СУБД, яка використовується. Зазвичай програма на початку сеансу видає користувачеві діалогове вікно із запитом ідентифікатора та пароля. Спеціалізовані або написані користувачами програми можуть використовувати ідентифікатор, жорстко закодований в програмі.

Стандарт SQL дозволяє програмі використовувати ідентифікатор авторизації, пов'язаний з конкретним набором інструкцій SQL (такий набір називається модулем), а не ідентифікатор конкретного користувача, який запустив програму. Такий механізм забезпечує можливість виконання програмою різних завдань від імені різних користувачів, навіть якщо при інших умовах цим користувачам заборонено доступ до відповідної інформації. Це зручна можливість, яка знаходить своє застосування в основних реалізаціях SQL.

26.4 Групи користувачів

У великих виробничих базах даних часто є групи користувачів зі схожими завданнями. Наприклад, в наведеній базі даних троє людей у відділі обробки замовлень утворюють таку групу користувачів зі спільними завданнями та правами доступу; двоє людей у фінансовому відділі утворюють іншу таку групу. У межах кожної групи всі користувачі працюють з однаковими даними і повинні мати ідентичні привілеї.

Згідно зі стандартом ANSI/ISO, групи користувачів можна утворити одним із трьох способів.

1 Кожному члену групи можна присвоїти один і той же ідентифікатор користувача, як показано на рисунку 26.2. Це спрощує управління системою безпеки, оскільки дозволяє встановити привілеї доступу до даних один раз (тому що ідентифікатор користувача один). Проте в такому випадку людей, які спільно використовують один ідентифікатор користувача, не можна буде відрізнити або ідентифікувати на дисплеї системного оператора та в звітах СУБД.

2 Всім членам групи можна присвоїти різні ідентифікатори користувача. Це дозволить вам розрізняти користувачів у звітах СУБД та встановлювати в подальшому різні привілеї для окремих користувачів. Проте привілеї доведеться встановлювати для кожного користувача окремо, що може бути трудомістким та збільшувати ймовірність виникнення помилок.

3 У найсучасніших СУБД, які підтримують цю можливість, можна створити роль, що містить необхідні привілеї. Роль представляє собою набір привілеїв об’єднаних одною назвою. Можна призначити кожному користувачеві його власний ідентифікатор та пов'язати з ним певну роль. Очевидно, що це найкращий вибір, оскільки можна відрізняти користувачів, не ускладнюючи при цьому адміністрування.

Вибір способу залежить від того, які компроміси допускаються в базі даних та прикладних програмах.

До впровадження підтримки ролей в Sybase та SQL Server був можливий третій варіант відносно груп користувачів. У цих СУБД існують ідентифікатори груп, які складаються з користувачів, пов'язаних якимось чином. Привілеї можуть бути надані як користувачам, що мають

персональні ідентифікатори, так і групам, які також мають власні ідентифікатори. В цьому випадку користувачі можуть здійснювати дії, дозволені як привілеями групи так і персональними привілеями. Таким чином, ідентифікатори груп спрощують управління системою безпеки. Однак вони не відповідають стандарту, і бази даних, в яких вони використовуються, можуть бути не переносими до іншої СУБД.

Деякі версії DB2 також підтримують групи користувачів, але іншим способом. Адміністратор бази даних DB2 може налаштувати її таким чином, що коли ви вперше підключаєтеся до DB2 та повідомляєте свій ідентифікатор користувача (відомий як ваш первинний ідентифікатор авторизації), DB2 автоматично шукає набір додаткових ідентифікаторів користувача (відомих як вторинні ідентифікатори авторизації), якими ви можете користуватися. Коли DB2 перевіряє ваші привілеї пізніше, вона перевіряє привілеї для всіх ідентифікаторів доступу – як первинних, так і вторинних. Адміністратор бази даних DB2 зазвичай встановлює вторинні ідентифікатори доступу, які збігаються з іменами груп користувачів і використовуються в RACF (Resource Access Control Facility, засіб керування доступом до ресурсів – засіб безпеки для мейнфреймів компанії IBM). Таким чином, використовуваний в DB2 підхід фактично вводить ідентифікатори груп, не розширюючи явно механізм ідентифікаторів користувача.

26.5 Об’єкти захисту

Системи захисту в SQL застосовуються щодо окремих об'єктів бази даних. У стандарті SQL-1 зазначені два типи об'єктів, які можна захистити – таблиці та представлення. Таким чином, кожна таблиця та кожне представлення можуть бути захищені індивідуально. Доступ до них може бути дозволений для одних користувачів та заборонений для інших. Пізніші версії стандарту SQL розширюють коло об'єктів захисту, включаючи до нього домени та користувацькі набори символів, а також додають нові типи захисту таблиць та представлень.

У більшості комерційних СУБД додатково можуть бути захищені інші типи об'єктів. Наприклад, у SQL Server важливим об'єктом захисту у базі даних є збережена процедура. За допомогою засобів захисту SQL

встановлюється, які користувачі можуть створювати та видаляти збережені процедури, а які користувачі можуть їх виконувати. У DB2 об'єктами захисту є табличні простори (physical tablespaces) – фізичні області зберігання таблиць. Адміністратор бази даних може для одних користувачів дозволяти створення нових таблиць у певній фізичній області, а для інших – забороняти. Інші реляційні СУБД можуть захищати інші об'єкти. В цілому, базовий механізм забезпечення безпеки в SQL – надання та скасування привілеїв на певні об'єкти за допомогою визначених інструкцій SQL. Такий механізм безпеки реалізований практично в усіх СУБД однаковим чином.

26.6 Привілеї

Привілеї користувача щодо даного об'єкту бази даних – це набір дій, які користувач має право виконувати з цим об'єктом. У стандарті SQL-1 для таблиць та представлень визначено чотири привілеї:

• Привілея SELECT дозволяє вибирати дані з таблиці або представлення. Маючи цей привілеї, можна задавати ім'я таблиці або представлення в розділі FROM інструкції SELECT або підзапиту.

• Привілея INSERT дозволяє додавати нові записи в таблицю або представлення. Маючи цю привілею, можна задавати ім'я таблиці або представлення в розділі INTO інструкції INSERT.

• Привілея DELETE дозволяє видаляти записи з таблиці або представлення. Маючи цю привілею, можна задавати ім'я таблиці або представлення в розділі FROM інструкції DELETE.

• Привілея UPDATE дозволяє змінювати записи в таблиці або представленні. Маючи цю привілею, можна задавати таблицю або представлення в інструкції UPDATE як цільову таблицю. Привілея UPDATE може бути обмежена окремими стовпцями таблиці або представлення, дозволяючи тим самим оновлювати тільки ці стовпці та забороняючи оновлювати інші.

Ці чотири привілеї підтримуються майже всіма комерційними реалізаціями SQL.

Розширені привілеї

У наступних версіях стандарту SQL базовий механізм керування привілеями стандарту SQL 1 значно розширено. До привілеїв SELECT, INSERT та UPDATE, які були визначені в SQL 1, додані нові можливості. З'явилась нова привілея REFERENCES, яка обмежує право користувача на створення зовнішніх ключів доступної йому таблиці для посилання на інші таблиці. Крім того, додано нову привілею USAGE, яка керує доступом до нових структур баз даних SQL – доменів, наборів символів, порядків сортування та правил конвертування тексту.

Розширення набули також привілеї SELECT, INSERT та UPDATE. Тепер вони можуть стосуватися конкретного стовпця або стовпців таблиці, а не тільки до всієї таблиці в цілому. Розглянемо їх корисність на прикладі бази даних.

Припустимо, ви хочете, щоб керівник відділу кадрів відповідав за додавання нових співробітників до таблиці SALESREPS. Він має вводити ідентифікатор прийнятого на роботу співробітника, його ім'я та іншу облікову інформацію. Однак за призначення новому співробітнику планового обсягу продаж, тобто за заповнення стовпця QUOTA, відповідає не він, а керівник відділу продажів.

Аналогічно, керівник відділу кадрів не повинен мати доступ до стовпця SALES наявних записів таблиці SALESREPS. Використовуючи нові можливості стандарту SQL, можна реалізувати цю схему, надавши керівникові відділу кадрів привілеї INSERT для відповідних стовпців. Інші стовпці (SALES і QUOTA) при створенні нового запису прийматимуть значення NULL. Керівник відділу продажів отримує привілеї UPDATE для стовпців SALES і QUOTA, щоб він міг призначати співробітникам планові обсяги продажу. Якщо не можна вказувати привілеї для конкретних стовпців, то доведеться або послаблювати згадані обмеження на доступ до окремих стовпців, або визначати додаткові представлення, лише для обмеження доступу.

Введення нової привілеї REFERENCES пов'язане з більш тонкою особливістю системи безпеки баз даних, що стосується визначення зовнішніх ключів та обмежень на значення стовпців. Проаналізуємо це на прикладі бази даних.

Припустимо, що електронний магазин компанії може створювати в базі даних нові таблиці (наприклад, йому може знадобитись таблиця з інформацією про новий товар), але співробітник магазину не має доступу до інформації про співробітників в таблиці SALESREPS. За такої схеми захисту може виявитися, що співробітник магазину ніяк не зможе дізнатися використовувані компанією ідентифікатори співробітників або дізнатися, чи був прийнятий на роботу новий співробітник.

Проте це не зовсім правильно. Службовець може створити нову таблицю та визначити один з її стовпців як зовнішній ключ для зв'язку з таблицею SALESREPS. Як ви пам'ятаєте, це означає, що допустимими значеннями цього стовпця будуть лише значення первинного ключа з таблиці SALESREPS, тобто допустимі ідентифікатори службовців. Щоб дізнатися, чи працює в компанії службовець з певним ідентифікатором, співробітнику магазину достатньо спробувати додати в свою таблицю новий рядок з цим значенням зовнішнього ключа. Якщо це вдасться, то такий службовець у компанії є, якщо ні – то службовця з таким ідентифікатором в компанії немає.

Ще більш серйозною проблемою є можливість створення нових таблиць з обмеженнями на значення стовпців. Припустимо, наприклад, що службовець намагається виконати таку інструкцію.

```
CREATE TABLE XYZ ( TRYIT DECIMAL ( 9,2 ) ,
        CHECK ( ( SELECT QUOTA
                        FROM SALESREPS
                WHERE NAME = ' VP Sales ')
BETWEEN 400000 AND 500000) );
```

Оскільки значення стовпця CHECK у створеній співробітником таблиці тепер пов'язане зі значенням стовпця QUOTA конкретного рядка таблиці SALESREPS, успішне виконання наведеної інструкції означає, що квота віце-президента по збуту знаходиться в зазначеному діапазоні. В іншому випадку можна спробувати аналогічну інструкцію, скоригувавши межі діапазону.

Зазначимо, однак, що тільки деякі реалізації SQL підтримують перевірку звернень до інших таблиць.

Цю можливість доступу до даних забезпечує введена в стандарт SQL привілея REFERENCES. Подібно до привілеїв SELECT, INSERT і UPDATE, вона призначається для окремих стовпців таблиці. Якщо у користувача є привілея REFERENCES для деякого стовпця, він може створювати нові таблиці, пов'язані з цим стовпцем якимось чином (наприклад, через зовнішній ключ або обмеження на значення стовпців, як у прикладі). Якщо СУБД не підтримує привілею REFERENCES, але підтримує зовнішні ключі та обмеження на значення стовпців, то іноді цю ж задачу може виконати привілея SELECT.

Стандарт SQL визначає нову привілею USAGE, яка керує доступом до доменів (наборів допустимих значень стовпців), користувацьких наборів символів, порядків сортування та правил конвертації тексту. Привілея USAGE просто дозволяє або забороняє використання цих об'єктів бази даних за назвою для окремих ідентифікаторів користувачів. Наприклад, маючи привілею USAGE на деякий домен, можна визначити нову таблицю та вказати цей домен як тип даних якогось з її стовпців. Без такої привілеї використовувати цей домен для визначення стовпців неможливо. Привілея USAGE служить, передусім, для спрощення адміністрування великих корпоративних баз даних, які використовуються та модифікуються декількома різними командами розробників.

Привілеї володіння

Під час створення таблиці за допомогою інструкції CREATE TABLE, ви стаєте її власником і отримуєте всі привілеї для цієї таблиці (SELECT, INSERT, DELETE, UPDATE та інші привілеї, які є у СУБД). Всі інші користувачі початково не мають жодних привілеїв для роботи з новоствореною таблицею. Щоб вони отримали доступ до таблиці, ви повинні явно надати їм відповідні привілеї за допомогою інструкції GRANT.

Так само за створення представлення за допомогою інструкції CREATE VIEW, ви стаєте його власником, але не обов'язково отримуєте щодо нього всі привілеї. Для успішного створення представлення

необхідно мати привілеї SELECT для кожної початкової таблиці представлення, тому привілею SELECT для роботи зі створеним представленням ви отримуєте автоматично. Щодо інших привілеїв (INSERT, DELETE та UPDATE), то ви отримуєте їх тільки у випадку, якщо мали їх для кожної початкової таблиці представлення.

Інші привілеї

У багатьох комерційних СУБД, крім привілеїв SELECT, INSERT, DELETE та UPDATE, щодо таблиць та представлень можуть бути надані додаткові привілеї. Наприклад, у Oracle та базах даних для мейнфреймів компанії IBM передбачені привілеї ALTER та INDEX. Маючи привілею ALTER для деякої таблиці, користувач може за допомогою інструкції ALTER TABLE модифікувати структуру цієї таблиці; маючи привілею INDEX, користувач може за допомогою інструкції CREATE INDEX створити індекс для таблиці.

У тих СУБД, де відсутні привілеї ALTER та INDEX, тільки власник таблиці може користуватися інструкціями ALTER TABLE та CREATE INDEX.

Часто в СУБД є додаткові привілеї не для таблиць та представлень, а для інших об'єктів. Наприклад, Oracle, Sybase та SQL Server підтримують привілею EXECUTE для збережених процедур, яка визначає, хто з користувачів має право її виконувати. У DB2 є привілея USE для табличних просторів, яка визначає, хто з користувачів може створювати таблиці в певному табличному просторі.

26.7 Представлення та безпека

Разом з привілеями, які обмежують доступ до таблиць, представлення також відіграють ключову роль у захисті даних. Створюючи представлення та дозволяючи користувачеві доступ до нього, а не до початкової таблиці, можна обмежити доступ користувача, дозволивши йому звертатися лише до певних стовпців та рядків. Таким чином, представлення дозволяють здійснювати чіткий контроль над тим, які дані доступні для того чи іншого користувача.

Допустимо, наприклад, що ви вирішили встановити наступне правило захисту даних в базі даних:

"*Персонал фінансового відділу має право отримувати з таблиці SALESREPS ідентифікатори та імена співробітників, а також ідентифікатори офісів; інформація про обсяг продажів та особисті плани повинні бути для них недоступні*".

Це правило можна реалізувати, створивши представлення:

```
CREATE VIEW REPINFO AS
        SELECT EMPL_NUM , NAME , REP_OFFICE
                FROM SALESREPS ;
```

та назначив привілеї SELECT для цього представлення користувачеві з ідентифікатором ARUSER (рис. 26.3). У даному прикладі для обмеження доступу до окремих стовпців використовується вертикальне представлення.

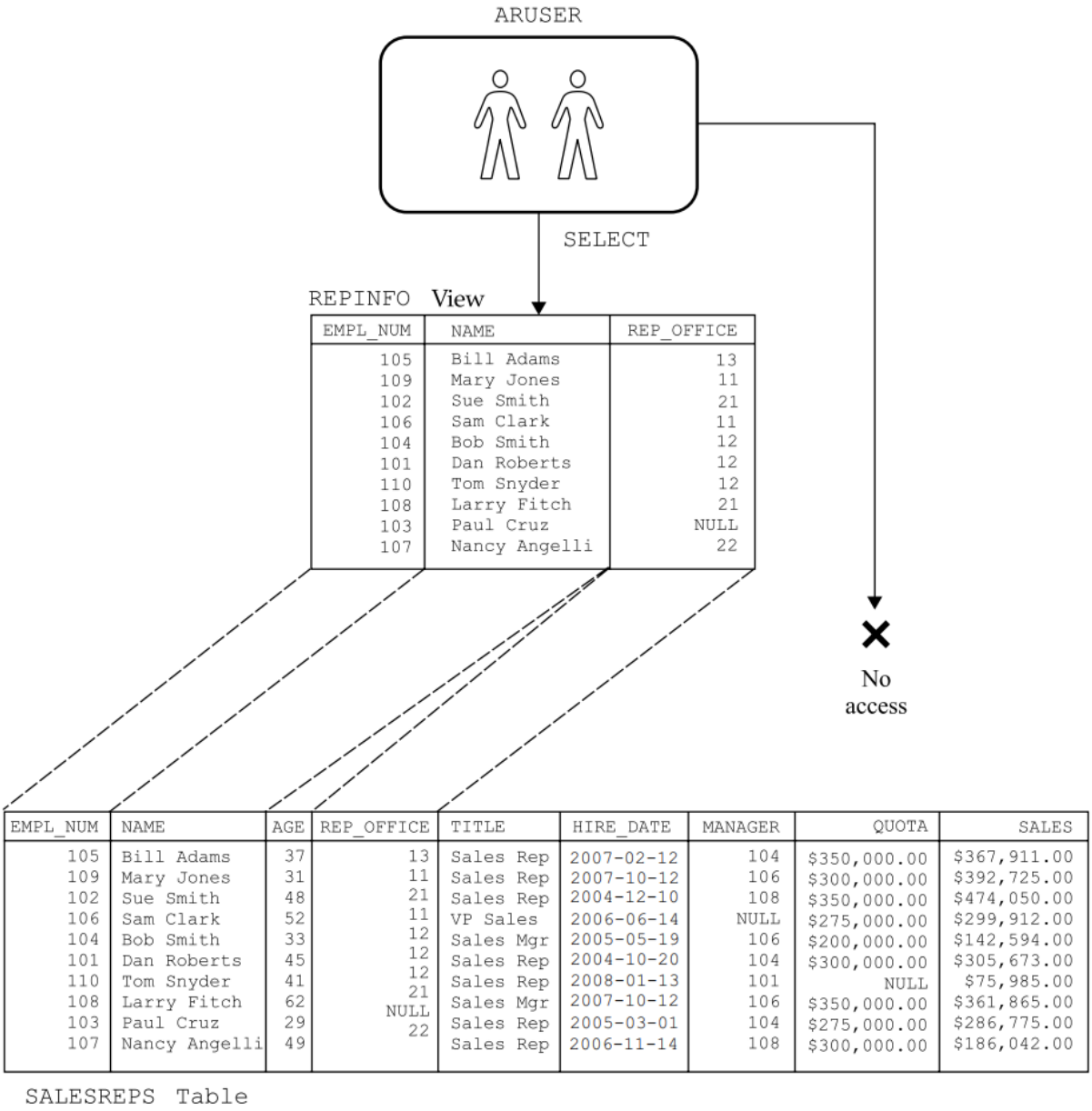

Рисунок 26.3 – Обмеження доступу до стовпців за допомогою представлення

За допомогою горизонтальних представлень також можна ефективно реалізувати правила захисту даних. Розглянемо наступне правило:

"*Менеджери з продажу кожного регіону повинні мати доступ до всіх даних, що стосуються співробітників лише їх регіону*".

Як показано на рисунку 26.4, можна створити два представлення – EASTREPS та WESTREPS, в яких містяться дані таблиці SALESREPS

окремо для кожного регіону, а потім надати дозвіл на доступ до відповідного представлення для кожного менеджера з продажу.

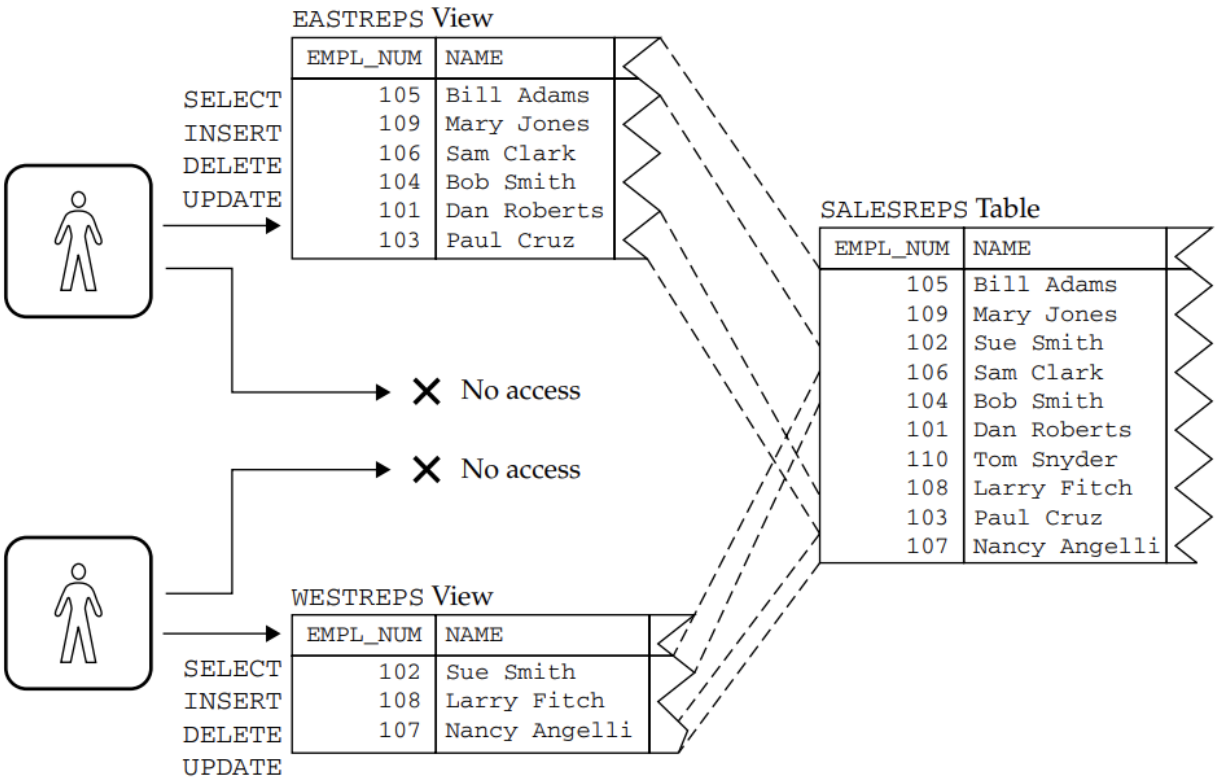

Рисунок 26.4 – Обмеження доступу до стовпців за допомогою представлення

Звичайно, представлення можуть бути набагато складнішими, ніж просто підмножина рядків і стовпців однієї таблиці, як у наведеному прикладі. Створюючи представлення за допомогою запиту з групуванням, можна надати користувачеві доступ до підсумкових даних, а не до детальних даних вихідної таблиці. В представленні можна об'єднувати дані з двох або більше таблиць, дозволяючи користувачу доступ саме до тих даних, які йому потрібні, і забороняючи доступ до всіх інших.

Обмеження для оновлення даних. Якщо представлення доступне лише для вибірки, то для нього можна встановити привілеї SELECT, але привілеї INSERT, DELETE та UPDATE для нього не доступні. Якщо користувач має оновлювати дані, що відображаються в доступному лише для вибірки представленні, йому слід надати дозвіл

на оновлення початковиих таблиць і він повинен мати можливість виконувати інструкції INSERT, DELETE та UPDATE стосовно цих таблиць.

Продуктивність. Оскільки СУБД перетворює кожне звернення до представлення на відповідні звернення до початкових таблиць, за використання представлень може значно збільшуватись складність операцій, які відтворюються з представленнями в базі даних. Тому представлення не можна легковажно застосовувати для обмеження доступу до даних – це неминуче призведе до зниження загальної продуктивності СУБД.

26.8 Надання привілеїв GRANT

Інструкція GRANT використовується для надання користувачам привілеїв під час роботи з об'єктами бази даних. Зазвичай інструкцією GRANT користуються власники таблиці або представлення, щоб надати іншим користувачам доступ до цих об’єктів. Як видно з рисунку 26.5, інструкція GRANT містить:

1 список наданих привілеїв,

2 ім'я таблиці або іншого об'єкту, для якого назначаються привілеї (для всіх об'єктів, крім таблиць та представлень, потрібно вказати тип об'єкта),

3 ідентифікатор користувача або ролі, яка отримує ці привілеї.

У більшості реалізацій SQL для надання привілеїв користувачеві його обліковий запис має вже бути в базі даних.

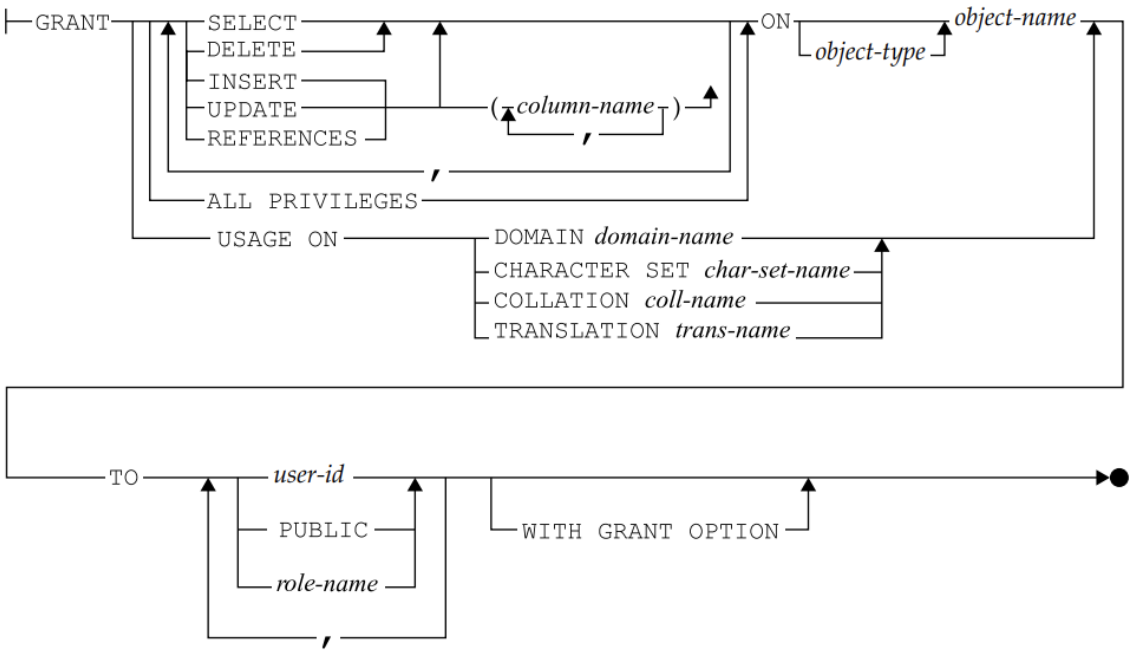

Рисунок 26.5 – Синтаксична діаграма інструкції GRANT

Синтаксична діаграма інструкції GRANT, зображена на рисунку, відповідає стандарту ANSI/150. Багато СУБД дотримуються синтаксису інструкції GRANT, який застосовується в DB2, який є гнучкішим і дозволяє задавати список ідентифікаторів користувачів і список таблиць, спрощуючи надання багатьох привілеїв. Наведемо декілька прикладів простих інструкцій GRANT для бази даних.

Надати користувачам з відділу обробки замовлень повний доступ до таблиці ORDERS.

```
GRANT SELECT , INSERT , DELETE , UPDATE
    ON ORDERS
    TO OPUSER ;
```

Дозволити користувачам фінансового відділу отримувати дані про замовників та додавати нових замовників до таблиці CUSTOMERS; користувачам відділу обробки замовлень дозволити лише вибірку з цієї таблиці.

```
GRANT SELECT , INSERT
    ON CUSTOMERS
    TO ARUSER ;
```

```
GRANT SELECT
    ON CUSTOMERS
    TO OPUSER ;
```

Дозволити Сэму Кларку додавати та видаляти інформацію про офіси.

```
GRANT INSERT , DELETE
    ON OFFI CES
    TO SAM;
```

При наданні великої кількості привілеїв або наданні привілеїв великій кількості користувачів в інструкції GRANT для зручності застосовуються два скорочення. Замість перерахування кожного привілею для певного об'єкта можна використати фразу ALL PRIVILEGES. Наведена нижче інструкція GRANT дозволяє віце-президенту з продажу Сему Кларку повний доступ до таблиці SALESREPS.

Надати віце-президенту Сему Кларку всі привелеї стосовно таблиці SALESREPS.

```
GRANT ALL PRIVI LEGES
    ON SALESREPS
    TO SAM;
```

Замість того, щоб надавати привілеї окремо кожному користувачу бази даних, можна використовувати ключове слово PUBLIC, щоб надати їх одразу всім користувачам, які мають право працювати з базою даних. Очевидно, що цією можливістю слід користуватися дуже

обережно. Наступна інструкція GRANT дозволяє всім користувачам отримувати дані з таблиці OFFICES.

Надати всім користувачам привілею SELECT стосовно таблиці OFFICES.

```
GRANT SELECT
    ON OFFI CES
    TO PUBLI C;
```

Зверніть увагу: така інструкція GRANT надає привілеї всім поточним і майбутнім користувачам бази даних, а не лише тим, хто має право працювати з базою даних в даний момент. Це усуває необхідність явно надавати привілегії новим користувачам під час їх появи.

Привілеї для роботи зі стовпцями

Стандарт SQL1 дозволяв надавати привілею UPDATE для окремих стовпців таблиці або представлення, а новіші версії стандарту надають можливість надавати так само привілеї SELECT, INSERT і REFERENCES.

Список стовпців розташовується після ключового слова SELECT, UPDATE, INSERT або REFERENCES і розміщується у круглих дужках. Наведемо вигляд інструкції GRANT, яка дозволяє співробітникам відділу обробки замовлень оновлювати лише стовпці з назвою компанії (COMPANY) та ідентифікатором прикріпленого за нею представника (CUST_REP) в таблиці CUSTOMERS.

Дозволити користувачам відділу обробки замовлень змінювати назви компаній, а також прикріплених до компаній представників.

```
GRANT UPDATE ( COMPANY , CUST_REP)
    ON CUSTOMERS
    TO OPUSER ;
```

Якщо список стовпців відсутній, то привілея діє для всіх стовпців таблиці або представлення.

Дозволити користувачам фінансового відділу змінювати всю інформацію про клієнтів.

```
GRANT UPDATE
    ON CUSTOMERS
    TO ARUSER ;
```

Стандарти SQL після SQL 1 підтримують надання привілеї SELECT для списку стовпців, подібно до наведеного нижче прикладу.

Дозволити користувачам фінансового відділу доступ лише для читання ідентифікаторів, імен і офісів представників з таблиці SALESREPS.

```
GRANT SELECT ( EMPL_NUM, NAME , REP_OFFICE )
    ON SALESREPS
    TO ARUSER ;
```

Ця інструкція GRANT усуває необхідність у створенні представлення REPINFO, показаному на рис. 26.3, а на практиці може зробити зайвим велику кількість представлень у промислових базах даних. Однак ця можливість наразі підтримується далеко не всіма провідними СУБД.

Передавання привілеїв (GRANT OPTION)

Якщо ви створюєте об'єкт у базі даних і стаєте його власником, то тільки ви можете надавати привілеї для роботи з цим об'єктом. Коли ви надаєте привілеї іншим користувачам, вони можуть користуватися об'єктом, але не можуть передавати ці привілеї комусь ще. Таким чином, власник об'єкта зберігає строгий контроль над тим, кому і які форми доступу до об'єкта дозволені.

Але існують ситуації, коли власник може бажати передати іншим користувачам право надавати привілеї для роботи зі своїм об'єктом. Наприклад, звернемося до представлень EASTREPS і WESTREPS у навчальній базі даних. Віце-президент з продажу Сем Кларк створив їх

і є їх власником. За допомогою наступної інструкції GRANT він може надати дозвіл менеджеру Ларрі Фічу з Лос-Анджелеського офісу користуватися представленням WESTREPS.

```
GRANT SELECT
    ON WESTREPS
    TO LARRY ;
```

Якщо Ларрі захоче надати Сью Сміт (ідентифікатор користувача SUE) дозвіл на доступ до представлення WESTREPS у зв'язку з тим, що вона складає прогноз обсягу продажів для офісу в Лос-Анджелесі, він не зможе зробити цього, оскільки попередній інструкції GRANT неявно забороняють таку дію. Потрібну привілею може надати тільки Сем Кларк, оскільки саме він є власником представлення.

Якщо Сему потрібно, щоб Ларрі на власний розсуд вирішував, кому надавати привілеї для роботи з представленням WESTREPS, він може скористатися наступним типом інструкції GRANT.

```
GRANT SELECT
    ON WESTREPS
    TO LARRY
WITH GRANT OPTION
```

Наявність ключового слова WITH GRANT OPTION означає, що інструкція GRANT, разом з привілеями, надає право надавати ці привілеї іншим користувачам.

Тепер Ларрі може виконати інструкцію GRANT:

```
GRANT SELECT
    ON WESTREPS
    TO SUE
```

що дозволяє Сью Сміт отримувати дані з представлення WESTREPS. На рисунку 26.6 процедура передачі привілеїв зображена графічно:

спочатку від Сема до Ларрі, а потім від Ларрі до Сью. Оскільки Ларрі не додав ключове слово WITH GRANT OPTION до своєї інструкції GRANT, ланцюжок закінчується на Сью: вона може отримувати дані з представлення WESTREPS, але не може надавати право доступу до нього іншому користувачеві. Однак, якби Ларрі додав ключове слово WITH GRANT OPTION до своєї інструкції GRANT, ланцюжок міг би продовжитися, оскільки Сью також мала б можливість надавати право доступу іншим користувачам.

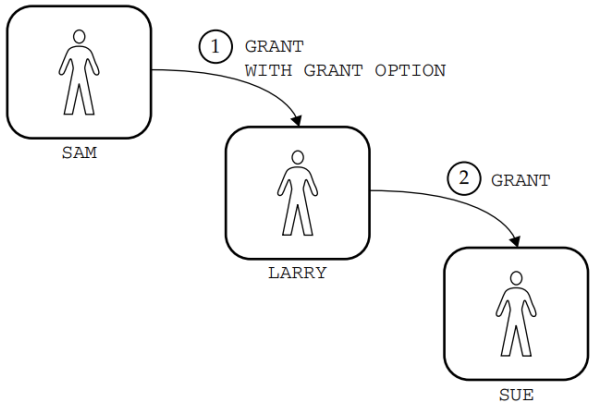

Рисунок 26.6 – Схема процедури передачі привілеїв

Так, Ларрі міг би діяти іншим чином: створити для Сью представлення, яке включає лише працівників Лос-Анджелеського офісу, і надати їй доступ до цього представлення.

```
CREATE VIEW LAREPS AS
        SELECT *
        FROM WESTREPS
        WHERE OFFICE = 21 ;

GRANT ALL PRIVI LEGES
        ON LAREPS
        TO SUE ;
```

Ларрі є власником представлення LAREPS, але не є власником представлення WESTREPS, на основі якого створюється нове представлення. Для ефективного забезпечення безпеки потрібно, щоб перед тим, як Ларрі може надати Сью привілею SELECT для роботи з представленням LAREPS, він мав не тільки привілею SELECT для роботи з представленням WESTREPS, але й право видачі цієї привілеї.

Якщо користувач має будь-які привілеї зі зазначенням GRANT OPTION, то він може надавати іншим користувачам не тільки самі ці привілеї, але й право їх передачі іншим. При цьому інші користувачі можуть в подальшому передавати привілеї також з правом передачі. Тому слід дуже обережно використовувати можливість передачі привілеїв. Зверніть увагу: право передачі стосується лише тих привілеїв, які зазначені в інструкції GRANT. Якщо ви хочете надати комусь деякі привілеї з правом передачі, а інші - без нього, то повинні скористатися двома окремими інструкціями GRANT.

Дозволити Ларрі Фічу витягувати, додавати, оновлювати та видаляти дані з представлення WESTREPS, а також дозволити йому надавати іншим користувачам дозвіл на вибірку даних.

```
GRANT SELECT
      ON WESTREPS
      TO LARRY
WITH GRANT OPTION ;

GRANT INSERT , DELETE , UPDATE
      ON WESTREPS
      TO LARRY ;
```

26.9 Скасування привілеїв (REVOKE)

У більшості реляційних баз даних привілеї, надані за допомогою інструкції GRANT, можуть бути вилучені за допомогою інструкції REVOKE. Синтаксична діаграма для інструкції REVOKE зображена на рисунку 26.7. Інструкція REVOKE, яка має структуру, аналогічну

структурі інструкції GRANT, містить набір привілеїв, які слід скасувати, об'єкт, до якого відносяться ці привілеї, та ідентифікатори користувачів, для яких скасовуються дані привілеї.

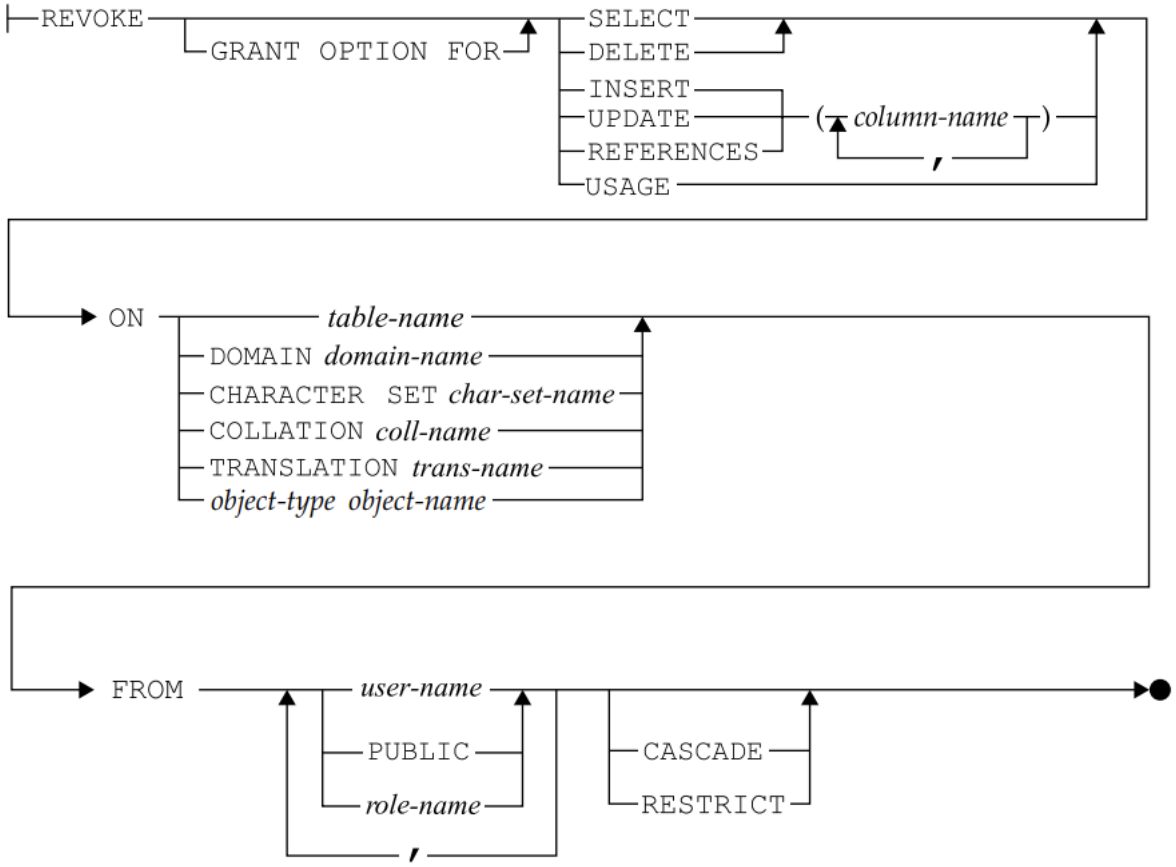

Рисунок 26.7 – Синтаксична діаграма для інструкції REVOKE

За допомогою інструкції REVOKE можна скасуватии всі або лише деякі з привілеїв, наданих раніше користувачеві. Наприклад, розглянемо таку послідовність інструкцій:

Надати певні привілеї для роботи з таблицею SALESREPS, а потім відмінити частину з них.

```
GRANT SELECT , INSERT , UPDATE
      ON SALESREPS
      TO ARUSER , OPUSER ;

REVOKE I NSERT , UPDATE
```

```
    ON SALESREPS
    FROM OPUSER ;
```

Спочатку привілеї INSERT та UPDATE для роботи з таблицею SALESREPS надаються двом користувачам, а потім в одного з них вилучаються. Однак привілеї SELECT залишаються в обох користувачів. Наведемо кілька додаткових прикладів інструкцій REVOKE.

Скасувати всі привілеї, що були надані раніше для роботи з таблицею OFFICES.

```
REVOKE ALL PRIVI LEGES
    ON OFFI CES
    FROM ARUSER ;
```

Скасувати привілеї UPDATE та DELETE у двох користувачів.

```
REVOKE UPDATE , DELETE
    ON OFFICES
    FROM ARUSER , OPUSER ;
```

Скасувати для всіх користувачів всі раніше надані привілеї для роботи з таблицею OFFICES.

```
REVOKE ALL PRIVILEGES
    ON OFFICES
    FROM PUBLIC;
```

За допомогою інструкції REVOKE ви можете вилучити лише ті привілеї, які ви раніше надали певному користувачу. У нього також можуть бути привілеї, надані іншими користувачами, і ваша інструкція REVOKE не впливає на них. Зверніть особливу увагу на те, що якщо два різних користувачі надають третьому користувачу однаковий привілей для того ж об'єкту, а потім один з них скасовує привілеї, то другий

привілей залишається дійсним і продовжує дозволяти доступ користувачу до об'єкту. Результат такого "перекриття привілеїв" ілюструється наступним прикладом.

Припустимо, віце-президент з продажу Сем Кларк надає Ларрі Фічу привілеї SELECT для роботи з таблицею SALESREPS і привілеї SELECT та UPDATE для роботи з таблицею ORDERS, використовуючи наступні інструкції.

```
GRANT SELECT
    ON SALESREPS
    TO LARRY ;

GRANT SELECT , UPDATE
    ON ORDERS
    TO LARRY ;
```

Через кілька днів віце-президент з маркетингу Джордж Уоткінс надає Ларрі привілеї SELECT та DELETE для роботи з таблицею ORDERS та привілеї SELECT для роботи з таблицею CUSTOMERS, використовуючи такі інструкції.

```
GRANT SELECT , DELETE
    ON ORDERS
    TO LARRY ;

GRANT SELECT
    ON CUSTOMERS
    TO LARRY ;
```

Таким чином Ларрі отримав привілеї для роботи з таблицею ORDERS з двох різних джерел. При цьому привілею SELECT для роботи з таблицею ORDERS він отримав з обох джерел. Кілька днів пізніше Сем скасовує привілеї, недавно надані Ларрі для роботи з таблицею ORDERS.

```
REVOKE SELECT , UPDATE
    ON ORDERS
    FROM LARRY ;
```

Після виконання цієї інструкції у Ларрі залишається привілея SELECT для роботи з таблицею SALESREPS, привілеї SELECT і DELETE для роботи з таблицею ORDERS і привілея SELECT для роботи з таблицею CUSTOMERS. Однак він втрачає привілею UPDATE для роботи з таблицею ORDERS.

26.10 Безпека на підставі ролей

Управління привілеями окремих користувачів може бути досить монотонною та трудомісткою роботою. У зв'язку з цим у стандарті SQL було додано поняття ролей. Нагадаємо, що роль – це набір привілеїв, об’єднаних певною назвою. У більшості сучасних реалізацій SQL ролі можуть бути надані окремим ідентифікаторам користувачів так само, як окремі привілеї. Більш того, багато реалізацій SQL постачаються з набором попередньо визначених ролей. Наприклад, привілеї, які зазвичай необхідні для роботи адміністратора бази даних, часто надаються постачальником СУБД у вигляді відповідної ролі.

Переваги використання ролей наступні.

• Ролі можуть існувати незалежно від ідентифікаторів користувачів. Наприклад, ми можемо створити роль для відділу обробки замовлень замість надання всім його співробітникам одного спільного ідентифікатора користувача OPUSER, як показано на рисунку 26.2. Якщо до відділу приєднується новий співробітник, одна інструкція GRANT з вказівкою ролі надає йому всі необхідні привілеї для роботи в цьому відділі.

• Ролі здатні "пережити" видалення користувачів. Адміністратору не потрібно турбуватися про втрату привілеїв за видалення певного користувача. Наприклад, якщо привілеї віце-президента з продажу Сема Кларка були надані за допомогою ролі, то список привілеїв цієї

ролі залишиться незмінним навіть після звільнення Сема з компанії та видалення його облікового запису. Роль може бути легко передана іншій особі, яка приймає цю посаду.

• Ролі підтримують стандартні привілеї. Застосування ролей в організації легко забезпечує однакові привілеї для всіх людей, які виконують однакову роботу.

• Ролі усувають монотонну і трудомістку роботу з надання привілеїв окремим користувачам. Ролі дозволяють надавати багато привілеїв однією простою командою. Під час додавання привілеїв до ролі або видалення з неї, всі зміни негайно відображаються на всіх користувачах, яким надано цю роль.

Створення та призначення привілеїв за допомогою ролей виконується дуже просто, якщо використовувати інструкції GRANT та REVOKE. На рисунку 26.5 інструкція GRANT дозволяє вказати ім'я ролі в реченні TO замість ідентифікатора користувача або ключового слова PUBLIC.

Аналогічно, ім'я ролі може використовуватися в реченні FROM інструкції REVOKE, як показано на рисунку 26.7.

Наведемо кілька прикладів використання таких інструкцій.

Створити роль OPUSER для користувачів з відділу обробки замовлень.

```
CREATE ROLE OPUSER ;
```

Призначити ролі привілеї, які є необхідними для всіх користувачів цієї ролі.

```
GRANT SELECT , INSERT , DELETE , UPDATE
    ON ORDERS
    TO OPUSER ;

GRANT SELECT
    ON CUSTOMERS
    TO OPUSER ;
```

```
GRANT UPDATE ( COMPANY , CUST_REP)
    ON CUSTOMERS
    TO OPUSER ;

GRANT SELECT
    ON SALESREPS
    TO OPUSER ;
```

Надати роль користувачам Julio, Sumit и Yolanda з відділу обробки замовлень.

```
GRANT OPUSER
    TO JULIO , SUMIT, YOLANDA ;
```

Надати право оновлення даних UPDATE для роботи з таблицею SALESREPS для ролі OPUSER.

Зауважимо, що всі три поточних користувача з даною роллю одразу отримують нову привілею.

```
GRANT UPDATE
    ON SALESREPS
    TO OPUSER ;
```

Надати роль OPUSER новому співробітнику відділу обробки замовлень – Francois.

Зауважимо, що цей співробітник одразу отримує всі привілеї даної ролі.

```
GRANT OPUSER
    TO FRANCOIS;
```

Скасувати роль OPUSER користувача Yolanda, яка переведена в інший відділ.

```
REVOKE OPUSER
    FROM YOLANDA ;
```

Зауважимо, що підтримка ролей в різних реалізаціях SQL є дещо різною.