Лекція 19

Вбудовані функції SQL

19.1 Двотабличний запит

Щоб зрозуміти, як SQL реалізує багатотабличні запити, найкраще почати з простого запиту, який з'єднує дані з двох різних таблиць.

Вивести всі замовлення, із вказанням номера замовлення і ціни, а також назви та кредитного ліміту клієнта, який зробив замовлення.

Рисунок 19.1. показує, що запитуються чотири елементи даних, які зберігаються у двох різних таблицях.

Таблиця ORDERS містить кількість і вартість кожного замовлення, але вона не включає в себе імена і кредитні ліміти клієнтів.

Таблиця CUSTOMERS містить список імен клієнтів і облікових записів клієнтів, але він не включає данні про замовлення.

Однак між двома таблицями існує зв'язок. Кожен рядок стовпчика CUST таблиці ORDERS містить ідентифікатор клієнта замовника, що відповідає одному з рядків стовпчика CUST_NUM. Очевидно, для того, щоб отримати бажані результати, оператор SELECT, який ви використовуєте для запиту, повинен якось враховувати це відношення між таблицями.

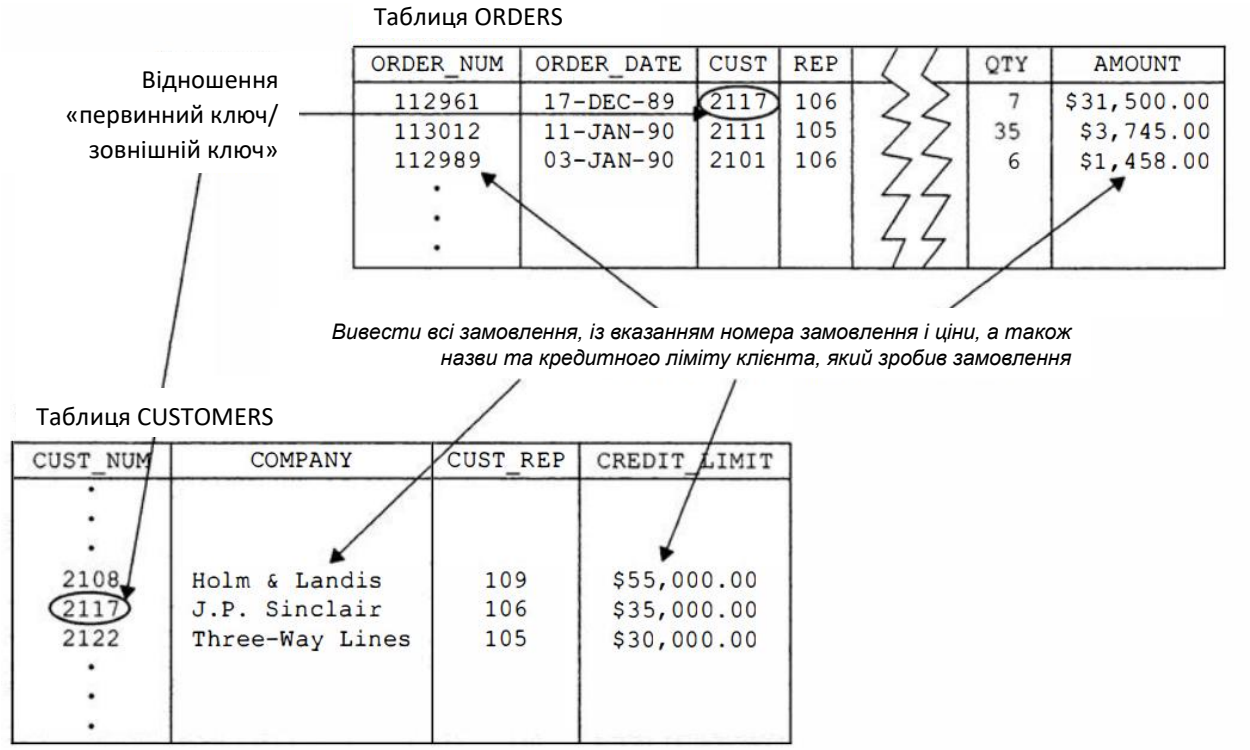

Рисунок 19.1. Двотабличний запит

Такий запит передбачає наступне:

• Кожен рядок в таблиці запитів формується з пари рядків: один рядок знаходиться в таблиці ORDERS, а інший рядок — в таблиці CUSTOMERS.

• Ця пара рядків визначається шляхом порівняння вмісту відповідних стовпчиків цих таблиць

Процес формування пар рядків шляхом порівняння вмісту відповідних стовпчиків називається *таблицею об'єднання*. Отримана таблиця, що містить дані з обох початкових таблиць, називається *з'єднанням* (приєднанням) двох таблиць. Розв'язок, заснований на точній рівності між двома стовпчиками, називається *з'єднанням за рівністю* (equi-join). З'єднання можуть також базуватися на інших типах порівнянь стовпців.

З'єднання є основою багатотабличних запитів в SQL. У реляційній базі даних вся інформація зберігається як явні значення даних в колонках, так що всі можливі відношення між таблицями можуть бути сформовані за допомогою відповідності вмісту пов'язаних стовпців. Таким чином,

з'єднання є потужним (і, до того ж, єдиним, за відсутності вказівників або інших механізмів поєднання рядків) інструментом для ідентифікації взаємозв'язків між даними. Оскільки SQL виконує багатотабличні запити за допомогою порівняння стовпців, не дивно, що оператор SELECT для багатоабличного запиту повинен містити умову вибору, яка визначає відношення між стовпцями.

Ось як виглядає оператор SELECT для запиту, наведеного вище.

```
SELECT ORDER_NUM, AMOUNT, COMPANY, CREDIT_LIMIT
  FROM ORDERS, CUSTOMERS
 WHERE CUST = CUST_NUM;
```

ORDER_NUM	AMOUNT	COMPANY	CREDIT_LIMIT
112989	$1,458.00	Jones Mfg.	$65,000.00
112968	$3,978.00	First Corp.	$65,000.00
112963	$3,276.00	Acme Mfg.	$50,000.00
112987	$27,500.00	Acme Mfg.	$50,000.00
112983	$702.00	Acme Mfg.	$50,000.00
113027	$4,104.00	Acme Mfg.	$50,000.00
112993	$1,896.00	Fred Levis Corp.	$65,000.00
113065	$2,130.00	Fred Levis Corp.	$65,000.00
113036	$22,500.00	Ace International	$35,000.00
113034	$632.00	Ace International	$35,000.00
113058	$1,480.00	Holm & Landis	$55,000.00
113055	$150.00	Holm & Landis	$55,000.00
113003	$5,625.00	Holm & Landis	$55,000.00

Такий запит має кілька особливостей. По-перше, оператор FROM містить дві таблиці. По-друге, умова вибору CUST = CUST NUM порівнює стовпці з двох різних таблиць. Назвемо ці стовпчики пов’язаними або погодженими (matching). Ця умова вибору, як і будь-яка інша, зменшує кількість рядків у таблиці результатів запиту. Оскільки запит є двотабличним, критерій вибору фактично зменшує кількість пар рядків у таблиці результатів.

Умова запиту дуже добре відображає сутність співставлення стовпців: Включити в таблицю результатів запиту тільки ті пари рядків, ідентифікатор клієнта яких (CUST) в таблиці ORDERS відповідає ідентифікатору клієнта (CUST_NUM) в таблиці CUSTOMERS.

Зауваження. В операторі SELECT не визначено як саме має виконуватися SQL-запит. Немає жодних інструкції на зразок «почати з

замовлень» або «почати з клієнтів». Натомість запит визначає, що саме повинно бути результатом запиту, залишаючи спосіб виконання для СУБД, в який він виконується.

19.2 Запити з використанням відношення "предок-нащадок"

Серед багатотабличних запитів найпоширенішими є запити з двох таблиць, пов'язаних природнім зв'язком «предок-нащадок». Запит про замовлення та клієнтів є прикладом такого запиту.

Кожне замовлення (нащадок) має відповідного замовника (предка), і кожен замовник (предок) може мати багато замовлень (нащадків). Пари рядків, з яких утворюються результати запитів, пов'язані відношенням "предок-нащадок". В базах даних SQL первинний і зовнішній ключі створюють відношення «предок-нащадок». Таблиця, що містить зовнішній ключ, є таблицею-нащадком, а таблиця з первинним ключем є таблицею-предком. Для використання відношення «предок» в запиті необхідно вказати умову вибору, в якій первинний ключ порівнюється з зовнішнім ключем.

Наведемо приклад такого запиту (рис. 19.3).

Вивести список всіх співробітників із зазначенням міст і регіонів, в яких вони працюють.

```
SELECT NAME, CITY, REGION
  FROM SALESREPS, OFFICES
 WHERE REP_OFFICE = OFFICE;
```

NAME	CITY	REGION
Mary Jones	New York	Eastern
Sam Clark	New York	Eastern
Bob Smith	Chicago	Eastern
Paul Cruz	Chicago	Eastern
Dan Roberts	Chicago	Eastern
Bill Adams	Atlanta	Eastern
Sue Smith	Los Angeles	Western
Larry Fitch	Los Angeles	Western
Nancy Angelli	Denver	Western

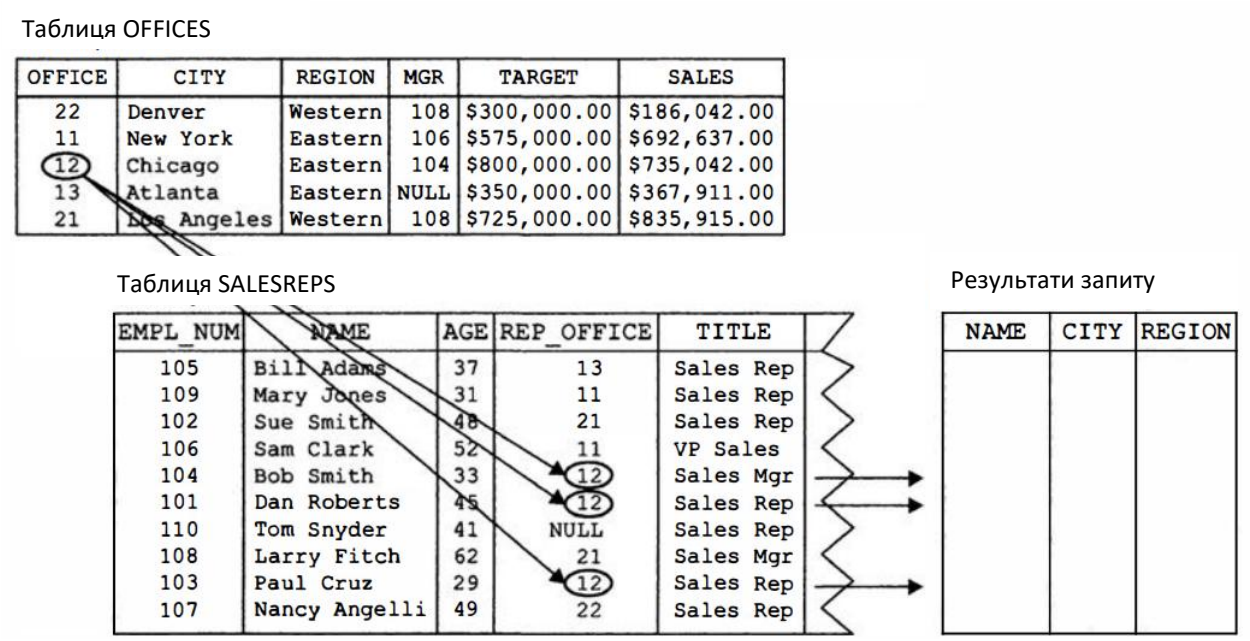

Рисунок 19.3. Запит з використанням відношення "предок-нащадок" між таблицями OFFICES та SALESREPS

Таблиця SALESREPS (нащадок) містить стовпчик REP_OFFICE, який є зовнішнім ключем для таблиці OFFICES (предка). Тут відношення «предок-нащадок» використовується з метою пошуку в таблиці OFFICE для кожного співробітника відповідного рядка, що містить місто і регіон, та включення його до результатів запиту.

Наведемо інший запит для цих самих двох таблиць, але тут вже ролі предка та нащадка змінюються (рис. 19.4).

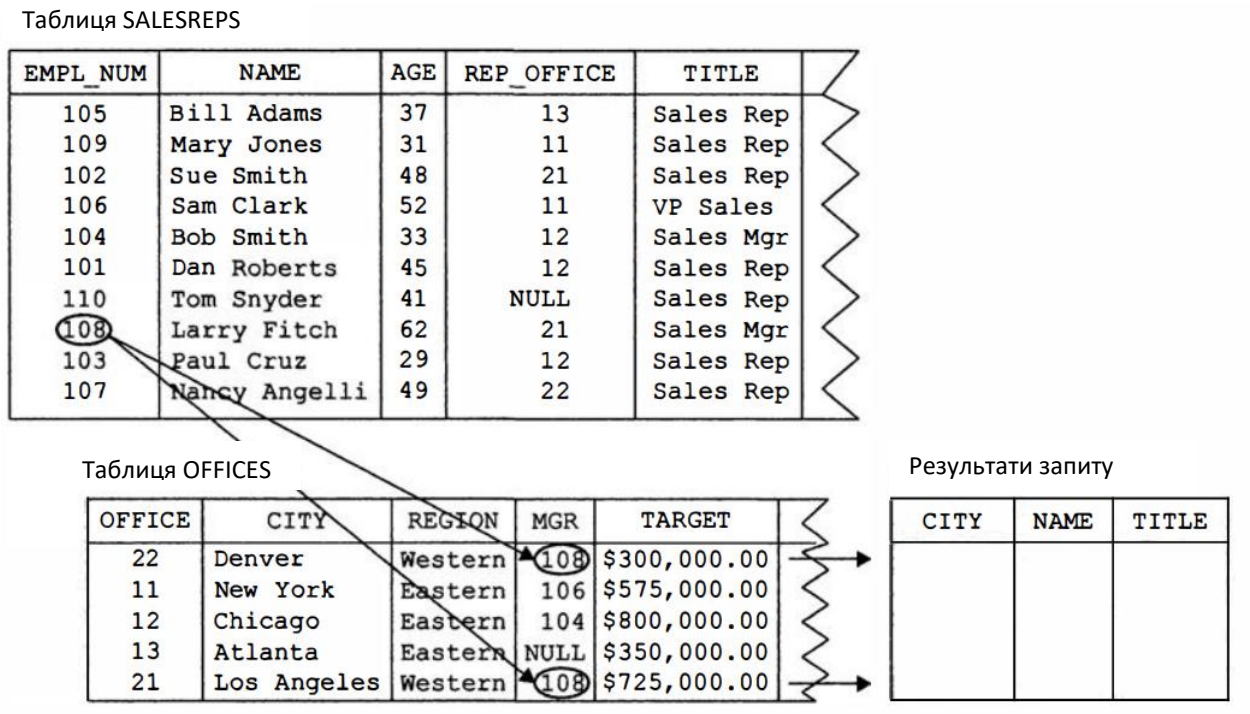

Рисунок 19.4. Інший запит з використанням відношення "предок-нащадок" між таблицями OFFICES та SALESREPS

Вивести список офісів з іменами та посадами їх керівників.

```
SELECT CITY, NAME, TITLE
  FROM OFFICES, SALESREPS
 WHERE MGR = EMPL_NUM;

CITY          NAME         TITLE
-----------   -----------  ---------
Chicago       Bob Smith    Sales Mgr
Atlanta       Bill Adams   Sales Rep
New York      Sam Clark    VP Sales
Denver        Larry Fitch  Sales Mgr
Los Angeles   Larry Fitch  Sales Mgr
```

Таблиця OFFICES (нащадок) містить стовпчик MGR, що є зовнішнім ключем для таблиці SALESREPS (предок). Це відношення використовується, щоб знайти відповідний рядок, що містить ім'я і посаду співробітника в таблиці SALESREPS для кожного офісу і включити його в результати запиту.

SQL не вимагає включення зв'язаних стовпців до результатів багатотабличного запиту. На практиці вони найчастіше не включаються, як у двох попередніх прикладах. Це тому, що первинний і зовнішній ключі часто є ідентифікаторами (наприклад, ідентифікатор офісу або ідентифікатор працівника в наведених прикладах), які важко запам'ятати людині, в той час як відповідні назви (міста, райони, імена, позиції) це набагато легше запам'ятати. Таким чином, природно, що оператор WHERE використовує ідентифікатори для приєднання до двох таблиць, а оператор SELECT використовує більш придатні для читання назви для створення стовпців результатів запиту.

19.3 Інший спосіб ідентифікації з'єднань

Найпростіший спосіб вказати зв'язані таблиці — це перелічити їх в операторі FROM інструкції SELECT, як показано у попередніх прикладах. Цей метод уточнення з'єднаних таблиць з'явився в найбільш ранніх реалізаціях SQL фірми IMB. Він включений в оригінальний стандарт SQL і підтримується всіма базами даних SQL.

Подальші версії стандарту значно розширили зв'язок і додали нові версії оператора FROM. Використовуючи ці версії, попередні приклади можна написати наступним чином.

Вивести список всіх працівників із зазначенням міст і регіонів, в яких вони працюють

```
SELECT NAME, CITY, REGION
  FROM SALESREPS JOIN OFFICES
    ON REP_OFFICE = OFFICE;

NAME            CITY          REGION
------------    ----------    -------
Mary Jones      New York      Eastern
Sam Clark       New York      Eastern
Bob Smith       Chicago       Eastern
Paul Cruz       Chicago       Eastern
Dan Roberts     Chicago       Eastern
Bill Adams      Atlanta       Eastern

Sue Smith       Los Angeles   Western
Larry Fitch     Los Angeles   Western
Nancy Angelli   Denver        Western
```

Вивести список всіх офісів із зазначенням імен та посад їх керівників.

```
SELECT CITY, NAME, TITLE
  FROM OFFICES JOIN SALESREPS
    ON MGR = EMPL_NUM;

CITY          NAME         TITLE
-----------   -----------  ---------
Chicago       Bob Smith    Sales Mgr
Atlanta       Bill Adams   Sales Rep
New York      Sam Clark    VP Sales
Denver        Larry Fitch  Sales Mgr
Los Angeles   Larry Fitch  Sales Mgr
```

В операторі FROM замість списку елементів, відокремлених комами, для опису операції приєднання використовується ключове слово JOIN. Зв'язані стовпці, що використовуються для з'єднання, вказуються в операторі ON в кінці оператора FROM. У цих простих прикладах використання нового синтаксису надає небагато переваг відносно старої версії оператора SELECT, але діапазон зв'язків, які можуть бути здійсненими за допомогою такого синтаксису є досить широким.

19.4 З’єднання з умовами вибору рядків

У багатотабличних запитах можна об'єднати умову вибору, яка вказує пов'язані стовпці з іншими критеріями вибору для подальшого звуження результатів запиту. Припустимо, що потрібно повторити попередній запит, але включати лише офіси, планові обсяги продажів яких є більшими за 600$.

Вивести список офісів, планові обсяги продажів яких є більшими за 600$.

```
SELECT CITY, NAME, TITLE
  FROM OFFICES, SALESREPS
 WHERE MGR = EMPL_NUM
   AND TARGET > 600000.00;

CITY          NAME         TITLE
-----------   -----------  ---------
Chicago       Bob Smith    Sales Mgr
Los Angeles   Larry Fitch  Sales Mgr
```

Згідно з першою умовою (MGR=EMPL_NUM), в таблицях OFFICES та SALESREPS відбираються пари рядів, які мають відповідні з'єднання «предок-нащадок»; згідно з другою умовою додатково відбираються лише пари рядів, де планова сума продажів перевищує $600,000. У цьому запиті умова з'єднання і умова відбору знаходяться в операторі WHERE. За використання нового синтаксису умови з'єднання знаходяться в операторі ON, а критерії відбору — в операторі WHERE, що полегшує розуміння запиту.

```
SELECT CITY, NAME, TITLE
  FROM OFFICES JOIN SALESREPS
    ON MGR = EMPL_NUM
 WHERE TARGET > 600000.00;

CITY          NAME          TITLE
------------  ------------  ----------
Chicago       Bob Smith     Sales Mgr
Los Angeles   Larry Fitch   Sales Mgr
```

19.5 Декілька пов'язаних стовпчиків

Таблиці ORDERS та PRODUCTS бази даних пов'язані парою складених ключів. Стовпці MFR та PRODUCT в таблиці ORDERS разом утворюють зовнішній ключ для таблиці PRODUCTS та пов'язані з його стовпцями MFR_ID та PRODUCT_ID відповідно. Для об'єднання таблиць, заснованих на цьому співвідношенні «предок-нащадок», необхідно вказати обидві пари пов'язаних стовпців, як показано в прикладі нижче.

```
SELECT ORDER_NUM, AMOUNT, DESCRIPTION
  FROM ORDERS, PRODUCTS
 WHERE MFR = MFR_ID
   AND PRODUCT = PRODUCT_ID;

ORDER_NUM      AMOUNT      DESCRIPTION
---------      ---------   ---------------
   113027      $4,104.00   Size 2 Widget
   112992        $760.00   Size 2 Widget
   113012      $3,745.00   Size 3 Widget
   112968      $3,978.00   Size 4 Widget
   112963      $3,276.00   Size 4 Widget
   112983        $702.00   Size 4 Widget
   113055        $150.00   Widget Adjuster
   113057        $600.00   Widget Adjuster
```

Умова відбору в даному запиті вказує SQL, що пов’язаними парами рядків таблиць ORDERS та PRODUCTS є ті, в яких пари пов’язаних стовпців містять ті ж самі значення. Альтернативна форма запиту вказує пов’язані стовпці так само.

```
SELECT ORDER_NUM, AMOUNT, DESCRIPTION
  FROM ORDERS JOIN PRODUCTS
    ON MFR = MFR_ID
   AND PRODUCT = PRODUCT_ID;
```

Багатостовчикові з'єднання зазвичай зустрічаються в запитах зі складеними зовнішніми ключами, як в прикладі вище. SQL не обмежує кількість стовпців в умовах з'єднання, але зазвичай з'єднання відображають реальні зв'язки між об'єктами, представленими в таблицях баз даних, і ці відношення, як правило, включають лише один-два зрідка більше стовпців таблиць.

19.6 Природні з’єднання

Часто зв'язані стовпчики, які використовуються для з’єднання двох таблиць, мають однакову назву в обох таблицях. Інша ситуація у прикладі бази даних, де первинний і зовнішній ключі мають трохи різні назви, що полегшує їх розрізнення. На практиці, однак, розробники баз даних часто використовують одну назву для стовпчика, який містить ідентифікатор клієнта або номер працівника в усіх таблицях, які містять ці дані. Припустимо, що ідентифікатор виробника й ідентифікатор товару мають назви MFR і PRODUCT в нашій базі даних в обох таблицях - як ORDERS, так і PRODUCTS. У цьому випадку, більшість природних зв'язків між двома таблицями були б за рівністю на основі назв стовпців, присутніх в обох таблицях. Таке з'єднання називається природним з'єднанням в стандарті SQL.

Синтаксис з'єднання SQL дозволяє легко визначити природне з'єднання.

```
SELECT ORDER_NUM, AMOUNT, DESCRIPTION
  FROM ORDERS NATURAL JOIN PRODUCTS;
```

Ця інструкція вказує, що СУБД повинна приєднувати таблиці ORDERS і PRODUCTS за всіма стовпцями з однаковою назвою в цих таблицях. У цьому прикладі це стовпці MFR і PRODUCT. Також, в цій ситуації - використовуючи новий синтаксис - можна явно вказати назви пов'язаних стовпчиків.

```
SELECT ORDER_NUM, AMOUNT, DESCRIPTION
  FROM ORDERS JOIN PRODUCTS
 USING (MFR, PRODUCT);
```

В операторі USING в дужках перелічуються пов’язані стовпці (які мають однакові назви в обох таблицях). Зауважемо, що оператор USING є значно компактнішим за оператор ON. Попередній запит є повністю еквівалентним до наступного (за умови однакових назв стовпців в обох таблицях).

```
SELECT ORDER_NUM, AMOUNT, DESCRIPTION
  FROM ORDERS JOIN PRODUCTS
    ON ORDERS.MFR = PRODUCTS.MFR
   AND ORDERS.PRODUCT = PRODUCTS.PRODUCT;
```

Часто, з'єднання із застосуванням оператора USING є кращим для природнього з’єднання за оператор NATURAL JOIN. Наприклад, якщо два різних адміністратори відповідають за підтримку таблиць ORDERS та PRODUCTS (дуже поширена ситуація для великої бази даних на виробництві), то можливо, що вони обидва вибирають одну і ту ж назву для нового стовпця, хоча ці стовпці можуть не мають нічого спільного. У цій ситуації, NATURAL JOIN буде намагатися використовувати нові стовпці для приєднання таблиць, що, ймовірно, призведе до помилки. Оператор USING захищає запит від помилок, пов'язаних з такими змінами в структурі бази даних. Крім того, оператор USING дозволяє вибрати окремі стовпчики для приєднання таблиць, в той час як NATURAL JOIN автоматично використовує всі стовпчики з відповідними назвами. Нарешті, якщо стовпців з такою ж назвою немає, то запит з NATURAL JOIN може повернути декартовий добуток або помилку в залежності від конкретної СУБД; запит з використанням USING поверне помилку.

19.7 Запити до трьох та більше таблиць

SQL дозволяє об'єднувати дані з трьох або більше таблиць за допомогою того ж методу, що і дані з двох таблиць. Ось простий приклад з'єднання трьох таблиць.

Вивести список замовлень вартістю понад $25 000, із зазначенням імені працівника, який прийняв замовлення та імені клієнта, який його зробив.

```
SELECT ORDER_NUM, AMOUNT, COMPANY, NAME
  FROM ORDERS, CUSTOMERS, SALESREPS
 WHERE CUST = CUST_NUM
   AND REP = EMPL_NUM
   AND AMOUNT > 25000.00;
```

```
ORDER_NUM      AMOUNT   COMPANY          NAME
---------    ----------  ---------------  -----------
   112987    $27,500.00  Acme Mfg.        Bill Adams
   113069    $31,350.00  Chen Associates  Paul Cruz
   113045    $45,000.00  Zetacorp         Larry Fitch
   112961    $31,500.00  J.P. Sinclair    Sam Clark
```

Як видно з рисунку 19.5, в цьому запиті використовуються два зовнішніх ключа таблиці ORDERS. Стовбець CUST є зовнішнім ключем для таблиці CUSTOMERS; він пов’язує кожне замовлення з клієнтом, який його зробив. Стовбець REP є зовнішнім ключем для таблиці SALESREPS, він пов’язує кожне замовлення з працівником, який його прийняв. Тобто, запит пов’язує кожне замовлення з відповідними клієнтом та працівником.

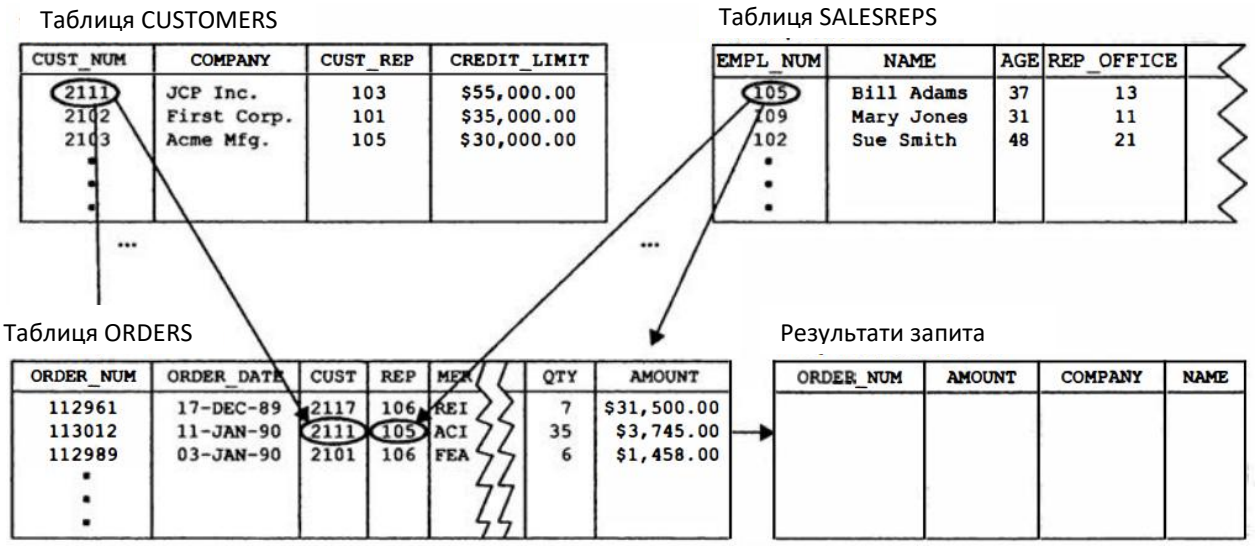

Рисунок 19.5. Запит до трьох таблиць

Альтернативний запис цього запиту більш наглядно вказує кожне з'єднання та пов'язані стовбці.

```
SELECT ORDER_NUM, AMOUNT, COMPANY, NAME
  FROM ORDERS JOIN CUSTOMERS ON CUST = CUST_NUM
              JOIN SALESREPS ON REP  = EMPL_NUM
 WHERE AMOUNT > 25000.00;
```

Ще один запит до трьох таблиць, в якому використовується інша комбінація відношень "предок-нащадок".

Вивести список замовлень вартістю вище за $25 000, який включає ім'я клієнта, який зробив це замовлення, та ім'я працівника, який є закріпленим за цим клієнтом.

```
SELECT ORDER_NUM, AMOUNT, COMPANY, NAME
  FROM ORDERS, CUSTOMERS, SALESREPS
 WHERE CUST = CUST_NUM
   AND CUST_REP = EMPL_NUM
   AND AMOUNT > 25000.00;
```

```
ORDER_NUM      AMOUNT    COMPANY          NAME
---------   ----------   ---------------  -----------
   112987   $27,500.00   Acme Mfg.        Bill Adams
   113069   $31,350.00   Chen Associates  Paul Cruz
   113045   $45,000.00   Zetacorp         Larry Fitch
   112961   $31,500.00   J.P. Sinclair    Sam Clark
```

На рисунку 19.6 показані взаємозв’язки, які використовуються у цьому запиті. У першому відношенні стовпчик CUST з таблиці ORDERS мвикористовується як зовнішній ключ таблиці CUSTOMERS. У другому відношенні зовнішнім ключем таблиці SALESREPS є стовпчик CUST_REP з таблиці CUSTOMERS. Простіше кажучи, цей запит пов'язує кожне замовлення з клієнтом, а кожного клієнта – з призначеним йому працівником.

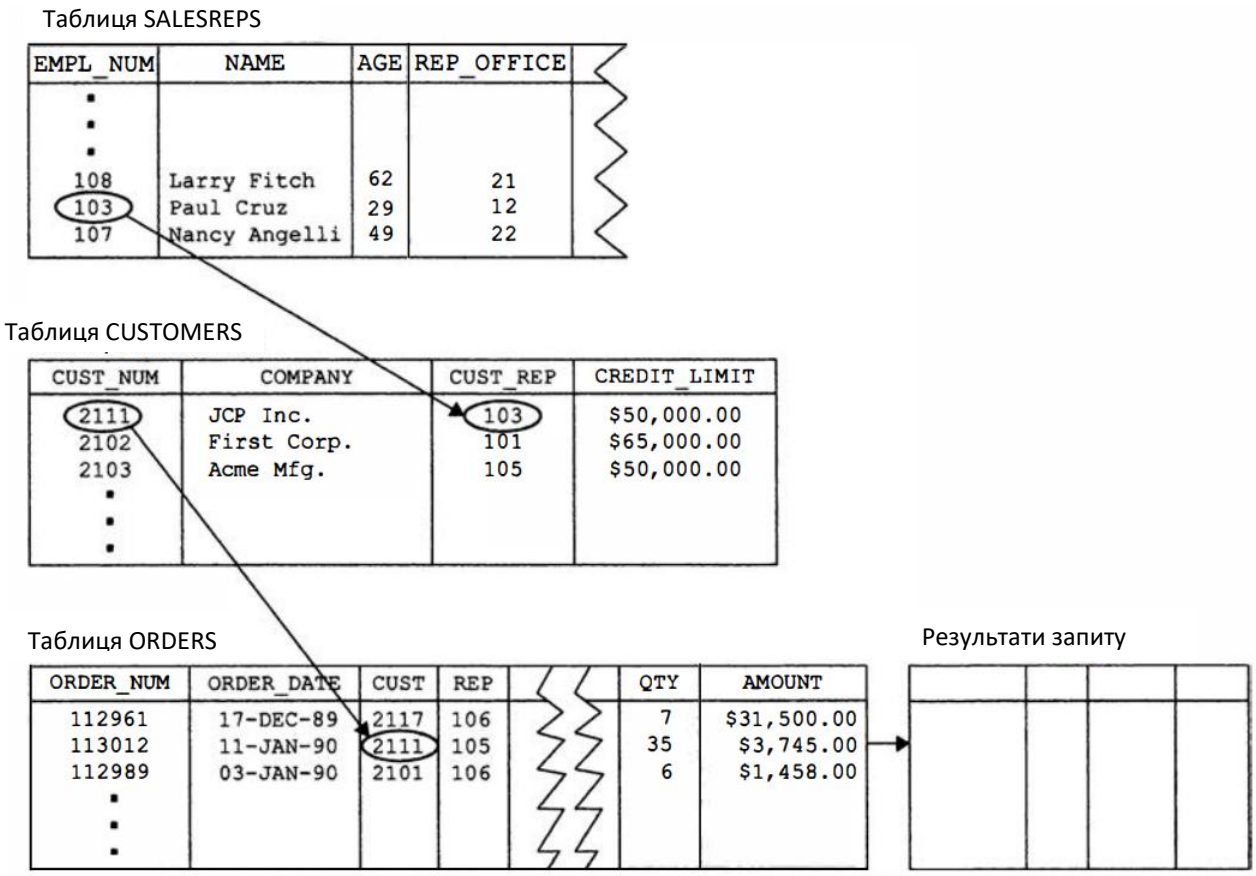

Рисунок 19.6. Запит до трьох таблиць с каскадними відношеннями "предок-нащадок"

Зверніть увагу, що порядок з'єднань в цих багатотабличних запитах не має значення. СУБД може з'єднати таблицю ORDERS з таблицею CUSTOMERS, а потім отриманий результат з таблицею SALESREPS. Альтернативно, він може спочатку підключити таблицю CUSTOMERS до таблиці SALESREPS і потім результат до таблиці ORDERS. У будь-якому випадку кінцевий результат буде однаковим, тому СКБД може виконувати з'єднання в тому порядку, який є найефективнішим. Однак

деякі складні поєднання чутливі до порядку. Однією з переваг нового синтаксису з'єднання в стандарті SQL є те, що в таких випадках він дозволяє визначити порядок виконання з'єднань.

Промислові застосунки часто мають запити на три або чотири таблиці, а в великих сховищах даних, інтелектуальні бізнес-запити легко включають більше десятка таблиць. Навіть в рамках невеликої навчальної бази даних, яка складається з п'яти таблиць легко створити запит до чотирьох таблиць, який би мав реальне значення.

Вивести список замовлень вартістю понад $25 000, включаючи ім'я клієнта, який зробив замовлення, ім'я співробітника і офіс, в якому працює співробітник.

```
SELECT ORDER_NUM, AMOUNT, COMPANY, NAME, CITY
  FROM ORDERS, CUSTOMERS, SALESREPS, OFFICES
 WHERE CUST = CUST_NUM
   AND CUST_REP = EMPL_NUM
   AND REP_OFFICE = OFFICE
   AND AMOUNT > 25000.00;
```

ORDER_NUM	AMOUNT	COMPANY	NAME	CITY
112987	$27,500.00	Acme Mfg.	Bill Adams	Atlanta
113069	$31,350.00	Chen Associates	Paul Cruz	Chicago
113045	$45,000.00	Zetacorp	Larry Fitch	Los Angeles
112961	$31,500.00	J.P. Sinclair	Sam Clark	New York

На рисунку 19.7 показані зв'язки "предок-нащадок", які використовуються в цьому запиті. Логічно він просто розширює на один крок послідовність зв'язків з попереднього запиту, пов’язуючи замовлення з клієнтом, клієнта з призначеним йому працівником, а працівника — з офісом, в якому він працює.

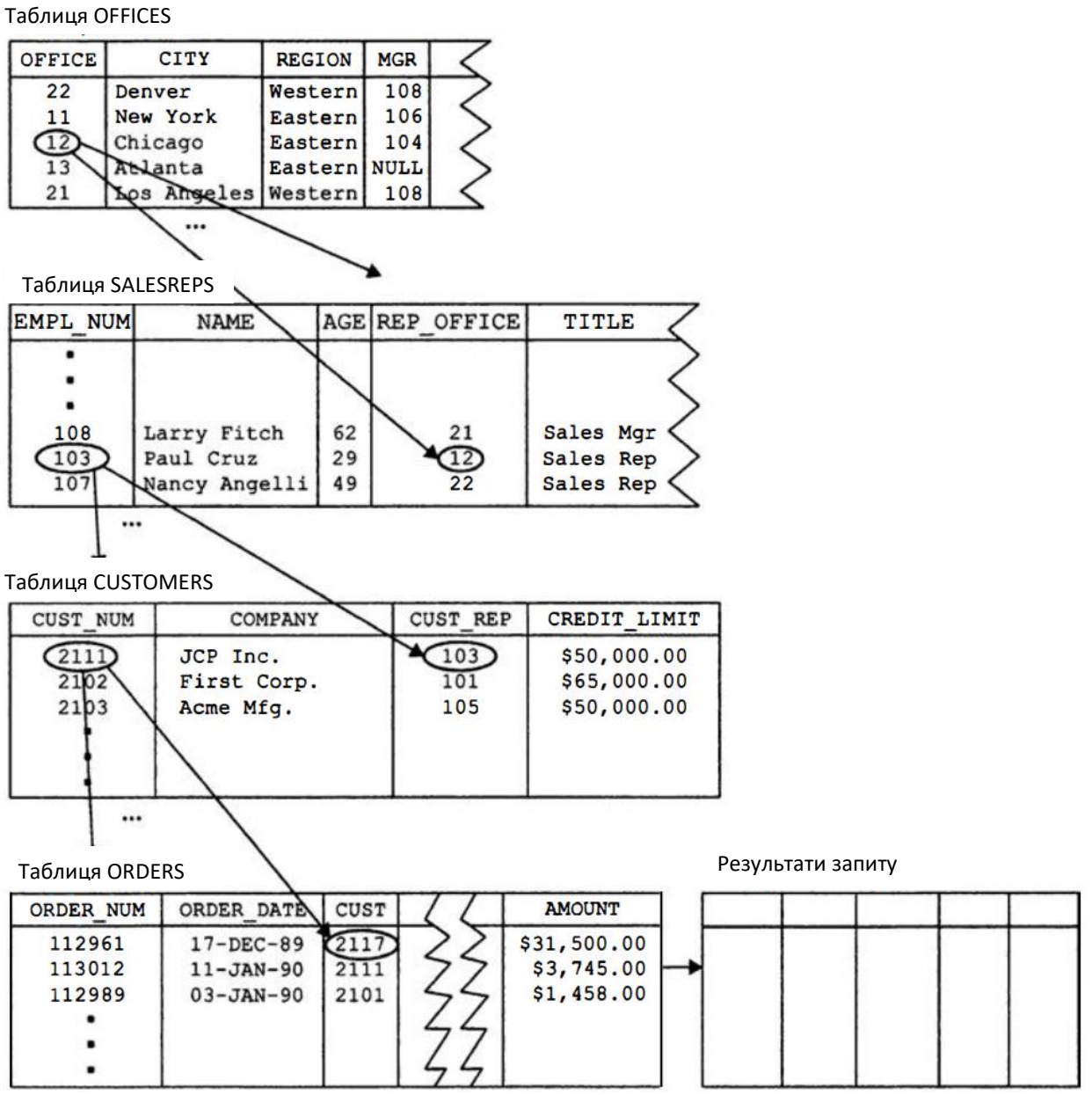

Рисунок 19.7. Поєднання за участю чотирьох таблиць

19.8 З’єднання за нерівністю

Термін «з'єднання» застосовується до будь-якого запиту, який з'єднує дані з двох таблиць бази даних шляхом порівняння значень у двох стовпцях цих таблиць. Найбільш поширеними є з'єднання, створені на основі рівності пов'язаних стовпців (з'єднання за рівністю). Але SQL дозволяє з'єднувати таблиці також за допомогою інших операцій порівняння. Наступний приклад використовує операцію порівняння "більше" (>) для з'єднання таблиць.

Вивести список всіх комбінацій працівників і офісів, де плановий обсяг продажів працівника більша за план будь-якого офісу, незалежно від того, де працює співробітник.

```
SELECT NAME, QUOTA, CITY, TARGET
  FROM SALESREPS, OFFICES
 WHERE QUOTA > TARGET;
```

NAME	QUOTA	CITY	TARGET
Bill Adams	$350,000.00	Denver	$300,000.00
Sue Smith	$350,000.00	Denver	$300,000.00
Larry Fitch	$350,000.00	Denver	$300,000.00

Як і в інших запитах для двох таблиць, кожен рядок результатів запиту отримана з пари рядків, які в цьому випадку містяться в таблицях SALESREPS і OFFICES. Згідно з умовою QUOTA > TARGET вибираються лише ті рядки, де значення атрибута QUOTA з таблиці SALESREPS перевищує значення атрибута TARGET з таблиці OFFICES.

Зауваження. Пари рядків, вибраних зі таблиць SALESREPS та OFFICES, пов'язані лише цією умовою; зокрема, рядок таблиці SALESREPS не обов’язково має відповідати працівникові, який працює в офісі, визначеному рядком таблиці OFFICES. Щоправда, цей приклад дещо штучний і показує, чому рідко виникає необхідність використання з’єднань за нерівністю. Однак, такі типи з’єднань можуть бути корисними в програмах підтримки рішень та інших програмах, що вивчають складніші зв'язки в базі даних.

19.9 Особливості багатотабличних запитів

Багатотабличні запити, які були розглянуті, не потребували використання спеціальних синтаксичних форм або будь-яких інших додаткових можливостей мови SQL, відмінних від тих, які використовуються для створення однотабличних запитів. Однак деякі багатотабличні запити не можуть бути створені без додаткових можливостей SQL.

Іноді в багатотабличних запитах потрібно використовувати кваліфіковані назви стовпчиків (*поле_таблиця* наприклад,

REP_OFFICE), щоб уникнути неоднозначності під час посилання на стовпчикі.

В багатотабличних запитах, вибір всіх стовпчиків SELECT(*) має особливий сенс: в багатотеленному запиті зірочка означає вибір всіх стовпчиків з усіх таблиць, вказаних оператором FROM.

Для створення багатотабличних запитів, які пов'язують таблицю саму з собою, можна використовувати самоз'єднання: багатотабличні запити використовують відношення, які існують в рамках однієї таблиці

• В операторі FROM можна використовувати псевдоніми таблиць для спрощення повних назв стовпців і забезпечення однозначності посилань на стовпці під час самозв'язування. Наприклад:

Вивести список імен, продажів і днів народження співробітників.

```
SELECT SALESREPS.NAME, QUOTA, SAM.BIRTHDAYS.BIRTH_DATE
  FROM SALESREPS, BIRTHDAYS
 WHERE SALESREPS.NAME = SAM.BIRTHDAYS.NAME;
```

Якщо замість імен двох таблиць використовувати псевдоніми S та B, то вводити та читати такий запит буде легше.

```
SELECT S.NAME, S.QUOTA, B.BIRTH_DATE
  FROM SALESREPS S, SAM.BIRTHDAYS B
 WHERE S.NAME = B.NAME;
```

19.20 Статистичні функції

SQL дозволяє отримати підсумки за допомогою набору статистичних функцій, які агрегують значення стовбців (column functions). Такі функції приймають дані всього стовпчика як аргумент і повертають єдине значення підсумку для даного стовпчика. Наприклад, функція AVG () приймає стовпчик чисел як аргумент і обчислює їх середнє значення. Нижче наведено запит, в якому функція AVG () використовується для обчислення середнього значення в двох стовпцях таблиці SALESREPS.

Визначити середні планові та середні фактичні продажі продавців.

```
SELECT AVG(QUOTA), AVG(SALES)
  FROM SALESREPS;

AVG(QUOTA)      AVG(SALES)
-----------     -----------
$300,000.00     $289,353.20
```

На рисунку 19.8 наведено схему виконання такого запиту. Перша функція приймає за аргументи всі значення, які містить стовпчик QUOTA, та обраховує їх середнє значення; друга функція обраховує середнє значення стовпчика SALES. Результатом запиту є рядок, який містить підсумки на підставі інформації, яка міститься в таблиці SALESREPS.

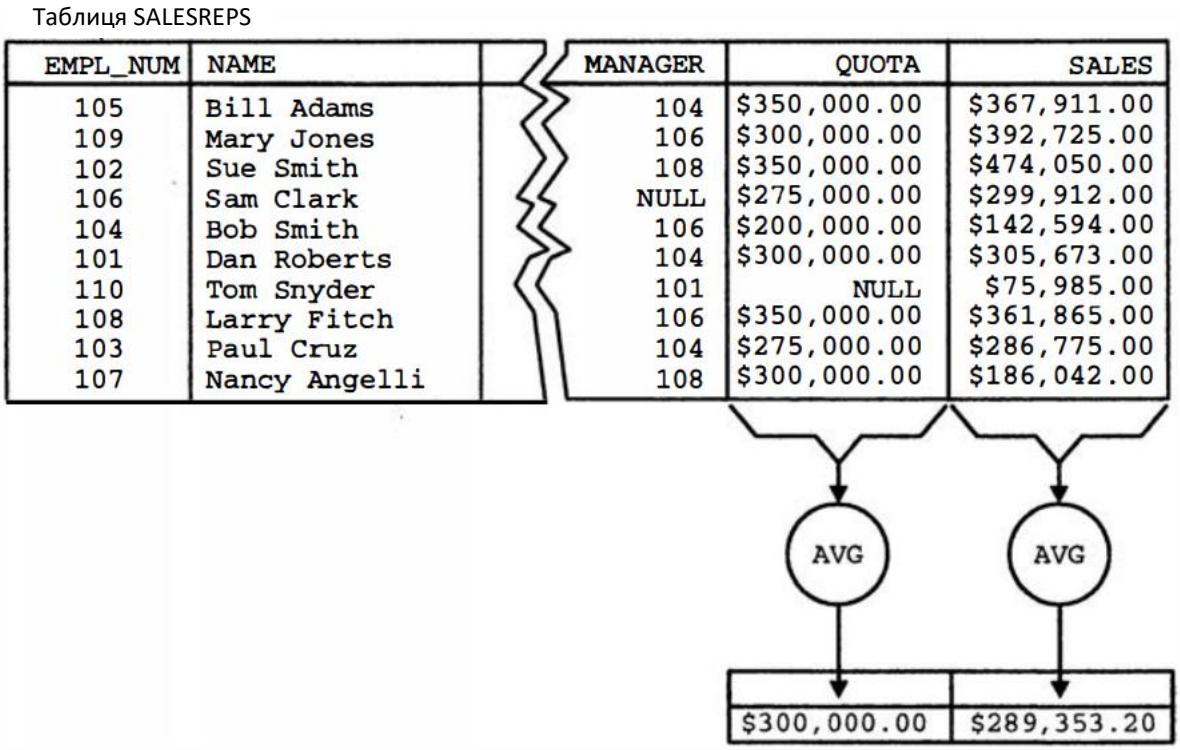

Рисунок 19.8. Виконання запиту з підсумком

В стандарті SQL визначено кілька таких функцій; крім того, багато розробників СУБД додають власні функції до своїх реалізацій SQL. Шість найбільш розповсюджених функцій, показано на рисунку 8.2. Вони дозволяють отримувати різні види підсумків.

• Функція SUM ( ) обчислює суму всіх значень стовпчика.

• Функція AVG ( ) обчислює середнє арифметичне всіх значень стовпчика.

• Функція MIN ( ) знаходить найменше серед усіх значень стовпчика.

• Функція MAX ( ) знаходить найбільше серед усіх значень стовпчика.

• Функція COUNT ( ) підраховує кількість значень, які містяться в стовпчику.

• Функція COUNT (*) підраховує кількість рядків таблиці результатів запиту (альтернативна форма функції COUNT ( )).

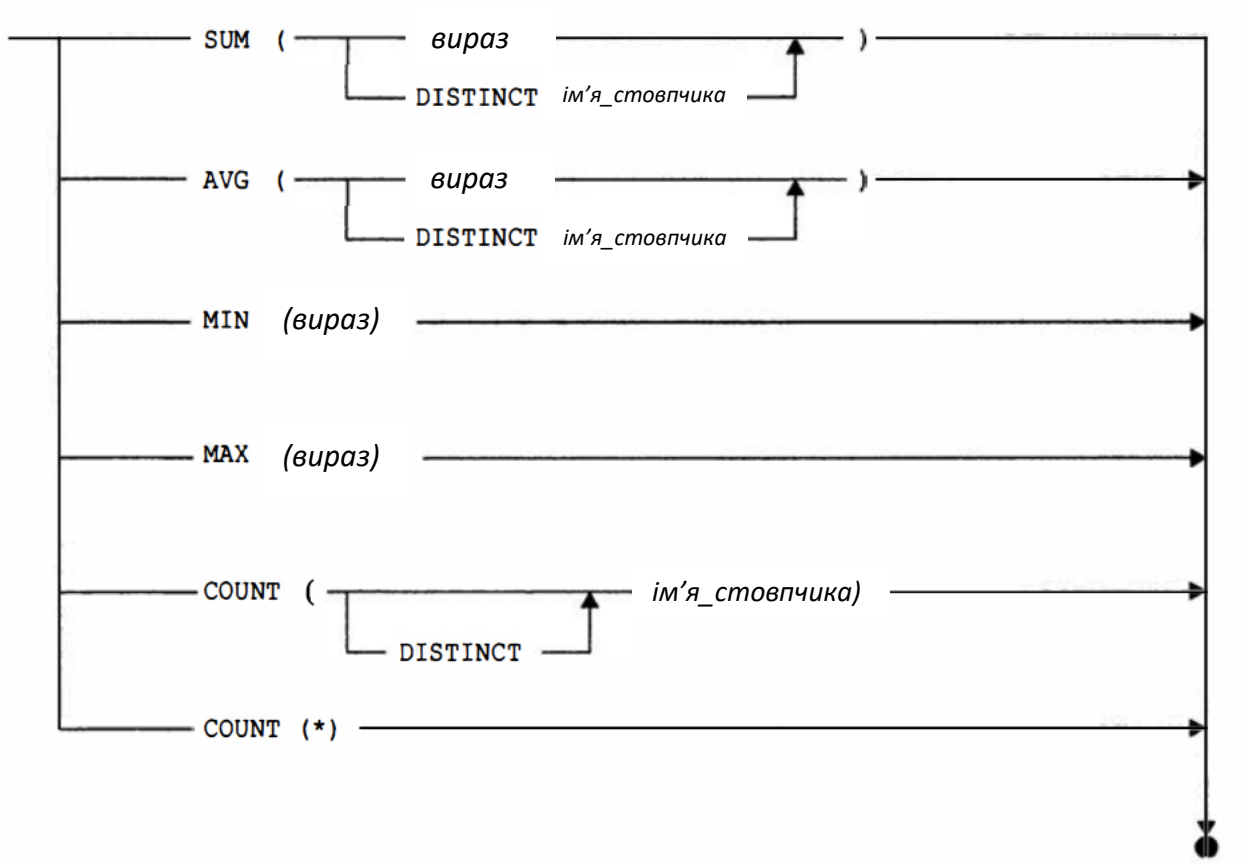

Рисунок 19.9. Синтаксична діаграма статистичних функцій

Аргументом статистичної функції може бути як проста назва стовпця так й вираз, як наведено в наступному запиті.

Визначити середній відсоток виконання плану в компанії.

```
SELECT AVG(100 * (SALES/QUOTA))
  FROM SALESREPS;

AVG(100 * (SALES/QUOTA))
------------------------
                  102.60
```

Під час виконання цього запиту СУБД створює тимчасовий стовпець, який містить значення 100 * (SALES/QUOTA) для кожного рядка таблиці SALESREPS, а потім обраховує середнє значення тимчасового стовпця.

Обрахунок суми значень стовпця

Статистична функція SUM () обраховує суму всіх значень стовпця. При цьому стовпець повинен мати числовий тип даних (має містити цілі числа, десяткові числа, числа с плаваючою крапкою або грошові величини). Результат, який повертає ця функція, має той самий тип даних, що й стовпець, проте точність результата може бути більшою. Наприклад, якщо застосувати функцію SUM () до даних стовпчика, який містить 16-разрядні цілі числа, вона може повернути за результат 32-разрядне ціле число. Нижче наведено кілька прикладів, використання функції SUM ( ) .

Визначити загальний плановий та загальний фактичний обсяги продаж.

```
SELECT SUM(QUOTA), SUM(SALES)
  FROM SALESREPS;

   SUM(QUOTA)         SUM(SALES)
-------------      -------------
$2,700,000.00      $2,893,532.00
```

Визначити суму всіх замовлень, які були прийняті Біллом Адамсом.

```
SELECT SUM(AMOUNT)
  FROM ORDERS, SALESREPS
 WHERE NAME = 'Bill Adams'
   AND REP = EMPL_NUM;

  SUM(AMOUNT)
------------
  $39,327.00
```

Визначення середнього значення стовпця

Статистична функція AVG () обчислює середнє арифметичне всіх значень стовпчика. Дані, які містяться в стовпчику, повинні мати числовий тип. Оскільки функція AVG () спочатку визначає суму всіх значень, а потім ділить цю суму на кількість цих значень, результат, який повертає ця функція може мати тип даних, який не співпадає з типом даних стовпця. Наприклад, якщо застосувати функцію AVG () до стовпчика цілих чисел, результат може бути або десятковим числом, або числом с плаваючою крапкою, залежно від СУБД, яка використовується.

Нижче наведено кілька прикладів використання функції AVG ( ) .

Визначити середню ціну товарів від виробника ACI.

```
SELECT AVG(PRICE)
  FROM PRODUCTS
 WHERE MFR_ID = 'ACI';

AVG(PRICE)
----------
   $804.29
```

Визначити середню вартість замовлень, які зробила компанія Acme Mfg. (клієнт 2103).

```
SELECT AVG(AMOUNT)
  FROM ORDERS
 WHERE CUST = 2103;

AVG(AMOUNT)
-----------
  $8,895.50
```

Обрахунок граничних значень

Статистичні функції MIN () и MAX () дозволяють знаходити відповідно найменше та найбільше значення в стовпчику. Стовпчик за цього може містити числа, рядки або значення дати та часу. Результат, який повертають ці функції, має той самий тип даних, що й стовпець.

Нижче наведено кілька прикладів використання цих функцій.

Визначити найбільший та найменший планові обсяги продажів.

```
SELECT MIN(QUOTA), MAX(QUOTA)
  FROM SALESREPS;

  MIN(QUOTA)    MAX(QUOTA)
------------  ------------
 $200,000.00  $350,000.00
```

Визначити дату першого замовлення.

```
SELECT MIN(ORDER_DATE)
  FROM ORDERS;

MIN(ORDER_DATE)
---------------
2007-01-04
```

Визначити найбільший відсоток виконання плану серед всіх співробітників.

```
SELECT MAX(100 * (SALES/QUOTA))
  FROM SALESREPS;

MAX(100 * (SALES/QUOTA))
------------------------
                  135.44
```

Якщо функції MIN () і MAX () застосовуються до числових даних, то числа порівнюються за арифметичними правилами (серед двох від'ємних чисел меншим є те, яке є більшим за модулем; нуль є меншим за будь-яке додатне число і більшим за будь-яке від'ємне число). Порівняння дат відбувається послідовно (більш ранні значення дати вважаються меншими, ніж пізніші дати). Часові інтервали порівнюються виходячи з їх тривалості (коротші проміжки часу вважаються меншими, ніж довші проміжки часу)

У випадку застосування функцій MIN () та MAX () до рядкових даних результат порівняння двох рядків залежить від таблиці кодування, яка використовується. За використання таблиці кодування ASCII (наприклад на персональних комп'ютерах): цифри їдуть перед літерами, а всі великі літери – перед маленькими. На мейнфреймах компанії BM, де використовується таблиця кодування EBCDIC, маленькі літери розташовані перед великими, а цифри їдуть за літерами. Нижче наведено порівняння послідовностей сортування, прийнятих в таблицях кодування ASCII та EBCDIC, на прикладі рядків впорядкованих за зростанням.

ASCII	EBCDIC
1234ABC	acme mfg.
5678ABC	zeta corp.
ACME MFG.	Acme Mfg.
Acme Mfg.	ACME MFG.
ZETA CORP.	Zeta Corp.
Zeta Corp.	ZETA CORP.
acme mfg.	1234ABC
zeta corp.	5678ABC

Відмінності в порядку сортування призводять до того, що той самий запит, який містить оператор ORDER BY, в різних системах може привести до різних результатів. Зберігання в таблицях символів національних алфавітів (наприклад, кирилиці) може викликати додаткові проблеми. В деяких СУБД для кожної мови використовується власний алгоритм сортування. В інших СУБД такі символи сортуються у відповідності з кодом символа. Для рішення цієї проблеми стандарт SQL містить підтримку національних наборів символів, користувацьких наборів символів тв альтернативних послідовностей сортування.

Підрахунок кількісті даних

Статистична функція COUNT ( ) підраховує кількість значень в стовпці; тип даних стовпця не має значення. Функція COUNT ( ) завжди повертає ціле число, незалежно від типу даних стовпця. Нижче наведено кілька запитів з використанням цієї функції.

Визначити кількість клієнтів компанії.

```
SELECT COUNT(CUST_NUM)
  FROM CUSTOMERS;

 COUNT(CUST_NUM)
----------------
             21
```

Визначити кількість працівників, які перевиконали план.

```
SELECT COUNT(NAME)
  FROM SALESREPS
 WHERE SALES > QUOTA$

COUNT(NAME)
-----------
          7
```

Визначити кількість замовлень вартістю більше за $25000.

```
SELECT COUNT(AMOUNT)
  FROM ORDERS
 WHERE AMOUNT > 25000.00;

COUNT(AMOUNT)
-------------
            4
```

Зверніть увагу, що функція COUNT () з переданими в неї даними стовпця не враховує значення NULL в цьому стовпці; це робить функція COUNT (*), яка підраховує всі рядки незалежно від їх значень. Якщо не враховувати можливості наявності значення NULL, то функція COUNT () ігнорує значення даних в стовпці та просто підраховує їх кількість. Тобто не має значення, який саме стовпець передано до функції COUNT () в якості аргумента. Тоді, останній приклад може бути переписаним наступним чином.

```
SELECT COUNT(ORDER_NUM)
  FROM ORDERS
 WHERE AMOUNT > 25000.00;

 COUNT(ORDER_NUM)
-----------------
                4
```

Важко уявити запит на зразок "підрахувати кількість вартостей замовлень" набагато легше уявити запит "підрахувати кількість замовлень". Тому в SQL було введено спеціальну статистичну функцію COUNT (*), яка підраховує кількість рядків, а не значень даних. Розглянемо попередній запит, переписаний за використання цієї функції.

```
SELECT COUNT(*)
  FROM ORDERS
 WHERE AMOUNT > 25000.00;

COUNT(*)
--------
       4
```

На практиці для підрахунку кількості рядків завжди використовується функція COUNT (*), а не COUNT ( ).

Статистичні функції в переліку стовпців результату

Якщо в список результуючих стовпців входить декілька статистичних функцій, або якщо аргумент функції є складним виразом, то розуміння запиту ускладнюється. Нижче наведено кроки виконання SQL-запитів, які були розширені за рахунок використання статистичних функцій. Ці правила є визначенням того, що означає запит, але не того, як СУБД насправді отримує результати запиту.

Для генерації результатів запиту SELECT слід виконати наступні кроки.

1. Якщо запит є об’єднанням (UNION) інструкцій SELECT, для кожної з них виконати кроки 2-5 для генерації окремих результатів запитів.

2. Сформувати добуток таблиць, які вказані в операторі FROM. Якщо там вказано лише одну таблицю, то добутком буде вона сама.

3. За наявності оператора WHERE застосувати умову, яка в ньому вказано до кожного рядка таблиці добутку, залишаючи лише ті рядки,

для умова є істиною, та відкидаючи рядки, для яких умова є хибною або дорівнює NULL.

4. Для кожного рядка з тих, що залишилися, розрахувати значення кожного елемента в списку результуючих стовпців таким чином створивши один рядок таблиці результатів запиту. При цьому за будь-якого посилання на стовпець використовується значення стовпця в поточному рядку, а в якості аргумента статистичної функції використовується весь набір рядків.

5. Якщо вказано предикат DISTINCT, слід видалити з таблиці результатів запиту всі рядки, що повторюються.

6. Якщо запит є об'єднанням інструкцій SELECT, слід об'єднати результати виконання окремих інструкцій в одну таблицю результатів запиту. Видалити з неї рядки, що повторюються, лише у випадку якщо в запиті не вказано предикат UNI ON ALL.

7. Якщо є оператор ORDER BY, відсортувати результати запиту, відповідно з вимогами цього оператора.

Одним з найкращих способів зрозуміти, як виконуються підсумкові запити зі статистичними функціями, це представити запит в два етапи. Спочатку визначити, якими є результати запиту без статистичних функцій, тобто якими є кілька рядків детального результату запиту. Потім визначити, як СУБД застосовує статистичні функції до деталізованих результатів запиту, повертаючи один отриманий рядок.

Наприклад, розглянемо наступний складний запит.

Визначити середню вартість замовлення, загальну вартість замовлення, середню вартість замовлення у відсотках до кредитного ліміту, а також середню вартість замовлення у відсотках від планового розміру продажів працівників.

```
SELECT AVG(AMOUNT), SUM(AMOUNT),
       (100 * AVG(AMOUNT/CREDIT_LIMIT)),
       (100 * AVG(AMOUNT/QUOTA))
  FROM ORDERS, CUSTOMERS, SALESREPS
 WHERE CUST = CUST_NUM
   AND REP = EMPL_NUM;
```

```
AVG(AMOUNT) SUM(AMOUNT) (100*AVG(AMOUNT/CREDIT_LIMIT))
----------- ----------- ------------------------------
  $8,256.37 $247,691.00                          24.44

 (100*AVG(AMOUNT/QUOTA))
 -----------------------
                    2.51
```

За відсутності статистичних функцій запит виглядав би наступним чином.

```
SELECT AMOUNT, AMOUNT, AMOUNT/CREDIT_LIMIT, AMOUNT/QUOTA
  FROM ORDERS, CUSTOMERS, SALESREPS
 WHERE CUST = CUST_NUM
   AND REP = EMPL_NUM;
```

Цей запит повернув би один рядок результатів запиту для кожного замовлення. Статистичні функції використовують стовпці таблиці результатів запитів для отримання однорядкової таблиці з результатами. В операторах SQL статистична функція може бути використана всюди, де можна вказати назву стовпця. Наприклад, це може бути вираз, який підсумовує або обраховуються значення двох статистичних функцій. Проте в деяких реалізаціях SQL, таких як ті, що базуються на стандарті SQL1, аргумент статистичної функції не може містити іншу статистичну функцію, оскільки отриманий вираз не бути мати сенсу. Іноді це правило формулюється наступним чином: «Функції не повинні бути вкладені».

Функції SUM ( ) , AVG (), MIN ( ) , MAX ( ) та COUNT () за аргумент приймають стовпець значень та повертають за результат одне значення. Розглянемо, що відбувається якщо в стовпці зустрічаються одне або кілька значень NULL. У стандарті ANSI/ISO сказано, що значення NULL статистичними функціями ігноруються.

Наступний запит демонструє, що статистична функція COUNT ( ) ігнорує всі значення NULL, які містяться в стовпці.

```
SELECT COUNT(*), COUNT(SALES), COUNT(QUOTA)
  FROM SALESREPS;

COUNT(*)    COUNT(SALES)      COUNT(QUOTA)
--------    ------------      ------------
      10              10                 9
```

В таблиці SALESREPS міститься десять рядків, тому функція COUNT (*) повертає число 10. В стовбці SALES міститься десять значень, до того ж жодне з них не дорівнює NULL, тому функція COUNT (SALES) також повертає число 10. Але в стовбці QUOTA міститься одне значення NULL – для працівника, якого було прийнято зовсім нещодавно. Функція COUNT (QUOTA) ігнорує це значення та повертає число 9. Саме через такі розбіжності замість функції COUNT ( ) для підрахунку рядків майже завжди використовується функція COUNT (*). Виключення складають випадки, коли не потрібно враховувати рядки, які містять значення NULL у певному стовпці.

Ігнорування значень NULL не впливає на результати, які повертають функції MIN ( ) та MAX (). Проте може привести до проблем за використання функцій SUM ( ) та AVG ( ).

В стандарті ANSI / ISO визначені наступні чіткі правила обробки значень NULL в статистичних функціях.

• якщо будь-які зі значень, які містяться в стовпці, дорівнюють NULL, то за обрахунку результата функції вони ігноруються.

• якщо всі значення в стовпці дорівнюють NULL, то функції SUM ( ) , AVG (), MIN ( ) та MAX () повертають значення NULL; функція COUNT () повертає нуль.

• якщо в стовпці відсутні значення (тобто стовпець порожній), то функції SUM ( ), AVG (), MIN () та MAX () повертають значення NULL; функція COUNT () повертає нуль.

• функція COUNT (*) підраховує кількість рядків незалежно від наявності чи відсутності в стовпці значень NULL, якщо рядків в таблиці нема, ця функція повертає нуль.

19.21 Видалення рядків, які повторюються (DISTINCT)

Ключове слово DISTINCT ставиться на початку списка стовпців, що повертаються та слугує для видалення рядків, які повторюються з таблиці результатів запиту. За допомогою цього предиката можна також вказивати, що перш ніж застосовувати до стовпця статистичної функції

з нього слід видалити всі значення, які повторюються. Для цього необхідно включити предикат DISTINCT перед аргументом статистичної функції одразу після дужки, що розкривається. Нижче наведені два запити, які ілюструють видалення значень, які повторюються, перед застосуванням статистичної функції.

Визначити кількість різних посад, які і∞снують в компанії.

```
SELECT COUNT(DISTINCT TITLE)
  FROM SALESREPS;

COUNT(DISTINCT TITLE)
---------------------
                    3
```

Визначити кількість офісів, в яких є працівники, які перевищили планові обсяги продажів.

```
SELECT COUNT(DISTINCT REP_OFFICE)
  FROM SALESREPS
 WHERE SALES > QUOTA;

COUNT(DISTINCT REP_OFFICE)
--------------------------
                         4
```

Ключове слово DISTINCT можна використовувати лише один раз в одному запиті. Якщо воно застосовується разом з аргументом однієї зі статистичних функцій, то не може бути використано з будь-яким іншим аргументом. Якщо DISTINCT вказаний перед списком стовпців, що повертаються, то воно не може бути використаним в жодній статистичній функції. Єдиним винятком з цього правила є випадок, коли DISTINCT використовується вдруге в рамках підпитання.

19.22 Застосування підзапитів

Підзапит — це запит всередині іншого SQL-запиту. Результати підзапитів використовуються СУБД для визначення результатів запиту вищого рівня, який містить даний підзапит. У найпростішому випадку підзапит міститься в операторі WHERE або HAVING іншої SQL-інструкції. Підзапити забезпечують ефективний природний спосіб обробки запитів, які виражаються через результати інших запитів. Наведемо приклад такого запиту.

Вивести список офісів, в яких планований обсяг продажів перевищує суму планованих продажів всіх співробітників.

В даному запиті необхідно отримати список офісів з таблиці OFFICES, для яких значення стовпця TARGET задовольняє певній умові. Логічно подумати, що інструкція SELECT, яка реалізує даний запит, має виглядати приблизно наступним чином:

```
SELECT CITY     FROM OFFICES     WHERE TARGET > ???
```

Тут величина ??? дорівнює сумі планових обсягів продажів всіх працівників, які працюють в даному офісі. Суму планових обсягів продажів для певного офісу (наприклад, для офісу з ідентифікатором 21) можна отримати за допомогою наступного запиту.

```
SELECT SUM(QUOTA)
  FROM SALESREPS
 WHERE REP_OFFICE = 21;
```

Однак введення такого запиту, написання результатів, а потім введення попереднього запиту з правильним значенням не є найефективнішим способом роботи. Краще спочатку написати перший запит, а потім замінити ? ? ? другим запитом.

```
SELECT CITY
  FROM OFFICES
 WHERE TARGET > (SELECT SUM(QUOTA)
                   FROM SALESREPS
                  WHERE REP_OFFICE = OFFICE)
```

Фактично це коректний SQL-запит. Внутрішній запит (підзапит) обраховує для кожного офісу суму планових обсягів продажів всіх працівників даного офісу. Головний (зовнішній) запит порівнює план продажів офіса з отриманою сумою та, залежно від результата порівняння, або додає даний офіс до таблиці результатів запиту, або ні. Разом головний та підлеглий запити створюють початковий запит та дістають з бази даних потрібну інформацію. Підлеглі SQL-запити зазвичай є частиною операторів WHERE або HAVING. В операторі WHERE вони допомогають вибирати з таблиці результатів запиту окремі рядки, а в операторі HAVING – групи рядків.

На рисунку 19.10 зображена структура підзапиту SQL.

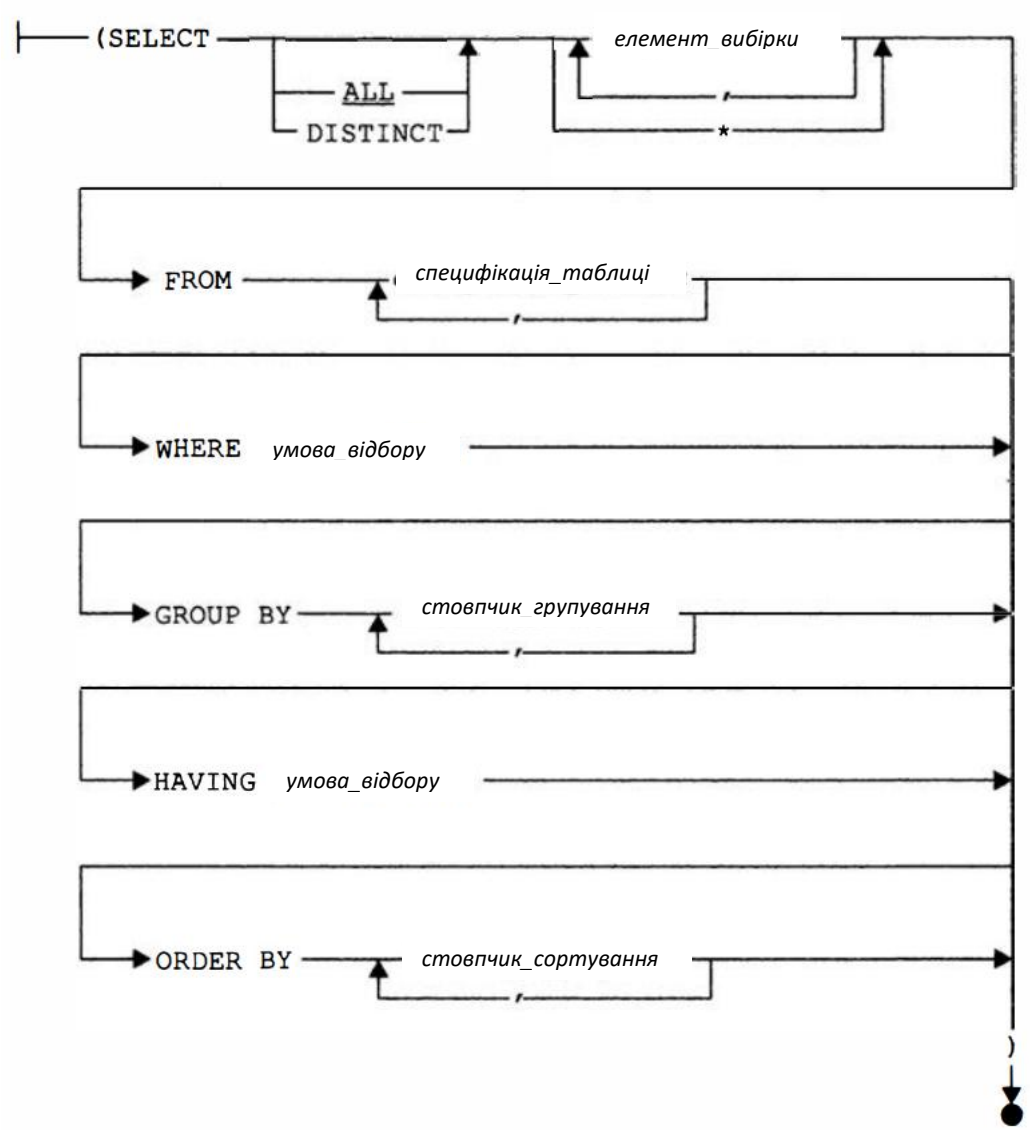

Рисунок 19.10. Синтаксична діаграма підзапиту

Підзапит завжди розташовується всередині круглих дужок, зберігаючи при цьому звичайну структуру інструкції SELECT, яка містить оператор FROM та можливо оператори WHERE, GROUP BY та HAVING. Структура цих операторів в підзапиті є ідентичною їх структурі в інструкції SELECT; в підзапиті ці оператори виконують свої звичайні функції. Проте між підзапиом та інструкцією SELECT є декілька відмінностей.

• У більшості випадків результати підзапиту завжди складаються з одного стовпця. Це означає, що в операторі SELECT підзапиту майже завжди вказує лише один елемент списку вибору.

• Хоча в підзапиті може знаходитися оператор ORDER BY, насправді він використовується дуже рідко. Результати підзапиту використовуються лише всередині головного запиту та користувачі його не бачать, тому немає сенсу їх сортування. Крім того, сортування великої кількості даних може негативно впливати на продуктивності.

• Назви стовпців у підзапиті можуть посилатися на стовпці таблиць головного запиту.

• В більшості реалізацій SQL підзапит не може бути об’єднанням (UNION) кількох різних інструкцій SELECT; допускається використання лише однієї інструкції SELECT. (Стандарт SQL послаблює це обмеження, дозволяючи створювати набагато потужніші запити).