ИСПОЛЬЗОВАНИЕ ВЛОЖЕННЫХ SQL-ЗАПРОСОВ

ЛЕКЦИЯ 3

ПОДЗАПРОСЫ

ВОПРОС 1

Подзапрос - очень мощное средство языка SQL. Он позволяет строить сложные иерархии запросов, многократно выполняемые в процессе построения результирующего набора или выполнения одного из операторов изменения данных (*DELETE* , *INSERT* , *UPDATE*).

Условно *подзапросы* иногда подразделяют на три типа, каждый из которых является сужением предыдущего:

табличный подзапрос, возвращающий набор строк и столбцов;

подзапрос строки, возвращающий только одну строку, но, возможно, несколько столбцов (такие *подзапросы* часто используются во *встроенном SQL*);

скалярный подзапрос, возвращающий значение одного столбца в одной строке.

ПОДЗАПРОС ПОЗВОЛЯЕТ РЕШАТЬ СЛЕДУЮЩИЕ ЗАДАЧИ:

- определять набор строк, добавляемый в таблицу на одно выполнение оператора *INSERT* ;
- определять данные, включаемые в представление, создаваемое оператором *CREATE VIEW* ;
- определять значения, *модифицируемые* оператором *UPDATE* ;
- указывать одно или несколько значений во фразах WHERE и HAVING оператора SELECT ;
- определять во фразе FROM таблицу как результат выполнения *подзапроса* ;
- применять *коррелированные подзапросы*.

Подзапрос называется коррелированным, если запрос, содержащийся в предикате, имеет ссылку на значение из таблицы (внешней к данному запросу), которая проверяется посредством данного предиката.

Подзапрос может быть указан как в предикате, определяемом фразой WHERE, так и в предикате по группам, определяемом фразой HAVING.

Например:

```
SELECT avg_f1, COUNT (f2) from tbl1
    GROUP BY avg_f1
    HAVING avg_f1 >(SELECT f1 FROM tbl1
                WHERE f3='a1');
```

В случае *коррелированного подзапроса* во фразе HAVING можно использовать только агрегирующие функции, так как каждый раз на момент выполнения *подзапроса* в качестве проверяемой строки, к значениям которой имеет доступ *подзапрос*, выступает результат группирования строк на основе агрегирующих функций основного запроса.

Например:

```
SELECT f1, COUNT(*), SUM(f2) from tbl1 t1
    GROUP BY f1
    HAVING SUM(f2)> (SELECT MIN(f2)*4
                     FROM tbl1 t1_in
                    WHERE t1.f1=t1_in.f1);
```

ПОСТРОЕНИЕ ПРЕЛИКАТА ЛЛЯ

Для использования результата *подзапроса* в предикате также применяются операторы ***ANY*** и ***ALL***, которые были подробно рассмотрены в предыдущих лекциях.

Приведем пример использования оператора ***ANY***:

SELECT f1,f2,f3 from tbl1

WHERE f3 = ANY (SELECT f3 FROM tbl2);

Данный оператор определяет, что в результирующий набор будут включены все строки, значение столбца f3 которых присутствует в таблице tbl2.

ПРИМЕНЕНИЕ ПОДЗАПРОСОВ В ОПЕРАТОРАХ ИЗМЕНЕНИЯ ДАННЫХ

ВОПРОС 2

- К операторам языка *DML*, кроме оператора ***SELECT***, относятся *операторы*, позволяющие изменять данные в таблицах. Это оператор ***INSERT*** , выполняющий добавление одной или нескольких строк в таблицу, оператор ***DELETE*** , удаляющий из таблицы одну или несколько строк, и оператор ***UPDATE*** , изменяющий значения столбцов таблицы.

ОПЕРАТОР INSERT

- Оператор *INSERT* имеет следующее формальное описание:

INSERT INTO table_name
[(field .,:)]
{ VALUES (value .,:) }
| subquery
| {DEFAULT VALUES};

Оператор ***INSERT*** может добавлять в таблицу как одну, так и несколько строк. Список полей (field .,:) указывает имена полей и порядок занесения в них значений из *списка значений*, определяемого фразой ***VALUES***, или как результат выполнения *подзапроса*.

- Список, определяемый фразой VALUES, называется конструктором значений таблицы и указывается в круглых скобках через запятую.
- Если список полей (field .,:) опущен, то порядок занесения значений будет соответствовать порядку столбцов, указанному в операторе *CREATE TABLE* при создании данной таблицы.
- Если для столбцов, на которые установлено ограничение NOT NULL, не указано добавляемых данных, то СУБД инициирует ошибку выполнения SQL-оператора.

ОПЕРАТОР *INSERT*

- Оператор *INSERT* демонстрирует копирование строк таблицы tbl2, выполняемое на основе *подзапроса*:

```
INSERT INTO tbl1(f1,f2,f3)
   (SELECT f1,f2,f3 FROM tbl2);
```

Очевидно, что количество полей, указываемое списком полей, и типы данных этих полей должны совпадать с количеством полей и их типами данных в конструкторе значений таблицы или в результирующем наборе, формируемом *подзапросом*.

ОПЕРАТОР DELETE

- Оператор *DELETE* имеет следующее формальное описание:

DELETE FROM table_name
[{ WHERE condition }
| { WHERE CURRENT OF cursor_name }];

- Оператор *DELETE* используется для удаления из таблицы строк, указанных условием во фразе WHERE (поисковое удаление, searched deletion) или WHERE CURRENT OF (позиционное удаление, positioned deletion).
- Позиционное удаление, определяемое фразой WHERE CURRENT OF, удаляя строки из курсора, соответственно удаляет их и из той таблицы базы данных, на базе которой был построен этот курсор.
- Если оператор *DELETE* применяется к какому-либо представлению, то данные удаляются также из созданной на основе последнего таблицы базы данных.
- Никогда нельзя забывать, что если фраза WHERE будет отсутствовать или предикат во фразе WHERE будет всегда принимать значение TRUE, то оператор *DELETE* удалит из таблицы все строки.

При вычислении значений столбцов можно применять условное выражение *CASE* и выражение CAST для приведения типов.

Например:

SELECT f1, CAST (f2 AS CHAR),
CAST (f3 AS CHAR) from tbl1;
UPDATE tbl2 SET f2 = (SELECT CAST (f2 AS CHAR) from tbl1 WHERE f1=1);UPDATE tbl2 SET f5 = (SELECT CAST (f5 AS DATE)
from tbl1 WHERE f1=1),
f6= CAST ('10/12/2003' AS DATE);

УСЛОВНОЕ ВЫРАЖЕНИЕ CASE

- Условное выражение *CASE* позволяет выбрать одно из нескольких значений на основании указываемого условия.
- Условное выражение *CASE* имеет следующее формальное описание:

{ CASE

{ expr WHEN expr THEN { expr | NULL }}

| { WHEN expr THEN { expr | NULL }}

[ELSE { expr | NULL }] END}

| { NULLIF {expr1,expr2) }

| {COALESCE (expr .,:) }

УСЛОВНОЕ ВЫРАЖЕНИЕ *CASE* МОЖЕТ БЫТЬ ЗАПИСАНО, СООТВЕТСТВЕННО, В ЧЕТЫРЕХ ФОРМАХ:

CASE с выражениями.
Например:

```
SELECT f1, CASE f3
    WHEN 'abc' THEN '1_abc' END
 FROM tbl1;
```

CASE с предикатами. Например:

```
SELECT f1, CASE
    WHEN f3= 'abc' THEN '1_abc‘
      ELSE f3 END FROM tbl1;
```

NULLIF - если выражения, указанные в скобках, не совпадают, то выбирается первое из этих значений, в противном случае устанавливается значение NULL. Например:

```
UPDATE tbl2 SET f3 = (SELECT NULLIF
(f3,'aaa')
                FROM tbl1 WHERE f1=1);
```

COALESCE - выбирается первое значение в списке, не равное NULL. Например:

```
INSERT INTO tbl1(f1,f2)
    VALUES (1+ COALESCE(SELECT
MAX(f1)
      FROM tbl1,  0 ), 100);
```

ДЛЯ УСПЕШНОГО ВЫПОЛНЕНИЯ ОПЕРАТОРА *UPDATE* ТРЕБУЕТСЯ РЯД УСЛОВИЙ, ВКЛЮЧАЮЩИЙ СЛЕДУЮЩИЕ:

- наличие соответствующих привилегий;
- для представления требуется определение его как изменяемого;
- при изменении представлений применяются ограничения WITH CHECK OPTION или WITH CASCADED CHECK OPTION, установленные при создании этого представления;
- в транзакциях "только чтение" изменение доступно только для временных таблиц;
- выражения, используемые для определения значений, не могут содержать *подзапросы* с агрегирующими функциями;
- для обновляемого курсора, указанного фразой FOR UPDATE, каждый изменяемый столбец также должен быть определен как FOR UPDATE ;
- в курсоре с фразой ORDER BY нельзя выполнять изменение столбцов, указанных в этой фразе.

СОЗДАНИЕ ПРЕДСТАВЛЕНИЙ

ВОПРОС 3

- ***Представление (view)***, иногда называемое также видом, определяет логическую таблицу, получаемую как результат выполнения сохраненного запроса. Представление - это некоторая логическая (*виртуальная) таблица*, которая формируется заново каждый раз, когда в SQL-операторе встречается ссылка на конкретное представление. Результирующий набор, создаваемый как результат выполнения запроса, определяющего данное *представление*, формируется из полей других таблиц *базы данных*. Таблицы, используемые в запросе для создания представления, называются простыми основными таблицами.
- *Представление* является объектом схемы и используется как логическая *таблица базы данных*.
- Для определения представления применяется оператор ***CREATE VIEW*** .

ОПЕРАТОР CREATE VIEW

- Оператор *CREATE VIEW* следующее формальное описание:

```
CREATE VIEW table_name [(field .,:) ]
  AS (SELECT_operator
     [WITH [CASCADED | LOCAL]
       CHECK OPTION ] );
```

- Список полей (field), указываемый после имени представления, позволяет переименовать столбцы основных таблиц, используемых в запросе. Это может потребоваться в случае совпадения имен столбцов при запросах, использующих объединение таблиц; для именования вычисляемых столбцов; для именования объединенных столбцов, полученных посредством соединения столбцов из двух таблиц, имеющих различные имена полей.

Оператор запроса *SELECT* , использующийся для построения представления, может иметь две формы:

Расширяемая форма оператора *SELECT* задается как конструкция *SELECT* *, не ограничивая жестко список полей, извлекаемых в запрос. Это позволяет не менять синтаксис представления при изменении оператором *ALTER TABLE* структуры таблицы: добавлении новых столбцов или удалении столбцов.

Постоянная форма оператора *SELECT* задается как конструкция *SELECT список_столбцов*, жестко фиксируя имена столбцов, входящих в запрос.

Как будет влиять изменение основных таблиц на представление, можно указать в операторе *ALTER TABLE* :

фраза RESTRICT определяет ограничение, отменяющее *изменение таблицы*, если на данный столбец есть ссылки в представлениях (а также в ограничениях и предикатах);

фраза CASCADE указывает, что все представления, использующие удаляемый столбец, также будут удалены (а также все *внешние ключи*, имеющие ссылки на удаляемый столбец или ограничения *FOREIGN* KEY).

Поддержка оператора *ALTER TABLE* необходима только для полного уровня соответствия стандарту, однако, большинство коммерческих СУБД реализует этот оператор, но с некоторыми изменениями и расширениями.

Следующий оператор иллюстрирует *изменение таблицы*, приводящее к удалению всех представлений, ссылающихся на столбец f2изменяемой таблицы:

ALTER TABLE tbl1 DROP COLUMN f2 CASCADE;

ИЗМЕНЕНИЕ ДАННЫХ В ПРЕДСТАВЛЕНИЯХ

- Если для представления указывается оператор DELETE, INSERT или UPDATE, то все изменения происходят как над представлением, так и над основными таблицами, используемыми для создания представления. Не во все представления можно внести изменения. Так, представления могут быть изменяемыми или постоянными.
- Стандарт позволяет внесение изменений всегда только в одну основную таблицу. Однако большинство коммерческих *СУБД* позволяют вносить изменения и в две связанные между собой таблицы, но с некоторыми оговорками.

ПРЕДСТАВЛЕНИЕ ЯВЛЯЕТСЯ ИЗМЕНЯЕМЫМ, ЕСЛИ ВЫПОЛНЕНЫ СЛЕДУЮЩИЕ УСЛОВИЯ:

запрос, используемый для создания представления, извлекает данные только из одной таблицы;

если в запросе, используемом для создания таблицы, в качестве таблицы выступает представление, то оно также должно быть изменяемым;

не разрешается никаких объединений таблиц, даже самой с собой;

в запросе, используемом для создания представления, нельзя ссылаться дважды на один и тот же столбец.

запрос, используемый для создания представления, не должен содержать вычислимых столбцов, агрегирующих функций и фраз *DISTINCT*, GROUP BY и HAVING ;

ОПЦИИ [WITH [CASCADED | LOCAL] CHECK OPTION

- Для *изменяемого представления* можно указывать фразу WITH CHECK OPTION, позволяющую предотвращать "потерю строк" в представлениях. Так, если эта фраза указана, то при внесении изменений в таблицу будет проверен предикат, указанный в запросе, использованном для создания таблицы. Если предикат не возвращает значение TRUE, то изменения не будут внесены.
- Например, если запрос создан оператором

```
CREATE VIEW v_tbl1 AS
  (SELECT f1,f2, f3 FROM tbl1 WHERE f2>100)
   WITH CHECK OPTION;,
```

- то вставка строки не будет произведена:

```
INSERT INTO v_tbl1 (f1,f2,f3)
       VALUES (1,50,'abc');
```

Так, для представления, созданного операторами

```
CREATE VIEW v_1 AS
   (SELECT f1,f2, f3 FROM tbl1 WHERE f2>100);,
CREATE VIEW v_2 AS
   (SELECT f1,f2, f3 FROM v_1 WHERE f2>50)
   WITH LOCAL CHECK OPTION;,
```

добавление строки будет выполнено:

```
INSERT INTO v_2 (f1,f2,f3)
      VALUES (1,30,'abc');.
```

Эта строка будет добавлена в основную таблицу, но не будет видна в представлении, посредством которого она была добавлена.

По умолчанию предполагается, что для WITH CHECK OPTION используется фраза CASCADED.

ТИПЫ ДАННЫХ

ВОПРОС 4

ТИПЫ ДАННЫХ

символьные:

- CHARACTER (*len*) ;
- CHAR (*len*) ;
- *CHARACTER VARYING* (*len*) ;
- CHAR *VARYING* (*len*) ;
- VARCHAR (*len*) ;
- *NATIONAL CHARACTER* (*len*) ;
- NATIONAL CHAR (*len*) ;
- NCHAR (*len*) ;
- *NATIONAL CHARACTER VARYING* (*len*) ;
- NATIONAL CHAR *VARYING* (*len*) ;
- NCHAR *VARYING* (*len*) ;

- двоичные:
  - BIT (*len*) ;
  - BIT *VARYING* (*len*) ;
- числовые:
  - *NUMERIC* ;
  - DECIMAL ;
  - DEC ;
  - INTEGER ;
  - INT ;
  - *SMALLINT* ;
  - FLOAT ;
  - REAL ;
  - *DOUBLE PRECISION* ;
- даты/времени:
  - DATE ;
  - TIME ;
  - TIME WITH *TIME ZONE* ;
  - *TIMESTAMP* ;
  - *TIMESTAMP* WITH *TIME ZONE* ;
- интервалы:
- *INTERVAL*.

Параметры типа, указываемые в скобках, в большинстве случаев можно при необходимости опускать.

Для символьных типов возможно указание фразы CHARACTER SET { set_name | using_form}, устанавливающей используемый набор символов.

ОПИСАНИЕ НАИБОЛЕЕ ЧАСТО

Функции даты/времени

Для работы с данными, имеющими тип даты/времени в языке SQL предусмотрены следующие функции:

CURRENT_TIME - определяет текущее время;

CURRENT_DATE - определяет текущую дату;

CURRENT_TIMESTAMP - определяет текущие дату и время.

Например:

```
INSERT INTO tbl1 (f1,f2,f3,f4)
    VALUES (1,100,'abc', CURRENT_DATE);
```

определяется установками компьютера.