ТЕМА 10. ОПРАТОРЫ INSERT, UPDATE, DELETE

DELETE

DELETE — удалить записи таблицы

Синтаксис

```
DELETE FROM [ ONLY ] имя_таблицы [ * ] [ [ AS ] псевдоним ]
    [ USING список_USING ]
    [ WHERE условие ]
    [ RETURNING * | выражение_результата [ [ AS ] имя_результата ] [,
...] ]
```

Описание

Команда DELETE удаляет из указанной таблицы строки, удовлетворяющие условию WHERE. Если предложение WHERE отсутствует, она удаляет из таблицы все строки, в результате будет получена рабочая, но пустая таблица.

Удалить строки в таблице, используя информацию из других таблиц в базе данных, можно двумя способами: применяя вложенные запросы или указав дополнительные таблицы в предложении USING. Выбор предпочитаемого варианта зависит от конкретных обстоятельств.

Предложение RETURNING указывает, что команда DELETE должна вычислить и возвратить значения для каждой фактически удалённой строки. Вычислить в нём можно любое выражение со столбцами целевой таблицы и/или столбцами других таблиц, упомянутых в USING. Список RETURNING имеет тот же синтаксис, что и список результатов SELECT.

Чтобы удалять данные из таблицы, необходимо иметь право DELETE для неё, а также право SELECT для всех таблиц, перечисленных в предложении USING, и таблиц, данные которых считываются в *условии*.

Параметры

имя_таблицы

Имя (возможно, дополненное схемой) таблицы, из которой будут удалены строки. Если перед именем таблицы добавлено ONLY, соответствующие строки удаляются только из указанной таблицы. Без ONLY строки будут также удалены из всех таблиц, унаследованных от указанной. При желании, после имени таблицы можно указать *, чтобы явно обозначить, что операция затрагивает все дочерние таблицы.

псевдоним

Альтернативное имя целевой таблицы. Когда указывается это имя, оно полностью скрывает фактическое имя таблицы. Например, в запросе DELETE FROM foo AS f дополнительные компоненты оператора DELETE должны обращаться к целевой таблице по имени f, а не foo.

список_USING

Список табличных выражений, позволяющий добавлять в условие WHERE столбцы из других таблиц. Он подобен списку таблиц, который можно задать в предложении FROM оператора SELECT; например, в нём можно определить псевдоним для таблицы. Целевую таблицу повторять в *списке_USING* нужно, только если требуется определить замкнутое соединение.

условие

Выражение, возвращающее значение типа boolean. Удалены будут только те строки, для которых это выражение возвращает true.

выражение_результата

Выражение, которое будет вычисляться и возвращаться командой DELETE после удаления каждой строки. В этом выражении можно использовать имена любых столбцов таблицы *имя_таблицы* или таблиц, перечисленных в списке USING. Чтобы получить все столбцы, достаточно написать *.

имя_результата

Имя, назначаемое возвращаемому столбцу.

Выводимая информация

В случае успешного завершения, DELETE возвращает метку команды в виде

```
DELETE число
```

Здесь *число* — количество удалённых строк. Заметьте, что это число может быть меньше числа строк, соответствующих *условию*, если удаления были подавлены триггером BEFORE DELETE. Если *число* равно 0, это означает, что запрос не удалил ни одной строки (это не считается ошибкой).

Если команда DELETE содержит предложение RETURNING, её результат будет похож на результат оператора SELECT (с теми же столбцами и значениями, что содержатся в списке RETURNING), полученный для строк, удалённых этой командой.

Замечания

Postgres Pro позволяет ссылаться на столбцы других таблиц в условии WHERE, когда эти таблицы перечисляются в предложении USING. Например, удалить все фильмы определённого продюсера можно так:

```
DELETE FROM films USING producers
  WHERE producer_id = producers.id AND producers.name = 'foo';
```

По сути в этом запросе выполняется соединение таблиц films и producers, и все успешно включённые в соединение строки в films помечаются для удаления. Этот синтаксис не соответствует стандарту. Следуя стандарту, эту задачу можно решить так:

```
DELETE FROM films
  WHERE producer_id IN (SELECT id FROM producers WHERE name = 'foo');
```

В ряде случаев запрос в стиле соединения легче написать и он может работать быстрее, чем в стиле вложенного запроса.

Примеры

Удаление всех фильмов, кроме мюзиклов:

```
DELETE FROM films WHERE kind <> 'Musical';
```

Очистка таблицы films:

```
DELETE FROM films;
```

Удаление завершённых задач с получением всех данных удалённых строк:

```
DELETE FROM tasks WHERE status = 'DONE' RETURNING *;
```

Удаление из tasks строки, на которой в текущий момент располагается курсор c_tasks:

```
DELETE FROM tasks WHERE CURRENT OF c_tasks;
```

Совместимость

Эта команда соответствует стандарту SQL, но предложения USING и RETURNING являются расширениями Postgres Pro, как и возможность использовать WITH с DELETE.

INSERT

INSERT — добавить строки в таблицу

Синтаксис

```
INSERT INTO имя_таблицы [ AS псевдоним ] [ ( имя_столбца [, ...] ) ]
    { DEFAULT VALUES | VALUES ( { выражение | DEFAULT } [, ...] ) [, ...] | запрос }
    [ RETURNING * | выражение_результата [ [ AS ] имя_результата ] [, ...] ]
```

Описание

INSERT добавляет строки в таблицу. Эта команда может добавить одну или несколько строк, сформированных выражениями значений, либо ноль или более строк, выданных дополнительным запросом.

Имена целевых столбцов могут перечисляться в любом порядке. Если список с именами столбцов отсутствует, по умолчанию целевыми столбцами становятся все столбцы заданной таблицы; либо первые *N* из них, если только *N* столбцов поступает от предложения VALUES или *запроса*. Значения, получаемые от предложения VALUES или *запроса*, связываются с явно или неявно определённым списком столбцов слева направо.

Все столбцы, не представленные в явном или неявном списке столбцов, получат значения по умолчанию, если для них заданы эти значения, либо NULL в противном случае.

Если выражение для любого столбца выдаёт другой тип данных, система попытается автоматически привести его к нужному.

С необязательным предложением RETURNING команда INSERT вычислит и возвратит значения для каждой фактически добавленной строки. В основном это полезно для получения значений, присвоенных по умолчанию, например, последовательного номера записи. Однако в этом предложении можно задать любое выражение со столбцами таблицы. Список RETURNING имеет тот же синтаксис, что и список результатов SELECT. В результате будут возвращены те строки, которые были успешно вставлены или изменены..

Чтобы добавлять строки в таблицу, необходимо иметь право INSERT для неё. Если указывается список столбцов, достаточно иметь право INSERT только для перечисленных столбцов.

Для применения предложения RETURNING требуется право SELECT для всех столбцов, перечисленных в RETURNING. Если для добавления строк применяется *запрос*, для всех таблиц или столбцов, задействованных в этом запросе, разумеется, необходимо иметь право SELECT.

Параметры

Добавление

имя_таблицы

Имя существующей таблицы (возможно, дополненное схемой).

псевдоним

Альтернативное имя, заменяющее *имя_таблицы*. Когда указывается этот псевдоним, он полностью скрывает реальное имя таблицы.

имя_столбца

Имя столбца в таблице *имя_таблицы*. Это имя столбца при необходимости может быть дополнено именем вложенного поля или индексом в массиве. (Когда данные вставляются только в некоторые поля столбца составного типа, в другие поля записывается NULL.)

DEFAULT VALUES

Все столбцы получат значения по умолчанию.

выражение

Выражение или значение, которое будет присвоено соответствующему столбцу.

DEFAULT

Соответствующий столбец получит значение по умолчанию.

запрос

Запрос (оператор SELECT), который выдаст строки для добавления в таблицу. Его синтаксис описан в справке оператора SELECT.

выражение_результата

Выражение, которое будет вычисляться и возвращаться командой `INSERT` после добавления или изменения каждой строки. В этом выражении можно использовать имена любых столбцов таблицы *`имя таблицы`*. Чтобы получить все столбцы, достаточно написать `*`.

`имя_результата`

Имя, назначаемое возвращаемому столбцу.

Выводимая информация

В случае успешного завершения, `INSERT` возвращает метку команды в виде

```
INSERT oid число
```

Здесь *`число`* представляет количество добавленных или изменённых строк. Если *`число`* равняется одному, а целевая таблица содержит oid, то в качестве *`oid`* выводится OID, назначенный добавленной строке. Эта одна строка должна быть добавлена, но не изменена. В противном случае в качестве *`oid`* выводится ноль.

Если команда `INSERT` содержит предложение `RETURNING`, её результат будет похож на результат оператора `SELECT` (с теми же столбцами и значениями, что содержатся в списке `RETURNING`), полученный для строк, добавленных или изменённых этой командой.

Примеры

Добавление одной строки в таблицу `films`:

```
INSERT INTO films VALUES
    ('UA502', 'Bananas', 105, '1971-07-13', 'Comedy', '82 minutes');
```

В этом примере столбец `len` опускается и, таким образом, получает значение по умолчанию:

```
INSERT INTO films (code, title, did, date_prod, kind)
    VALUES ('T_601', 'Yojimbo', 106, '1961-06-16', 'Drama');
```

В этом примере для столбца с датой задаётся указание `DEFAULT`, а не явное значение:

```
INSERT INTO films VALUES
    ('UA502', 'Bananas', 105, DEFAULT, 'Comedy', '82 minutes');
INSERT INTO films (code, title, did, date_prod, kind)
    VALUES ('T_601', 'Yojimbo', 106, DEFAULT, 'Drama');
```

Добавление строки, полностью состоящей из значений по умолчанию:

```
INSERT INTO films DEFAULT VALUES;
```

Добавление нескольких строк с использованием многострочного синтаксиса `VALUES`:

```
INSERT INTO films (code, title, did, date_prod, kind) VALUES
    ('B6717', 'Tampopo', 110, '1985-02-10', 'Comedy'),
    ('HG120', 'The Dinner Game', 140, DEFAULT, 'Comedy');
```

В этом примере в таблицу `films` вставляются некоторые строки из таблицы `tmp_films`, имеющей ту же структуру столбцов, что и `films`:

```
INSERT INTO films SELECT * FROM tmp_films WHERE date_prod < '2004-05-07';
```

Добавление одной строки в таблицу `distributors` и получение последовательного номера, сгенерированного благодаря указанию `DEFAULT`:

```
INSERT INTO distributors (did, dname) VALUES (DEFAULT, 'XYZ Widgets')
   RETURNING did;
```

UPDATE

UPDATE — изменить строки таблицы

Синтаксис

```
UPDATE [ ONLY ] имя_таблицы [ * ] [ [ AS ] псевдоним ]
    SET { имя_столбца = { выражение | DEFAULT } |
          ( имя_столбца [, ...] ) = [ ROW ] ( { выражение | DEFAULT } [, ...] ) |
          ( имя_столбца [, ...] ) = ( вложенный_SELECT )
        } [, ...]
    [ FROM список_FROM ]
    [ WHERE условие ]
    [ RETURNING * | выражение_результата [ [ AS ] имя_результата ] [, ...] ]
```

Описание

UPDATE изменяет значения указанных столбцов во всех строках, удовлетворяющих условию. В предложении SET должны указываться только те столбцы, которые будут изменены; столбцы, не изменяемые явно, сохраняют свои предыдущие значения.

Изменить строки в таблице, используя информацию из других таблиц в базе данных, можно двумя способами: применяя вложенные запросы или указав дополнительные таблицы в предложении FROM. Выбор предпочитаемого варианта зависит от конкретных обстоятельств.

Предложение RETURNING указывает, что команда UPDATE должна вычислить и возвратить значения для каждой фактически изменённой строки. Вычислить в нём можно любое выражение со столбцами целевой таблицы и/или столбцами других таблиц, упомянутых во FROM. При этом в выражении будут использоваться новые (изменённые) значения столбцов таблицы. Список RETURNING имеет тот же синтаксис, что и список результатов SELECT.

Для выполнения этой команды необходимо иметь право UPDATE для таблицы, или как минимум для столбцов, перечисленных в списке изменяемых. Также необходимо иметь право SELECT для всех столбцов, значения которых считываются в *выражениях* или *условии*.

Параметры

имя_таблицы

Имя таблицы (возможно, дополненное схемой), строки которой будут изменены. Если перед именем таблицы добавлено ONLY, соответствующие строки изменяются только в указанной таблице. Без ONLY строки будут также изменены во всех таблицах, унаследованных от указанной. При желании, после имени таблицы можно указать *, чтобы явно обозначить, что операция затрагивает все дочерние таблицы.

псевдоним

Альтернативное имя целевой таблицы. Когда указывается это имя, оно полностью скрывает фактическое имя таблицы. Например, в запросе UPDATE foo AS f дополнительные компоненты оператора UPDATE должны обращаться к целевой таблице по имени f, а не foo.

имя_столбца

Имя столбца в таблице *имя_таблицы*. Имя столбца при необходимости может быть дополнено именем вложенного поля или индексом массива. Имя таблицы добавлять к имени целевого столбца не нужно — например, запись UPDATE table_name SET table_name.col = 1 ошибочна.

выражение

Выражение, результат которого присваивается столбцу. В этом выражении можно использовать предыдущие значения этого и других столбцов таблицы.

DEFAULT

Присвоить столбцу значение по умолчанию (это может быть NULL, если для столбца не определено некоторое выражение по умолчанию).

вложенный_SELECT

Подзапрос SELECT, выдающий столько выходных столбцов, сколько перечислено в предшествующем ему списке столбцов в скобках. При выполнении этого подзапроса должна быть получена максимум одна строка. Если он выдаёт одну строку, значения столбцов в нём присваиваются целевым столбцам; если же он не возвращает строку, целевым столбцам присваивается NULL. Этот подзапрос может обращаться к предыдущим значениям текущей изменяемой строки в таблице.

список_FROM
Список табличных выражений, позволяющий использовать в условии WHERE и выражениях присваивания столбцы из других таблиц. Этот список подобен тому, что задаётся в предложении FROM оператора SELECT. Заметьте, что целевую таблицу нужно добавлять в *список_FROM* только при формировании замкнутого соединения (в этом случае она должна фигурировать в *списке_FROM* под псевдонимом).

условие
Выражение, возвращающее значение типа boolean. Изменены будут только те стоки, для которых это выражение возвращает true.

выражение_результата
Выражение, которое будет вычисляться и возвращаться командой UPDATE после изменения каждой строки. В этом выражении можно использовать имена любых столбцов таблицы *имя_таблицы* или таблиц, перечисленных в списке FROM. Чтобы получить все столбцы, достаточно написать *.

имя_результата
Имя, назначаемое возвращаемому столбцу.

Выводимая информация
В случае успешного завершения, UPDATE возвращает метку команды в виде

```
UPDATE число
```

Здесь *число* обозначает количество изменённых строк, включая те подлежащие изменению строки, значения в которых не были изменены. Если *число* равно 0, данный запрос не изменил ни одной строки (это не считается ошибкой).
Если команда UPDATE содержит предложение RETURNING, её результат будет похож на результат оператора SELECT (с теми же столбцами и значениями, что содержатся в списке RETURNING), полученный для строк, изменённых этой командой.

Примеры
Изменение слова Drama на Dramatic в столбце kind таблицы films:

```
UPDATE films SET kind = 'Dramatic' WHERE kind = 'Drama';
```

Изменение значений температуры и сброс уровня осадков к значению по умолчанию в одной строке таблицы weather:

```
UPDATE weather SET temp_lo = temp_lo+1, temp_hi = temp_lo+15, prcp = DEFAULT
  WHERE city = 'San Francisco' AND date = '2003-07-03';
```

Выполнение той же операции с получением изменённых записей:

```
UPDATE weather SET temp_lo = temp_lo+1, temp_hi = temp_lo+15, prcp = DEFAULT
  WHERE city = 'San Francisco' AND date = '2003-07-03'
  RETURNING temp_lo, temp_hi, prcp;
```

Такое же изменение с применением альтернативного синтаксиса со списком столбцов:

```
UPDATE weather SET (temp_lo, temp_hi, prcp) = (temp_lo+1, temp_lo+15, DE-
FAULT)
  WHERE city = 'San Francisco' AND date = '2003-07-03';
```

Увеличение счётчика продаж для менеджера, занимающегося компанией Acme Corporation, с применением предложения `FROM`:

```
UPDATE employees SET sales_count = sales_count + 1 FROM accounts
  WHERE accounts.name = 'Acme Corporation'
  AND employees.id = accounts.sales_person;
```

Выполнение той же операции, с вложенным запросом в предложении `WHERE`:

```
UPDATE employees SET sales_count = sales_count + 1 WHERE id =
  (SELECT sales_person FROM accounts WHERE name = 'Acme Corporation');
```

Изменение имени контакта в таблице счетов (это должно быть имя назначенного менеджера по продажам):

```
UPDATE accounts SET (contact_first_name, contact_last_name) =
    (SELECT first_name, last_name FROM salesmen
     WHERE salesmen.id = accounts.sales_id);
```

Подобный результат можно получить, применив соединение:

```
UPDATE accounts SET contact_first_name = first_name,
                    contact_last_name = last_name
  FROM salesmen WHERE salesmen.id = accounts.sales_id;
```

Обновление статистики в сводной таблице в соответствии с текущими данными:

```
UPDATE summary s SET (sum_x, sum_y, avg_x, avg_y) =
    (SELECT sum(x), sum(y), avg(x), avg(y) FROM data d
     WHERE d.group_id = s.group_id);
```

Попытка добавить новый продукт вместе с количеством. Если такая запись уже существует, вместо этого увеличить количество данного продукта в существующей записи. Чтобы реализовать этот подход, не откатывая всю транзакцию, можно использовать точки сохранения:

```
BEGIN;
-- другие операции
SAVEPOINT sp1;
INSERT INTO wines VALUES('Chateau Lafite 2003', '24');
-- Предполагая, что здесь возникает ошибка из-за нарушения уникальности ключа,
-- мы выполняем следующие команды:
ROLLBACK TO sp1;
UPDATE wines SET stock = stock + 24 WHERE winename = 'Chateau Lafite 2003';
-- Продолжение других операций и в завершение...
COMMIT;
```