PostgreSQL Foreign Key

Summary: in this tutorial, you will learn about PostgreSQL foreign key and how to add foreign keys to tables using foreign key constraints.

Introduction to PostgreSQL Foreign Key Constraint

A foreign key is a column or a group of columns in a table that reference the primary key of another table.

The table that contains the foreign key is called the referencing table or child table. And the table referenced by the foreign key is called the referenced table or parent table.

A table can have multiple foreign keys depending on its relationships with other tables.

In PostgreSQL, you define a foreign key using the foreign key constraint. The foreign key constraint helps maintain the referential integrity of data between the child and parent tables.

A foreign key constraint indicates that values in a column or a group of columns in the child table equal the values in a column or a group of columns of the parent table.

PostgreSQL foreign key constraint syntax

The following illustrates a foreign key constraint syntax:

```
[CONSTRAINT fk_name] FOREIGN KEY(fk_columns) REFERENCES
parent_table(parent_key_columns) [ON DELETE delete_action] [ON UPDATE update_action]
```

```
[CONSTRAINT fk_name]
   FOREIGN KEY(fk_columns)
   REFERENCES parent_table(parent_key_columns)
   [ON DELETE delete_action]
   [ON UPDATE update_action]
```

In this syntax:

- First, specify the name for the foreign key constraint after the `CONSTRAINT` keyword. The `CONSTRAINT` clause is optional. If you omit it, PostgreSQL will assign an auto-generated name.
- Second, specify one or more foreign key columns in parentheses after the `FOREIGN KEY` keywords.
- Third, specify the parent table and parent key columns referenced by the foreign key columns in the `REFERENCES` clause.

- Finally, specify the delete and update actions in the `ON DELETE` and `ON UPDATE` clauses.

The delete and update actions determine the behaviors when the primary key in the parent table is deleted and updated. Since the primary key is rarely updated, the `ON UPDATE action` is not often used in practice. We'll focus on the `ON DELETE` action.

PostgreSQL supports the following actions:

- SET NULL
- SET DEFAULT
- RESTRICT
- NO ACTION
- CASCADE

PostgreSQL foreign key constraint examples

The following statements create the `customers` and `contacts` tables:

`DROP TABLE IF EXISTS customers; DROP TABLE IF EXISTS contacts; CREATE TABLE customers(customer_id INT GENERATED ALWAYS AS IDENTITY, customer_name VARCHAR(255) NOT NULL, PRIMARY KEY(customer_id)); CREATE TABLE contacts(contact_id INT GENERATED ALWAYS AS IDENTITY, customer_id INT, contact_name VARCHAR(255) NOT NULL, phone VARCHAR(15), email VARCHAR(100), PRIMARY KEY(contact_id), CONSTRAINT fk_customer FOREIGN KEY(customer_id) REFERENCES customers(customer_id));`

```
DROP TABLE IF EXISTS customers;
DROP TABLE IF EXISTS contacts;

CREATE TABLE customers(
   customer_id INT GENERATED ALWAYS AS IDENTITY,
   customer_name VARCHAR(255) NOT NULL,
   PRIMARY KEY(customer_id)
);

CREATE TABLE contacts(
   contact_id INT GENERATED ALWAYS AS IDENTITY,
   customer_id INT,
   contact_name VARCHAR(255) NOT NULL,
   phone VARCHAR(15),
   email VARCHAR(100),
   PRIMARY KEY(contact_id),
   CONSTRAINT fk_customer
      FOREIGN KEY(customer_id)
```

```
      REFERENCES customers(customer_id)
);
```

In this example, the `customers` table is the parent table and the `contacts` table is the child table.

Each customer has zero or many contacts and each contact belongs to zero or one customer.

The `customer_id` column in the `contacts` table is the foreign key column that references the primary key column with the same name in the `customers` table.

The following foreign key constraint `fk_customer` in the `contacts` table defines the `customer_id` as the foreign key:

```
CONSTRAINT fk_customer FOREIGN KEY(customer_id) REFERENCES customers(customer_id)
```

```
CONSTRAINT fk_customer
   FOREIGN KEY(customer_id)
      REFERENCES customers(customer_id)
```

Because the foreign key constraint does not have the `ON DELETE` and `ON UPDATE` action, they default to `NO ACTION`.

NO ACTION

The following inserts data into the `customers` and `contacts` tables:

```
INSERT INTO customers(customer_name) VALUES('BlueBird Inc'), ('Dolphin LLC'); INSERT INTO contacts(customer_id, contact_name, phone, email) VALUES(1,'John Doe','(408)-111-1234','john.doe@bluebird.dev'), (1,'Jane Doe','(408)-111-1235','jane.doe@bluebird.dev'), (2,'David Wright','(408)-222-1234','david.wright@dolphin.dev');
```

```
INSERT INTO customers(customer_name)
VALUES('BlueBird Inc'),
      ('Dolphin LLC');

INSERT INTO contacts(customer_id, contact_name, phone, email)
VALUES(1,'John Doe','(408)-111-1234','john.doe@bluebird.dev'),
      (1,'Jane Doe','(408)-111-1235','jane.doe@bluebird.dev'),
      (2,'David Wright','(408)-222-1234','david.wright@dolphin.dev');
```

The following statement deletes the customer id 1 from the `customers` table:

```
DELETE FROM customers WHERE customer_id = 1;
```

```
DELETE FROM customers
WHERE customer_id = 1;
```

Because of the `ON DELETE NO ACTION`, PostgreSQL issues a constraint violation because the referencing rows of the customer id 1 still exist in the `contacts` table:

```
ERROR: update or delete on table "customers" violates foreign key constraint "fk_customer" on table "contacts" DETAIL: Key (customer_id)=(1) is still referenced from table "contacts". SQL state: 23503
```

```
Code language: Shell Session (shell)
```

The `RESTRICT` action is similar to the `NO ACTION`. The difference only arises when you define the foreign key constraint as `DEFERRABLE` with an `INITIALLY DEFERRED` or `INITIALLY IMMEDIATE` mode. We'll discuss more on this in the subsequent tutorial.

SET NULL

The `SET NULL` automatically sets `NULL` to the foreign key columns in the referencing rows of the child table when the referenced rows in the parent table are deleted.

The following statements drop the sample tables and re-create them with the foreign key that uses the `SET NULL` action in the `ON DELETE` clause:

```
DROP TABLE IF EXISTS contacts; DROP TABLE IF EXISTS customers; CREATE TABLE customers( customer_id INT GENERATED ALWAYS AS IDENTITY, customer_name VARCHAR(255) NOT NULL, PRIMARY KEY(customer_id) ); CREATE TABLE contacts( contact_id INT GENERATED ALWAYS AS IDENTITY, customer_id INT, contact_name VARCHAR(255) NOT NULL, phone VARCHAR(15), email VARCHAR(100), PRIMARY KEY(contact_id), CONSTRAINT fk_customer FOREIGN KEY(customer_id) REFERENCES customers(customer_id) ON DELETE SET NULL ); INSERT INTO customers(customer_name) VALUES('BlueBird Inc'), ('Dolphin LLC'); INSERT INTO contacts(customer_id, contact_name, phone, email) VALUES(1,'John Doe','(408)-111-1234','john.doe@bluebird.dev'), (1,'Jane Doe','(408)-111-1235','jane.doe@bluebird.dev'), (2,'David Wright','(408)-222-1234','david.wright@dolphin.dev');
```

```
DROP TABLE IF EXISTS contacts;
DROP TABLE IF EXISTS customers;

CREATE TABLE customers(
```

```
    customer_id INT GENERATED ALWAYS AS IDENTITY,
    customer_name VARCHAR(255) NOT NULL,
    PRIMARY KEY(customer_id)
);

CREATE TABLE contacts(
    contact_id INT GENERATED ALWAYS AS IDENTITY,
    customer_id INT,
    contact_name VARCHAR(255) NOT NULL,
    phone VARCHAR(15),
    email VARCHAR(100),
    PRIMARY KEY(contact_id),
    CONSTRAINT fk_customer
        FOREIGN KEY(customer_id)
        REFERENCES customers(customer_id)
        ON DELETE SET NULL
);

INSERT INTO customers(customer_name)
VALUES('BlueBird Inc'),
       ('Dolphin LLC');

INSERT INTO contacts(customer_id, contact_name, phone, email)
VALUES(1,'John Doe','(408)-111-1234','john.doe@bluebird.dev'),
       (1,'Jane Doe','(408)-111-1235','jane.doe@bluebird.dev'),
       (2,'David Wright','(408)-222-1234','david.wright@dolphin.dev');
```

The following statements insert data into the `customers` and `contacts` tables:

`INSERT INTO customers(customer_name) VALUES('BlueBird Inc'), ('Dolphin LLC'); INSERT INTO contacts(customer_id, contact_name, phone, email) VALUES(1,'John Doe','(408)-111-1234','john.doe@bluebird.dev'), (1,'Jane Doe','(408)-111-1235','jane.doe@bluebird.dev'), (2,'David Wright','(408)-222-1234','david.wright@dolphin.dev');`


```
INSERT INTO customers(customer_name)
VALUES('BlueBird Inc'),
       ('Dolphin LLC');

INSERT INTO contacts(customer_id, contact_name, phone, email)
VALUES(1,'John Doe','(408)-111-1234','john.doe@bluebird.dev'),
       (1,'Jane Doe','(408)-111-1235','jane.doe@bluebird.dev'),
       (2,'David Wright','(408)-222-1234','david.wright@dolphin.dev');
```

To see how the `SET NULL` works, let's delete the customer with id 1 from the `customers` table:

```
DELETE FROM customers WHERE customer_id = 1;
```

```
DELETE FROM customers
WHERE customer_id = 1;
```

Because of the `ON DELETE SET NULL` action, the referencing rows in the `contacts` table set to NULL. The following statement displays the data in the `contacts` table:

```
SELECT * FROM contacts;
```

	contact_id integer	customer_id integer	contact_name character varying (255)	phone character varying (15)	email character varying (100)
1	3	2	David Wright	(408)-222-1234	david.wright@dolphin.dev
2	1	[null]	John Doe	(408)-111-1234	john.doe@bluebird.dev
3	2	[null]	Jane Doe	(408)-111-1235	jane.doe@bluebird.dev

As can be seen clearly from the output, the rows that have the `customer_id` 1 now have the `customer_id` sets to `NULL`

CASCADE

The `ON DELETE CASCADE` automatically deletes all the referencing rows in the child table when the referenced rows in the parent table are deleted. In practice, the `ON DELETE CASCADE` is the most commonly used option.

The following statements recreate the sample tables. However, the delete action of the `fk_customer` changes to `CASCADE`:

```
DROP TABLE IF EXISTS contacts; DROP TABLE IF EXISTS customers; CREATE TABLE customers( customer_id INT GENERATED ALWAYS AS IDENTITY, customer_name VARCHAR(255) NOT NULL, PRIMARY KEY(customer_id) ); CREATE TABLE contacts( contact_id INT GENERATED ALWAYS AS IDENTITY, customer_id INT, contact_name VARCHAR(255) NOT NULL, phone VARCHAR(15), email VARCHAR(100), PRIMARY KEY(contact_id), CONSTRAINT fk_customer FOREIGN KEY(customer_id) REFERENCES customers(customer_id) ON DELETE CASCADE ); INSERT INTO customers(customer_name) VALUES('BlueBird Inc'), ('Dolphin LLC'); INSERT INTO contacts(customer_id, contact_name, phone, email) VALUES(1,'John Doe','(408)-111-1234','john.doe@bluebird.dev'), (1,'Jane Doe','(408)-111-1235','jane.doe@bluebird.dev'), (2,'David Wright','(408)-222-1234','david.wright@dolphin.dev');
```

```
DROP TABLE IF EXISTS contacts;
DROP TABLE IF EXISTS customers;

CREATE TABLE customers(
   customer_id INT GENERATED ALWAYS AS IDENTITY,
   customer_name VARCHAR(255) NOT NULL,
   PRIMARY KEY(customer_id)
);

CREATE TABLE contacts(
   contact_id INT GENERATED ALWAYS AS IDENTITY,
   customer_id INT,
   contact_name VARCHAR(255) NOT NULL,
   phone VARCHAR(15),
   email VARCHAR(100),
   PRIMARY KEY(contact_id),
   CONSTRAINT fk_customer
      FOREIGN KEY(customer_id)
      REFERENCES customers(customer_id)
      ON DELETE CASCADE
);

INSERT INTO customers(customer_name)
VALUES('BlueBird Inc'),
      ('Dolphin LLC');

INSERT INTO contacts(customer_id, contact_name, phone, email)
VALUES(1,'John Doe','(408)-111-1234','john.doe@bluebird.dev'),
      (1,'Jane Doe','(408)-111-1235','jane.doe@bluebird.dev'),
      (2,'David Wright','(408)-222-1234','david.wright@dolphin.dev');
```

The following statement deletes the customer id 1:

```
DELETE FROM customers WHERE customer_id = 1;
```

Because of the `ON DELETE CASCADE` action, all the referencing rows in the `contacts` table are automatically deleted:

```
SELECT * FROM contacts;
```

	contact_id integer	customer_id integer	contact_name character varying (255)	phone character varying (15)	email character varying (100)
1	3	2	David Wright	(408)-222-1234	david.wright@dolphin.dev

SET DEFAULT

The `ON DELETE SET DEFAULT` sets the default value to the foreign key column of the referencing rows in the child table when the referenced rows from the parent table are deleted.

Add a foreign key constraint to an existing table

To add a foreign key constraint to the existing table, you use the following form of the ALTER TABLE statement:

```
ALTER TABLE child_table ADD CONSTRAINT constraint_name FOREIGN KEY (fk_columns)
REFERENCES parent_table (parent_key_columns);
```

```
ALTER TABLE child_table
ADD CONSTRAINT constraint_name
FOREIGN KEY (fk_columns)
REFERENCES parent_table (parent_key_columns);
```

When you add a foreign key constraint with `ON DELETE CASCADE` option to an existing table, you need to follow these steps:

First, drop existing foreign key constraints:

```
ALTER TABLE child_table DROP CONSTRAINT constraint_fkey;
```

```
ALTER TABLE child_table
DROP CONSTRAINT constraint_fkey;
```

First, add a new foreign key constraint with `ON DELETE CASCADE` action:

```
ALTER TABLE child_table ADD CONSTRAINT constraint_fk FOREIGN KEY (fk_columns)
REFERENCES parent_table(parent_key_columns) ON DELETE CASCADE;
```

```
ALTER TABLE child_table
ADD CONSTRAINT constraint_fk
FOREIGN KEY (fk_columns)
REFERENCES parent_table(parent_key_columns)
ON DELETE CASCADE;
```

In this tutorial, you have learned about PostgreSQL foreign keys and how to use the foreign key constraint to create foreign keys for a table.