SQLWorkshop — Tutorial D3

version **#v1.0.6**

Assistant C/UNIX 2021 `<assistants@tickets.assistants.epita.fr>`

Copyright

Contents

*https://intra.assistants.epita.fr

Today, let us dive into what we kept aside during the previous tutorial: database management. Knowing how to query or manage rows from a table is one thing, but knowing how to create and alter those tables is essential. You will learn how to create your whole database from scratch and, more importantly, how to do it right.

1 Handling the database

1.1 Constraints

Let us start with a quick explanation of the concept of constraints.

A column of a table has a type, determining the kind of data that can be inserted as a value. However, most of the time, you want to add constraints to your columns. For example, a column containing a product price should probably only accept positive values but there is no standard data type that accepts only positive numbers.

To that end, you can define constraints on columns and tables at the creation or alteration of the table. Constraints give you as much control over the data in your tables as you wish. If a user attempts to store data in a column that would violate a constraint, an error is raised.

You can also name a constraint. This clarifies error messages and allows you to refer to the constraint when you need to change it. So, to specify a named constraint, use the key word CONSTRAINT, followed by an identifier, itself followed by the constraint definition. If you do not specify a constraint name in this way, the system chooses a name for you.

Two kinds of constraints exist: *column* constraint and *table* constraint. Column constraints apply to a column, and table constraints can apply to the whole table. A column constraint can be re-written into a table constraint that applies to only one column.

Here is a list of commonly used constraints, you should recognize most of them:

- NOT NULL: ensures that a column cannot have a NULL value
- UNIQUE: ensures that all values in a column are different for each row in a table
- CHECK: ensures that all values in a column satisfy a specific condition
- DEFAULT: sets a default value for a column when no value is specified
- PRIMARY KEY: specifies a column or columns of a table, that uniquely identify each row in the table. Two rows cannot have the same value(s) for column(s) from the primary key, and none of those fields can have null values. A table can have only one primary key. When multiple fields are used as a primary key, they are called a composite key. PostgreSQL doesn't force you to have one but many other object-relational database management systems will, so you should have one in each table
- FOREIGN KEY: is used to link tables together. You can have multiple foreign keys in the same table. A foreign key must reference a unique column, ideally a primary key, from another table.

These constraints will be further explained later.

1.2 Creation

1.2.1 Table creation

To create a table, you can use the command `CREATE TABLE`. As you may have guessed, this table will allow you to store your data ordered in columns. When using this command, you will be able to define the name of your columns, the type of their associated data and their constraints (define a primary key, prevent the user to enter a null value...).

```
CREATE TABLE table_name
(
    column1 TYPE [COLUMN CONSTRAINTS],
    column2 TYPE [COLUMN CONSTRAINTS],
    column3 TYPE [COLUMN CONSTRAINTS],

    [TABLE CONSTRAINTS...]
);
```

1.2.2 Example

In this example, FOREIGN KEY is here to forbid the creation of a rating for a movie which isn't in the movie table.

```
CREATE TABLE movie
(
    movie_id                    SERIAL,
    movie_title                 VARCHAR(64) NOT NULL,
    PRIMARY KEY(movie_id)       -- movie_id is the primary key
);

CREATE TABLE rating
(
    rating_author       VARCHAR(64) NOT NULL,
    rating_value        INT NOT NULL,
    rating_comment      VARCHAR(64),
    rating_movie        INT REFERENCES movie(movie_id),         -- foreign key
    PRIMARY KEY(rating_author, rating_movie)
);
```

1.3 Deletion

1.3.1 Deletion of a table

To delete a table, you must use the `DROP TABLE` statement. A good practice is to always use `DROP TABLE IF EXISTS`.

```
DROP TABLE IF EXISTS table_name;
```

If you want to delete a table and *everything* related to it, you can use the `CASCADE` parameter.

```
DROP TABLE IF EXISTS table_name CASCADE;
```

1.4 Alter table

If you want to modify something in an existing table (add, delete or update), you can use the `ALTER TABLE` statement.

1.4.1 Add a column in a table

```
ALTER TABLE table_name
ADD column_name datatype;
```

1.4.2 Delete a column in a table

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

1.4.3 Modify a table

To modify the type of a column, here is the syntax:

```
ALTER TABLE table_name
ALTER COLUMN column_name TYPE new_column_type;
```

Many other things can be altered. Check all possibilities here: http://www.postgresql.org/docs/9.6/static/sql-altertable.html

1.5 Exercises

After each request, do not forget that you can use the `\d <table_name>` command to check that your table schema is correct.

1.5.1 Request 1

Create a table `can` with 3 columns: an index `id`, a name `name`, and a capacity `capacity_cl`. Types are respectively: serial, varchar(64), and int. Here, `id` is the primary key of the table.

1.5.2 Request 2

Now, create a table `student` with 4 columns: `id`, `firstname`, `lastname`, `assistant`. Types are respectively: serial, varchar(64), varchar(64), boolean, and the student's id is the primary key of the table.

1.5.3 Request 3

Alter the table `student` to remove the `id`, and replace it by the column `login` varchar(8) as primary key.

1.5.4 Request 4

Create a table `student_can` to store the cans bought by students. It must contain references to the two other tables.

2 Check

A `CHECK` constraint is the most generic constraint type. It allows you to specify that the value in a certain column satisfies a boolean expression. For instance, to require positive product prices, you could use:

```
CREATE TABLE products (
    product_id INTEGER,
    name VARCHAR(64),
    price FLOAT CHECK (price > 0)
);
```

If the insertion in this table does not respect the check constraint, it will raise an exception, as a non-fulfilled `NOT NULL` constraint would.

As said previously, a constraint can be named, which is quite useful when dealing with CHECK constraints.

```
CREATE TABLE products (
    product_id INTEGER,
    name VARCHAR(64),
    price FLOAT CONSTRAINT positive_price CHECK (price > 0)
);
```

A check constraint can also refer to several columns. Say you store a regular price and a discounted price, and you want to ensure that the discounted price is lower than the regular price:

```
CREATE TABLE products (
    product_id INTEGER,
    name VARCHAR(64),
    price FLOAT CHECK (price > 0),
    discounted_price INTEGER CHECK (discounted_price > 0),
    CHECK (price > discounted_price)
);
```

As it is referring to more than one column, the constraint must be a table constraint.

You can also put more than one test in a CHECK constraint:

```
CHECK (discounted_price > 0 AND price > discounted_price)
```

2.1 Exercises

2.1.1 Request 1

Drop the *can* table and insert it again with a check that the capacity is always positive.

2.1.2 Request 2

On the address table in the given files, check that the zip_code is of length 5.

3 Not Null

The NOT NULL constraint simply specifies that a column must not accept any null value.

```
CREATE TABLE products (
    product_no INTEGER NOT NULL,
    name TEXT NOT NULL,
    price NUMERIC
);
```

A not-null constraint is always written as a column constraint. It is functionally equivalent to creating a check constraint: CHECK (column_name IS NOT NULL).

Of course, a column can have more than one constraint. Just write the constraints one after another:

```
CREATE TABLE products (
    product_no INTEGER NOT NULL,
    name TEXT NOT NULL,
    price NUMERIC NOT NULL CHECK (price > 0)
);
```

The order does not matter. It does not necessarily determine in which order the constraints are checked.

4 Unique

UNIQUE constraints ensure that the data contained in a column is unique among all the rows in the table.

```
CREATE TABLE products (
    product_no INTEGER UNIQUE,
    name TEXT,
    price NUMERIC
);
```

To define a unique constraint for a group of columns, write it as a table constraint with the column names separated by commas:

```
CREATE TABLE example (
    a INTEGER,
    b INTEGER,
    c INTEGER,
    UNIQUE (a, c)
);
```

This specifies that the combination of values from the table is unique for each row. Values from one of the selected columns do not need to be unique.

further

In general, a unique constraint is violated during an insertion or an update if there is more than one row in the table where the values of all the columns included in the constraint are equal. However, two null values are never considered equal in this comparison. It means that even in the presence of a unique constraint it is possible to store duplicate rows that only contain null values in at least one of the constraint's columns.

5 Keys

5.1 Introduction

As you have already seen, FOREIGN KEY and PRIMARY KEY are used to link two tables together using a join. Let us describe more accurately these two constraints.

5.2 Primary keys

A primary key constraint indicates that a column, or group of columns, can be used as a unique identifier for rows in the table. This requires values to be both unique and not null. `example INTEGER UNIQUE NOT NULL` and `example INTEGER PRIMARY KEY` will accept the exact same data.

```
CREATE TABLE products (
    product_no INTEGER PRIMARY KEY,
    name TEXT,
    price NUMERIC
);
```

Primary keys can span more than one column; the syntax is similar to the unique constraint:

```
CREATE TABLE example (
    a INTEGER,
    b INTEGER,
    c INTEGER,
    PRIMARY KEY (a, c)
);
```

Only one primary key can be specified for a table. PostgreSQL does not force you to have one but many other object-relational database management systems will, so you should have one in each table.

Primary keys are useful both for documentation purposes and for client applications. For example, a GUI application that allows modifying row values probably needs to know the primary key of a table to be able to identify rows uniquely. There are also various ways in which the database system makes use of a primary key, if one has been declared ; for example, the primary key defines the default target column(s) for foreign keys referencing its table.

5.3 Foreign keys

A foreign key constraint specifies that the values in a column (or a group of columns) match the values appearing in a column (or group of columns) of another table. We say this maintains the referential integrity between two related tables. This column (or group of columns) must be UNIQUE, or equivalent (such as PRIMARY KEY).

We use the “products” table defined earlier and we want to ensure that the “orders” table only contains orders of products that actually exist. So we define a foreign key constraint in the “orders” table that references the “products” table:

```
CREATE TABLE orders (
    order_id INTEGER PRIMARY KEY,
    product_no INTEGER REFERENCES products (product_no),
    quantity INTEGER
);
```

Now it is impossible to create orders with non-null product_no entries that do not appear in the products table.

A foreign key can also constrain and reference a group of columns. As usual, it then needs to be written in table constraint form. The number and type of the constrained columns need to match the number and type of the referenced columns.

```
CREATE TABLE t1 (
    a INTEGER PRIMARY KEY,
    b INTEGER,
    c INTEGER,
    FOREIGN KEY (b, c) REFERENCES other_table (c1, c2)
);
```

You can assign a name for a foreign key constraint.

A table can have more than one foreign key constraint. A foreign and primary key can overlap.

5.4 On Delete

We know that foreign keys disallow creation of orders that do not relate to any products. However, what would happen if a product was removed after the creation of an order referencing it? The `ON DELETE` keyword responds to this problem:

```
CREATE TABLE order_items (
    product_no INTEGER REFERENCES products ON DELETE RESTRICT,
    order_id INTEGER REFERENCES orders ON DELETE CASCADE,
    quantity INTEGER,
    PRIMARY KEY (product_no, order_id)
);
```

Restricting and cascading deletes are the two most common options. RESTRICT prevents deletion of a referenced row. CASCADE specifies that when a referenced row is deleted, rows referencing it should be automatically deleted as well.

The default behaviour, or NO ACTION, prevents deletion like RESTRICT. The difference is that NO ACTION allows the check to be deferred until later in the transaction (like TRIGGERS), whereas RESTRICT does not.

There are two other options: SET NULL and SET DEFAULT. These cause the referencing columns in the referencing rows to be set to NULL or their default values, respectively, when the referenced row is deleted. Note that the values you are setting should respect all constraints. For example, if an action specifies SET DEFAULT but the default value does not satisfy the foreign key constraint, the operation will fail.

5.5 On Update

Analogous to ON DELETE there is also ON UPDATE which is invoked when a referenced column is changed (updated). The possible actions are the same. In this case, CASCADE means that the updated values of the referenced columns should be copied into the referencing rows.

5.6 Exercises

5.6.1 Request 1

Create three tables: a table of player with their pseudo (two players can not have the same), a table of characters (the characters are all different) from Super Smash Bros Ultimate and a table of match between two players, with the character used, which player won the match, and the number of match in this configuration.

6 Normal Forms

6.1 Introduction

Database normalization is the process of structuring a relational database in accordance with a series of so-called normal forms in order to reduce data redundancy and improve data integrity. By doing it, you remove any redundancy in the row and the column but also protect from most errors due to insertion, update or deletion.

This table will be used to illustrate normal forms during all this part. (But as it is too big to be printed in one line, it is broken into two tables so the second one should be appended to the first one.)

Book	Author	Book Type	Price
Beginning MySQL Database Design and Optimization	Chad Russell	Hardcover	49.99

Subject	Pages	Thickness	Publisher	Publisher Country	Genre ID	Genre Name
MySQL, Database, Design	520	Thick	Apress	USA	1	Tutorial

6.2 1NF

To satisfy 1NF, you need to ensure that the values in each column of a table are atomic. You also need to have a PRIMARY KEY in every table.

In the initial table, Subject contains a set of values, meaning it does not comply. One way to respect the first normal form would be to separate the duplicities into multiple columns. For that, you could do multiple columns. However, using this method, you won't be able to add another subject.

To do this, the correct way is to create a table of subjects. Here, You will also need a table of links between the subjects and the books. To do that, you need to choose a primary key for each table. For the subject table, you can use the name of the subject or you can add a serial to use an ID. For the book table, the primary key will be a tuple of two columns. Indeed, a book is represented by a title, an author. To do that, during the table creation, those two columns should be created with a unique constraint and a primary key constraint.

Book	Author	Book Type	Price	Pages	Thickness	Genre ID	Genre Name	Publisher ID
Beginning MySQL Database Design and Optimization	Chad Russell	Hardcover	49.99	520	Thick	1	Tutorial	1

Subject ID	Subject name
1	MySQL
2	Database
3	Design

Publisher_ID	Name	Country
1	Apress	USA

Book	Subject ID
Beginning MySQL Database Design and Optimization	1
Beginning MySQL Database Design and Optimization	2
Beginning MySQL Database Design and Optimization	3

6.3 2NF

To satisfy 2NF, you need to respect 1NF and for each non key attribute to depend on all the columns in the primary key but also in all the candidate keys. The candidate keys are all the columns that are not in the primary key but on which other columns depend.

Book	Author	Book Type	Price	Pages	Thickness	Genre ID	Genre Name	Publisher ID
Beginning MySQL Database Design and Optimization	Chad Russell	Hardcover	49.99	520	Thick	1	Tutorial	1
The Relational Model for Database Management: Version 2	E.F.Codd	Paperback	39.99	528	Thick	2	Popular science	2
Beginning MySQL Database Design and Optimization	Chad Russell	E-book	22.34	520	Thick	1	Tutorial	1
The Relational Model for Database Management: Version 2	E.F.Codd	E-book	13.88	528	Thick	2	Popular science	2

Here, the price depends on the book name and on the author but also on the book type. The name and the author constitute the primary key so values should depend on it. However, the book type is a candidate key as the price depends on it. To respect 2NF, you need all values to depend on all candidate keys, so things like publisher should also depend on the book type, but here it doesn't. To achieve 2NF here, you need to remove the book type from the table and create a new one. This new table will have the book name, the book type and the price. The book name is here for the join with the book table. The primary key in the new table will be a tuple of the book name and the book type and by doing so, it will conform to 2NF too.

Book	Author	Pages	Thickness	Genre ID	Genre Name	Publisher ID
Beginning MySQL Database Design and Optimization	Chad Russell	520	Thick	1	Tutorial	1
The Relational Model for Database Management: Version 2	E.F.Codd	528	Thick	2	Popular science	2

Book	Book Type	Price
Beginning MySQL Database Design and Optimization	Hardcover	49.99
The Relational Model for Database Management: Version 2	Paperback	39.99
Beginning MySQL Database Design and Optimization	E-book	22.34
The Relational Model for Database Management: Version 2	E-book	13.88

6.4 3NF

To satisfy 3NF, you need to respect 2NF and for all values to depend only on primary key and candidate key.

Here the genre name depend on the genre id but the genre name is neither the primary key nor a candidate key. So you need to create a new table with the genre id and the genre name. By doing that you can remove the genre name from the book table. The new book table is now conform to 3NF because every column that are neither primary key nor candidate key depends only on those.

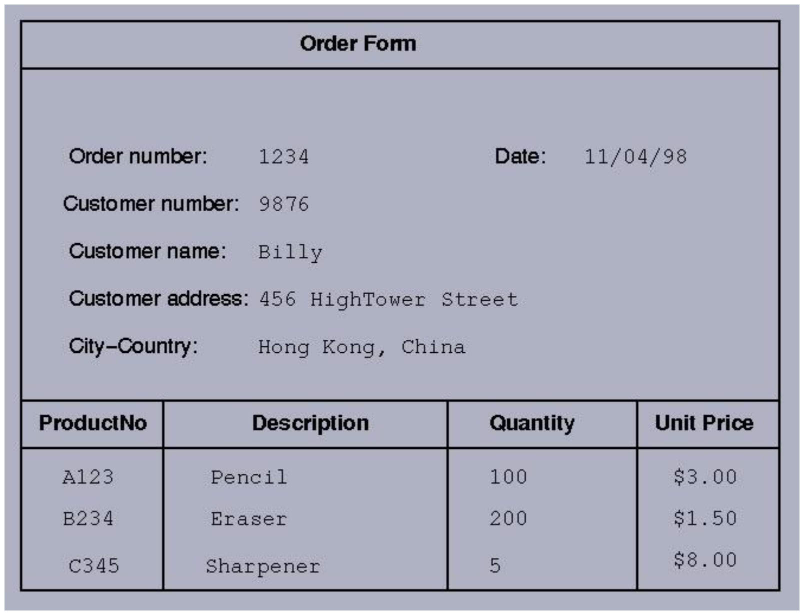
Book	Author	Pages	Thickness	Genre ID	Publisher ID
Beginning MySQL Database Design and Optimization	Chad Russell	520	Thick	1	1
The Relational Model for Database Management: Version 2	E.F.Codd	528	Thick	2	2

Genre ID	Genre Name
1	Tutorial
2	Popular science

6.5 Exercises

6.5.1 Exercise 1

Write the normalization of this table so that 3NF is respected. You will need more than one table to obtain a normalized result.



Order Form

Order number: 1234 Date: 11/04/98

Customer number: 9876

Customer name: Billy

Customer address: 456 HighTower Street

City–Country: Hong Kong, China

ProductNo	Description	Quantity	Unit Price
A123	Pencil	100	$3.00
B234	Eraser	200	$1.50
C345	Sharpener	5	$8.00

The only way out is through