Learning Objectives

Learners will be able to...

- Match CRUD operation to SQL statement
- Apply INSERT statement
- Use UPDATE statement
- Understand DELETE statement
- Match common SQL data types

info

Learners should:

Have a familiarity with the syntax and structure of SQL statements.

CRUD and SQL

We interact with information every day: we **read** feeds on social networks, watch streams, **create** blog entries, send messages in instant messengers. If you need to add something, we can **update** the entries, and if something has become unnecessary, we **delete** unnecessary data.

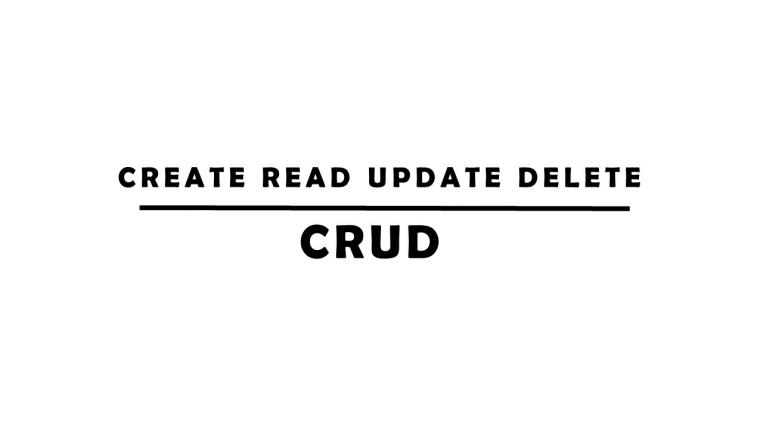


Image with Create, Read, Update, and Delete

CRUD operations include 4 functions: Create, Read, Update and Delete. These are the basic methods of working with databases. CRUD operations are for editing program data. Let's take a closer look at what each operation means via Rest API and Database:

1. **Create**: REST API equivalent is **POST** - The POST method allows you to enter information into the source/database. SQL statement for create data is **INSERT**.
2. **Read**: REST API equivalent is **GET** - the GET method allows you to get information from the source/database. You are already familiar with the expression for getting data from the database - this is a **SELECT**.
3. **Update**: REST API equivalent is **PUT** - The PUT method allows you to update existing information in the source/database. SQL statement for update data is **UPDATE**.
4. **Delete**: REST API equivalent is **DELETE** - The DELETE method to delete existing information from the source/database. SQL statement for update data is **DELETE**.)
   We'll look at each of the operators in more detail later on.

INSERT Statement

Insert button picture

The **INSERT** statement inserts new records into a table.

There are two ways to write an INSERT INTO statement.

The first way defines both the column names and the values to be inserted:

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

Database have no actor Jackie Chan, lets fix it.

Run command in psql shell:

```
INSERT INTO actor (actor_id, first_name, last_name, last_update)
VALUES ( 201, 'JACKIE', 'CHAN', '2023-07-01 04:34:33' );
```

If you try to run the command a second time, you will see an error similar to the one shown below:

```
ERROR:  duplicate key value violates unique constraint
"actor_pkey"
DETAIL:  Key (actor_id)=(201) already exists.
```

This happened because the DBMS was told that the value in the `actor_id` column should not be repeated.

But the convenience of this way of inserting values is that we can list not all fields, but only those that are indicated in the table as necessary. In this case, these are two fields: `first_name` and `last_name`.

And the working version of the insert will look like this:

```
INSERT INTO actor (first_name, last_name)
VALUES ('JACKIE', 'CHAN');
```

Moreover, we can list the fields in any order, the main thing is not to confuse the orders in `VALUES`:

```
INSERT INTO actor (last_name, first_name)
VALUES ('CHAN', 'JACKIE');
```

Try inserting another Jackie Chan into the table.

After execution, you will see another entry, which will have its own `actor_id`.

If you are adding values for all columns of a table, you do not need to specify the column names in the SQL query. However, make sure the order of the values is in the same order as the columns in the table. The INSERT INTO syntax would look like this:

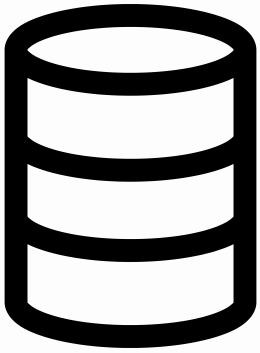
```
INSERT INTO table_name
VALUES (value1, value2, value3, ...);
```

And Jackie example:

```
INSERT INTO actor
VALUES ( 205, 'JACKIE', 'CHAN', '2023-07-01 04:34:33' );
```

But it's best to always specify a list of columns when adding data to a table. This avoids errors when changing the structure of the table, when the code you wrote will stop working if columns were added or removed to the table.

DATA TYPE Introduction



Picture of a database

If you continue experimenting with inserting data into a table, you may notice that it is not always possible to do this.

For example, these queries will cause the following errors.

1. Number instead of correct formatted datetime.
   ```
   sql     INSERT INTO actor (actor_id, first_name, last_name,
   last_update)     VALUES ( 211, 'JACKIE', 'CHAN', 100 );
   txt-hide-clipboard     ERROR:  column "last_update" is of type
   timestamp without time zone but expression is of type integer
   LINE 1: ...st_name, last_update) VALUES ( 211, 'JACKIE', 'CHAN',
   100 );
   ^     HINT:  You will need to rewrite or cast the expression.
   ```
2. Incorrectly formatted datetime
   ```
   sql     INSERT INTO actor (actor_id, first_name, last_name,
   last_update)      VALUES ( 211, 'JACKIE', 'CHAN', '10' );
   txt-hide-clipboard     ERROR:  invalid input syntax for type
   timestamp: "10"     LINE 1: ...t_name, last_update) VALUES ( 211,
   'JACKIE', 'CHAN', '10' );
   ^
   ```
3. Long string in actor name
   ```
   sql     INSERT INTO actor (actor_id, first_name, last_name,
   last_update)      VALUES ( 213, 'very-very-very-very-very-very-
   very-very-very-very-very-very-very-very-long', 'CHAN', '2023-07-01' );
   txt-hide-clipboard     ERROR:  value too long for type character
   varying(45)
   ```

This is because the DBMS supports certain data types for each column and checks for their correctness.

There are common data types for SQL, but the full list of supported types will depend on the specific DBMS implementation and you should check the documentation.

Common SQL types

Numeric types

Name	Description	Range
SMALLINT	small-range integer	-32768 to +32767
INT	typical choice for integer	-2147483648 to +2147483647
BIGINT	large-range integer	-9223372036854775808 to +9223372036854775807
DECIMAL, NUMERIC	user-specified precision, exact	up to 131072 digits before the decimal point; up to 16383 digits after the decimal point
FLOAT	variable-precision, inexact	1E-307 to 1E+308
REAL	variable-precision, inexact	1E-37 to 1E+37

SQL Date and Time Data Types

Name	Description
DATE	date (no time of day), stores date in the format YYYY-MM-DD
TIME	time of day (no date), stores time in the format HH:MM:SS
TIMESTAMP	both date and time, stores number of seconds passed since the Unix epoch (‘1970-01-01 00:00:00’ UTC)

SQL Character and String Data Types

Name	Description
CHAR	fixed-length, blank padded
VARCHAR, VARCHAR(max)	variable-length with limit
TEXT	variable unlimited length

UPDATE Statement

Update image

The `UPDATE` statement is used to update existing data in a table.

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

UPDATE changes the values of the specified columns in all rows that satisfy the **WHERE** clause.

```
UPDATE actor
SET first_name = 'First', last_name = 'Last'
WHERE actor_id = 201;
```

The example above will update the value of the first and last name of the actor with `actor_id = 201` to `First` and `Last` values.

You could try to run it and check values in database.

important

Warning

Be careful when updating records in a table! Notice the WHERE clause in the UPDATE statement. The WHERE clause specifies which record(s) that should be updated.

The **SET** clause lists only the columns that will be changed. In the example above, we updated two fields, but we can update one or more than two by listing the required fields separated by commas.

Also in the set operator, you can refer to the fields available in the query and perform operations on it that are available for this data type, for example, string concatenation.

```
UPDATE actor
SET first_name = first_name || ' ' || last_name
WHERE actor_id = 202;
```

important

Warning

Be careful when specifying where. If you forget to specify it, all values in the table will be updated!

DELETE Statement

The **DELETE** statement is used to delete a row from the database.

```
DELETE FROM table_name WHERE condition;
```

In order to delete the previously created actor with ID 201, we need to execute the following request.

```
DELETE FROM actor WHERE actor_id = 201;
```

You can check after running this query that the fetch returns 0 rows.

important

Warning

If you forget to specify the WHERE clause, all records in the table will be deleted!

In real applications, it is very common to not physically remove data from a table using the DELETE operator, but to add an additional field with a indicator that the data has been deleted.