SQL Errors: Five Common SQL Mistakes

Aldo Zelen
Aldo is a data architect with a passion for the cloud. From leading a team of data professionals to coding a data warehouse in the cloud, Aldo has experience with the whole lifecycle of data-intensive projects. Aldo spends his free time forecasting geopolitical events in forecasting tournaments.

Tags: HOW TO HOW TO IN SQL

As you learn SQL, watch out for these common coding mistakes

You've written some SQL code and you're ready to query your database.

You input the code and …. no data is returned. Instead, you get an error message.

Don't despair! Coding **errors** are common in any programming language, and SQL is no exception. In this post, we'll discuss five **common mistakes** people make when writing **SQL**.

Watch Your Language (and Syntax)

The most common **SQL error** is a **syntax error**. What does syntax mean? Basically, it means a set arrangement of words and commands. If you use improper syntax, the database does not know what you're trying to tell it.

To understand how syntax works, we can think of a spoken language. Imagine saying to a person "Nice dof" when you mean "Nice dog". The person does not know what "dof" means. So when you tell your database to find a TABEL instead of a TABLE, the database does not know what it needs to do.

People tend to make the same kinds of **syntax** mistakes, so their **errors** are usually easy to spot and very much the same. After you read this article, you should be able to remember and avoid (or fix) these *common mistakes*. Knowing what errors to look for is very important for novice *SQL* coders, especially early on. New coders tend to make more mistakes and spend more time looking for them.

> Looking for the best hands-on SQL learning experience? Here's our SQL Practice Set course.

The types of **SQL errors** we will look at are:

1. Misspelling Commands
2. Forgetting Brackets and Quotes
3. Specifying an Invalid Statement Order
4. Omitting Table Aliases
5. Using Case-Sensitive Names

Ready? Let's start.

SQL Errors:

1. Misspelling Commands

This is the most common type of *SQL mistake* among rookie and experienced developers alike. Let's see what it looks like. Examine the simple SELECT statement below and see if you can spot a problem:

```
SELECT *
FORM dish
WHERE NAME = 'Prawn Salad';
```

If you run this query, you'll get an error which states:

```
Syntax error in SQL statement "SELECT * FORM[*] dish WHERE NAME = 'Prawn Salad';";
SQL statement: SELECT * FORM dish WHERE NAME = 'Prawn Salad'; [42000-176]
```

Each database version will tell you the exact word or phrase it doesn't understand, although the error message may be slightly different.

What is wrong here? You misspelled FROM as FORM. Misspellings are commonly found in keywords (like SELECT, FROM, and WHERE), or in table and column names.

Most common *SQL* spelling *errors* are due to:

- **"Chubby fingers"** where you hit a letter near the right one: SELEVT or FTOM or WJIRE
- **"Reckless typing"** where you type the right letters in the wrong order: SELETC or FORM or WHEER

Solution:

Use an SQL editor that has syntax highlighting: the `SELECT` and `WHERE` keywords will be highlighted, but the misspelled FORM will not get highlighted.

If you're learning with INTERACTIVE SQL COURSES IN LEARNSQL.COM, the code editor puts every `SELECT` statement keyword in light purple. If the keyword is black, as it is with any other argument, you know there's a problem. (In our example, FORM is black).

So if we correct our statement we get:

```
SELECT *
FROM dish
WHERE NAME = 'Prawn Salad'
```

The keyword is now the right color and the statement executes without an error.

2. Forgetting Brackets and Quotes

Brackets group operations together and guide the execution order. In SQL (and in all of the programming languages I use), the following order of operations …

```
SELECT *
FROM artist
WHERE first_name = 'Vincent' and last_name = 'Monet' or last_name
```

… is not the same as:

```
SELECT *
FROM artist
WHERE first_name = 'Vincent' and (last_name = 'Monet' or last_name
```

Can you figure out why?

A very *common SQL mistake* is to forget the closing bracket. So if we look at this erroneous statement :

```
SELECT *
FROM artist
WHERE first_name = 'Vincent' and (last_name = 'Monet' or last_name
```

We get an error code with the position of the error (the 102nd character from the beginning):

```
ERROR: syntax error at or near ";" Position: 102
```

Remember: **brackets always come in pairs**.

The same is true with single quotes (‘ ‘) or double quotes (” ”). There is no situation in SQL where we would find a quote (either a single quote or a double

quote) without its mate. Column text values can contain one quote (e.g. `exp.last_name = "O'Reilly"`) and in these situations we must mix two types of quotes or use escape characters. (In SQL, using escape characters simply means placing another quote near the character you want to deactivate – e.g. `exp.last_name = 'O''Reilly.`)

Solution:

Practice, practice, practice. Writing more *SQL* code will give you the experience you need to avoid these *mistakes*. And remember people usually forget the **closing**bracket or quotation mark. They rarely leave out the opening one. If you're running into problems, take a close look at all your closing punctuation!

3. Invalid statement order

When writing SELECT statements, keep in mind that there is a predefined keyword order needed for the statement to execute properly. There is no leeway here.

Let's look at an example of a correctly-ordered statement:

```
SELECT name
FROM dish
WHERE name = 'Prawn Salad'
GROUP BY name
HAVING count(*) = 1
ORDER BY name;
```

There's no shortcut here; you simply have to remember the correct keyword order for the SELECT statement:

- `SELECT` identifies column names and functions
- `FROM` specifies table name or names (and `JOIN` conditions if you're using multiple tables)

- `WHERE` defines filtering statements
- `GROUP BY` shows how to group columns
- `HAVING` filters the grouped values
- `ORDER BY` sets the order in which the results will be displayed

You cannot write a `WHERE` keyword before a `FROM`, and you can't put a `HAVING` before a `GROUP BY`. The statement would be invalid.

Let's look at what happens when you mix up the statement order. In this instance, we'll use the *common SQL error* of placing `ORDER BY` before `GROUP BY`:

```
SELECT name
FROM dish
WHERE name = 'Prawn Salad'
ORDER BY name
GROUP BY name
HAVING count(*) = 1
```

The error message we see is pretty intimidating!

```
Syntax error in SQL statement "SELECT name FROM dish WHERE name = 'Prawn Salad'
ORDER BY name GROUP[*] BY name HAVING count(*) = 1;"; SQL statement:
SELECT name FROM dish WHERE name = 'Prawn Salad' ORDER BY name GROUP BY
name HAVING count(*) = 1; [42000-176]
```

Solution:

Don't be discouraged! You can see that all of the keywords are highlighted correctly and all the quotations and brackets are closed. So now you should check the statement order. When you're just beginning your SQL studies, I suggest using a `SELECT` order checklist. If you run into a problem, refer to your list for the correct order.

> Tired of listening to theory and just want to type some queries? Get yourself into learn-free querying with our SQL Practice track.

4. Omitting Table Aliases

When joining tables, creating table aliases is a popular practice. These aliases distinguish among columns with the same name across tables; thus the database will know which column values to return. This is not mandatory when we're joining different tables, since we can use the full table names. But it is mandatory if we join a table to **itself**.

Suppose we're writing an SQL statement to find an exhibition's current location and the location from the previous year:

```
SELECT *
FROM exhibit
    JOIN exhibit ON (id = previous_id);
```

The database would return an error:

```
Ambiguous column name "id"; SQL statement: SELECT * FROM exhibit JOIN exhibit ON (id = previ
```

Note: Whenever you encounter "ambiguous column name" in your error message, you surely need table aliases.

The correct statement (with aliases) would be:

```
SELECT ex.* , exp.name
FROM exhibit
    JOIN exhibit ON (ex.id = exp.previous_id);
```

Solution:

Practice using table aliases for single-table `SELECT` statements. Use aliases often – they make your SQL more readable.

5. Using Case-Sensitive Names

This error only occurs when you need to write non-standard names for tables or database objects.

Let's say that you need to have a table named **LargeClient** and for some reason you add another table called **LARGECLIENT**. As you already know, object names in databases are usually case-insensitive. So when you write a query for the **LargeClient** table, the database will actually query **LARGECLIENT**.

To avoid this, you must put double quotes around the table name. For example:

```
SELECT * FROM
"LargeClient"
 WHERE cust_name = 'Mijona';
```

When creating a table, you will need to use double quotes if:

- The table will have a case-sensitive name.
- The table name will contain special characters. This includes using a blank space, like "Large Client".

Solution:

Avoid using these names if you can. If not, remember your double quotes!

Everybody Makes SQL Mistakes

Those are the five most **common errors** in **SQL** code. You'll probably make them many times as you learn this language. Remember, everybody makes mistakes writing code. In fact, making mistakes is a normal and predictable part of software development.

So don’t be discouraged. When you make mistakes in the future, try to analyze your code in a structured way. With a structured analysis, you can find and correct your errors quicker.

If you would like to learn about some other syntactic mistakes that I’ve not included here, please let me know. In an upcoming article, we’ll look at non-syntactic errors. These return or modify data and are therefore much more dangerous. Subscribe to our blog so you won’t miss it!