Chapter 61: Clean Code in SQL

How to write good, readable SQL queries, and example of good practices.

Section 61.1: Formatting and Spelling of Keywords and Names

Table/Column Names

Two common ways of formatting table/column names are `CamelCase` and `snake_case`:

```
SELECT FirstName, LastName
FROM Employees
WHERE Salary > 500;
```

```
SELECT first_name, last_name
FROM employees
WHERE salary > 500;
```

Names should describe what is stored in their object. This implies that column names usually should be singular. Whether table names should use singular or plural is a heavily discussed question, but in practice, it is more common to use plural table names.

Adding prefixes or suffixes like `tbl` or `col` reduces readability, so avoid them. However, they are sometimes used to avoid conflicts with SQL keywords, and often used with triggers and indexes (whose names are usually not mentioned in queries).

Keywords

SQL keywords are not case sensitive. However, it is common practice to write them in upper case.

Section 61.2: Indenting

There is no widely accepted standard. What everyone agrees on is that squeezing everything into a single line is bad:

```
SELECT d.Name, COUNT(*) AS Employees FROM Departments AS d JOIN Employees AS e ON d.ID =
e.DepartmentID WHERE d.Name != 'HR' HAVING COUNT(*) > 10 ORDER BY COUNT(*) DESC;
```

At the minimum, put every clause into a new line, and split lines if they would become too long otherwise:

```
SELECT d.Name,
       COUNT(*) AS Employees
FROM Departments AS d
JOIN Employees AS e ON d.ID = e.DepartmentID
WHERE d.Name != 'HR'
HAVING COUNT(*) > 10
ORDER BY COUNT(*) DESC;
```

Sometimes, everything after the SQL keyword introducing a clause is indented to the same column:

```
SELECT   d.Name,
         COUNT(*) AS Employees
FROM     Departments AS d
JOIN     Employees AS e ON d.ID = e.DepartmentID
WHERE    d.Name != 'HR'
HAVING   COUNT(*) > 10
```

```
ORDER BY COUNT(*) DESC;
```

(This can also be done while aligning the SQL keywords right.)

Another common style is to put important keywords on their own lines:

```
SELECT
    d.Name,
    COUNT(*) AS Employees
FROM
    Departments AS d
JOIN
    Employees AS e
    ON d.ID = e.DepartmentID
WHERE
    d.Name != 'HR'
HAVING
    COUNT(*) > 10
ORDER BY
    COUNT(*) DESC;
```

Vertically aligning multiple similar expressions improves readability:

```
SELECT Model,
       EmployeeID
FROM Cars
WHERE CustomerID = 42
  AND Status     = 'READY';
```

Using multiple lines makes it harder to embed SQL commands into other programming languages. However, many languages have a mechanism for multi-line strings, e.g., `@"..."` in C#, `"""..."""` in Python, or `R"(...)"` in C++.

Section 61.3: SELECT *

`SELECT *` returns all columns in the same order as they are defined in the table.

When using `SELECT *`, the data returned by a query can change whenever the table definition changes. This increases the risk that different versions of your application or your database are incompatible with each other.

Furthermore, reading more columns than necessary can increase the amount of disk and network I/O.

So you should always explicitly specify the column(s) you actually want to retrieve:

```
--SELECT *                                 don't
  SELECT ID, FName, LName, PhoneNumber  -- do
  FROM Emplopees;
```

(When doing interactive queries, these considerations do not apply.)

However, `SELECT *` does not hurt in the subquery of an EXISTS operator, because EXISTS ignores the actual data anyway (it checks only if at least one row has been found). For the same reason, it is not meaningful to list any specific column(s) for EXISTS, so `SELECT *` actually makes more sense:

```
-- list departments where nobody was hired recently
SELECT ID,
       Name
FROM Departments
```

```
WHERE NOT EXISTS (SELECT *
                  FROM Employees
                  WHERE DepartmentID = Departments.ID
                    AND HireDate >= '2015-01-01');
```

Section 61.4: Joins

Explicit joins should always be used; implicit joins have several problems:

- The join condition is somewhere in the WHERE clause, mixed up with any other filter conditions. This makes it harder to see which tables are joined, and how.
- Due to the above, there is a higher risk of mistakes, and it is more likely that they are found later.
- In standard SQL, explicit joins are the only way to use outer joins:

```
SELECT d.Name,
       e.Fname || e.LName AS EmpName
FROM      Departments AS d
LEFT JOIN Employees   AS e ON d.ID = e.DepartmentID;
```

- Explicit joins allow using the USING clause:

```
SELECT RecipeID,
       Recipes.Name,
       COUNT(*) AS NumberOfIngredients
FROM      Recipes
LEFT JOIN Ingredients USING (RecipeID);
```

(This requires that both tables use the same column name.
USING automatically removes the duplicate column from the result, e.g., the join in this query returns a single `RecipeID` column.)