Introduction to SQL

Qitian Liao

University of California, Berkeley

Contents

1 Manipulation

Get up and running with SQL by learning commands to manipulate data stored in relational databases.

1.1 Introduction to SQL

SQL, **S**tructured **Q**uery **L**anguage, is a programming language designed to manage data stored in relational databases. SQL operates through simple, declarative statements. This keeps data accurate and secure, and helps maintain the integrity of databases, regardless of size.

The SQL language is widely used today across web frameworks and database applications. Knowing SQL gives you the freedom to explore your data, and the power to make better decisions. By learning SQL, you will also learn concepts that apply to nearly every data storage system.

The statements covered in this course use SQLite Relational Database Management System (RDBMS). You can also access a glossary of all the SQL commands taught in this chapter.

1.2 Relational Databases

In one line of code, we can return information from a relational database.

```
SELECT * FROM celebs;
```

A *relational database* is a database that organizes information into one or more tables. Here, the relational database contains one table.

A *table* is a collection of data organized into rows and columns. Tables are sometimes referred to as *relations*. Here the table is `celebs`.

A *column* is a set of data values of a particular type. Here, `id`, `name`, and `age` are the columns.

A *row* is a single record in a table.

All data stored in a relational database is of a certain data type. Some of the most common data types are:

- `INTEGER`, a positive or negative whole number
- `TEXT`, a text string
- `DATE`, the date formatted as YYYY-MM-DD
- `REAL`, a decimal value

1.3 Statements

The code below is a SQL statement. A *statement* is text that the database recognizes as a valid command. Statements always end in a semicolon `;`.

```
CREATE TABLE table_name (
   column_1 data_type,
   column_2 data_type,
   column_3 data_type
);
```

Let us break down the components of a statement:

1. `CREATE TABLE` is a *clause*. Clauses perform specific tasks in SQL. By convention, clauses are written in capital letters. Clauses can also be referred to as commands.
2. `table_name` refers to the name of the table that the command is applied to.
3. `(column_1 data_type, column_2 data_type, column_3 data_type)` is a *parameter*. A parameter is a list of columns, data types, or values that are passed to a clause as an argument. Here, the parameter is a list of column names and the associated data type.

The structure of SQL statements vary. The number of lines used does not matter. A statement can be written all on one line, or split up across multiple lines if it makes it easier to read.

1.4 Create

`CREATE` statements allow us to create a new table in the database. You can use the `CREATE` statement anytime you want to create a new table from scratch. The statement below creates a new table named celebs.

```
1 CREATE TABLE celebs (
2    id INTEGER,
3    name TEXT,
4    age INTEGER
5 );
```

1. `CREATE TABLE` is a clause that tells SQL you want to create a new table.
2. `celebs` is the name of the table.
3. `(id INTEGER, name TEXT, age INTEGER)` is a list of parameters defining each column, or attribute in the table and its data type:
   - `id` is the first column in the table. It stores values of data type `INTEGER`
   - `name` is the second column in the table. It stores values of data type `TEXT`
   - `age` is the third column in the table. It stores values of data type `INTEGER`

1.5 Insert

The `INSERT` statement inserts a new row into a table. You can use the `INSERT` statement when you want to add new records. The statement below enters a record for Justin Bieber into the celebs table.

```
1 INSERT INTO celebs (id, name, age)
2 VALUES (1, "Justin Bieber", 22);
```

1. `INSERT INTO` is a clause that adds the specified row or rows.
2. `celebs` is the name of the table the row is added to.
3. `(id, name, age)` is a parameter identifying the columns that data will be inserted into.
4. `VALUES` is a clause that indicates the data being inserted. `(1, "Justin Bieber", 22)` is a parameter identifying the values being inserted.

- `1` is an integer that will be inserted into the `id` column
- `"Justin Bieber"` is text that will be inserted into the `name` column
- `22` is an integer that will be inserted into the `age` column

1.6 Select

`SELECT` statements are used to fetch data from a database. In the statement below, `SELECT` returns all data in the name column of the celebs table.

```
1 SELECT name FROM celebs;
```

1. `SELECT` is a clause that indicates that the statement is a query. You will use `SELECT` every time you query data from a database.
2. `name` specifies the column to query data from.
3. `FROM celebs` specifies the name of the table to query data from. In this statement, data is queried from the `celebs` table.

You can also query data from all columns in a table with `SELECT`.

```
1 SELECT * FROM celebs;
```

`*` is a special wildcard character that we have been using. It allows you to select every column in a table without having to name each one individually. Here, the result set contains every column in the `celebs` table.

`SELECT` statements always return a new table called the *result set*.

1.7 Alter

The `ALTER TABLE` statement adds a new column to a table. You can use this command when you want to add columns to a table. The statement below adds a new column `twitter_handle` to the `celebs` table.

```
1 ALTER TABLE celebs
2 ADD COLUMN twitter\_handle TEXT;
```

1. `ALTER TABLE` is a clause that lets you make the specified changes.
2. `celebs` is the name of the table that is being changed.
3. `ADD COLUMN` is a clause that lets you add a new column to a table:
   - `twitter_handle` is the name of the new column being added
   - `TEXT` is the data type for the new column
4. `NULL` is a special value in SQL that represents missing or unknown data. Here, the rows that existed before the column was added have `NULL` (∅) values for `twitter_handle`.

1.8 Update

The `UPDATE` statement edits a row in a table. You can use the `UPDATE` statement when you want to change existing records. The statement below updates the record with an `id` value of `4` to have the `twitter_handle` `@taylorswift13`.

```
1 UPDATE celebs
2 SET twitter_handle = "@taylorswift13"
3 WHERE id = 4;
```

1. `UPDATE` is a clause that edits a row in the table.
2. `celebs` is the name of the table.
3. `SET` is a clause that indicates the column to edit.
    - `twitter_handle` is the name of the column that is going to be updated
    - `@taylorswift13` is the new value that is going to be inserted into the `twitter_handle` column.
    - `WHERE` is a clause that indicates which row(s) to update with the new column value. Here the row with a `4` in the `id` column is the row that will have the `twitter_handle` updated to `@taylorswift13`.

1.9 Delete

The `DELETE FROM` statement deletes one or more rows from a table. You can use the statement when you want to delete existing records. The statement below deletes all records in the `celeb` table with no `twitter_handle`:

```
1 DELETE FROM celebs
2 WHERE twitter_handle IS NULL;
```

1. `DELETE FROM` is a clause that lets you delete rows from a table.
2. `celebs` is the name of the table we want to delete rows from.
3. `WHERE` is a clause that lets you select which rows you want to delete. Here we want to delete all of the rows where the `twitter_handle` column `IS NULL`.
4. `IS NULL` is a condition in SQL that returns true when the value is `NULL` and false otherwise.

1.10 Constraints

Constraints that add information about how a column can be used are invoked after specifying the data type for a column. They can be used to tell the database to reject inserted data that does not adhere to a certain restriction. The statement below sets *constraints* on the `celebs` table.

```
1 CREATE TABLE celebs (
2    id INTEGER PRIMARY KEY,
3    name TEXT UNIQUE,
4    date_of_birth TEXT NOT NULL,
5    date_of_death TEXT DEFAULT "Not Applicable"
6 );
```

1. PRIMARY KEY columns can be used to uniquely identify the row. Attempts to insert a row with an identical value to a row already in the table will result in a *constraint violation* which will not allow you to insert the new row.
2. UNIQUE columns have a different value for every row. This is similar to PRIMARY KEY except a table can have many different UNIQUE columns.
3. NOT NULL columns must have a value. Attempts to insert a row without a value for a NOT NULL column will result in a constraint violation and the new row will not be inserted.
4. DEFAULT columns take an additional argument that will be the assumed value for an inserted row if the new row does not specify a value for that column.

1.11 Review

We have learned six commands commonly used to manage data stored in a relational database and how to set constraints on such data.

SQL is a programming language designed to manipulate and manage data stored in relational databases.

- A *relational database* is a database that organizes information into one or more tables.
- A *table* is a collection of data organized into rows and columns.

A *statement* is a string of characters that the database recognizes as a valid command.

- CREATE TABLE creates a new table.
- INSERT INTO adds a new row to a table.
- SELECT queries data from a table.
- ALTER TABLE changes an existing table.
- UPDATE edits a row in a table.
- DELETE FROM deletes rows from a table.

Constraints add information about how a column can be used.

2 Queries

Now we learn the most commonly used SQL commands to query a table in a database

2.1 Introduction

Now, we will be learning different SQL commands to query a single table in a database.

One of the core purposes of the SQL language is to retrieve information stored in a database. This is commonly referred to as querying. Queries allow us to communicate with the database by asking questions and returning a result set with data relevant to the question.

We will be querying a database with one table named `movies`.

Fun fact: IBM started out SQL as SEQUEL (**S**tructured **E**nglish **QUE**ry **L**anguage) in the 1970's to query databases.

2.2 Select

Previously, we learned that `SELECT` is used every time you want to query data from a database and `*` means all columns.

Suppose we are only interested in two of the columns. We can select individual columns by their names (separated by a comma):

```
1 SELECT column1, column2
2 FROM table_name;
```

To make it easier to read, we moved `FROM` to another line.

Line breaks do not mean anything specific in SQL. We could write this entire query in one line, and it would run just fine.

2.3 As

Knowing how `SELECT` works, suppose we have the code below:

```
1 SELECT name AS "Titles"
2 FROM movies;
```

`AS` is a keyword in SQL that allows you to rename a column or table using an alias. The new name can be anything you want as long as you put it inside of single quotes. Here we renamed the `name` column as `Titles`.

- Although it is not always necessary, it's best practice to surround your aliases with single quotes.
- When using `AS`, the columns are not being renamed in the table. The aliases only appear in the result.

2.4 Distinct

When we are examining data in a table, it can be helpful to know what distinct values exist in a particular column.

DISTINCT is used to return unique values in the output. It filters out all duplicate values in the specified column(s).

For instance,

```
1 SELECT tools
2 FROM inventory;
```

might produce:

tools
Hammer
Nails
Nails
Nails

By adding DISTINCT before the column name,

```
1 SELECT DISTINCT tools
2 FROM inventory;
```

the result would now be:

tools
Hammer
Nails

Filtering the results of a query is an important skill in SQL. It is easier to see the different possible genres in the movie table after the data has been filtered than to scan every row in the table.

```
1 SELECT DISTINCT genre
2 FROM movies;
```

2.5 Where

We can restrict our query results using the WHERE clause in order to obtain only the information we want.

Following this format, the statement below filters the result set to only include top rated movies (IMDb ratings greater than 8):

```
1 SELECT *
2 FROM movies
3 WHERE imdb_rating > 8;
```

1. WHERE clause filters the result set to only include rows where the following *condition* is true.
2. imdb_rating $>$ 8 is the condition. Here, only rows with a value greater than 8 in the imdb_rating column will be returned.

The $>$ is an operator. Operators create a condition that can be evaluated as either *true* or *false*.

Comparison operators used with the WHERE clause are:

- = equal to

- != not equal to
- > greater than
- < less than
- >= greater than or equal to
- <= less than or equal to

2.6 Like

`LIKE` can be a useful operator when you want to compare similar values.

The `movies` table contains two films with similar titles, 'Se7en' and 'Seven'.

In order to select all movies that start with 'Se' and end with 'en' and have exactly one character in the middle,

```
1 SELECT *
2 FROM movies
3 WHERE name LIKE "Se_en";
```

- `LIKE` is a special operator used with the `WHERE` clause to search for a specific pattern in a column.
- `name LIKE 'Se_en'` is a condition evaluating the `name` column for a specific pattern.
- `Se_en` represents a pattern with a wildcard character.

The `_` means you can substitute any individual character here without breaking the pattern. The names `Seven` and `Se7en` both match this pattern.

The percentage sign `%` is another wildcard character that can be used with `LIKE`.

This statement below filters the result set to only include movies with names that begin with the letter 'A':

```
1 SELECT *
2 FROM movies
3 WHERE name LIKE "A%";
```

`%` is a wildcard character that matches zero or more missing letters in the pattern. For example:

- `A%` matches all movies with names that begin with letter 'A'
- `%a` matches all movies that end with 'a'

We can also use `%` both before and after a pattern:

```
1 SELECT *
2 FROM movies
3 WHERE name LIKE "%man%";
```

Here, any movie that contains the word 'man' in its name will be returned in the result.

`LIKE` is not case sensitive. 'Batman' and 'Man of Steel' will both appear in the result of the query above.

2.7 Is Null

More often than not, the data you encounter will have missing values. Unknown values are indicated by NULL.

It is not possible to test for NULL values with comparison operators, such as = and !=.

Instead, we will have to use these operators:

- IS NULL
- IS NOT NULL

To filter for all movies *with* an IMDb rating:

```
1 SELECT name
2 FROM movies
3 WHERE imdb_rating IS NOT NULL;
```

2.8 Between

The BETWEEN operator is used in a WHERE clause to filter the result set within a certain *range*. It accepts two values that are either numbers, text or dates.

For example, this statement filters the result set to only include movies with years from 1990 up to, and including 1999.

```
1 SELECT *
2 FROM movies
3 WHERE year BETWEEN 1990 AND 1999;
```

When the values are text, BETWEEN filters the result set for within the alphabetical range.

In this statement, BETWEEN filters the result set to only include movies with names that begin with the letter 'A' up to, but not including ones that begin with 'J'.

```
1 SELECT *
2 FROM movies
3 WHERE name BETWEEN "A" AND "J";
```

However, if a movie has a name of simply 'J', it would actually match. This is because BETWEEN goes *up to* the second value — up to 'J'. So the movie named 'J' would be included in the result set but not 'Jaws'.

2.9 And

Sometimes we want to combine multiple conditions in a WHERE clause to make the result set more specific and useful.

One way of doing this is to use the AND operator. Here, we use the AND operator to only return 90's romance movies.

```
1 SELECT *
2 FROM movies
3 WHERE year BETWEEN 1990 AND 1999
4    AND genre = "romance";
```

- `year BETWEEN 1990 AND 1999` is the 1st condition.
- `genre = 'romance'` is the 2nd condition.
- `AND` combines the two conditions.

With `AND`, both conditions must be true for the row to be included in the result.

2.10 Or

Similar to `AND`, the `OR` operator can also be used to combine multiple conditions in `WHERE`, but there is a fundamental difference:

- `AND` operator displays a row if *all* the conditions are true.
- `OR` operator displays a row if *any* condition is true.

Suppose we want to check out a new movie or something action-packed:

```
1 SELECT *
2 FROM movies
3 WHERE year > 2014
4    OR genre = "action";
```

- `year > 2014` is the first condition.
- `genre = "action"` is the second condition.
- `OR` combines the two conditions.

With `OR`, if any of the conditions are true, then the row is added to the result.

2.11 Order By

It is often useful to list the data in our result set in a particular order.

We can *sort* the results using `ORDER BY`, either alphabetically or numerically. Sorting the results often makes the data more useful and easier to analyze.

For example, if we want to sort everything by the movie's title from A through Z:

```
1 SELECT *
2 FROM movies
3 ORDER BY name;
```

- `ORDER BY` is a clause that indicates you want to sort the result set by a particular column.
- `name` is the specified column.

Sometimes we want to sort things in a decreasing order. For example, if we want to select all of the well-received movies, sorted from highest to lowest by their year:

```
1 SELECT *
2 FROM movies
3 WHERE imdb_rating > 8
4 ORDER BY year DESC;
```

- DESC is a keyword used in ORDER BY to sort the results in *descending order* (high to low or Z-A).
- ASC is a keyword used in ORDER BY to sort the results in *ascending order* (low to high or A-Z).

The column that we ORDER BY does not even have to be one of the columns that we are displaying.

Note: ORDER BY always goes after WHERE (if WHERE is present).

2.12 Limit

Most SQL tables contain hundreds of thousands of records. In those situations, it becomes important to cap the number of rows in the result.

For instance, imagine that we just want to see a few examples of records.

```
1 SELECT *
2 FROM movies
3 LIMIT 10;
```

LIMIT is a clause that lets you specify the maximum number of rows the result set will have. This saves space on our screen and makes our queries run faster.

Here, we specify that the result set cannot have more than 10 rows.

LIMIT always goes at the very end of the query. Also, it is not supported in all SQL databases.

2.13 Case

A CASE statement allows us to create different outputs (usually in the SELECT statement). It is SQL's way of handling if-then logic.

Suppose we want to condense the ratings in movies to three levels:

- *If the rating is above 8, then it is Fantastic.*
- *If the rating is above 6, then it is Poorly Received.*
- *Else, Avoid at All Costs.*

```
1 SELECT name,
2  CASE
3   WHEN imdb_rating > 8 THEN "Fantastic"
4   WHEN imdb_rating > 6 THEN "Poorly Received"
5   ELSE "Avoid at All Costs"
6  END
7 FROM movies;
```

- Each WHEN tests a condition and the following THEN gives us the string if the condition is true.
- The ELSE gives us the string if all the above conditions are false.
- The CASE statement must end with END.

In the result, you have to scroll right because the column name is very long. To shorten it, we can rename the column to 'Review' using AS:

```sql
SELECT name,
 CASE
  WHEN imdb_rating > 8 THEN "Fantastic"
  WHEN imdb_rating > 6 THEN "Poorly Received"
  ELSE "Avoid at All Costs"
 END AS "Review"
FROM movies;
```

2.14 Review

We just learned how to query data from a database using SQL. We also learned how to filter queries to make the information more specific and useful. Let us summarize:

- SELECT is the clause we use every time we want to query information from a database.
- AS renames a column or table.
- DISTINCT return unique values.
- WHERE is a popular command that lets you filter the results of the query based on conditions that you specify.
- LIKE and BETWEEN are special operators.
- AND and OR combines multiple conditions.
- ORDER BY sorts the result.
- LIMIT specifies the maximum number of rows that the query will return.
- CASE creates different outputs.