The Complete SQL Cheat Sheet

Created by David Freitag
dataengineeringprojects.io

Data sample

Here’s the data we’ll be using for our query examples.

```
-- Table Name: offices
-- Description: this table contains data about company offices
-- around the world.

+----+----------+----------------+---------------+-------------+
| id | city     | country        | num_employees | date_opened |
+----+----------+----------------+---------------+-------------+
|  1 | New York | United States  |           250 |  1998-02-03 |
|  2 | Chicago  | United States  |           100 |  2001-03-12 |
|  3 | London   | United Kingdom |            75 |  2011-06-20 |
|  4 | Berlin   | Germany        |            55 |  2009-05-01 |
|  5 | Prague   | Czech Republic |            99 |  2019-09-18 |
+----+----------+----------------+---------------+-------------+
```

Our first query: SELECT all records from the table.

```sql
-- Select all rows and columns from the "offices" table.

SELECT *
FROM offices;
```

Basic querying

Using SELECT, AS, and LIMIT in queries.

```sql
-- Only select the id and city columns.

SELECT id, city
FROM offices;
```

```sql
-- Rename a column using AS.

SELECT
  id AS office_id,
  city
FROM offices;
```

```sql
-- Use LIMIT to only receive 3 rows of results.

SELECT
  *
FROM offices
LIMIT 3;
```

Using WHERE for filtering

Using WHERE with comparison operators:

```sql
-- You can use WHERE with <, >, <=, >=, =, and <>.

SELECT
  city,
  num_employees
FROM offices
WHERE num_employees > 100;
```

```sql
-- You can also use WHERE to find rows where the value in a
-- column lies between two numbers.

SELECT
  city,
  num_employees
FROM offices
WHERE num_employees BETWEEN 50 AND 100;
```

Using WHERE with logical operators:

```sql
-- Use WHERE with LIKE to match strings.

SELECT
  country,
  num_employees
FROM offices
WHERE city LIKE 'New York';
```

```sql
-- Use WHERE with IN to match sets of values.

SELECT
  country,
  num_employees
FROM offices
WHERE city IN ('New York', 'London');
```

```sql
-- Use WHERE with IS NULL or IS NOT NULL to find all rows with
-- a NULL value in a certain column.

SELECT
  country,
  num_employees
FROM offices
WHERE num_employees IS NOT NULL;
```

Combining operators

Using WHERE with AND, OR, NOT, and IN.

```sql
-- Find all offices in the United states with
-- more than 100 employees.

SELECT
  city,
  num_employees
FROM offices
WHERE num_employees > 100
  AND country = 'United States';
```

```sql
-- Find all offices that are in either the United States
-- or in the United Kingdom.

SELECT
  city,
  country
FROM offices
WHERE country = 'United States'
   OR country = 'United Kingdom';
```

```sql
-- Find all offices that are not in the United States.

SELECT
  city,
  country
FROM offices
WHERE country NOT IN ('United States');
```

```sql
-- Use parentheses to combine AND and OR. This query finds
-- the offices with more than 50 employees that are in
-- either the United States or the Czech Republic.

SELECT
  city, country
FROM offices
WHERE num_employees > 50
  AND (country = 'United States' OR country = 'Czech Republic');
```

Wildcards and ORDER BY

Using the % wildcard and ordering results.

```sql
-- Use the % wildcard operator to find offices with a country
-- whose name starts with the string, "United".

SELECT
  city, country, num_employees
FROM offices
WHERE country LIKE 'United%';
```

```sql
-- This will order by the rows by the number of employees in
-- descending order (most to least). Drop the DESC keyword
-- to order in ascending order instead.

SELECT
  city, country, num_employees
FROM offices
ORDER BY num_employees DESC;
```

Grouping and aggregating data

Use the GROUP BY keyword along with aggregate functions.

```sql
-- This will give you the average number of employees, grouped
-- by country.

SELECT
  country,
  AVG(num_employees) AS avg_emp
FROM offices
GROUP BY country;
```

```sql
-- Find the min, max, sum, and average number of employees
-- grouped by country.

SELECT
  country,
  MIN(num_employees) AS min_emp,
  MAX(num_employees) AS max_emp,
  SUM(num_employees) AS sum_emp,
  AVG(num_employees) AS avg_emp
FROM offices
GROUP BY country;
```

```sql
-- Count the total number of rows grouped by country, which
-- gives us the total number of offices in each country.

SELECT
  country,
  COUNT(*) AS cnt_offices
FROM offices
GROUP BY country;
```

```sql
-- Count the number of cities in each country where there is
-- an office.

SELECT
  country, COUNT(city) AS cnt_city
FROM offices
GROUP BY country;
```

To filter data you've aggregated, use HAVING:

```sql
-- Find the countries and the count of offices in those
-- countries where there is more than 1 office.

SELECT
  country,
  COUNT(*) AS cnt_offices
FROM offices
GROUP BY country
HAVING COUNT(*) > 1;
```

CASE WHEN statements

Use CASE WHEN to create if/then logic in your queries.

```sql
-- Use a CASE WHEN statement to abbreviate "United States".
-- The ELSE statement will keep all other country names
-- as their original strings.

SELECT
  CASE WHEN country = 'United States'
       THEN 'USA'
       ELSE country
       END AS country_short
FROM offices;
```

```sql
-- Combine SUM() with CASE WHEN to get a count of the number
-- of offices that have more than 100 employees.

SELECT
  SUM(CASE WHEN num_employees > 100
           THEN 1
           ELSE 0
           END) AS sum_offices_100_plus
FROM offices;
```

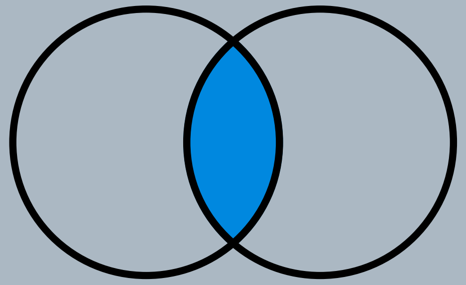
DISTINCT for unique values

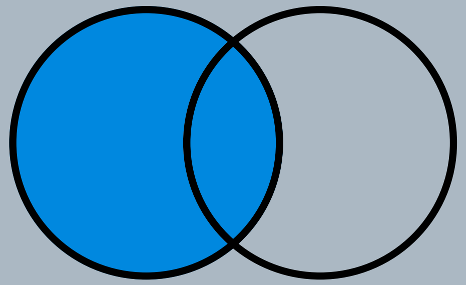
Use DISTINCT with SELECT for finding uniques:

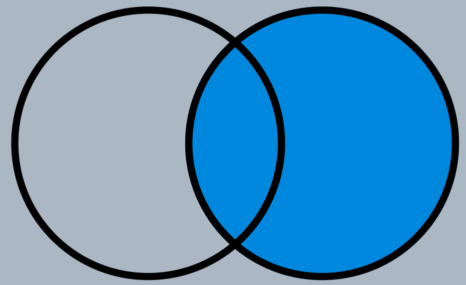
```sql
-- Use DISTINCT to get a list of countries where each value
-- appears only once (i.e. no duplicate rows.)

SELECT DISTINCT
  country
FROM offices;
```

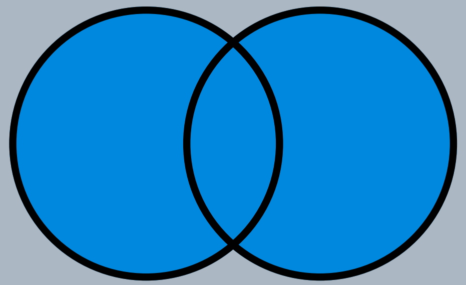
SQL joins

The 4 main SQL join types:


INNER JOIN


LEFT JOIN


RIGHT JOIN


FULL OUTER JOIN

```sql
-- Join our "offices" table to a table of "employees"
-- using the ON statement. Whenever the 'id' column
-- from the "offices" table matches the 'office_id'
-- column in the "employees" table, the join occurs.

-- Note that we use also use table name aliasing with AS.

SELECT
  o.country,
  o.city,
  e.employee_name
FROM offices AS o
INNER JOIN employees AS e
  ON o.id = e.office_id;
```

```sql
-- We can also include multiple join conditions. Here
-- we only perform a left join when the offices are located
-- in the United States.

SELECT
  o.country,
  o.city,
  e.employee_name
FROM offices AS o
LEFT JOIN JOIN employees AS e
  ON o.id = e.office_id
  AND o.country = 'United States';
```

Stack results with UNION ALL

Use UNION ALL to stack query results together.

```sql
-- Stack results into a single column that contains both
-- countries and cities.

SELECT DISTINCT
  city
FROM offices

UNION ALL

SELECT DISTINCT
  country
FROM offices;
```

CTE’s and subqueries

Split your query into multiple queries:

```sql
-- Create a CTE (Common Table Expression), which serves as a
-- kind of temp table we can query from later in the main
-- part of our query.

WITH unique_cities AS (
  SELECT DISTINCT
    city
  FROM offices
)

SELECT
  *
FROM unique_cities;
```

```sql
-- Create a subquery and combine it with NOT IN to find
-- only the cities with offices > 100 employees.

SELECT DISTINCT
  city
FROM offices
WHERE city NOT IN (
  SELECT city
  FROM offices
  WHERE num_employees > 100);
```

Manipulating data

Manipulate data using a variety of functions.

```sql
-- Manipulate strings using UPPER() and LOWER().
-- Select a substring using SUBSTR().

SELECT DISTINCT
  UPPER(city) AS uppercase_city,
  LOWER(country) AS lowercase_country,
  SUBSTR(city, 1, 3) AS first_3_city_name
FROM offices;
```

```sql
-- Create a single column with combined city + country
-- and add a comma plus a space between the two.

SELECT
  CONCAT(city, ', ', country) AS comb_city_country
FROM offices;
```

```sql
-- Extract components out of date columns to create new
-- columns. This one is just the year each office opened.

SELECT
  EXTRACT(YEAR FROM date_opened) AS yr_opened
FROM offices;
```

```sql
-- Find the number of characters in a value in a column. This
-- query shows how to find the names of cities and their lengths.

SELECT
  city,
  LENGTH(city) AS length_city_name
FROM offices;
```

```sql
-- COALESCE() returns the first non-null value inside the
-- parentheses. If num_employees is NULL, the the column is
-- filled with the value 0 instead.

SELECT
  COALESCE(num_employees, 0) AS non_null_num_employees
FROM offices;
```

Window functions

Use window functions to combine aggregating and grouping of data.

```sql
-- You can create something called a window function, which is
-- an aggregation applied over a grouping (defined with the
-- PARTITION BY statement). This query gives us the number
-- of employees per country.

SELECT DISTINCT
  country,
  SUM(num_employees) OVER(PARTITION BY country) AS num_emp_in_country
FROM offices;
```

```sql
-- You can even take things a step further by including
-- an ORDER BY statement along with your PARTITION BY
-- statement as part of the window function. This query
-- finds the FIRST_VALUE() in the column num_employees for
-- each country, ordered by date_opened, i.e. how many employees
-- are in the office that was the first one opened in each
-- country?

SELECT DISTINCT
  country,
  FIRST_VALUE(num_employees) OVER(PARTITION BY country
              ORDER BY date_opened) AS num_first
FROM offices;
```

```sql
-- You can also use DENSE_RANK() and ROW_NUMBER() to get
-- different kinds of "rankings." Here's the breakdown of each
-- when ties occur in the rankings:
-- RANK()         1,1,1,4,5 - ties allowed
-- DENSE_RANK()   1,1,1,2,3 - ties allowed
-- ROW_NUMBER()   1,2,3,4,5 - ties not allowed

-- If there are no ties, all 3 functions return the same results.

SELECT
  country, city, date_opened,
  RANK() OVER(ORDER BY date_opened) AS rank_num,
  DENSE_RANK() OVER(ORDER BY date_opened) AS dense_rank_num,
  ROW_NUMBER() OVER(ORDER BY date_opened) AS row_num
FROM offices
ORDER BY date_opened;
```