**Basic Math and Stats with SQL**

Depending on the mathematical operation, the data type returned is as follows:

* Two integers return an integer
* A numeric on either side of the operator returns a numeric
* Anything with a floating-point number returns a floating point number of type double precision

Exponentiation, root and factorial functions returns numeric and floating-point types even when the input is an integer.

If we are working with Integers, and need to force decimal division, you can cast one integer to a numeric format.

Syntax when carrying out operations on columns in tables

SELECT col\_1,col\_2 etc,

Col\_1 + col\_2 as new\_col

FROM us\_counties\_2010

**Joining Tables in a Relational Database**

Syntax of SQL joins:

SELECT \*

FROM table\_a JOIN table\_b

ON table\_a.key\_column = table\_b.foreign\_key\_column

Similar to a normal SQL syntax, but we follow the first FROM table\_a with a JOIN table\_b. We give a ON keyword to specify that we want the tables to join on columns **values** being equal between the 2 tables.

We can also do JOIN ON table\_a.key\_column >= table\_b.foreign\_key\_column.

A primary key is a column or collection of columns whose values uniquely identify each row in a table.

A valid primary key enforces certain constraints:

* The primary key column must have a unique value for each row
* The column of the primary key cannot have missing values.

Adding a foreign\_key to a different table:

dept\_id integer REFERENCES departments (dept\_id)

The above syntax creates a foreign key in the other table, given that the referring table (in this case departments) contains the column we are referring to.

The UNIQUE constrain makes sure that the values in a column or the combination of values in more than one column are unique.

Unique combinations are made using integers, as using strings would require repeating texts over large number of rows. Also if a name is changed, this would mean changing the value in many different rows, instead of 1 table (the case if you use integers).

**JOIN types**

1. JOIN: Returns rows from both tables where matching values are found in the joined columns of both tables Also called INNER JOIN
2. LEFT JOIN: Returns every row of the left table, plus the rows which match values in the joined column of the right table. When there is no matching result in right table, relevant to rows in left table, no values from the right table is shown.
3. RIGHT JOIN: Returns every row from the right table and the matching rows of the left table.
4. FULL OUTER JOIN: Returns all the rows from the left and right tables. Then it joins the tables depending on matching values. Any non matched rows will be shown empty on the particular empty side.
5. CROSS JOIN: Returns every possible combination of rows from both tables.

**Using NULL to find rows with missing values**

For large tables, we need to filter to show all rows with missing values. To do this we employ the keyword NULL.

When a SQL join returns empty rows in one of the tables, those columns don’t come back empty, but instead comes back with the value NULL.

**Types of Table Relationships**

* One to one
* One to many
* Many to many

**One to one**

There is only one match for an id in each of the two tables. One row matches one other row with the same id/primary key

**One to many relationship**

A key value in the first table will have multiple matches in the second tables, joined columns.

**Many to many relationship**

Multiple rows in the first table will have multiple matching rows in the second table.

**Selecting specific columns in a JOIN**

When joining tables, and querying specific columns, we need to include the columns and the table name. Else the columns reference will be ambiguous.

Instead we need to add the table name in front of each column we are querying.

**Simplifying JOIN syntax with Table aliases**

Syntax:

SELECT table\_alias1.col\_name, table\_alias2.col\_name

FROM table\_name1 as table\_alias1 LEFT JOIN table\_name2 as table\_alias2

**JOINING multiple tables**