**Database Overview**

* Databases are systems to use and store data
* Great for large amounts of data in comparison to excel spreadsheets

|  |  |
| --- | --- |
| **Spreadsheets** | **Databases** |
| * Great for one-time analysis * Quick with reasonable data size * Small learning curve | * Handle mass amounts of data * Can automate for reuse * Great for data integrity |

Basic Operators

* **<** : less than
* **>** : greater than
* **<=** : less than or equal to
* **>=** : greater than or equal to
* **=** : equal to
* **<>** or **!=** : not equal to
* Random note – use **‘\_’** for any string/text or date value only.

**SQL Fundamentals**

Main Functions: Calls most commonly used

* **SELECT**: most common, used to retrieve information/columns
  + Syntax
    - *SELECT column\_name  
      FROM table\_name*
  + ***SELECT DISTINCT***: variation of SELECT, returns unique values in a column
    - Syntax
      * *SELECT DISTINCT column  
        FROM table*
* **FROM**: used to choose the table from which to retrieve information
  + Syntax
    - *SELECT column  
      FROM table*
* **WHERE**: specifies conditions for rows, acts like a filter
  + Syntax
    - *SELECT column  
      FROM table  
      WHERE condition for rows*
* **ORDER BY**: sorts output by the chosen column in ascending (***ASC***) or descending (***DESC***) order.
  + Including ASC or DESC is optional. If not included function defaults to ASC
  + Syntax
    - *SELECT column 1, column 2, column 3  
      FROM table  
      ORDER BY column 1, column 3 ASC*
* **LIMIT**: allows to limit number of rows returned.
  + Always the last command in a query
  + Syntax
    - *SELECT column  
      FROM table  
      WHERE condition  
      ORDER BY column  
      LIMIT #*

Keywords: Calls used with other functions

The following are the most commonly used.

* ***BETWEEN***: used to match a value against a range of values
  + Similar to using >= for the low end of a range and <= for the high end of a range
  + Commonly used with dates in the format of ‘YYYY-MM-DD’
  + Syntax
    - *WHERE value BETWEEN 1 AND 5*
    - *WHERE date\_column BETWEEN ‘2020-02-20’ AND ‘2022-02-22’*
* ***AND/OR***: used to add more criteria to a function
  + Is also used with keywords like BETWEEN
  + Syntax
    - *WHERE value is > 1 AND value is < 5*
    - *WHERE value is >1 OR value is < 5*
    - *WHERE column = ‘Color’ AND value = ‘Blue’*
    - *WHERE color\_column = ‘Red’ OR color\_column = ‘Blue’*
* ***IN***: creates a condition to check to see if a value is included in a list
  + Can be used instead of multiple AND keywords with a WHERE function.
  + Syntax – (using last example from above)
    - *WHERE color\_column IN (‘red’, ‘blue’)*
* ***NOT***: used to say no, or the opposite, to a condition or keyword
  + Syntax
    - *WHERE color\_column NOT IN (‘red’, ‘blue’)*
    - *WHERE value NOT BETWEEN 1 AND 5*
* ***LIKE***: allows us to perform pattern matching
  + This keyword is case sensitive
  + Wildcard characters to be used with this keyword – ‘**%**’ and ‘\_’
    - **%** matches any sequence of characters
      * Syntax – all names that begin with ‘A’
        + *WHERE name LIKE ‘A%’*
    - **\_** matches any single character
      * Syntax – get all mission impossible movies
        + *WHERE movie LIKE ‘Mission Impossible \_’*
* ***ILIKE***: allows to perform a matching pattern, just like LIKE, but it is not case sensitive
  + Syntax – all names that begin with ‘A’
    - WHERE name ILIKE ‘a%’
      * This returns all names that start with the letter ‘a’, regardless of the names’ capitalization.
  + Syntax – get all mission impossible movies
    - WHERE movie ILIKE ‘mission impossible \_’
      * This returns all the Mission Impossible movies, regardless of the movies’ capitalization in their title. This would not work for LIKE.

**GROUP BY Statements**

Aggregate Functions: commonly used to take multiple inputs and return them in a single output

The following are the most used. These functions do not work without ( ) around the chosen data. When using aggregate functions, you cannot pull multiple columns + aggregate functions without using GROUP BY, as the results of the none-aggregated columns will have more rows than the aggregated ones and therefore produce an ERROR MESSAGE regarding syntax.

* ***AVG ( )*** – returns the average value
  + Returns output in a float data type; the numbers have multiple decimal places
  + Use **ROUND ( )** to round-up to the desired amount of decimal places.
  + Syntax without ROUND ( ) – i.e., output is *12.34567890123456*
    - *SELECT column1, AVG(column2)  
      FROM table  
      GROUP BY column1*
  + Syntax with ROUND ( ) to get 2 decimal points in output – i.e., output is *12.35*
    - *SELECT column1, ROUND(AVG(column2), 2)  
      FROM table  
      GROUP BY column1*
      * The # after the comma in ROUND ( ) represents the desired number of decimal places you wish to see.
* ***COUNT ( )*** – returns the number/count of values
  + It can be used with DISTINCT to count unique values in a column
  + Syntax
    - *SELECT COUNT (column)  
      FROM table*
    - *SELECT COUNT (DISTINCT (column))  
      FROM table*
      * Notice with DISTINCT you need to be careful with ( )
* ***MAX ( )*** – returns the maximum value
  + Syntax
    - *SELECT MAX(column)  
      FROM table*
* ***MIN ( )*** – returns the minimum value
  + Syntax
    - *SELECT MIN(column)  
      FROM table*
* ***SUM ( )*** – returns the sum of values
  + Syntax
    - *SELECT SUM(column)  
      FROM table*

GROUP BY Operators

* **GROUP BY**: aggregates columns per some category
  + Must come after FROM or WHERE
  + For timestamps, use **DATE ( )** function to convert into a categorical column, as it’s too specific to be one as is.
  + DATE output will have the following format ‘YYYY-MM-DD’
    - Syntax – timestamp value format = ‘YYYY-MM-DD HH:MM:SS.SSSSS’
      * *SELECT DATE(timestamp\_column) 🡪 converts to only date!  
        FROM table*
* **HAVING**: filters after an aggregation has taken place
  + Allows you to use aggregation result as a filter
  + Syntax
    - *SELECT company, SUM(sales)  
      FROM table  
      WHERE company != ‘Google’  
      GROUP BY company  
      HAVING SUM(sales) > 100*

**JOINS**

**JOIN** Statement: allows us to combine information from different tables

* ***AS*** clause: creates an alias for columns, tables, or results.
  + Syntax
    - *SELECT SUM(column) AS new\_name  
      FROM table*
    - *SELECT abc.column  
      FROM table AS abc*
  + AS must always go directly after the name you want to change.
* ***ON*** clause: helps join tables with a column from each.
  + Syntax
    - *SELECT \*  
      FROM tableA  
      JOIN tableB  
       ON tableA.column1 = tableB.column1*
* ***USING*** clause: functions just as the ON clause, but works with columns from different tables that have the same name to shorten the process.
  + Syntax
    - *SELECT \*  
      FROM tableA  
      JOIN tableB  
      USING (column1)*
* **INNER JOIN**: will output matching set of records from both tables
  + If only JOIN is written in PostgreSQL, it defaults to INNER JOIN
  + Syntax
    - *SELECT \*  
      FROM tableA  
      INNER JOIN tableB  
       ON tableA.col\_match = tableB.col\_match*

OUTER JOINs

Different types of OUTER JOINs allow us to specify how to deal with values that are only present in one the tables. The different types include FULL OUTER, LEFT OUTER, & RIGHT OUTER JOIN.

* **FULL OUTER JOIN**: grabs everything from both tables
  + If only **OUTER JOIN** is typed in, it defaults to a FULL OUTER JOIN.
  + When output is produced, anytime there are values missing from a column, because the data doesn’t match on both sides, the value will be ‘*null*’.
  + Syntax
    - *SELECT \*  
      FROM tableA  
      FULL OUTER JOIN tableB  
       ON tableA.column = tableB.column*
  + Syntax with WHERE: creates the exact opposite result of an INNER JOIN
    - *SELECT \*  
      FROM tableA  
      OUTER JOIN tableB  
       USING (column)  
      WHERE tableA.id IS* ***NULL*** *OR tableB.id IS* ***NULL***
  + The output from the example above shows the results that are unique to only table A or table B. It is the exact opposite of INNER JOIN results.
* **LEFT OUTER JOIN:** gives results present in the left table
  + If there is not a match with values in the right table, the result will be **NULL**
  + If you write **LEFT JOIN**, it will default to LEFT OUTER JOIN
  + Syntax
    - *SELECT \*  
      FROM tableA  
      LEFT JOIN tableB  
       USING (column)*
  + Syntax with WHERE: grabs rows unique to TableA
    - *SELECT \*  
      FROM tableA  
      LEFT JOIN tableB  
       USING (column)  
      WHERE tableB.id IS* ***NULL***
* **RIGHT OUTER JOIN**: gives results present in the right table, same to LEFT JOIN but with the tables switched between the FROM and JOIN functions.
  + If there is not a match with values in the right table, the result will be **NULL**
  + If you write **RIGHT JOIN**, it will default to RIGHT OUTER JOIN
  + If tables are switched in order, it produces the same results as a LEFT JOIN
  + Syntax
    - *SELECT \*  
      FROM tableA  
      RIGHT JOIN tableB  
       USING (column)*
  + Syntax with WHERE: grabs rows unique to TableA
    - *SELECT \*  
      FROM tableA  
      RIGHT JOIN tableB  
       USING (column)  
      WHERE tableA.id IS* ***NULL***
* **UNION**: used to combine outputs of two or more SELECT statements
  + Basically, concatenates two or more results
  + Syntax
    - *SELECT \* FROM table1  
      UNION  
      SELECT \* FROM table2*

**Advance SQL Commands**

Timestamps & EXTRACT

When choosing time data types for tables and databases, must be careful because not all data (like time zone) might be necessary. If too much is available, you can always remove some.

To pull time-type functions you must be aware of some different parameters. To learn about the different parameters, use **SHOW** to output the value of run-time parameters. Use **SHOW ALL** to see the current parameters your pc is working on. i.e., *SHOW TIMEZONE* outputs the pc’s current time zone it is working on.

* ***TIME*** – contains only time
* ***DATE*** – contains only date
* ***TIMESTAMP*** – contains date & time
* ***TIMESTAMPTZ***– contains date, time, and time zone
* ***TIMEZONE*** – gives time zone
* ***NOW ( )*** – gives timestamp with time zone (GMT)
* ***TIMEOFDAY*** – gives timestamp with time zone as a string
* ***CURRENT\_TIME*** – gives current time with time zone
* ***CURRENT\_DATE*** – gives current date in international format
* ***EXTRACT ( )*** – allows you to extract a sub-component of a date value
  + Use with YEAR, MONTH, DAY, WEEK, QUARTER
  + Syntax
    - *EXTRACT (YEAR FROM date\_col)*
* ***AGE ( )*** – calculates and returns the current age of a given timestamp
  + Output will look like 13 years 1 month 5 days 01:23:45.678901234
  + Syntax
    - *AGE(date\_col)* 🡪 takes current date & calculates how old timestamp is
* ***TO\_CHAR ( )***: general job is to convert data types to text
  + Particularly useful for timestamp formatting
  + Syntax
    - *TO\_CHAR(date\_col, ‘mm-dd-yyyy’)*

Mathematical Functions

Many math functions are available to use. To find them, go online as there are many resources detailing them all. The following are basic examples of some math functions:

* Division
  + *SELECT value\_col / value\_col2 FROM table*
* Rounded Division
  + *SELECT ROUND(value\_col / value\_col2, 2) FROM table*
* Percentage
  + *SELECT ROUND(value\_col / value\_col2, 2) \* 100 FROM table*
* Multiplication
  + *SELECT value\_col \* 2 FROM table*

String Functions & Operations

The following are basic examples of functions and operators to manipulate string values.

* **LENGTH**: gives back # of characters of a value
  + *SELECT LENGTH(col) FROM table*
* Concatenation using ||
  + *SELECT first\_name || ‘a’ || last\_name FROM table*
* Combining concatenation with other functions to create new employee emails
  + *SELECT LOWER(LEFT(first\_name, 1)) || LOWER(last\_name)|| ‘@gmail.com’  
    FROM table*
    - **LOWER** – makes the output lowercase
    - **LEFT** ( col\_name, # ) – pulls the left-most characters of a string value. # refers to number of characters to include, going from left to right.

Subquery

Allows you to create more complex queries by creating a query that’s based on the results of another query.

* Syntax – we want the values from the table that were above the average of that same table’s values.
  + *SELECT col\_1, col\_num  
    FROM table  
    WHERE col\_num > (SELECT AVG(col\_num)  
     FROM table)*
* Subqueries are run first because they are inside a parenthesis, similar to PEMDAS
* **IN** can also be used with subqueries to create a “list”
  + Syntax – want to include values that are inside of another table
    - *SELECT col\_1, col\_2  
      FROM table1  
      WHERE col\_1 IN (SELECT col\_1  
       FROM table2)*
* **EXISTS**: used to test for the existence of rows in subqueries
  + Syntax
    - *SELECT col\_name  
      FROM table  
      WHERE EXISTS (SELECT col\_name  
       FROM table  
       WHERE condition)*

Self-Join: a query where the table is joined to itself

Useful for comparing values in a column within the same table. It basically joins 2 copies of the same table – an alias is necessary in order to maintain the data organized.

* Syntax:
  + *SELECT tableA.col, tableB.col  
    FROM table AS tableA  
    JOIN table AS tableB  
     ON tableA.some\_col = tableB.other\_col*

SYNTAX ORDER

**SELECT** column1, SUM(column2)  
**FROM** table1  
**JOIN** table2  
 **ON** table1.column = table2.column  
**WHERE** column1 = ‘Red’  
**GROUP BY** column1  
**HAVING** SUM(column2) > 5  
**ORDER BY** column1 DESC  
**LIMIT** 10

**Creating Databases & Tables**

**Data types:**

* + - * + Boolean (TRUE or FALSE)
        + Character (text)
        + Numeric (integers, float,…)
        + Temporal (date, time, interval,…)
        + UUID
        + Array
        + JSON

**Primary & Foreign Keys**

Primary Key: column or group of columns to identify a row uniquely in a table (non-null), (ID = 1,2,3,4,…). [PK]

Foreign Key:

Under the ‘schemas’ you can view the dependencies; the tables and and columns.

Under schemas > tables > constraints you can view the primary (golden key icon)/foreign keys

* **XXXX**: allows to limit number of rows returned.
  + Always the last command in a query
  + Syntax
    - *SELECT column  
      FROM table  
      WHERE condition  
      ORDER BY column  
      LIMIT #*

**Preguntas**

Is there a way to deal with NAs? Taking the average,…

Can I make comments in .sql files? To upload to Github.