SQL TUTORIAL NOTES

SELECT

- Queries don't change any data in the tables.
- Capitalizing the words "Select" and "From" is convention to make the queries more readable.
- SQL treats one space, multiple spaces and/or a line break as the same thing.
    - For instance:
        - `SELECT *        FROM tutorial.us_housing_units`

          is the same as

        -
          ```
          SELECT *
            FROM tutorial.us_housing_units
          ```
- All of the columns are name in lower case and use underscores instead of spaces → `tutorial.us_housing_units.`
- The table name itself also uses underscores instead of spaces.
- Most people just avoid space in columns, due to is annoying dealing with them. For instance, if you wanted to refer to a column named with spaces, you'll have to user double quotes every time → `"my column with spaces"`
- You can rename columns to include spaces. For instance, if you wanted the west column to appear as West Region in the results, you would have to:

  ```
        Select west As "West Region"
              From tutorial.us_housing_units
  ```

- Without the double quotes, that query would read "West" and "Region" as separate object and would return an error. The results will only return capital letters if you put the column name into double quotes. For example:

  ```
              Select west as West_Region, South as South_Region
                    From tutorial.us_housing_units
  ```

Automatic LIMIT in Mode

- It restricts how many rows the SQL query returns. The default is 100.
- Limits are used to keep their queries from taking too long to run and return.
- Example:

  ```
        SELECT *
          FROM tutorial.us_housing_units
         LIMIT 100
  ```

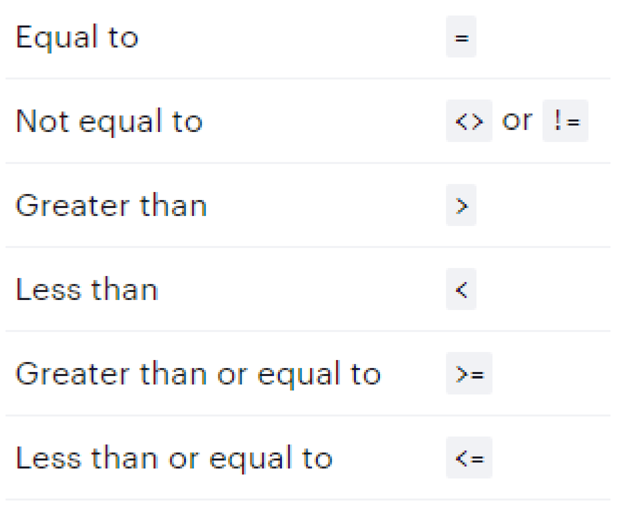
SQL Where

- It's mainly used to filter the result of a select statement.
- For instance, let's get all of those rows whose month is the first one.

  ```
  SELECT *
        FROM tutorial.us_housing_units
    WHERE month = 1
  ```

- The order is: Select → From → Where

SQL Comparison operators



Equal to	=
Not equal to	<> or !=
Greater than	>
Less than	<
Greater than or equal to	>=
Less than or equal to	<=

Figure 1. Comparison operators in SQL.

- Most of these operators are applied over numerical columns. We can use them with non-numerical columns though, by enclosing them in single quotes.
- *SQL USES SINGLE QUOTES TO REFERENCE TO COLUMN VALUES.*
- When these operators are used over non-numerical columns, the filters are based on alphabetical order.
- See the examples of both cases.
    - –Numerical columns

        -
          ```
          SELECT *
            FROM tutorial.us_housing_units
           WHERE west > 30
          ```
    - -Non-numerical columns
      ```
      SELECT *
        FROM tutorial.us_housing_units
       WHERE month_name != 'January'
      ```

- It's important to mention that SQL considers 'Ja' > 'J' because it has an extra letter. It's worth noting that most dictionaries would list 'Ja' after 'J' as well.

Arithmetic in SQL

- In SQL arithmetic is only allowed across columns on values in a given row, this is, you can only add values in multiple columns from the same row together, using + or –.

```
SELECT year,
       month,
       west,
       south,
       west + south AS south_plus_west
  FROM tutorial.us_housing_units
```

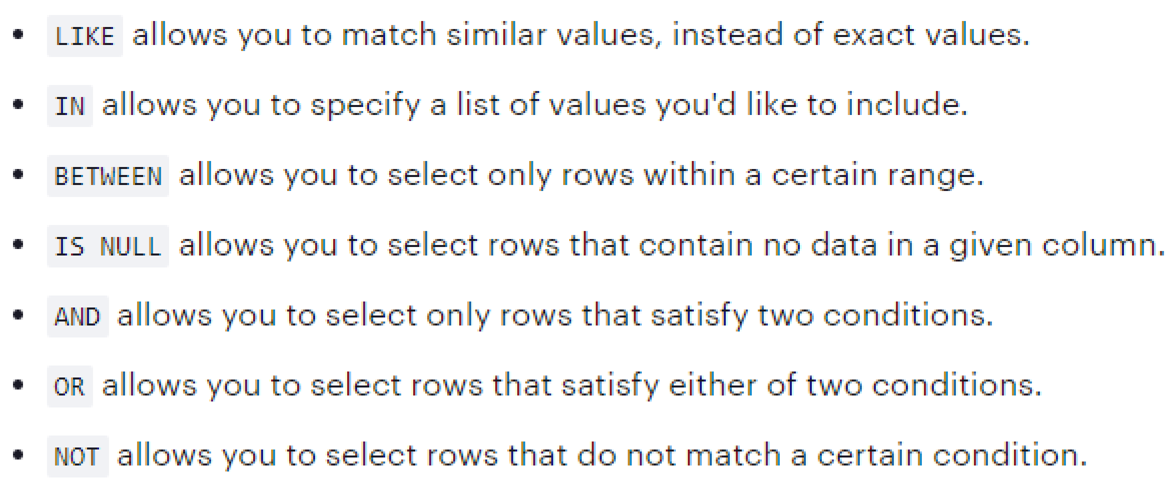
- It's possible to chain several arithmetic functions, including both columns names and actual numbers.

```
SELECT year,
       month,
       west,
       south,
       west + south - 4 * year AS nonsense column
  FROM tutorial.us_housing_units
```

- To add values across multiple rows, you have to use aggregate functions. (We'll talk about later on).
- As in (I'd say) in any software, we can add parentheses to manage the order of the operations. Sometimes it makes sense to use parentheses even when it's not absolutely necessary just to make your query easier to read. For example.

```
SELECT year,
       month,
       west,
       south,
       (west + south)/2 AS south_west_avg
  FROM tutorial.us_housing_units
```

SQL LOGICAL OPERATORS



- `LIKE` allows you to match similar values, instead of exact values.
- `IN` allows you to specify a list of values you'd like to include.
- `BETWEEN` allows you to select only rows within a certain range.
- `IS NULL` allows you to select rows that contain no data in a given column.
- `AND` allows you to select only rows that satisfy two conditions.
- `OR` allows you to select rows that satisfy either of two conditions.
- `NOT` allows you to select rows that do not match a certain condition.

Figure 2. SQL Logical Operators

- *Like* is a logical operator (case sensitive) that allows you to match on similar values rather than exact ones.
  - *For instance, let's get a list of songs whose group start with "Snoop"*

    -
      ```
      Select *
      From tutorial.billboard_top_100_year_end
      ```

```
Were  "group" LIKE 'Snoop%'
```

- Notice that on this case the word "group" appears in between quotes, because the word group is actually the name of a function in SQL.
- Generally speaking, enclosing a word in quotes, indicates that you're referring to a column name.
- The symbol "%" is called wildcard and represent any character or set of characters.
- The non-case sensitive of the logical operator *LIKE* is *ILIKE.*
- *The character "_" (underscore) is another wildcard to substitute a single character*

    -
      ```
      SELECT *
        FROM tutorial.billboard_top_100_year_end
       WHERE artist ILIKE 'dr_ke'
      ```
- *Exercises:*
    - *Query to show all of rows for which Ludacris was a member of the group*

        -
          ```
          Select *
          From tutorial.billboard_top_100_year_end
          Where "group" ilike '%Ludacris%'
          ```
    - Query that returns all rows for which the first artist listed in the group has a name that begin with "DJ"

SQL IN

- Logical operator that allows you t specify a list of values that you'd like to include in the results. Examples:

    -
      ```
      SELECT *
        FROM tutorial.billboard_top_100_year_end
      WHERE year_rank IN (1, 2, 3)
      ```

    -
      ```
      SELECT *
        FROM tutorial.billboard_top_100_year_end
       WHERE artist IN ('Taylor Swift', 'Usher',
      'Ludacris')
      ```
- Write a query that shows all of the entries for Elvis and M.C. Hammer.asdfas

    -
      ```
      SELECT *
      ```

    -
      ```
        FROM tutorial.billboard_top_100_year_end
      ```

    -
      ```
       WHERE "group" IN ('M.C. Hammer', 'Hammer', 'Elvis
      Presley')sadfasdf
      ```

SQL BETWEEN

- Logical operator that allows you to select only rows that are within a specific range. It has to be paired with the operator *AND*. Example.

    -
      ```
      SELECT *
        FROM tutorial.billboard_top_100_year_end
       WHERE year_rank BETWEEN 5 AND 10
      ```
- Between includes the range bounds (in the previous case, 5 and 10). So, this query will return exactly the same result.

    -
      ```
      SELECT *
        FROM tutorial.billboard_top_100_year_end
       WHERE year_rank >= 5 AND year_rank <= 10dsfasd
      ```
- Write a query that shows all top 100 songs from January 1, 1985, through December 31, 1990.

    -
      ```
      Select *
      FROM tutorial.billboard_top_100_year_end
      Where year BETWEEN 1985 AND 1990
      ```

SQL IS NULL

- Logical operator that allows to exclude rows with missing data from your results. Example:

```
SELECT *
  FROM tutorial.billboard_top_100_year_end
 WHERE artist IS NULL
```

- `WHERE artist = NULL` will not work—you can't perform arithmetic on null values.
- Write a query that shows all of the rows for which song_name is null.

```
SELECT *
  FROM tutorial.billboard_top_100_year_end
 WHERE song_name IS NULL
```

SQL AND

- Logical operator that allows you to select only rows that satisfy two conditions.

```
SELECT *
  FROM tutorial.billboard_top_100_year_end
 WHERE year = 2012 AND year_rank <= 10
```

- It can be used as many times as you need it.

```
SELECT *
  FROM tutorial.billboard_top_100_year_end
 WHERE year = 2012
   AND year_rank <= 10
   AND "group" ILIKE '%feat%'
```

- Write a query that surfaces all rows for top-10 hits for which Ludacris is part of the Group.

```
Select *
FROM tutorial.billboard_top_100_year_end
Where year_rank <= 10 and "group" ilike '%Ludacris%'
```

- Write a query that surfaces the top-ranked records in 1990, 2000, and 2010.

```
SELECT *
  FROM tutorial.billboard_top_100_year_end
 WHERE year_rank = 1
   AND year IN (1990, 2000, 2010)
```

- Write a query that lists all songs from the 1960s with "love" in the title.

```
Select *
From tutorial.billboard_top_100_year_end
Where year >= 1960 and
      year < 1970 and
      song_name ilike '%love%'
```

SQL OR

- It allows you to select rows that satisfy either or two conditions.

```
SELECT *
     FROM tutorial.billboard_top_100_year_end
     WHERE year_rank = 5 OR
     artist = 'Gotye'
```

- OR, AND can be used together, using parenthesis. Example:

  -
    ```
    SELECT *
       FROM tutorial.billboard_top_100_year_end
        WHERE year = 2013 AND
       ("group" ILIKE '%macklemore%' OR
       "group" ILIKE '%timberlake%')
    ```
- Write a query that returns all rows for top-10 songs that featured either Katy Perry or Bon Jovi.

  -
    ```
    SELECT *
          FROM tutorial.billboard_top_100_year_end
         WHERE year_rank <= 10 AND
         ("group" ILIKE '%katy perry%' OR
         "group" ILIKE '%bon jovi%')
    ```
- Write a query that returns all songs with titles that contain the word "California" in either the 1970s or 1990s.

  -
    ```
    SELECT *
      FROM tutorial.billboard_top_100_year_end
     WHERE song_name LIKE '%California%' AND
    (year BETWEEN 1970 AND 1979 OR
    year BETWEEN 1990 AND 1999)
    ```
- Write a query that lists all top-100 recordings that feature Dr. Dre before 2001 or after 2009.

  -
    ```
    SELECT *
      FROM tutorial.billboard_top_100_year_end
     WHERE "group" ILIKE '%dr. dre%'
       AND (year <= 2000 OR year >= 2010)
    ```

SQL NOT

- Logical operator you can put before any conditional statement to select those rows which that statement is false. Example:

  -
    ```
    SELECT *
      FROM tutorial.billboard_top_100_year_end
     WHERE year = 2013 AND
    year_rank NOT BETWEEN 2 AND 3
    ```
- Using NOT with < and > usually doesn't make sense because you can simply use the opposite comparative operator instead. Example.

  -
    ```
    SELECT *
      FROM tutorial.billboard_top_100_year_end
     WHERE year = 2013
       AND year_rank NOT > 3
    ```

- Can be turned into…

  -
    ```
    SELECT *
      FROM tutorial.billboard_top_100_year_end
     WHERE year = 2013
       AND year_rank <= 3
    ```
- NOT is commonly used with LIKE. I.e.:

  -
    ```
    SELECT *
      FROM tutorial.billboard_top_100_year_end
     WHERE year = 2013
       AND "group" NOT ILIKE '%macklemore%'
    ```
- Not is also frequently used to identify not-null rows. I.e.

  -
    ```
    SELECT *
      FROM tutorial.billboard_top_100_year_end
     WHERE year = 2013
       AND artist IS NOT NULL
    ```
- Write a query that returns all rows for songs that were on the charts in 2013 and do not contain the letter "a".

  -
    ```
    SELECT *
          FROM tutorial.billboard_top_100_year_end
         WHERE song_name NOT ILIKE '%a%'
           AND year = 2013
    ```

SQL ORDER BY

- It allows you to reorder the results based on the data of one or more columns. I.e.

  -
    ```
    SELECT *
      FROM tutorial.billboard_top_100_year_end
     ORDER BY artist
    ```
- It sorts the result alphabetically and from min to max when it comes of numerical data by default → ORDER BY artist ASC (the asc part isn't necessary due to it's the default value)
- It's possible to sort in reverse order by replacing (or just adding) the word DESC.

  -
    ```
    SELECT *
      FROM tutorial.billboard_top_100_year_end
     WHERE year = 2013
     ORDER BY year_rank DESC
    ```
- Write a query that returns all rows from 2012, ordered by song title from Z to A.

  -
    ```
    SELECT *
      FROM tutorial.billboard_top_100_year_end
     WHERE year = 2012
     ORDER BY song_name DESC
    ```

ORDERING DATA BY MULTIPLE COLUMNS:

- It can be useful if your data falls into categories and you'd like to organize rows by date, but keeping all of the results within a given category together. I.e., The following query makes the most recent years come first, but orders top-rank songs before lower-ranked songs.

  -
    ```
    SELECT *
      FROM tutorial.billboard_top_100_year_end
      WHERE year_rank <= 3
     ORDER BY year DESC, year_rank
    ```
- From the previous query we can get first that columns in the order by clause must be separated by commas, second the DESC operator only affects to the columns that precedes and finally, the results are sorted by the first column mentioned (year) and then by the next one (year_rank). Out of the curiosity, this is the same query with the columns position swiped in the clause ORDER BY.

  -
    ```
    SELECT *
      FROM tutorial.billboard_top_100_year_end
     WHERE year_rank <= 3
     ORDER BY year_rank, year DESC
    ```
- Last but not least, you can substitute the column names by number in the clause order by. The numbers will correspond to the order in which you list columns in the Select clause. I.e.

  -
    ```
    SELECT *
      FROM tutorial.billboard_top_100_year_end
     WHERE year_rank <= 3
     ORDER BY 2, 1 DESC
    ```
- This functionality is supported by MODE, but not by every flavour of SQL. So, the documentation of your respective flavour.
- When order by is executed with a row limit (clause LIMIT), the ordering clause is executed first.
- Write a query that returns all rows from 2010 ordered by rank, with artists ordered alphabetically for each song.

  -
    ```
    SELECT *
     FROM tutorial.billboard_top_100_year_end
    WHERE year = 2010
    ORDER BY year_rank, artist
    ```
- Write a query that shows all rows for which T-Pain was a group member, ordered by rank on the charts, from lowest to highest rank (from 100 to 1).

  -
    ```
    Select *
    FROM tutorial.billboard_top_100_year_end
    Where "group" ilike '%T-Pain%'
    Order by year_rank DESC
    ```

Using comments

- **Use – (two dashes) to comment whatever is on the right of them. I.e.**

  -
    ```
    SELECT *  --This comment won't affect the way the code runs
      FROM tutorial.billboard_top_100_year_end
     WHERE year = 2013
    ```
- Use ***/* comments */*** **to generate multiline comments. I.e.**

  -
    ```
    /* Here's a comment so long and descriptive that
    it could only fit on multiple lines. Fortunately,
    it, too, will not affect how this code runs. */
    SELECT *
      FROM tutorial.billboard_top_100_year_end
     WHERE year = 2013
    ```
- Write a query that returns songs that ranked between 10 and 20 (inclusive) in 1993, 2003, or 2013. Order the results by year and rank, and leave a comment on each line of the WHERE clause to indicate what that line does.

  -
    ```
    SELECT *
      FROM tutorial.billboard_top_100_year_end
     WHERE year IN (2013, 2003, 1993)  --Select the relevant
    years
       AND year_rank BETWEEN 10 AND 20  --Limit the rank to 10-
    20
     ORDER BY year, year_rank
    ```