**SQL**

**for Data Analyst / Data Scientist**

**By Gundala Nagaraju (Raju)**

**Base Query:**

**SELECT** \* **FROM** **ACCOUNT**;

**Query returns every column and every row of the table called ACCOUNT table.**

**SELECT** \* **FROM** **ACCOUNT** **LIMIT** 10;

**Query returns every column and first 10 rows from ACCOUNT table.**

**Selecting Specific Columns:**

**SELECT** account\_id, district\_id, frequency **FROM** **ACCOUNT**;

**Query returns every row of account\_id, district\_id, frequency from ACCOUNT table.**

**Data Types in SQL:** In SQL, we have different data types, But the following 7 are the most important.

1. **Integer -** A whole number without a fractional part (E.g. 8, 26, 701).

2. **Decimal -** A number with a fractional part. E.g. 3.14, 3.141592654, 961.1241250

3. **Boolean -** A binary value. It can be either TRUE or FALSE.

4. **Date -** Speaks for itself. You can also choose the format. E.g. 2017-12-31

5. **Time -** You can decide the format of this, as well. E.g. 23:59:59

6. **Timestamp -** The date and the time together. E.g. 2017-12-31 23:59:59:000001

7. **Text -** This is the most general data type. But it can be alphabetical letters only,

or a mix of letters and numbers and any other characters. E.g. hello1, Insurance,

Test123

**Filtering (WHERE Clause):**

**SELECT** \* **FROM** **ACCOUNT WHERE** account\_id = 10;

**"Horizontal filtering.", Query returns every column from ACCOUNT table - but only those rows where the value in account\_id is 10. Obviously, this can be something text, a number (integer or decimal), date or any other data format, too.**

**Advanced Filtering:**

**Comparison operators help you compare two values. (Usually, a value that you define in your query and values that exist in your SQL table.) Mostly, they are mathematical symbols, with a few exceptions:**

|  |  |
| --- | --- |
| **Comparison operator** | **What does it mean?** |
| **=** | **Equal to** |
| **<>** | **Not Equal to** |
| **!=** | **Not Equal to** |
| **<** | **Less Than** |
| **<=** | **Less Than or Equal to** |
| **>** | **Greater Than** |
| **>=** | **Greater Than or Equal to** |
| **LIKE** | **“%string%’ contains string value** |
| ***IN*** | **(‘val1’, ‘val2’, ‘val3’) contains val1, val2, val3** |
| ***NOT IN*** | **(‘val1’, ‘val2’, ‘val3’) eliminates val1, val2, val3** |

**Few Examples:**

**SELECT** \* **FROM** CARD **WHERE** **TYPE** <> 'gold'; **or**

**SELECT** \* **FROM** CARD **WHERE** **TYPE** != 'gold';

**Query returns every column from CARD table, but only those rows where the value in TYPE is NOT Equal to 'gold'.**

**SELECT** \* **FROM** CARD **WHERE** disp\_id <= 100;

**Query returns every column from CARD table, but only those rows where the value in TYPE is Less Than or Equal to 100.**

**SELECT** \* **FROM** CARD **WHERE** **TYPE** **LIKE** '%junior%';

**Query returns every column from CARD table, but only those rows where the value in**

**TYPE contains the 'junior' string.**

**Multiple Conditions:** You can use more than one condition to filter. For that, we have two logical operators: **OR, AND**

**SELECT** \* **FROM** CARD

**WHERE** card\_id **IN** (8,12,24,27)

**AND** **TYPE** **LIKE** '%junior%';

**Query returns every column from CARD table, but only those rows where the value in card\_id is 8, 12, 24, 27 AND the value in TYPE contains the 'junior' string.**

**SELECT** \* **FROM** CARD

**WHERE** card\_id < 100

**OR** **TYPE** **LIKE** '%junior%';

**Query returns every column from CARD table, but only those rows where the value in card\_id is Less Than 100 OR the value in TYPE contains the 'junior' string.**

**BETWEEN Operator**

The BETWEEN operator is a logical operator that allows you to specify whether a value in a range or not.

The BETWEEN operator is often used in the WHERE clause of the SELECT, UPDATE, and DELETE statements.

**SELECT** \*

**FROM** **transaction**

**WHERE** balance **BETWEEN** 5000 **AND** 10000

;

**Query returns every column from transaction table, but only those rows where the value in balance is Less Than 100 OR the value in TYPE contains the 'junior' string.**

Note: The BETWEEN operator returns true if the value of the '**balance**' is greater than or equal to (>=) the value of '**balance**' and less than or equal to (<= ) the value of the '**balance**', otherwise, it returns zero.

The NOT BETWEEN returns true if the value of '**balance**' is less than (<) the value of the '**balance**' or greater than (>)the value of the value of '**balance**', otherwise, it returns 0.

**Proper Formatting:** You can use line breaks and indentations for nicer formatting. It won't have any effect on your output. Be careful and put a semicolon at the end of the query though!

**SELECT** \* **FROM** CARD

**WHERE** card\_id < 100

**AND** **TYPE** **LIKE** '%junior%'

**LIMIT** 20;

**Sorting Values:**

**SELECT** \* **FROM** CARD

**WHERE** card\_id < 100

**order** **BY** disp\_id

**LIMIT** 20;

**Query returns every row and column from CARD Table, where card\_id is Less Than 100 ordered by disp\_id, in ascending order (by default) & LIMIT to 20 rows.**

**SELECT** \* **FROM** CARD

**WHERE** card\_id < 100

**order** **BY** disp\_id **DESC**

**LIMIT** 20;

**Query returns every row and column from CARD Table, where card\_id is Less Than 100 ordered by disp\_id, in descending order & LIMIT to 20 rows.**

**UNIQUE VALUES:**

**SELECT** **DISTINCT** **TYPE** **FROM** CARD

**WHERE** card\_id < 100;

**It returns every unique value from TYPE column from CARD table.**

**Correct Keyword Order:** SQL is extremely sensitive to keyword order. So, make sure you keep it right order

**1. SELECT - columns**

**2. FROM – the table**

**3. WHERE - condition 1 (Between or not between)**

**4. AND, OR – condition 2**

**5. ORDER BY column name**

**6. LIMIT**

**SQL Functions for Aggregation:** In SQL, there are five important aggregate functions for data analysts/scientists

**• COUNT()**

**• SUM()**

**• AVG()**

**• MIN()**

**• MAX()**

**Few Examples:**

**SELECT** **COUNT**(\*) **FROM** **client**

**WHERE** client\_id > 200;

**Query counts the number of rows in the CLIENT table in which the value in client\_id is**

**GREATER THAN 200.**

**SELECT** **SUM**(amount) **FROM** loan

**WHERE** account\_id < 500;

**Query sums the amount column in the LOAN table in which the value in account\_id is**

**LESS THAN 500.**

**SELECT** **COUNT**(\*), **SUM**(amount), **MAX**(amount), **MIN**(amount), **AVG**(amount)

**FROM** loan **WHERE** account\_id < 100;

**SQL GROUP BY:** The GROUP BY clause is usually used with an aggregate function (COUNT, SUM, AVG, MIN, MAX). It groups the rows by a given column value (specified after GROUP BY) then calculates the aggregate for each group and returns that to the screen.

**SQL HAVING**: The execution order of the different SQL keywords doesn't allow you to filter with the WHERE clause on the result of an aggregate function (COUNT, SUM, etc.). This is because WHERE is executed before the aggregate functions. But that's what HAVING is for:

**SELECT** account\_id, **SUM**(amount) **FROM** ordert

**WHERE** account\_id < 20

**GROUP** **BY** account\_id

**HAVING** **COUNT**(\*) > 1;

**SELECT** **COUNT**(\*), **SUM**(amount), **MAX**(amount), **MIN**(amount), **AVG**(amount)

**FROM** ordert

**WHERE** account\_id < 50

**GROUP** **BY** account\_id

**HAVING** **COUNT**(\*) > 1;

**SQL ALIASES:** You can rename columns, tables, subqueries

**SELECT** account\_id, **SUM**(amount) **as** sum\_of\_amount **FROM** ordert

**WHERE** account\_id < 20

**GROUP** **BY** account\_id;

**SQL JOIN:** You can JOIN two (or more) SQL tables based on column values.

**SELECT** a.\* **FROM** **ACCOUNT** a, district b

**WHERE** a.district\_id = b.a1;

**SELECT** a.\*, b.\* **FROM** card a, disp b

**WHERE** a.disp\_id = b.disp\_id

**AND** a.disp\_id < 100;

**Constraints** are the rules enforced on data columns on table. These are used to limit the type of

data that can go into a table.

This ensures the accuracy and reliability of the data in the database. Constraints could be column level or table level.

Column level constraints are applied only to one column, whereas table level constraints are applied to the whole table.

Following are commonly used constraints available in SQL

**NOT NULL Constraint**: Ensures that a column cannot have NULL value.

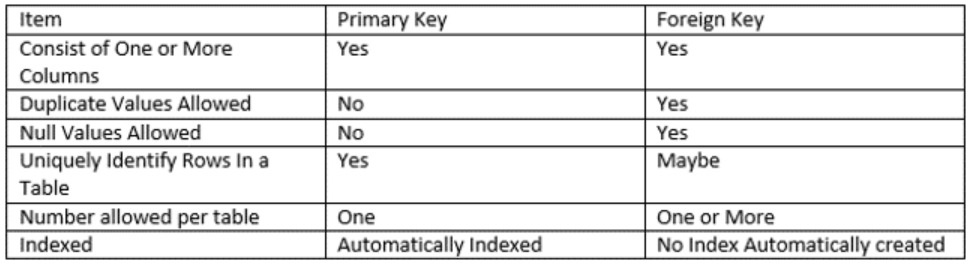
**DEFAULT Constraint**: Provides a default value for a column when none is specified.

**UNIQUE Constraint**: Ensures that all values in a column are different.

**PRIMARY Key**: Uniquely identified each rows/ records in a database table.

**FOREIGN Key**: Uniquely identified a rows/records in any another database table.

**INDEX**: Use to create and retrieve data from the database very quickly.

****

**SUBQUERIES:**You can run SQL queries within SQL queries. (Called subqueries.) Even queries within queries within queries. The point is to use the result of one query as an input value of another query.

**SELECT** \* **FROM** **client** a

**WHERE** a.client\_id = (**SELECT** b.client\_id

**FROM** disp b

**WHERE** b.client\_id = 2);