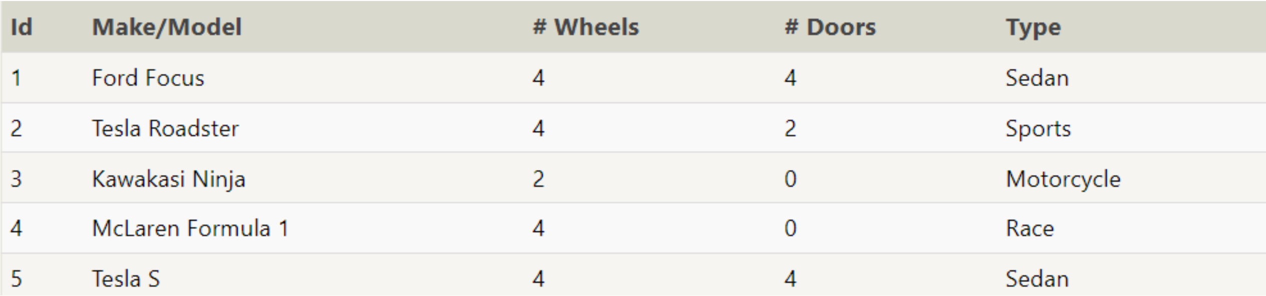
SQL Summary & Revision Guide

~ Rahul Narang

Structured Query Language (SQL) is the standard programming language for managing data in relational databases. With SQL, individuals can retrieve, modify, insert, and delete multiple data records using a single query.

Some of the popular databases: SQLite, MySQL, Postgres, Oracle. As we know that, SQL is used for querying relational databases, so we need to have a model. A relational database represents a collection of related (two-dimensional) tables. Consider each table as a dataframe (in python's language) and query/join those tables which you want to play around with.

Vehicle Table



Id	Make/Model	# Wheels	# Doors	Type
1	Ford Focus	4	4	Sedan
2	Tesla Roadster	4	2	Sports
3	Kawakasi Ninja	2	0	Motorcycle
4	McLaren Formula 1	4	0	Race
5	Tesla S	4	4	Sedan

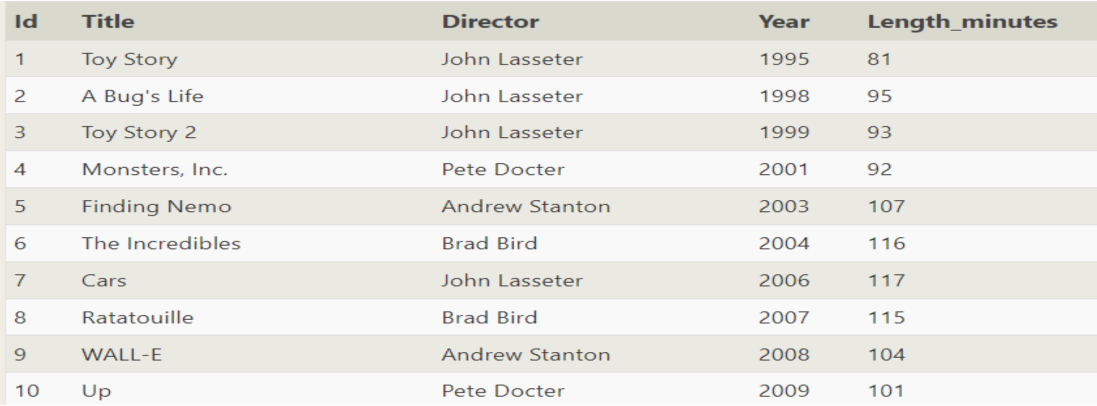
I will be dividing this document into SQL conceptual segments (which I have come across) which is helpful for data analysis purposes. I will be updating this document regularly.

Data Retrieval

To retrieve data from a SQL database, we need to write SELECT statements, which are often referred to as queries/query. Before writing any query, we need to understand the business problem statement and design the logic to obtain the required information.

Most basic query to retrieve columns from tables is SELECT * FROM TABLE_NAME. The result of this query will be a two-dimensional set of rows and columns (if mentioned after SELECT STATEMENT). We can specific column name to after SELECT STATEMENT to retrieve the desired columns.

Table: movies



Id	Title	Director	Year	Length_minutes
1	Toy Story	John Lasseter	1995	81
2	A Bug's Life	John Lasseter	1998	95
3	Toy Story 2	John Lasseter	1999	93
4	Monsters, Inc.	Pete Docter	2001	92
5	Finding Nemo	Andrew Stanton	2003	107
6	The Incredibles	Brad Bird	2004	116
7	Cars	John Lasseter	2006	117
8	Ratatouille	Brad Bird	2007	115
9	WALL-E	Andrew Stanton	2008	104
10	Up	Pete Docter	2009	101

Q. Find Title and Year for each film.

Select Title, Year from movies;

Constraints

Constraints help ensure that data entered into the database meets certain criteria or conditions, thereby maintaining consistency as per the requirements. Below is the basic format and we can add different more and more complex constraints as well.

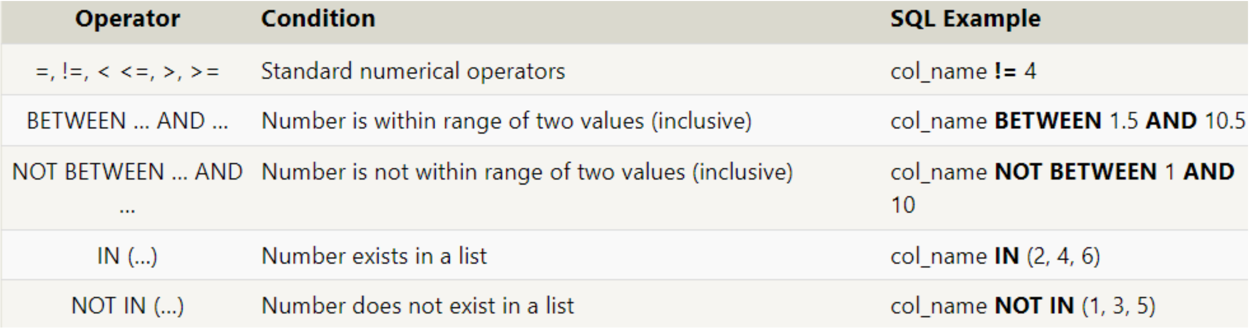
SELECT column1,column2

FROM table1

WHERE condition1

AND/OR condition2

AND/OR …..



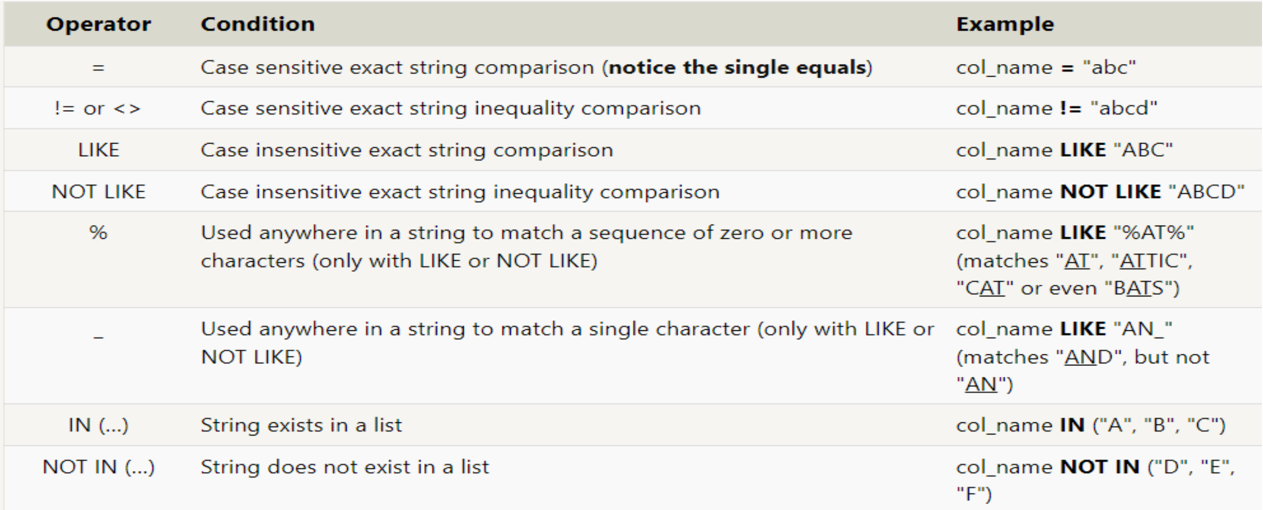
Operator	Condition	SQL Example
=, !=, < <=, >, >=	Standard numerical operators	col_name **!=** 4
BETWEEN … AND …	Number is within range of two values (inclusive)	col_name **BETWEEN** 1.5 **AND** 10.5
NOT BETWEEN … AND …	Number is not within range of two values (inclusive)	col_name **NOT BETWEEN** 1 **AND** 10
IN (…)	Number exists in a list	col_name **IN** (2, 4, 6)
NOT IN (…)	Number does not exist in a list	col_name **NOT IN** (1, 3, 5)

Q. Find the movies not released in the years between 2000 and 2010. Use table movies on page 1.

*SELECT * FROM movies*

where Year not between 2000 and 2010;

Text Specific Operators



Operator	Condition	Example
=	Case sensitive exact string comparison (**notice the single equals**)	col_name **=** "abc"
!= or <>	Case sensitive exact string inequality comparison	col_name **!=** "abcd"
LIKE	Case insensitive exact string comparison	col_name **LIKE** "ABC"
NOT LIKE	Case insensitive exact string inequality comparison	col_name **NOT LIKE** "ABCD"
%	Used anywhere in a string to match a sequence of zero or more characters (only with LIKE or NOT LIKE)	col_name **LIKE** "%AT%" (matches "AT", "ATTIC", "CAT" or even "BATS")
_	Used anywhere in a string to match a single character (only with LIKE or NOT LIKE)	col_name **LIKE** "AN_" (matches "AND", but not "AN")
IN (…)	String exists in a list	col_name **IN** ("A", "B", "C")
NOT IN (…)	String does not exist in a list	col_name **NOT IN** ("D", "E", "F")

Q. Find all the Toy Story movies

```
SELECT Title,director FROM movies
WHERE title like "Toy Story%";
```

Filtering, Sorting, Ordering

In SQL, the DISTINCT keyword is used in a SELECT statement to retrieve unique values from a specified column or combination of columns. It filters out duplicate rows from the result set, returning only distinct values.

```
SELECT DISTINCT column1, column2
FROM table1
WHERE condition(s);
```

SQL provides a way to sort your results by a given column in ascending or descending order using the ORDER BY clause

```
SELECT column1, column2
FROM table
WHERE condition(s)
ORDER BY column ASC/DESC;
```

Limiting results to a subset

The LIMIT will reduce the number of rows to return, and additionally OFFSET can specify where to begin counting the number rows from. These two when used together can be very effective.

```
SELECT column1, column2
FROM table
WHERE condition(s)
ORDER BY column ASC/DESC
LIMIT num_limit OFFSET num_offset;
```

Q. List the next five movies sorted alphabetically from the movies table.

```
SELECT *
FROM movies
ORDER BY Title asc
limit 5 OFFSET 5;
```

Interesting Problem

Q. List all the cities west of Chicago, ordered from west to east

```
Select city from north_american_cities
where Longitude < (
SELECT Longitude FROM north_american_cities
where city='Chicago' )
ORDER BY Longitude
```

Normalization and JOIN

Database normalization is useful because it minimizes duplicate data in any single table and allows for data in the database to grow independently of each other. As a trade-off, queries get slightly more complex since they should be able to find data from different parts of the database, and performance issues can arise when working with many large tables.

There are 1NF, 2NF, 2NF Normalization forms also which although isn’t a part of SQL Querying but it aims at organizing data to minimize redundancy and dependency issues. I believe a Data Architect can play a big role here.

Using the JOIN clause in a query, we can combine row data across two separate tables using this unique key / primary key.

```
SELECT column1, column2
FROM table1
INNER JOIN table2
ON table1.id = table2.id
WHERE condition(s)
ORDER BY columns ASC/DESC
LIMIT num_limit OFFSET num_offset;
```

Types of Joins

INNER JOIN

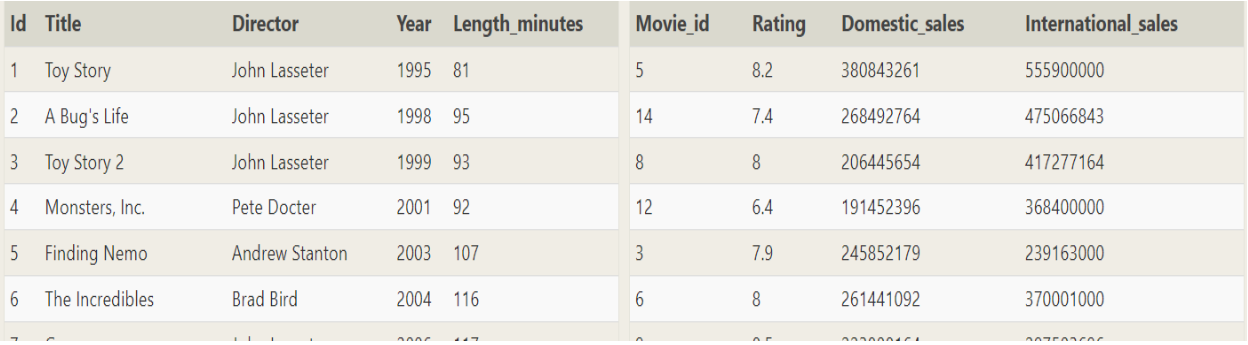
LEFT JOIN (or LEFT OUTER JOIN)

RIGHT JOIN (or RIGHT OUTER JOIN)

FULL JOIN (or FULL OUTER JOIN)

CROSS JOIN (Returns the Cartesian product of the two tables, meaning it returns all possible combinations of rows from both tables.)

Table 1: movies ; Table 2: Boxoffice



Id	Title	Director	Year	Length_minutes
1	Toy Story	John Lasseter	1995	81
2	A Bug's Life	John Lasseter	1998	95
3	Toy Story 2	John Lasseter	1999	93
4	Monsters, Inc.	Pete Docter	2001	92
5	Finding Nemo	Andrew Stanton	2003	107
6	The Incredibles	Brad Bird	2004	116

Movie_id	Rating	Domestic_sales	International_sales
5	8.2	380843261	555900000
14	7.4	268492764	475066843
8	8	206445654	417277164
12	6.4	191452396	368400000
3	7.9	245852179	239163000
6	8	261441092	370001000

Q. Show the sales numbers for each movie that did better internationally rather than domestically.

```
SELECT m.id,m.title,b.Domestic_sales, b.International_sales

FROM movies m

left JOIN Boxoffice b

on m.Id = b.Movie_Id

WHERE b.Domestic_sales< b.International_sales ;
```

OUTER JOINs

LEFT JOIN includes rows from table1 regardless of whether a matching row is found in table2.

RIGHT JOIN keeps the rows from table2 regardless of whether a match is found in table1.

FULL JOIN simply means rows from both tables.

```
SELECT table1.column1, table2.column

FROM table1

INNER/LEFT/RIGHT/FULL JOIN table2

   ON table1.id = table2.matching_id

WHERE condition(s)

ORDER BY column, … ASC/DESC

LIMIT num_limit OFFSET num_offset;
```

Q. Is it good/bad to have NULLS in your SQL database?

Actually, the answer to this question is both good and bad. If you have a database, where you are storing raw data in the (similar to a datalake) it’s is fine to have NULLS in it. But, in a scenario where you have a numerical field and we see most of the fields when queried gives NULL than we can replace those NULLs with 0s and with empty strings for text data.

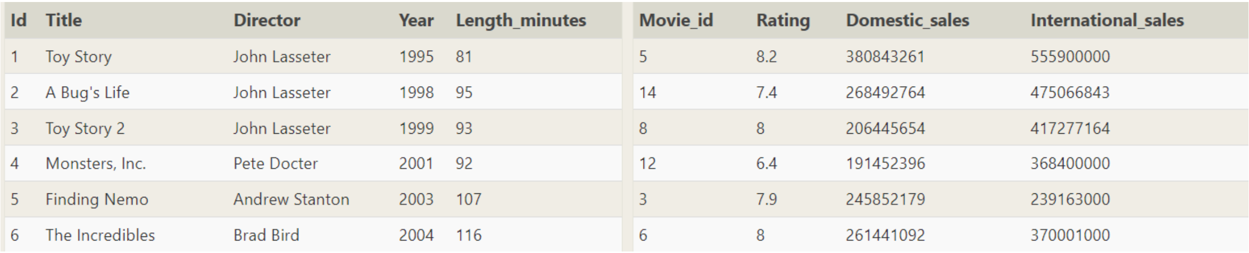
```
SELECT column1, column2, …

FROM table

WHERE column IS/IS NOT NULL

AND/OR condition2

AND/OR …;
```

Queries with expression

These expressions can use mathematical and string functions along with basic arithmetic to transform values when the query is executed. Remember, database has its own supported set of mathematical, string, and date functions that can be used in a query, which you can find on their official websites.

```
SELECT weight / 2 AS half_weight

FROM weight_table

WHERE ABS(BMI) * 5 > 50;
```

Table 1: movies ; Table 2 : boxoffice



Id	Title	Director	Year	Length_minutes
1	Toy Story	John Lasseter	1995	81
2	A Bug's Life	John Lasseter	1998	95
3	Toy Story 2	John Lasseter	1999	93
4	Monsters, Inc.	Pete Docter	2001	92
5	Finding Nemo	Andrew Stanton	2003	107
6	The Incredibles	Brad Bird	2004	116

Movie_id	Rating	Domestic_sales	International_sales
5	8.2	380843261	555900000
14	7.4	268492764	475066843
8	8	206445654	417277164
12	6.4	191452396	368400000
3	7.9	245852179	239163000
6	8	261441092	370001000

Q. List all movies that were released on even number years

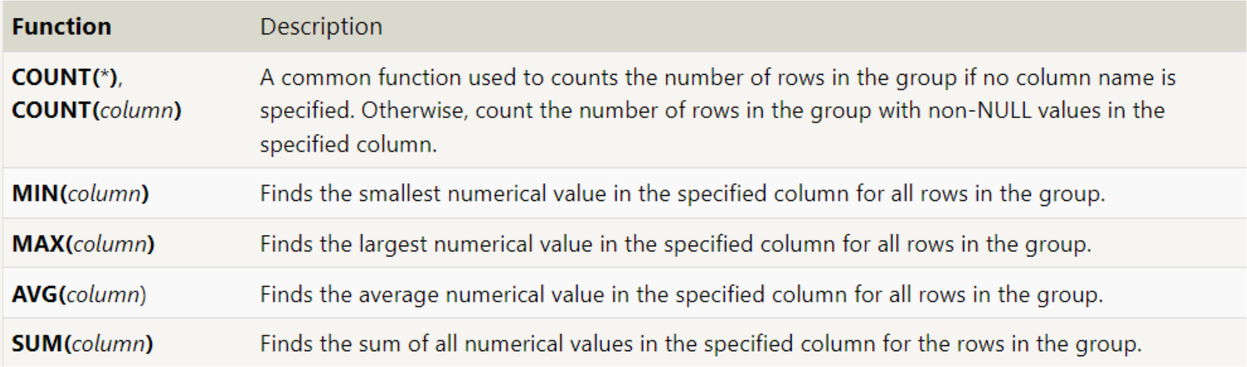
```
SELECT title, year

FROM movies

  JOIN boxoffice

    ON movies.id = boxoffice.movie_id
```

```
where Year%2 ==0;
```

Aggregates

```
SELECT AGG_FUNC(column_or_expression) AS aggregate_description, …
FROM table1
WHERE expression;
```

Here are some common aggregate functions that I have used in my day-to-day analysis and operations.



Function	Description
COUNT(*****), **COUNT(***column***)**	A common function used to counts the number of rows in the group if no column name is specified. Otherwise, count the number of rows in the group with non-NULL values in the specified column.
MIN(*column***)**	Finds the smallest numerical value in the specified column for all rows in the group.
MAX(*column***)**	Finds the largest numerical value in the specified column for all rows in the group.
AVG(*column*)	Finds the average numerical value in the specified column for all rows in the group.
SUM(*column***)**	Finds the sum of all numerical values in the specified column for the rows in the group.

GROUP BY clause works by grouping rows that have the same value in the column specified. Consider it as putting the field inside a pivot rows box in excel.

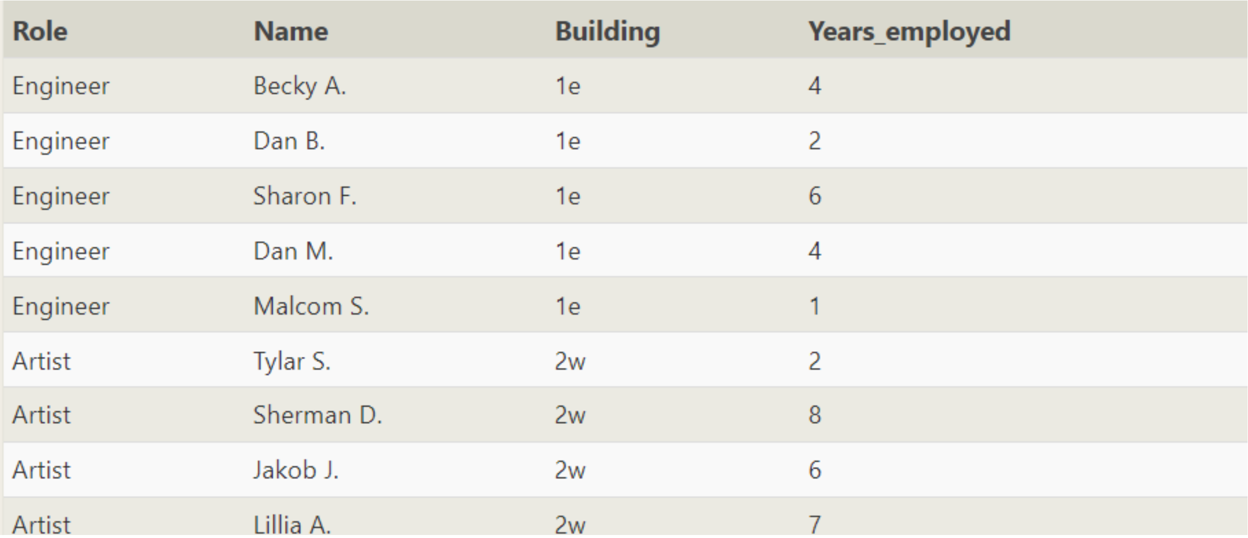
NOTE:

GROUP BY clause is executed after the WHERE clause (which filters the rows which are to be grouped), then how exactly do we filter the grouped rows?

Answer. HAVING. HAVING clause constraints is written the similar fashion as the WHERE clause constraints but is applied to the grouped rows.

```
SELECT group_by_column, AGG_FUNC(column_expression) AS aggregate_result_alias, …
FROM table1
WHERE condition1
GROUP BY column1
HAVING group_condition;
```

table employee



Role	Name	Building	Years_employed
Engineer	Becky A.	1e	4
Engineer	Dan B.	1e	2
Engineer	Sharon F.	1e	6
Engineer	Dan M.	1e	4
Engineer	Malcom S.	1e	1
Artist	Tylar S.	2w	2
Artist	Sherman D.	2w	8
Artist	Jakob J.	2w	6
Artist	Lillia A.	2w	7

Q. Find the total number of years employed each role

```
SELECT Role, sum(Years_employed) as Years

FROM employees

group by Role

;
```

<u>MOST IMPORTANT CONCEPT OF SQL</u>

Order of execution of a Query

1. FROM and JOINs
2. WHERE
3. GROUP BY
4. HAVING
5. SELECT
6. DISTINCT
7. ORDER BY
8. LIMIT / OFFSET

Q. Find the total domestic and international sales that can be attributed to each director.

```
SELECT director, sum(Domestic_sales+International_sales) as sales
FROM movies
   INNER JOIN boxoffice
       ON movies.id = boxoffice.movie_id
GROUP BY director;
```

NOTE: The information presented in this document is also present across internet.