SQL CHEAT-SHEET

1 DATABASE

```
CREATE DATABASE database_name;
--Creates a new database
```

```
USE database_name;
--Uses the specified database
```

2 SQL COMMANDS

• DATABASE

```
CREATE TABLE table_name
( column1 datatype,
column2 datatype,
column3 datatype);
--The create table statement creates a new table in a database
```

```
ALTER TABLE table_name
ADD column_name datatype;
--The Alter table statement is used to modify the columns of an existing table and add a new column.
```

```
ALTER TABLE table_name
ADROP COLUMN column_name;
--The Alter table statement is used to modify the columns of an existing table and Drop column.
```

```
ALTER TABLE table_name
RENAME col_name TO col_newname;
--Renames the column names in the existing table.
```

```
DROP TABLE table_name;
--Drop deletes both structure and records in the table.
```

```
TRUNCATE TABLE table_name;
--Truncate deletes the table but not the structure
```

• DDL COMMANDS

```
INSERT INTO table_name
VALUES (value1, value2);
```

```
UPDATE table_name
SET column1 = value1,column2 = value2
WHERE some_column =some_value;
--The UPDATE statement is used to edit records (rows) in a table.
```

```
DELETE FROM table_name
WHERE some_column = some_value;
--The delete statement is used to delete records (rows) in a table.
```

• DML COMMANDS

```
GRANT SELECT,UPDATE on table_1
TO user_1, user_2;
--Used to grant a user access privileges to a database.
```

```
GRANT SELECT,UPDATE on table_1
TO user_1, user_2;
--Used to grant a user access privileges to a database.
```

• DCL COMMANDS

COMMIT	Saves all the transactions made on a database.
ROLLBACK	It is used to undo transactions which are not yet been saved.

• TCL COMMANDS

```
SELECT col1,col2.. FROM table_name;
--Retrieve data from specified columns in the table
```

```
SELECT * FROM table_name;
--Retrieve the data from all fields in the table.
```

```
SELECT col1,col2..FROM table_name
WHERE condition;
--Used to filter the records based on a particular condition.
```

7 SQL CONSTRAINTS

NOT NULL	Specifies that this column cannot store a NULL value.
UNIQUE	Specifies that this column can have only UNIQUE values
PRIMARY KEY	It is a field using which it is possible to uniquely identify each row in a table.
FOREIGN KEY	It is a field using which it is possible to uniquely identify each row in some other table.
CHECK	It validates if all values in a column satisfy some particular condition or not
DEFAULT	It specifies a default value for a column when no value is specified for that field

8 OPERATORS

AND	The NOT operator allows the negotiation of the condition.
OR	The OR operator allows multiple conditions to be combined. Records match either condition.
NOT	The AND operator allows multiple conditions to be combined. Records must match both conditions
ALL	It compares a value to all the values in another set.
ANY	It compares the values in the list according to the condition.
BETWEEN	The BETWEEN operator can be used to filter by a range of values.
LIKE	The LIKE operator can be used inside of a WHERE clause to match a specified pattern.
% WILDCARD	The % wildcard can be used in a LIKE operator pattern to match zero or more unspecified character(s).
_ WILDCARD	The _ wildcard can be used in a LIKE operator pattern to match any single unspecified character.
IN	The IN operator is used to compare the specified value. AS - Columns or tables can be aliased using the AS clause.
EXIST	It is used to search for the presence of a row in a table.

• WHERE

```
SELECT column_name FROM table_name
WHERE column_name IS NULL;
--Column values can be NULL or have no value. These records can be matched using the IS NULL and IS NOT NULL operators.
```

• UNION

```
SELECT col1, col2 FROM table_name
UNION
SELECT col1, col2 FROM table_name;
--Combine rows from two queries without any duplicates.
```

• UNION ALL

```
SELECT col1, col2 FROM table_name
UNION ALL
SELECT col1, col2 FROM table_name;
--Combine rows from two queries with duplicates
```

• INTERSECT

```
SELECT col1, col2 FROM table_name
INTERSECT
SELECT col1, col2 FROM table_name;
--Return the common rows of two queries.
```

• UNION

```
SELECT col1, col2 FROM table_name
MINUS
SELECT col1, col2 FROM table_name;
--Returns the values from the first table after removing the values from the second table.
```

9 QUERYING DATA

```
SELECT DISTINCT(column_name)
FROM table_name;
--Unique values of the columns are retrieved from the table.
```

```
SELECT * FROM table_name LIMIT 5;
--Limit is used to limit the result set to the specified number of rows.
```

```
SELECT col1, col2 FROM table_name
ORDER BY col1 ASC [DESC];
--Sort the result set in ascending or descending order
```

```
SELECT col1, col2 FROM table_name
ORDER BY col1 LIMIT n OFFSET offset;
--Skip offset of rows and return the next n rows based on LIMIT.
```

```
SELECT col1, aggregate(col2)
FROM table_name GROUP BY col1;
--GROUP BY Groups rows using an aggregate function
```

```
SELECT col1, aggregate(col2)
FROM table_name
GROUP BY col1 HAVING condition;
--Filter groups using the HAVING clause.
```

```
DESC table_name;

--Describes the structure of the table.
```

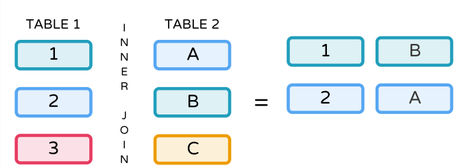
10 CASE

```
SELECT column_name,
CASE
WHEN condition THEN 'output'
WHEN condition THEN 'output'
.
.
.
ELSE 'output'
END 'new_colname' FROM table_name;
```

11 JOINS

• INNER JOIN

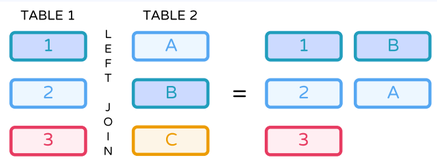
```
SELECT col1, col2
FROM table_name t1
INNER JOIN table_name t2
ON condition;
```



Only returns rows that meet the join condition

• LEFT JOIN

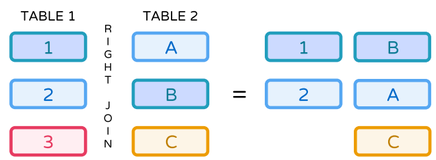
```
SELECT col1, col2
FROM table_name t1
LEFT JOIN table_name t2
ON condition;
```



Returns all rows from the table on the left side of the join and matched rows from the right side of the join

• RIGHT JOIN

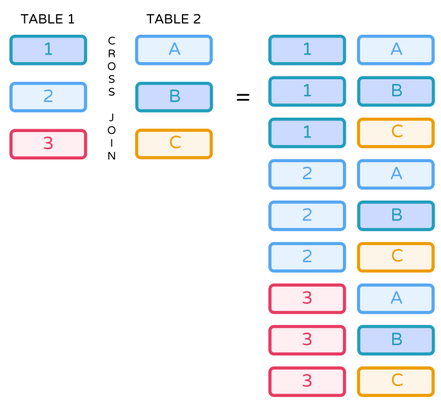
```
SELECT col1, col2
FROM table_name t1
RIGHT JOIN table_name t2
ON condition;
```



Returns all rows from the table on the right side of the join and matched rows from the left side of the join

• CROSS JOIN

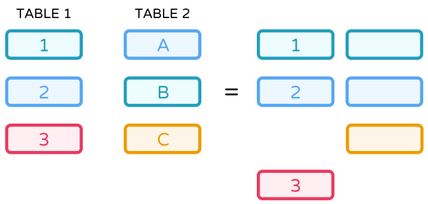
```
SELECT col1, col2
FROM table_name t1
CROSS JOIN table_name t2
ON condition;
```



Produce A Cartesian Product Of Rows In Tables

• FULL JOIN

```
SELECT col1, col2
FROM table_name t1
FULL JOIN table_name t2
ON condition;
```



Returns all rows from bith sides even if join condition is not met

12 AGGREGATE FUNCTIONS

AVG()	returns the average of a list
SUM()	returns the total of a list.
COUNT()	returns the number of elements of a list.
MIN()	returns the minimum value of a list.
MAX()	returns the maximum value of a list.

13 ADVANCED AGGREGATE FUNCTIONS

OVER()	It is a window function used inside every analytical function
PARTITION BY ()	Creates a partition internally and later performs the specified operations.
ROW_NUMBER ()	Provides row numbers for all the rows based on a specified column in the table.
RANK()	Ranking is assigned to the rows based on a specified column. Skips the rank when it contains the same values.
DENSE_ RANK()	Ranking is assigned to the rows based on a specified column. Ranks are not skipped.
PERCENT_ RANK()	Assigns the rank to the specified column within the range of 0-1.
LAG()	The first value becomes NULL. Compares the current value with the previous value.
LEAD()	The last value becomes NULL. Compares the current value with the next value.
FIRST_ VALUE()	Gives the first value to all rows.
LAST_ VALUE()	Gives the last value to all rows.
last_ value()	Gives the last value to all rows.
Nth value()	Gives Nth value to all rows.
NTILE()	Divides the rows to 'n' number of small buckets.
cume_ dist()	The cumulative percentage of the records is calculated from the first row to the current row for the specified column.

14 VIEWS

```
SELECT VIEW view_name
AS SELECT * FROM table_name;
```

*--It creates a **simple** view.*

```
SELECT VIEW view_name
AS SELECT col1, col2
FROM table_name t1
INNER JOIN table_name t2
ON condition;
```

*--It creates a **complex** view*

```
CREATE TEMPORARY VIEW view_name
AS SELECT col1, col2
FROM table_name;
```

*--It creates a **temporary** view.*

```
DROP VIEW view_name;
```

*--**Delete** a view.*

15 SQL TRIGGERS

All query elements are processed in a very strict order: Query execution order.

FROM	the database gets the data from tables in FROM clause and if necessary performs the JOINs,
WHERE	the data are filtered with conditions specified in the WHERE clause,
GROUP BY	the data are grouped by conditions specified in the WHERE clause,
AGGREGATE FUNCTIONS	the aggregate functions are applied to the groups created in the GROUP BY phase,
HAVING	the groups are filtered with the given condition,
WINDOW FUNCTIONS	Ranking is assigned to the rows based on a specified column. Ranks are not skipped.
SELECT	the database selects the given columns,
DISTINCT	repeated values are removed,
UNION/ INTERSECT/ EXCEPT	the database applies set operations,
ORDER BY	the results are sorted,
OFFSET	the first rows are skipped,
LIMIT/ FETCH/TOP	only the first rows are selected

16 ADVANCE SQL

• COUNT ( )

Function with the **DISTINCT** clause eliminates the repetitive appearance of the same data. The **DISTINCT** can come only once in a given select statement.

• COUNT (*)

When * is used as an argument, it simply **counts** the total number of rows **including the NULLs**.

• COUNT (1)

With **COUNT(1)**, there is a misconception that it counts records from the first column. What **COUNT(1)** really does is that it replaces all the records you get from query results with the value 1 and then counts the rows meaning it even replaces a **NULL with 1** meaning it takes NULLs into consideration while counting.

• COUNT(COLUMN_NAME)-

When a column name is used as an argument, it simply counts the total number of rows excluding the NULLs meaning it will not take NULLs into consideration.

COUNT(DISTINCT EXPR,[EXPR...])

APPROX_COUNT_DISTINCT
is useful if an exact result is not required.

• compared to **SELECT COUNT(DISTINCT<COLUMN>)**, which calculates the exact number of distinct values in a column of the table, **APPROX_COUNT_DISTINCT** can run much faster and consume significantly less memory.

• the implementation of **APPROX_COUNT_DISTINCT()** has a much smaller memory requirement as compared to the **COUNT(DISTINCT)** function

• LOWER():

The **LOWER()** function converts a string to **lowercase**.

• UPPER():

The **UPPER()** function converts a string to **uppercase**.

• REGEXP_CONTAINS():

• Returns **TRUE** if the value is a **partial match** for the regular expression, regex.

• If the regex argument is invalid, the function returns an error.

• You can search for a full match by using ^ **(beginning of text)** and **$ (end of text)**.

Syntax: **REGEXP_CONTAINS**(value, regex)

• REGEXP_EXTRACT():

• Returns the first substring in value that matches the regular expression, regex. Returns NULL if there is no match.

• If the regular expression contains a capturing group, the function returns the substring that is matched by that capturing group.

• If the expression does not contain a capturing group, the function returns the entire matching substring.

• Returns an **error if**:
--the regular expression is **invalid**
--the regular expression has **more than one** capturing group

Syntax: **regexp_extract**(value, regex)

17 SQL INDEXES

• A SQL index is a quick lookup table used to **quickly retrieve** data from a database.

• Indexes are generally used for **large databases**. They are small, fast, and less memory-consuming.

• Indexing a table or view surely improves the performance of queries and applications.

CREATING A SEARCH INDEX:

Indexes are created for one or more columns in a table, using the **CREATE INDEX** command.

Syntax: **CREATE SEARCH INDEX** index_name **ON** `dataset.table_name` (column_name);

DROPPING A SEARCH INDEX:

An index can be dropped using SQL DROP command.

Syntax: **DROP SEARCH INDEX** index_name **ON** `dataset.table_name`;

18 SQL PARTITIONING

Partitioning is a database process where very large tables are divided into multiple smaller, individual parts.

• By splitting a large table, queries that access only limited amount of data can run faster because there is less data to scan.

• You can partition BigQuery tables by:

-time-unit column: tables are partitioned based on a timestamp, date, or datetime column in the table.

-ingestion time: tables are partitioned based on the timestamp when bigquery ingests the data.

-integer range: tables are partitioned based on an integer column.

use the **CREATE TABLE** statement with a **SELECT AS** clause for the query. include a **PARTITION BY** clause to configure the partitioning.

• COPYING INDIVIDUAL PARTITIONS:

• You can copy the data from one or more partitions to another table.

• Copying partitions is not supported by Console.

• However you can copy them using the API

• DELETING A PARTITION

• You can delete an individual partition from a partitioned table.

• But you can't delete the special NULL or UN-PARTITIONED partitions.

• You can only delete one partition at a time