**SQL NOTES**

**1. MANIPULATION**

**2. QUERIES**

**3. OPERATORS**

**4. AGGREGATE FUNCTIONS**

**5. WINDOW FUNCTIONS**

**6. MULTIPLE TABLES**

**7. STRING FUNCTIONS**

**8. DATE AND TIME FUNCTIONS**

**9. DATA COMMANDS**

**COMMANDS / SYNTAX:**

**1. ------------MANIPULATION----------**

**\* COLUMN CONSTRAINTS:**

**-PRIMARY KEY** constraint can be used to uniquely identify the row.  
 (There can only be one PRIMARY KEY column per table & Cannot be nul)

**-UNIQUE** columns have a different value for every row.  
 (Can have multiple unique columns.)

**-NOT NULL** columns must have a value.

**-DEFAULT** assigns a default value for the column when no value is specified

**\* CREATE TABLE -** statement creates a new table in a database.  
 **CREATE TABLE** table name (column 1 data type, column 2 data type.....)

**EXAMPLE:**  
 **CREATE TABLE** table name (  
 id **INTEGER PRIMARY KEY**,  
 name **TEXT UNIQUE**,  
 grade **INTEGER NOT NULL**  
 age **INTEGER DEFAULT** 10  
 );

**\* INSERT INTO -** adds a new row to a table.  
 ***--- Insert into columns in order ---***  
 **INSERT INTO** table\_name  
 **VALUES** (value\_1, value\_2)

***---Insert into clomuns by name ---***  
 **INSERT INTO** table\_name (column1, column2)  
 **VALUES** (value1, value2)

**\* ALTER TABLE -** To modify the columns of an existing table.  
 **ALTER TABLE** table\_name  
 **ADD** column\_name data\_type;  
 **CHANGE COLUMN** old\_column\_name new\_column\_name data\_type;  
 **- ADD COLUMN -** Used to add a new column.  
 **- CHANGE COLUMN** - Used to rename a column.

**\*DELETE -** Used to delete rows in a table. (IF not using WHERE all records are deleted)  
 **DELETE FROM** table\_name  
 **WHERE** column = vlaue

**\* UPDATE -** used to edit rows in a table with SET to indicate the column and new value  
 **UPDATE** tabel\_name  
 **SET** column1 = value1, column 2 = value2 OR column1 = value - 1  
 **WHERE** column = value

**---------------------------------------------------------------------------------------------------------------------------**

**2. ------------QUERIES------------**

**\* SELECT-** Selects columns (use \* for all columns)(can select multiple(SELECT name, year))

**\* LIMIT -** max number of rows to be displayed

**\* AS -** Renames a column or table (used normally before FROM)

**\* DISTINCT**: - Filters out values.   
 SELECT DISTINCT (value)  
 FROM (table)

**\* CONDITIONAL with WHERE clause:**  
 SELECT \*FROM (table)  
 WHERE (colum)( conditional) (ex:WHERE numbers < 5;)

**\* WILD CARDS:**  
 - **\_ Wild Card** - Matches any single unspecified character  
 SELECT name FROM movies  
 WHERE name LIKE '\_ove';   
 --(Matches anything with a single character followed by 'ove')

**- % Wild Card** - Matches 0 or more of any unspecified number of charcters  
 SELECT name FROM movies  
 WHERE name LIKE 'The%';  
 --(Matches any movie that begins with 'the' followed by 0 or more characters)

**\* ORDER BY:** - Used to sort a colum alphabetically or numerically by ASC or DESC  
 SELECT \* FROM (table)  
 ORDER BY year DESC;  
 --(Orders the year by descending order (the default is ascending)(always after WHERE if present)

**\* CASE -** Creates different outputs (SQL if-then) usually in SELECT statement (ends with end) SELECT (column), (use ' , ' for CASE)  
 CASE  
 WHEN (condition) THEN (output)  
 WHEN (condition) THEN (output)  
 ELSE (output)  
 END (use END AS (new column name) to shorten column name)  
 FROM (table)

**\* HAVING -** SImilar to WHERE but used with aggregate functions. (comes after GROUP BY, but before ORDER BY and LIMIT)  
 SELECT COUNT(solumn name) FROM (table)  
 HAVING COUNT(column name) (condition);

**\*\*\*EXAMPLE OF MULTIPLE COMMANDS QUERY\*\*\***

**SELECT** name, year, imdb\_rating,  
 **CASE**  
 **WHEN** imdb\_rating > 7 **THEN** 'Great'  
 **WHEN** imdb\_rating > 5 **THEN** 'Alright'  
 **WHEN** imdb\_rating **IS NULL THEN** 'null'  
 **ELSE** 'Not good'  
 **END AS** 'Score'  
 **FROM** movies  
 **WHERE** year < 2010  
 **ORDER BY** imdb\_rating **DESC**  
 **LIMIT** 10;

**---------------------------------------------------------------------------------------------------------------------------**

**3. ------------OPERATORS------------**

**\* IS NULL -** Unknown values/missing values

**\* IS NOT NULL -** Checks for values not NULL

**\* AND** - Displays rows if ALL conditions are true

**\* OR -** Displays rows if ANY condition is true

**\* LIKE Operator:** - Used with WHERE to match specific pattern  
 SELECT names FROM (table)  
 WHERE name LIKE 'star%';  
 --(Matches names from table that begin with 'star'. Can also be used like '%star%.)

**\* BETWEEN: -** Filters results with a certain range  
 SELECT \* FROM (table)  
 WHERE number BETWEEN 1 AND 10  
 --(Filters numbers/ range from 1 up to and including 10)

**--------------------------------------------------------------------------------------------------------------------------**

**4. ------------AGGREGATE FUNCTIONS------------**

**\*AGGREGATE FUNCTIONS -** perform a calculation on a set of values and return a single value

**\* COUNT( ) -** Count the number of rows (non empty values)  
 COUNT(\*) - Counts every row (includes null)  
 COUNT((column name)) - Counts rows in the column (doesnt include null)

**\* SUM(** **) -** Sum of values in that column  
 SELECT SUM(column name) FROM (table);

**\* MAX(** **) -** Largest value in that column  
 SELECT MAX(column name) FROM (table);

**\* MIN(** **) -** Smallest value in a column  
 SELECT MIN(column name) FROM (table name);

**\* AVG( ) -** Calculate average of values in a column  
 SELECT AVG(column name) FROM (table name);

**\* ROUND( ) -** Rounds values in a column to number of decimal places specified by integer (takes 2 arguments column name, and an integer)  
 SELECT name, ROUND(price, integer) FROM (table);   
 (returns names and the rounded price)

**\*GROUP BY( ) -** Arrange identical data into groups Used with aggregate functions in collaboration with SELECT. (comes after any WHERE but before ORDER BY or LIMIT)  
 **SELECT** column\_name **FROM** table\_name  
 **GROUP BY** column\_name  
 **ORDER BY** column\_name;

**\* YEAR( )-** Filter by year from a **DATE** column.  
 **WHERE YEAR (**date\_column\_name**) >** 1990;  
 (Shows rows with the year over 1990)

**---------------------------------------------------------------------------------------------------------------------------**

**5. ------------ WINDOW FUNCTIONS -------------**

**\* WINDOW FUNCTIONS -** perform calculations across a set of table rows related to the current row. (can use window and aggregate functions)

**\* OVER( ) -** Defines the window of rows for the function to operate on, specifying **partitioning** and **ordering.** SQL processes each row within this window according to the specified function (like SUM, AVG, etc.) **SUM(**column**) OVER** (**PARTITION BY, ORDER BY**) can also use just **PARTITION** BY or **ORDER** BY.

**\* PARTITION -** dividing the result set into smaller subsets, or "**partitions**," based on the values of one or more columns. Each **partition** is then processed independently by the **window function**.  
 -**SUM**(available\_copies) **OVER** (**PARTITION BY** category): The **SUM**() function calculates  
 the total number of available copies within each **partition** (i.e., for each category).

**\* ROW NUMBER( ) -** assign a unique row number to each row, ordered by the title column.  
 -**SELECT** column1 **ROW NUMBER () OVER** (**ORDER BY** column1) **AS** 'New\_column\_name'   
 **FROM** table\_name;

**\* RANK( ) -** Assigns a rank to each row within a partition of a result set. Rows with equal values receive the same rank, and the next rank is incremented by the number of tied rows**.**  
 - **RANK**() **OVER** (**ORDER BY** publication\_date): This assigns a rank to each book based on the publication\_date. Books with the same publication date receive the same rank, and the next rank is incremented by the number of tied rows.

**\* DENSE RANK( ) -** Similar to **RANK**(), but the next rank is incremented by 1 regardless of the number of tied rows.

**\* NTILE(n)-** helps you distribute rows into a specified number of groups, making it easier to analyze data in segments. (used like percentiles)  
 **(n) -** number of segments

**\* LEAD( ) -** provides access to a row at a specified physical offset following the current row within the result set (take this table, move forward by this much, use this default (if provided)  
 - **LEAD**(column, offset, default)  
 **column**: The column from which to retrieve the value.  
 **offset**: The number of rows forward from the current row (default is 1).  
 **default**: The value to return if the offset goes beyond the result set (optional).  
 - **SELECT** column1, column2,   
 **LEAD**(column2, 1) OVER (ORDER BY column1) AS next\_column2  
 **FROM** table1;

**\* LAG( ) -** Opposite of **LEAD**. **LAG** looks back and **LEAD** looks forward.

**\* WINDOW FUNCTION EXAMPLES:**  
 **MIN**(publication\_date) **OVER** (**PARTITION BY** category **ORDER BY** publication\_date) \*This calculates the minimum publication date within each category.

---------------------------------------------------------------------------------------------------------------------------

**6. -------------- MULTIPLE TABLES---------------**

**\* USING TABLES AND COLUMN NAMES-** table\_name.column\_name

**\* JOIN -** Joins tables with another table (will only return results matching the condition specified by ON). (columns from tables must match)  
 -**INNER JOIN** is the default JOIN

**- LEFT JOIN** keeps all rows from the first table even if there is no matching rows.  
 (Will omit the unmatched row from the second table)

**- CROSS JOIN** used to combine each row form one table with each row from another  
 table. (helpful for creating all possible combinations for the rows in two tables)

**\* ON -** How to combine the tables (what to match the column information with)

**SELECT \* FROM** (table name)  
 **JOIN** (other table name)  
 **ON** table name.column name = table name.column name;

**SELECT** table name.column name, other\_tablename.columnname  
 **FROM** (table name)  
 **JOIN** (other table name)  
 **ON** table.column = other\_table.column

**\*UNION -** Combins results from multiple SELECT and filters duplicates  
 **SELECT** column\_name (Tables must have same number of columns)  
 **FROM** table\_name (Columns must have the same data types in the same  
 **UNION** order as the first table)  
 **SELECT** column\_name  
 **FROM** table\_name;

**\* FOREIGN KEY -** PRIMARY KEY for one table appears in a differnet table.

**\* WITH -** Stores the result of a query in a temporary table using a nickname.  
 **WITH** (new\_temp\_name) **AS** (  
 **SELECT** \* **FROM** (table\_name)

---------------------------------------------------------------------------------------------------------------------------

**7. ------------STRING FUNCTIONS------------**

**\* CONCAT -** Used to connect or join multiple strings to a single string.  
 **SELECT CONCAT(**'(', col1, ', ''', col2, ''', ''', 'name', ''')'**)**  
 **FROM** your\_table;  
 OUTPUT:  
 (123, 'example', 'name')

---------------------------------------------------------------------------------------------------------------------------

**8. ------------DATE ANDTIME FUNCTIONS -------------**

**\* CURDATE( ) -** Returns the current date

**\* DATE\_SUB(date, INTERAL) -** Subtracts specified time interval from a date

**\* DATE\_ADD(date, INTERVAL) -** Adds specific time interval to a date

**\* NOW( ) -** Returns current date & time

**\* YEAR( date ) -** Extracts the year from a date

**\* MONTH ( date ) -** Extracts month from a date

**\*DAY( date ) -** Extracts the day of the month from date

**\* DATEDIFF( date1, date2 ) -** Returns the difference in days between 2 dates

---------------------------------------------------------------------------------------------------------------

**9. --------------DATA COMMANDS-------------**

**INTO OUTFILE -** Output of select will be written to a file  
 CSV EXAMPLE:  
 **SELECT** \* **FROM** table\_name  
 **INTO OUTFILE** 'C:\path\_to\_file.csv'  
 **FIELDS TERMINATED BY** ',' -- fields(columns) seperated by comma  
 **ENCLOSED BY** '"' -- each field enclosed with double quotes  
 **LINES TERMINATED BY** 'n\' -- each row terminated by new line