SQL (Structured Language Query) Notes

# Introduction

* Database usually comprises of 4 categories
  + Tables: store data
  + Views: shows specific data
  + Stored Procedures: tasks
  + Functions: returning values from calculations
* Primary and Foreign keys (Relational database): Primary keys are unique ids in one table while foreign keys in other tables point to primary keys
* ; needed at the end of lines
* -- or # to comment, /\* \*/ for multi-line
* SQL is not case sensitive

# Data Types

* INT, BIGINT, FLOAT, DOUBLE, BOOLEAN
* DECIMAL(10, 2) → Max 10 digits with 2 after the decimal
* VARCHAR(n) → max n chars in string, CHAR(n) → fixed length n chars
* TEXT → Used for long text, can’t be indexed
* DATE → YYYY-MM-DD, DATETIME → YYYY-MM-DD HH:MM:SS, TIME → HH:MM:SS

# SELECT Statement

* *USE database\_name* → Selects the database
* *SELECT (\*, column\_name) FROM table\_name* → \* selects all columns but you can specify which column to extract from a specified table
* *WHERE* → specify a value in a column
* *ORDER BY* → order by a column

# SELECT Clause

* *SELECT column\_1, column\_2* → select multiple columns
* Arithmetic operators → + - \* / %
* *AS* → give an alias to a column name (space requires string ‘ ’)
* *SELECT DISTINCT* → removes duplicates

# WHERE Clause

* *WHERE* → Used after selecting columns for conditionals
* Comparison operators: = > >= < <= (!= <> Same things)
* Standard data format → ‘YYYY-MM-DD’ → Can perform math operations
* BETWEEN n AND m

# AND, OR, NOT Operators

* *AND* → Satisfy multiple conditions usually in a where clause, has priority over *OR*
* *OR* → Satisfy at least one condition
* *NOT* → Negate a condition
  + *NOT* reverses all conditions in parentheses (*OR* becomes *AND*)

# LIKE Statement

* *LIKE* → Finds matches based on a pattern
* % any number of characters, \_ specified number of characters
  + str% → Starts with
  + %str → Ends with
  + %str% → Contains an
  + \_\_\_\_ → 4 characters
  + M\_\_\_ → M then 3 characters
  + M\_\_\_% → M then at least 3 characters

# IN and IS Operator

* *IN (LIST)* → Looks for values in a list
* *IS* → Equivalent to =, except it must be used for NULL

# GROUP BY

* *GROUP BY* → Groups rows that have the same value in one or more columns to perform aggregate functions. If you group by more than one column, each combination becomes a unique group
  + Aggregation Functions
    - COUNT(), SUM()
    - AVG(), MAX(), MIN(), STDDEV(), VARIANCE()

# ORDER BY

* *ORDER BY column ASC/DESC* → Orders by a column in ascending or descending order
  + Can order by multiple columns, but whatever is written first is used to order first

# HAVING

* *HAVING* → Used after *GROUP BY* to filter grouping conditions
  + *WHERE* filters before grouping, *HAVING* filters after aggregation
  + Only used on aggregation functions

# LIMITS and Aliasing

* *LIMIT #*→ Limits the number of rows in output
* *LIMIT #, #* → Start at one number and go the other number of rows
* *SELECT column AS new\_column* → Give alias to a column

# JOINS

* Inner Joins: Return rows that have the same specified elements in both tables
  + *FROM table\_1 INNER JOIN table\_2 ON column\_1 = column\_2*
* Outer Joins:
  + Left Join: Returns all rows from the left table, and matching rows from the right. If there’s no match in the right table, it fills in NULL. The left table is whatever is declared after FROM
    - *FROM table\_1 LEFT OUTER JOIN table\_2 ON …*
  + Right Join: Returns all rows from the right table, and matching rows from the left. If there’s no match in the left table, it fills in NULL (Useless use *LEFT*)
    - *FROM table\_1 RIGHT OUTER JOIN table\_2 ON …*
* Self Joins: Joining a table to itself
  + *FROM table\_1 JOIN table\_1 ON … (Aliases are required due to ambiguity)*
* Joining more than 2 tables together: Use multiple *JOIN* statements

# UNIONS

* Combining rows together
* *UNION* between 2 select statements to combine the rows
  + It is by default distinct so all duplicates will be removed
  + *UNION ALL* to include the duplicates

# STRING Functions

* Built in functions to analyze text
  + *LENGTH()* -> gives length
  + *UPPER(), LOWER()* -> upper/lowercase
  + *CONCAT()* -> Combines strings
  + *TRIM(), LTRIM(), RTRIM()* -> Remove whitespaces
  + *SUBSTRING(*string, start, n*)* -> Extract a substring
  + *LEFT()/ RIGHT(*string, n*)* -> Get n characters from left of right
  + *REVERSE()* -> Reverse the string
  + *INSTR()* -> Position of substring
  + *REPLACE(*string, n, m*)* -> Replace all n with m in a string
  + *LOCATE*(string, n) -> Position of n in string

# CASE Statements

* Used after a SELECT statement, similar to an if/elif/else statement
  + *CASE*

*WHEN … THEN …*

*WHEN … THEN …*

*ELSE*

*END AS* column\_name

# Subqueries

* A query inside another query. We filter the result of another query
* Placed in (), used in a WHERE, SELECT, or FROM clause

# Window Functions

* Do calculations across a window of rows. It’s like a GROUP BY but it doesn’t collapse the rows
* *OVER()* applies a function to a specific number of windows of rows
* *PARTITION BY* splits the table into groups based on a column
* Rolling/Cumulative Total: *SUM() OVER(PARTITION BY … ORDER BY)* will produce a rolling total
* *ROW\_NUMBER() OVER(PARTITION BY … ORDER BY)* produces row numbers. A partition can be used to further divide how we give out row numbers
* *RANK() OVER(PARTITION BY … ORDER BY)* → Rank will assign a position number within a partition when ordered. Same as *ROW\_NUMBER()* but there can be ties
* *DENSE\_RANK() OVER(PARTITION BY … ORDER BY)* →A tie will not cause a skip on the rank number

# CTEs (Common Table Expression)

* Mini-table defined at the top of a query which can then be used like a normal table. You can only use a CTE directly after creating it. It can be used to combine tables together
* *WITH CTE\_name AS ()* to define a CTE

# Temporary Tables

* *CREATE TEMPORARY TABLE table\_name (columns and data type)* to create a temporary table
  + Data type and columns can be declared. Common data types are VARCHAR(), INT(), FLOAT(), DATE
  + *CREATE TABLE …* will create a new table in the database (not temporary)
  + *INSERT INTO table\_name VALUES ()* to insert information by row into table
  + Use a Select statement to get values to insert into the temp table from an existing table

# Stored Procedures

* A way to store SQL queries to reuse. It can be called again anytime.
* *CREATE PROCEDURE procedure\_name()* + Query to create a produce
* *CALL procedure\_name()* to run the procedure
* To have multiple queries inside a stored procedure
  + At top, *DELIMITER $$*
  + *BEGIN* and *END $$* around the query
  + Then, change *DELIMITER ;*
* *DROP PROCEDURE IF EXISTS procedure\_name* is good practice to override an existing procedure by the same name
* Parameters: We can allow a stored procedure to take inputs
  + Parameters are declared in () during initialization with a data type

# Inserting Data into Tables and Editing Data

* Deleting Rows and Columns
  + Rows: *DELETE FROM table\_name WHERE*
  + Columns: *ALTER table\_name DROP COLUMN column*
* Insertion for Rows
  + *INSERT INTO table\_name (column1, column2)*
  + *VALUES (value1, value2)*
* Editing
  + *UPDATE table\_name*
  + *SET column\_name = value*
  + *WHERE condition*

# Triggers and Events

* Triggers: Run automatically in response to a data change. Usually to fill another table if one table in the database is edited
  + *CREATE TRIGGER trigger\_name*
    - *AFTER INSERT ON table / AFTER UPDATE ON table* Trigger if an insert or update is made
    - *FOR EACH ROW*  Trigger activation on all rows
    - *BEGIN triggercode END;* What happens after the trigger takes place
      * Trigger Code includes an INSERT INTO another\_table with VALUES. Use NEW. to insert anything new
* Events: A scheduled automator
  + *CREATE EVENT event\_name*
    - *ON SCHEDULE EVERY ## SECOND/ AT ‘YYYY-MM-DD’*
    - *DO*
    - *BEGIN/END* statements with a *DELETE*

# Functions

* Used to make calculations, usually called in SELECT and WHERE statements. IF and ELSE statements can be used in between the BEGIN/END. Make sure to END IF
  + *CREATE FUNCTION function\_name(input1 TYPE, input2 TYPE)*
  + *RETURNS return\_type*
  + *DETERMINISTIC or NOT DETERMINISTIC*
  + *BEGIN/END* statement with a *RETURN* to return value
* Appending Results from Functions
  + *ALTER TABLE table\_name*
  + *ADD COLUMN column\_name datatype*
  + Then create the function
    - *UPDATE table\_name*
    - *SET column\_name = function();*

# Table Duplication

* *CREATE TABLE new\_table LIKE old\_table* → Duplicates the old table but only the column names and types are pulled
* *INSERT INTO new\_table SELECT \* FROM old\_table* → Duplicate the data