**Introduction to SQL**

Overview

* Can use the SQLlite server to practice – use the files saved in the folder to practice SQL queries.
* Run one line at a time
* Commands end with a semi-colon (;)
* SQL is case insensitive and ignores whitespace
* Convention
  + SQL keywords in capitals
  + Table names, column names in lower case
* SQL is subdivided into the following categories
  + DDL (Data Definition Language): Commands that define the structure of the database, such as CREATE, ALTER, and DROP statements.
  + DML (Data Manipulation Language): Commands that manipulate data stored in the database, such as INSERT, UPDATE, DELETE.
  + DCL (Data Control Language): Commands that control access to data within the database, such as GRANT and REVOKE.
  + DQL (Data Query Language): Commands used to query the database and retrieve data.

Types of SQL operation

* Querying the data
  + Querying allows a user to gain insights from the data, by selecting specific data and performing operations upon it
* Editing tables of data
  + Inserting records, deleting records, or modifying existing records falls under the second category, editing the contents of the tables
* Editing the actual database structure
  + Editing the database structure itself includes defining tables, creating, dropping, and altering them

Querying the Data

SELECT

* Basic SELECT statements
  + SELECT \* FROM people;
    - Selects all from the people table
  + SELECT forename, age FROM people;
    - Selects the specified columns from the people table
  + SELECT DISTINCT hair\_colour FROM people;
    - Selects only those values that are unique from the specified column in the people table
  + SELECT people.forename FROM people;
    - Uses dot notation to select columns from the table
* Conditional SELECT statements
  + WHERE
    - SELECT \* FROM people WHERE nationality = 'British';
      * Sets condition for selection
    - SELECT \* FROM people WHERE (nationality <> 'British') AND (age > 30);
      * Can combine multiple conditions
      * <> means does not equal
  + IN
    - SELECT \* FROM people WHERE pid IN (1, 5, 9);
      * Condition based on appearance in a list
  + NOT
    - SELECT \* FROM people WHERE pid NOT IN (1, 5, 9);
      * Inverse condition
  + LIKE
    - Can be used as part of a condition to carry out fuzzy matching
    - Uses the \_ for single characters and % for multiple characters
      * LIKE ‘p\_n’ would match to pan, pin, or pen, but not to person, or penguin, as it only matches single character replacements.
      * LIKE ‘p%n’ however, will look for any number of characters between the p and n, so both pan and penguin will be matched.
        + This also includes matching zero characters.
    - The searching capacity of LIKE is generally case sensitive.
      * To get around this, use the LOWER (or UPPER) keyword and wrap it around the column the filter is being applied to with brackets ().
      * SELECT \* FROM people WHERE LOWER(nationality) LIKE '%d%';
    - SELECT \* FROM people WHERE forename LIKE 'Alic%';
      * Forename must start with ‘Alic’
    - SELECT \* FROM people WHERE forename LIKE '\_o%';
      * Forename must contain an ‘o’
* Sorting SELECT statements
  + ORDER BY
    - SELECT \* FROM people ORDER BY forename DESC;
      * Order selection and define descending scale
    - SELECT \* FROM people ORDER BY age ASC, nationality DESC;
      * Order by multiple conditions in the order specified

Arithmetic

* Standard operators
  + + (Add)
  + - (Subtract)
  + \*\*\* (Multiply)
  + / (Divide)
  + % (Modulo)
* Direct operations on columns
  + SELECT forename, age \* 365 FROM people;
  + SELECT pID \* age FROM people;

Aggregates

* Standard aggregation functions
  + COUNT() - counts the number of records
    - SELECT COUNT(pid) FROM people;
  + SUM() - returns the sum of the values of the records
    - SELECT SUM(age) FROM people WHERE eye\_colour = 'Blue';
  + AVG() - returns the mean average of the values of records
    - SELECT AVG(age) FROM people WHERE forename LIKE 'A%';
  + NB: NULL values are ignored in these statistics.

Comments and Whitespace

* Comments
  + Single line comments use a double dash (--)
    - --DROP TABLE people
  + Multiple line comments use the /\* and \*/ convention
    - /\*
    - DROP TABLE people
    - DROP TABLE personaldata
    - \*/

Data Tables

Data Types

* VARCHAR(n) - String/Character - Variable length (up to n)
* CHAR(n) - String/Character - Fixed length (n)
* INTEGER - Integer
* SERIAL - Incrementing Integer - Auto-Incrementing integer for Primary Key
* DATE - Stores in the form YYYY-MM-DD
* DATETIME - Form YYYY-MM-DD HH:MI:SS

NULLS

* Nulls get placed anywhere there is an absence of data.
* They cannot be detected using ordinary conditions, and have to use separate statements to pull them out
  + IS NULL and IS NOT NULL.
    - SELECT \* FROM people WHERE nationality IS NULL;
    - SELECT \* FROM people WHERE nationality IS NOT NULL;

Keys

* Primary Keys
  + A primary key is a field in a table which uniquely identifies each record, and so each value must be unique.
  + Each table can only have one primary key.
  + If an entry is attempted to be inserted into a table with a value in the defined primary key column that already exists, then an error will be thrown
  + NB. Primary key can consist of multiple columns - a composite key.
* Foreign Keys
  + Foreign keys are columns or combinations of columns used to link two tables together
  + The foreign key in a set of data refers to the primary key in another set of data.
  + While foreign keys do not need to be unique, if the column has been formally defined as a foreign key, it will prevent associated primary key records from being deleted.
    - CREATE TABLE IF NOT EXISTS transactions (
    - transaction\_id INTEGER PRIMARY KEY,
    - customer\_id INTEGER,
    - product\_code INTEGER,
    - product\_name VARCHAR(255),
    - quantity INTEGER,
    - FOREIGN KEY(customer\_id) REFERENCES people(pid)
    - );
* Cascade
  + To get around this issue, of records being protected by primary key / foreign key interactions, CASCADE can be used.
  + CASCADE will take any changes in a table, and propogate (or cascade) them through any related tables.
    - CREATE TABLE transactions (
    - transaction\_id INTEGER PRIMARY KEY,
    - customer\_id INTEGER,
    - product\_code INTEGER,
    - product\_name VARCHAR(255),
    - quantity INTEGER,
    - FOREIGN KEY(customer\_id)
    - REFERENCES people(pid)
    - ON DELETE CASCADE
    - ON UPDATE CASCADE
    - );

Data Manipulation

INSERT

* INSERT is used to insert records into existing tables of data.
* Any inserted records will have to conform to any primary/foreign key constraints which have been set up:
  + No duplication of primary keys.
  + No adding non-existent foreign keys.
* Example
  + INSERT INTO people
  + VALUES (4, 'Dani', 34, 'Irish', 'Black', 'Brown');
  + Error will be thrown if datatypes are not as specified in the table
  + Can check datatypes
    - SELECT typeof(pid) FROM people;

UPDATE (using SET)

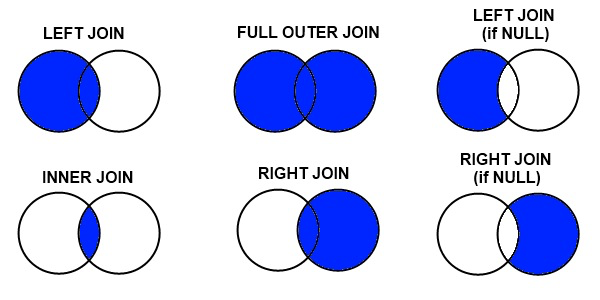
* UPDATE is used to change existing records in a table of data.
  + UPDATE people
  + SET forename = 'Alicia'
  + WHERE forename = 'Alice';
* NB. Use the WHERE condition to check what is going to be modified before carrying out the modification

DELETE

* DELETE is used to delete existing records in a table of data.
  + DELETE FROM people
  + WHERE forename = 'Bob';
* NB. Use the WHERE condition to check what is going to be modified before carrying out the modification

Performing Joins

Types of Join



* Left Join
  + Returns all the data from the first table, and any data from the second data which can be linked.
  + Data is linked if it has the same column values.
* Right Join
  + Returns all the data from the second table, and any linked data from the first.
    - Note: A Left Join from table a to table b, is the same as a Right Join from table b to table a.
* Inner Join
  + Keeps only rows which have common values in both tables.
* Outer Join
  + Keeps all data, and tries to match up what it can.

Considerations when Joining

* Joining has noteworthy effects when a value within the joining columns has duplicates - B in the above example.
* Example
  + SELECT transactions.\*, people.forename
  + FROM transactions
  + LEFT JOIN people
  + ON people.pid = transactions.customer\_id;
  + This has:
    - selected all the columns from transactions,
    - selected the forename column from people
    - joined the pid column of people, to the customer\_id column of transactions
* Joins can also be carried out on multiple columns and can include conditional statements
  + SELECT tableA.\*, tableB.\*
  + FROM tableA
  + LEFT JOIN tableB
  + ON (tableA.column1 = tableB.column2 AND tableA.column3 = tableB.column4)
  + OR (tableA.column5 = tableB.column6);

Duplicate Data (Joins) – needs further investigation

* Many-to-One
* One-to-Many
* Many-to-Many

Aliasing

* Table names can be aliased to a new name
* There are two different ways to do this, either with, or without the AS keyword.
  + SELECT t.\*, p.forename
  + -- first method with AS
  + FROM transactions AS t
  + -- second method without AS
  + LEFT JOIN people p
  + ON p.pid = t.customer\_id;

Database Operations

CREATE

* CREATE TABLE IF NOT EXISTS people (will create, if it does not already exist, a new table by the name of people).
* Within the brackets, and on separate lines for ease-of-understanding, are lines of the form: COLUMN\_NAME DATA\_TYPE [CONSTRAINT],.
* Example:
  + CREATE TABLE IF NOT EXISTS people (
    - pid INTEGER PRIMARY KEY,
    - forename VARCHAR(255),
    - age INTEGER CHECK (age < 150),
    - nationality VARCHAR(255),
    - hair\_colour VARCHAR(255),
    - eye\_colour VARCHAR(255),
    - email\_address VARCHAR(255) UNIQUE
    - );
* CHECK constraint
  + CHECK is used to place a specific condition on a column, such as a number must be more than 0, or a date more than 1900-01-01.
    - INSERT INTO people(age) VALUES (160)
      * This would return an error, as the check has been imposed above when creating the table
* UNIQUE constraint
  + UNIQUE acts like a primary key, in that it requires every value placed into a column to be unique, such as an email address which while not used as a key, should be different for every user in the database.
    - INSERT INTO people(email\_address) VALUES ('alice@gmail.com');
      * This would return an error, as the unique constraint has been imposed above when creating the table
* Once the table has been created, it can then be populated using the INSERT function:
  + INSERT INTO people (pid, forename, age, nationality, hair\_colour, eye\_colour, email\_address)
    - VALUES (1,'Alice', 29, 'British', 'Brown', 'Blue', alice@gmail.com'), (2, 'Bob', 32, 'Canadian', 'Brown', 'Blue', 'bob@gmail.com'), (3, 'Cleo', 29, 'Danish', 'Brown', 'Blue', 'cleo@gmail.com');
* Checking existing constraints
  + If we did not create the table ourselves we can check to see what constraints have been placed on the columns by querying the table. This returns the statements used to create the table.
    - SELECT sql FROM sqlite\_master WHERE name = "people";
  + Whilst it is not as nicely formatted, this does give you the ability to see the CHECK conditions, data types and UNIQUE statements.

DROP

* Dropping a table will delete the entire table, and all data stored within it - **be very careful with its use**.
  + DROP TABLE IF EXISTS transactions;
  + DROP DATABASE 'sql.sqlite';

ALTER

* The ALTER command can be used to make changes to tables of data - for example adding a new column to an existing table or renaming a column.
  + ALTER TABLE people ADD height\_cm INTEGER;
* ALTER can also be used to change constraints on tables - such as adding or removing primary and foreign keys.
  + This is done simply in other flavours of SQL, using the below syntax (not in SQLlite)
    - ALTER TABLE people DROP CONSTRAINT people\_pkey;
* Renaming columns
  + Most types of SQL
    - ALTER TABLE people RENAME COLUMN age TO age\_years;
  + SQLlite
    - Have to create a new table, transfer the values across and then delete the old table

Additional things to look into

8.1 Race Conditions

* Some database management systems allow queries for the same or different programs to be called at the same time (in parallel). If one query is finding the value of a record, and the other is updating the value of that same record then the order of execution will make a difference on what value is returned. This is known as a race condition.
* This is a nice case study on Race Conditions - [Fixing a Race Condition. I’d like to share my journey of fixing… | by Mark McDonald | In the weeds | Medium](https://medium.com/in-the-weeds/fixing-a-race-condition-c8b475fbb994)

8.2 ACID Transactions

* ACID transactions are a set of properties that a database transaction can have that enforce the integrity of the database.
* The acronym ACID stands for:
  + Atomicity
  + Consistency
  + Isolation
  + Durability
* Good databases are created, and their transactions designed by using ACID properties.
* For a primer on ACID Transactions click here - [ACID Database Properties: Ensuring Strong Data Consistency (yugabyte.com)](https://www.yugabyte.com/acid/)

8.3 RDBMS systems

* We have so far briefly discussed that for SQL we have used relational databases. This means that values within a table can relate to other tables, this means we can run queries across multiple tables to get interesting results.
* Relational Database Management Systems (RDBMS) are the programs that manage the databases themselves. These make accessing and working with the databases easier than just directly interacting with data.
* Examples for RDBMS include; Oracle Database, MySQL and Microsoft SQL Server.
* The RDBMS used will dictate the flavour of SQL which needs to be written in.
* Why not find out which RDBMS your team / department uses?

8.4 NoSQL

* Relational databases are one method of storing data, they are particularly good at storing structured data - data that is tabular and easily fits rows and columns. This, however, is not the only type of data that is stored by organisations.
* Non-relational databases use unique keys to refer to specific instances of data, without using the rigid structure of related columns. This allows the database to be flexible, as the data type and size of each record can be variable. This makes non-relational database management systems particularly good at storing data such as audio, video and text files in a scalable manner.
* NoSQL is a commonly used database that doesn’t rely on structured data.