1. Introduction to SQL

The **database** is a program that helps store data and provides functionality for adding, modifying, and querying that data, and doing that all fast. Databases come in many forms, but a really popular type of databases is called a relational database.

It stores each kind of data in a table, which is kind of like storing data in spreadsheet. A row represents an item, and column represents properties about that item.

Most database come with a query language to interact with the database. **SQL** is a language designed entirely for accessing databases.

2. Creating a table and inserting data

We can create a table in database with **CREATE TABLE** clause. We also need to append name-type of columns into the parentheses.

When creating a columns in a table, we should set a primary key. The **primary key** is a single, or a group of fields or columns that can uniquely identify a row in a table. Using a primary key, we can uinquely identifies each row of a table and gets a unique index for each primary key column that helps with faster access.

```sql
/* Create a table with named groceries */

CREATE TABLE groceries (
    id INTEGER PRIMARY KEY,
    name TEXT ,
    quantitiy INTEGER
);
```

We can put data into a table with **INSERT INTO** clause.

```sql
/* Insert a data into table */

INSERT INTO groceries VALUES (1, "Bananas", 4);
INSERT INTO groceries VALUES (2, "Peanut Butter", 1);
INSERT INTO groceries VALUES (3, "Dark chocolate bars", 2);
```

3. Querying the table

We use **SELECT** clause to retreive rows from a table. An asterisk(*) can be used to retreive all rows of table.

We can sort retrieved rows in specific order by using **ORDER BY** clause.

```sql
/* SELECT name column from groceries table */

SELECT name FROM groceries;

/* SELECT all columns from groceries table */

SELECT * FROM groceries ORDER BY aisle;
```

Anytime we want to filter rows in specific conditions, **WEHRE** clause is used.

```sql
/* Get rows which aisle is bigger than 5 */

SELECT * FROM groceries WHERE aisle > 5 ORDER BY aisle;
```

4. Aggregating data

An **aggregate function** performs a calculation on a set of values, and returns a single value. Except for COUNT(*), aggergate functions ignore null values. Aggregate functions often used with the GROUP BY clause of the SELECT cluase.

```sql
/* Total sum of quantity of groceries table */

SELECT SUM(quantity) FROM groceries;

/* Calculate sum of quantity grouped by aisle */

SELECT aisle, SUM(quantity) FROM groceries GROUP BY aisle;
```

Because the SQL engine just picked the first item out of it, we really shouldn't be using something different from what we're grouping by.

5. Project : Design a store database

Create your own store! Your store should sell one type of things, like clothing or bikes, whatever you want your store to specialize in. You should have a table for all the items in your store, and at least 5 columns for the kind of data you think you'd need to store. You should sell at least 15 items, and use select statements to order your items by price and show at least one statistic about the items.

```sql
/* Create table named store */

CREATE TABLE store (
    id INTEGER PRIMARY KEY,
    name TEXT,
    category TEXT,
    price REAL,
    rating REAL
);

/* INSERT records on store */

INSERT INTO store VALUES
    (1, 'abc', 'foods', 3.0, 4.1),
    (2, 'chocolate', 'foods', 1.9, 3.8),
    (3, 'pen-tech', 'grocery', 4.4, 4.5),
    (4, 'cake', 'foods', 12.9, 3.7),
    (5, 'product1', 'grocery', 14.0, 4.43),
    (6, 'product2', 'facny', 3.2, 1.89),
    (7, 'product3', 'foods', 27.0, 3.5),
    (8, 'product4', 'grocery', 30.0, 3.5),
    (9, 'product5', 'fancy', 15.0, 4.3),
    (10, 'product6', 'fancy', 1.6, 4.1),
    (11, 'product7', 'kits', 2.0, 4.1),
    (12, 'product8', 'kits', 47.0, 3.9),
    (13, 'product9', 'kits', 28.0, 2.8),
    (14, 'product10', 'foods', 15.2, 2.1),
    (15, 'product11', 'fancy', 3.2, 5.0),
    (16, 'product12', 'kits', 4.5, 4.5);

/* View Table sorted by price */
SELECT * FROM store ORDER BY price DESC;

/* Calculate average rating grouped by category */
SELECT category, AVG(rating) FROM store GROUP BY category;
```