Data Engineering in the Cloud

Basic SQL Queries - Part I

Xuemao Zhang
East Stroudsburg University

January 18, 2025

Outline

Before performing data loading into a data warehouse, we need to review the traditional relational database

- Installation of DBeaver
- Basic SQL Queries
  - CREATE DATABASE - creates a new database.
  - CREATE TABLE - creates a new table.
  - SELECT - extracts data from a database.
  - ALTER and ADD - add a column in an existing table.
  - UPDATE - updates data in a database.
  - DELETE - deletes data from a database.
  - INSERT INTO - inserts new data into a database.
  - CREATE INDEX - create indexes in tables
- Window functions

DBeaver

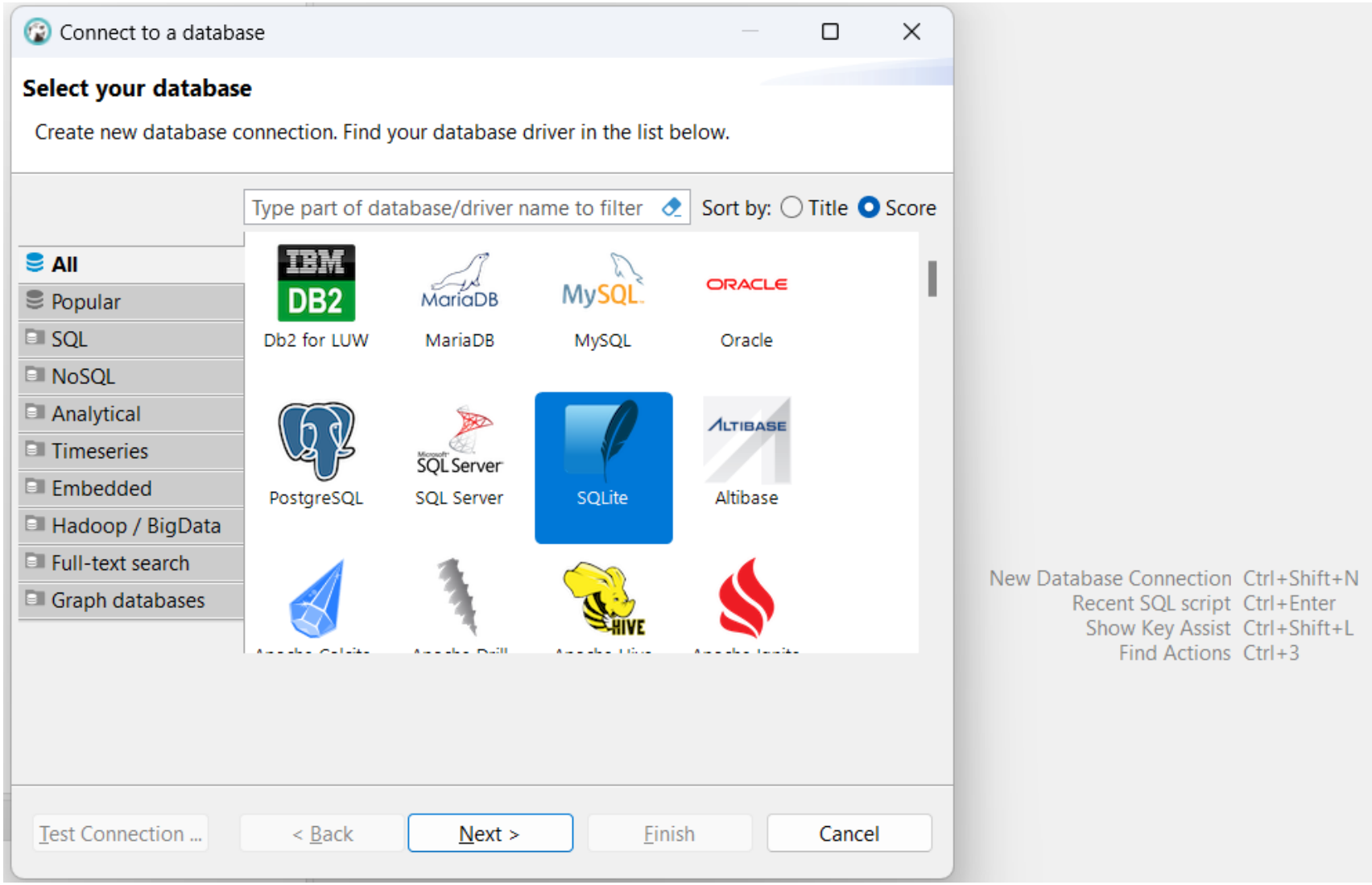
- DBeaver is free and open source universal database tool for developers and database administrators
- It is free and open-source.
- It is multiplatform.
- It supports any database having a JDBC driver.
  - One tool for all data sources.
  - A JDBC (Java Database Connectivity) driver is a software component that enables Java applications to interact with databases. It acts as an interface between Java applications and database management systems (DBMS), allowing Java code to execute SQL queries and retrieve results from databases.

DBeaver

- Installation of DBeaver Community https://dbeaver.io/download/

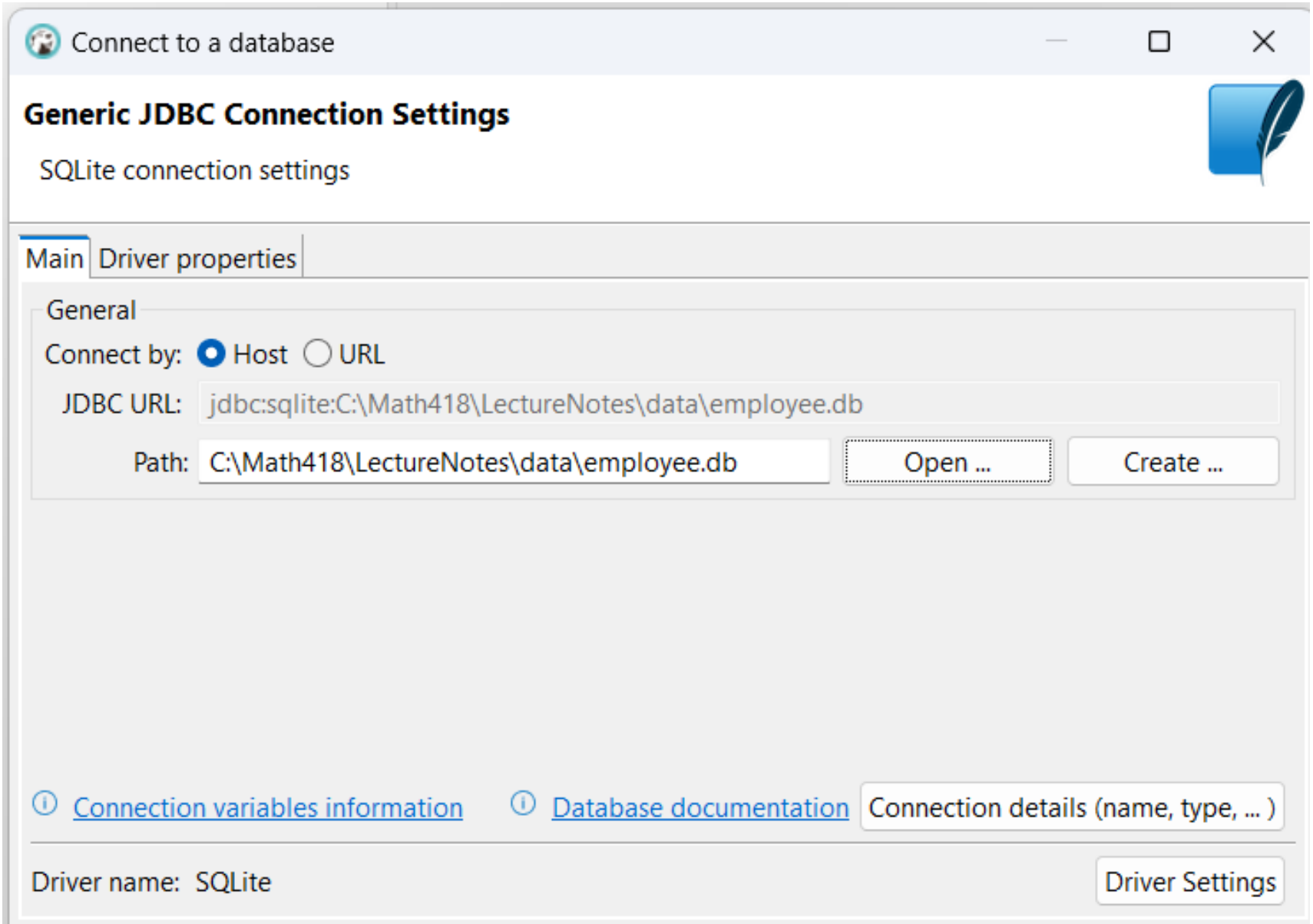
DBeaver

- New Database Connection



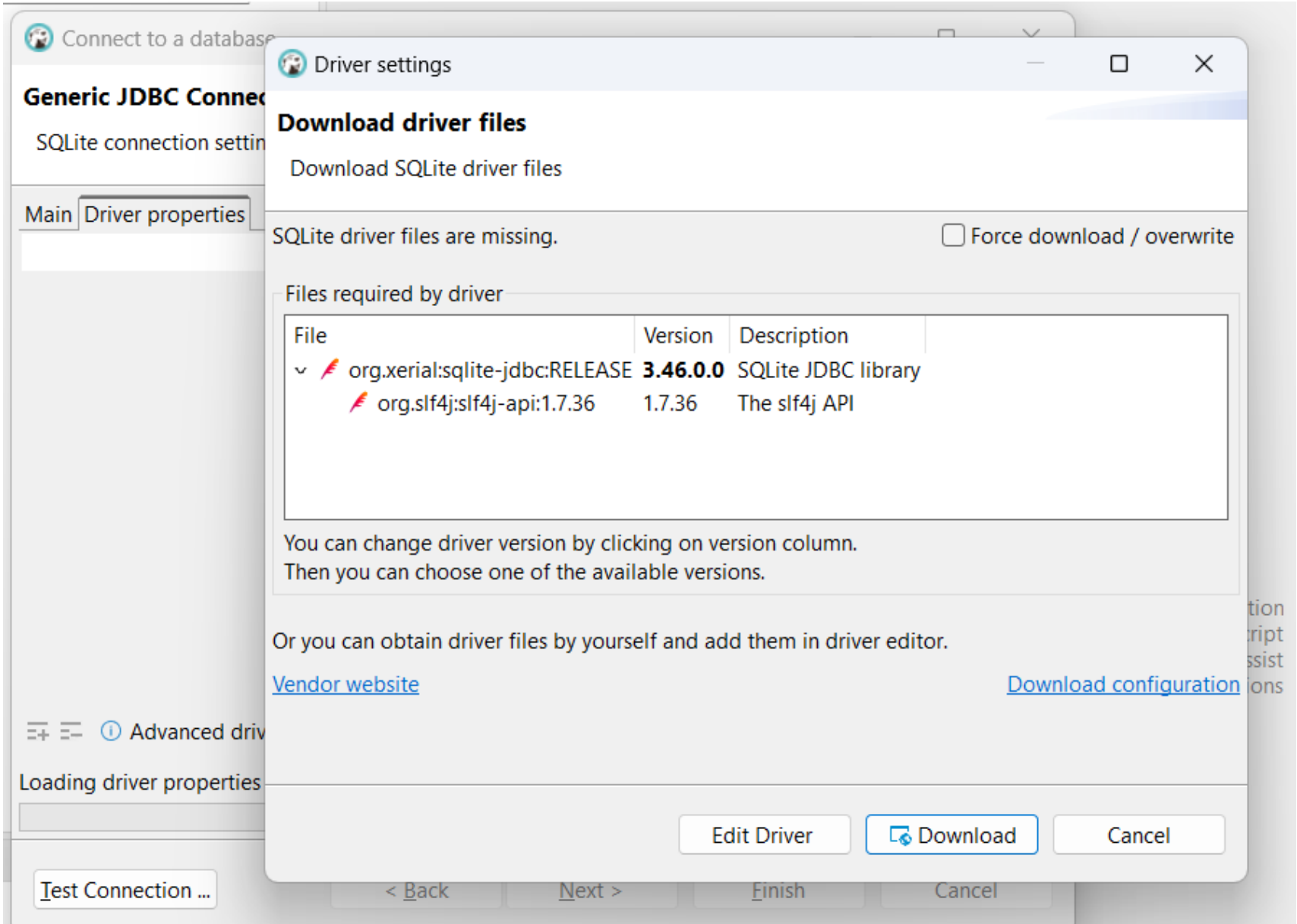
DBeaver

- Let's connect an SQLite database file `employee.db`



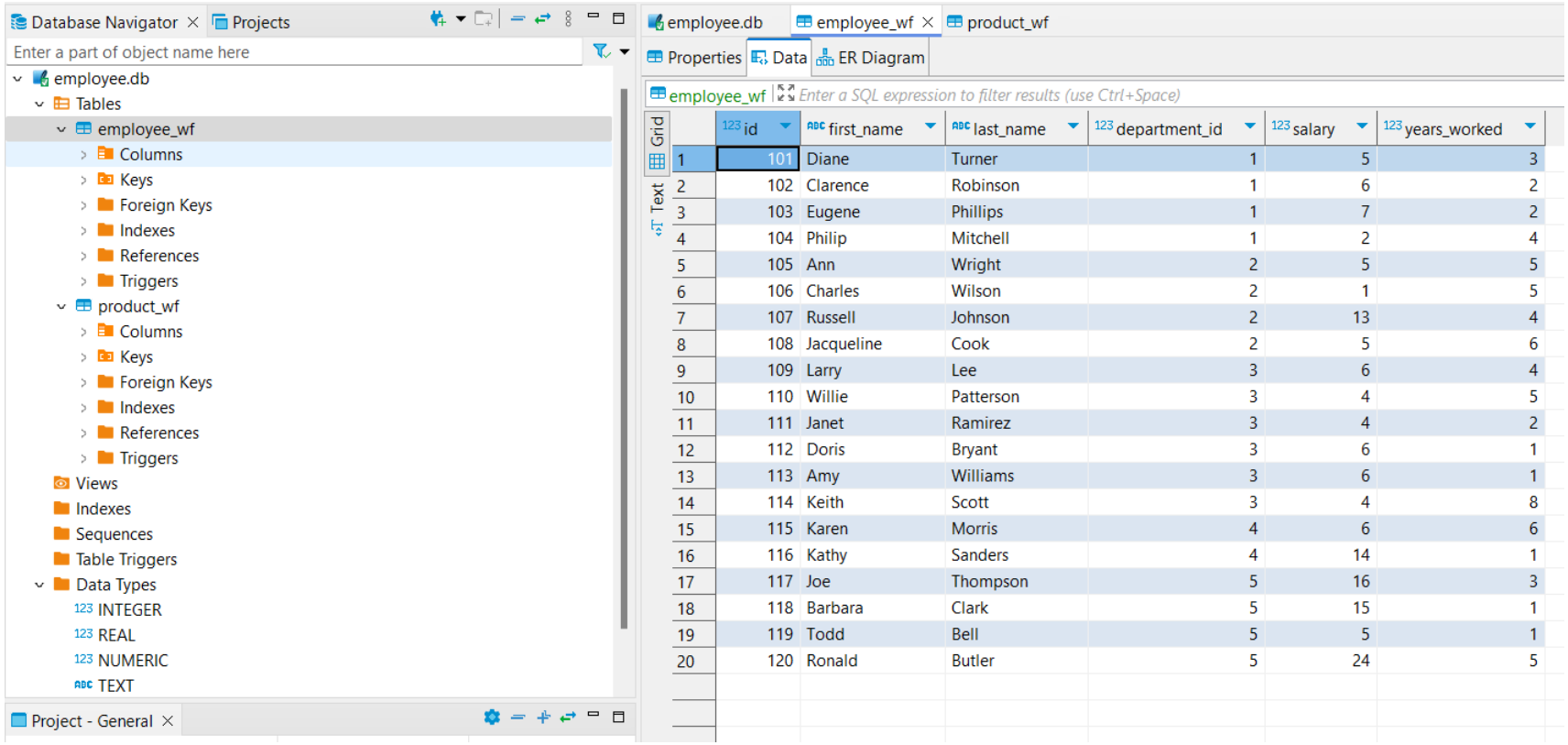
DBeaver

- We need to download the driver files



DBeaver

- This database has two tables: employee_wf and product_wf



SQL Key Words

- SQL Tutorial https://www.w3schools.com/sql/sql_syntax.asp

Some of The Most Important SQL **Key Words**:

- `SELECT` - extracts data from a database
- `UPDATE` - updates data in a database
- `DELETE` - deletes data from a database
- `INSERT INTO` - inserts new data into a database
- `CREATE DATABASE` - creates a new database
- `ALTER DATABASE` - modifies a database
- `CREATE TABLE` - creates a new table
- `ALTER TABLE` - modifies a table
- `DROP TABLE` - deletes a table
- `CREATE INDEX` - creates an index (search key)
- `DROP INDEX` - deletes an index

Basic SQL Commands - SELECT

SELECT	desired attributes
FROM	one or more tables
WHERE	conditions on rows of the tables are satisfied

- Retrieving an entire table
  - The **alias** `ew` can be used to refer to the employee_wf table in other parts of the query.

```
SELECT * FROM employee_wf ew;
```

- Return the first 5 rows only using key word `LIMIT`

```
SELECT * FROM employee_wf ew LIMIT 5;
```

Basic SQL Commands - SELECT

- Select columns

```
SELECT first_name, last_name, salary FROM employee_wf;
```

- Find unique `years_worked` using `DISTINCT`

```
SELECT DISTINCT years_worked FROM employee_wf;
```

Basic SQL Commands - SELECT

- The `ORDER BY` keyword is used to sort the result-set in ascending or descending order.

SELECT *
FROM table
ORDER BY columns ASC|DESC;

- Oder the data by variable `years_worked`

```
SELECT * FROM employee_wf
ORDER BY years_worked ASC;
```

Basic SQL Commands - SELECT

- Select columns with `years_worked >= 4`

```
SELECT first_name, last_name, salary
FROM employee_wf
WHERE years_worked >= 4;
```

```
SELECT first_name, last_name, salary, years_worked
FROM employee_wf
WHERE years_worked >= 4;
```

- A boolean expression in the **WHERE** clause may contain the following operators or any combination:
  - `AND`
  - `NOT`
  - `OR`

Basic SQL Commands - SELECT

- Select columns with `price >= 1000` and `brand == 'Apple'`

```
SELECT product_name, price
FROM product_wf
WHERE price >= 1000 AND brand == 'Apple';
```

- Select columns with `price` between 600 and 1200

```
SELECT product_name, price
FROM product_wf
WHERE price >= 600 AND price <= 1200;
```

Basic SQL Commands - SELECT

- Select columns with brand is not Apple

```
SELECT product_name, price, brand
FROM product_wf
WHERE NOT brand ==’Apple’;

SELECT product_name, price, brand
FROM product_wf
WHERE brand !=’Apple’;
```

Basic SQL Commands - SELECT

- Select columns with `brand` is Apple or Dell

```
SELECT product_category, product_name, price, brand
FROM product_wf
WHERE brand ='Apple' or brand ='Dell';
```

- Select columns with `species_id` is `brand` is Apple or Dell and `product_category=='Phone'`

```
SELECT product_category, product_name, price, brand
FROM product_wf
WHERE (brand ='Apple' or brand ='Dell') and product_category='Pho
```

Basic SQL Commands - SELECT

- The `CREATE TABLE` statement copies data from one table into a new table.
- For example, we save the data with brand ==‘Apple’ as a table
  - We create another table

```
CREATE TABLE product_apple AS
SELECT *
FROM product_wf
WHERE brand = 'Apple';

SELECT * FROM product_apple;
```

Basic SQL Commands - SELECT

- The `GROUP BY` statement groups rows that have the same values into summary rows.

SELECT	column1 aggregate(column2)
FROM	table name
WHERE	condition
GROUP BY	column1
ORDER BY	column1;

```
SELECT  brand, avg(price)
FROM product_wf
GROUP BY brand;
```

- SQLite Aggregate Functions: https://www.sqlite.org/lang_aggfunc.html

Basic SQL Commands - SELECT

- Some other arithmetic summaries: Min, Max, Count and Sum/Total

```
SELECT  min(price) as min_price, max(price) as max_price,
count(price) as n_price, sum(price) as sum_price
FROM product_wf;
```

Basic SQL Commands - SELECT

- The `DROP TABLE` statement is used to drop an existing table in a database.

```
DROP TABLE product_apple;
```

Basic SQL Commands - ALTER and ADD

- The `ALTER TABLE` statement is used to add, delete, or modify columns in an existing table.

  ALTER TABLE table name
  ADD column name datatype;

- For example, we add another column age

```
ALTER TABLE employee_wf
ADD age integer;
```

Basic SQL Commands - UPDATE

- The `UPDATE` statement is used to modify the existing records in a table.

UPDATE	table name
SET	column1 = value1, column2 = value2, ...
WHERE	condition;

Note: The `WHERE` clause specifies which record(s) that should be updated. If you omit the `WHERE` clause, all records in the table will be updated! You cannot undo an update unless you ran it inside a transaction
https://www.sqlitetutorial.net/sqlite-transaction/

- Let's update the first record

```
UPDATE employee_wf
SET age=40
WHERE id = 101;

select * from employee_wf
where id = 101;
```

Basic SQL Commands - DELETE

- The `DELETE` statement is used to delete existing records in a table.
  DELETE FROM table name
  WHERE condition;

Note: The `WHERE` clause specifies which record(s) that should be deleted. If you omit the `WHERE` clause, all records in the table will be deleted! Again, you cannot undo a delete.

- Let's delete the first record

```
DELETE FROM surveys
WHERE record_id = 1;

select * from surveys
where record_id = 1; /*an empty row will be returned*/
```

Basic SQL Commands - INSERT INTO

- The `INSERT INTO` statement is used to insert new records/rows in a table.

1. Specify both the column names and the values to be inserted:
   INSERT INTO table name (column1, column2, column3, ...)
   VALUES (value1, value2, value3, ...);
2. If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query. However, make sure the order of the values is in the same order as the columns in the table.
   INSERT INTO table name
   VALUES (value1, value2, value3, ...);

- Let's insert one more row
  - Be careful, the Primary Key cannot be NULL

```
INSERT INTO employee_wf (id, first_name, last_name, department_id,sa
VALUES (101,    'Diane',    'Turner',   1,  5,  3,  40);

select * from employee_wf
where id = 101;
```

Basic SQL Commands - CREATE INDEX

- Indexes are used to retrieve data from the database more quickly than otherwise.
- The users cannot see the indexes, they are just used to speed up searches/queries.
- **Note**: Updating a table with indexes takes more time than updating a table without (because the indexes also need an update). So, only create indexes on columns that will be frequently searched against.

1. Creates an index on a table. Duplicate values are allowed:
   CREATE INDEX index name
   ON table name (column1, column2, ...);
2. Creates a unique index on a table. Duplicate values are not allowed:
   UNIQUE UNIQUE INDEX index name
   ON table name (column1, column2, ...);

Basic SQL Commands - CREATE INDEX

- The SQL statement below creates an index named `idx_dpt` on the `department_id` column in the `employee_wf` table:

```
CREATE INDEX idx_dpt
ON employee_wf (department_id);
```

- If you want to create an index on a combination of columns, you can list the column names within the parentheses, separated by commas:

```
CREATE INDEX idx_dptid
ON employee_wf (department_id, id);
```

Basic SQL Commands - CREATE INDEX

- An index can be dropped using the `DROP` command.

DROP INDEX index name;

```
DROP INDEX idx_dpt;
DROP INDEX idx_dptid;
```

Basic SQL Commands - CREATE INDEX

Testing Index performance:

- To test if indexes will begin to decrease query times, you can run a set of queries on your database, record the time it takes those queries to finish, and then begin creating indexes and rerunning your tests.
  - To do this, try using the `EXPLAIN QUERY PLAN` clause to see how SQLite will execute the query
  - This data base is too small:(

```
EXPLAIN QUERY PLAN SELECT *
FROM employee_wf
WHERE department_id = 3;
```

Window functions

Types of window functions

- Aggregate: avg, sum, min, max, count
- Ranking: rank, dense_rank, row_number, ntile
- Value: first_value, lag, lead

```
/*average by department*/
SELECT department_id, avg(salary) from employee_wf
group by department_id;
```

Window functions

```
SELECT department_id, avg(salary) from
employee_wf group by department_id;

/*the sum of the salary column over all rows in  employee_wf */
SELECT *, SUM(salary) OVER() as total_salary from employee_wf;
```

Window functions

```
/* total salary for each department,*/
SELECT *, SUM(salary) OVER(PARTITION by department_id)
as total_salary from employee_wf;

/*For each employee based on department, percentage of salary*/
select
    *,
    (salary * 100) /(sum(salary) over( PARTITION by department_id))
FROM
    employee_wf;
```

License