



CoGrammar

SQL and SQLite

**SKILLS
FOR LIFE**

SKILLS BOOTCAMPS



Department
for Education

Software Engineering Lecture Housekeeping

- The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly.
(FBV: Mutual Respect.)
- No question is daft or silly - **ask them!**
- There are **Q&A sessions** midway and at the end of the session, should you wish to ask any follow-up questions. Moderators are going to be answering questions as the session progresses as well.
- If you have any questions outside of this lecture, or that are not answered during this lecture, please do submit these for upcoming Open Classes. You can submit these questions here: [Open Class Questions](#)

Software Engineering Lecture Housekeeping cont.

- For all **non-academic questions**, please submit a query: www.hyperiondev.com/support
- Report a **safeguarding** incident: www.hyperiondev.com/safeguardreporting
- We would love your **feedback** on lectures: [Feedback on Lectures](#)

Progression Criteria

✓ **Criterion 1: Initial Requirements**

- Complete 15 hours of Guided Learning Hours and the first four tasks within two weeks.

✓ **Criterion 2: Mid-Course Progress**

- Software Engineering: Finish 14 tasks by week 8.
- Data Science: Finish 13 tasks by week 8.

✓ **Criterion 3: Post-Course Progress**

- Complete all mandatory tasks by 24th March 2024.
- Record an Invitation to Interview within 4 weeks of course completion, or by 30th March 2024.
- Achieve 112 GLH by 24th March 2024.

✓ **Criterion 4: Employability**

- Record a Final Job Outcome within 12 weeks of graduation, or by 23rd September 2024.



Lecture Objectives

1. Introduction
2. What is SQL?
3. Why SQL?
4. SQL keywords
5. Creating and modifying tables
6. Retrieving data from tables
7. Deleting rows and tables



Relational Databases




Introduction

- A relational database is a set of tables that store data to easily retrieve and manipulate data.
- They are widely used in many applications, such as facebook, google, e-commerce and many more.



Creating a Relational Database

- 
- Once a schema (database) is designed the next step is to create the actual schema and it's tables.
 - Once the tables are created we can use SQL to insert and manipulate our statements.

What is SQL?

- ❖ SQL stands for Structured Query Language
- ❖ SQL is a database language that is composed of commands that enable users to create databases or table structures, perform various types of data manipulation and data administration as well as query the database to extract useful information.

Aspects of SQL?

- ❖ Data Definition Language (DDL):
 - Defines databases
 - Defines views
 - Defines access rights
- ❖ Data Manipulation Language (DML):
 - INSERT
 - UPDATE
 - DELETE
 - SELECT

Why SQL?

- ❖ SQL is easy to learn since its vocabulary is relatively simple. Its basic command set has a vocabulary of fewer than 100 words. It is also a non-procedural language, which means that the user specifies what must be done and not how it should be done. This aligns with our declarative approach towards programming.
- ❖ Users do not need to know the physical data storage format or the complex activities that take place when a SQL command is executed in order to issue a command.

Important Keywords

- **CREATE TABLE:** Creates a new table
- **NOT NULL:** Ensures that a column doesn't contain null values
- **UNIQUE:** Ensures that there are no repetitions
- **PRIMARY KEY:** Defines a primary key
- **FOREIGN KEY:** Defines a foreign key
- **DROP TABLE:** Deletes a table entirely

Important Keywords ...

- **INSERT:** Inserts rows into a table
- **SELECT:** Select attributes from a row
- **WHERE:** Restricts the selection of rows based on a conditional expression.
- **UPDATE:** Modifies an attribute's values in a table
- **ORDER BY:** Orders the selected rows by a specific column.
Can specify ASCENDING or DESCENDING
- **DELETE:** Deletes one or more rows from a table

Special Operators

(Used in Condition Statements)

- **BETWEEN:** Checks if a value is within range
- **IS NULL:** Checks if a value is null
- **LIKE:** Checks if a string matches a given pattern (regular expression)
- **IN:** Checks if a value is in a given list
- **EXISTS:** Checks if a query returns any rows
- **DISTINCT:** Limits to unique values

Aggregate Functions

(Used when specifying columns)

- Eg. **SELECT COUNT**(student_id) **FROM** students;
- **COUNT**: Number of rows with non-null values for a given column
- **MIN**: Returns minimum value for a given column
- **MAX**: Returns maximum value for a given column
- **SUM**: Returns the sum of a given column
- **AVG**: Returns the average of a given column

Create Table

- To create new tables in SQL, you use the **CREATE TABLE** statement.
- Pass all the columns you want in the table, as well as their data types and constraints, as arguments to the CREATE TABLE function.

The syntax of the CREATE TABLE statement:

```
CREATE TABLE table_name (  
    column1_name datatype constraint,  
    column2_name datatype constraint,  
    ...  
);
```


Create Table Example

- Table names use the snake_case convention with plural nouns.
- Columns name use the snake_case convention with singular nouns.

```
CREATE TABLE employees (  
    employee_id int NOT NULL,  
    last_name varchar(255) NOT NULL,  
    first_name varchar(255),  
    address varchar(255),  
    phone_number varchar(255),  
);
```

Inserting Values

There are two ways to write the INSERT INTO command:

1. Specify both the column names and the values to be inserted.

```
INSERT INTO employees (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

2. You do not have to specify the column names if you are adding values for all of the columns of the table. However, you should ensure that the order of the values is in the same order as the columns in the table.

```
INSERT INTO table_name
VALUES (value1, value2, value3,...);
```

Inserting Values Example

1. Specify both the column names and the values to be inserted.

```
INSERT INTO employees (employee_id, last_name, first_name,  
address, phone_number)  
VALUES (1234, 'Smith', 'John', '25 Oak Rd', '0837856767');
```

2. Specify the values only.

```
INSERT INTO employees  
VALUES (1, 'Smith', 'John', '25 Oak Rd', '0837856767');
```

Retrieving Data

- The SELECT statement is used to fetch data from a database. The data returned is stored in a result table, known as the result-set.
- To select all the columns in a table:

```
SELECT * FROM table_name;
```

- To select specific columns from a table:

```
SELECT column1, column2,...  
FROM table_name;
```

Retrieving Data Example

- To select all the columns in a table:

```
SELECT * FROM employees;
```

- To select specific columns from a table:

```
SELECT first_name, last_name,...  
FROM employees;
```

Ordering Data

- You can use the ORDER BY command to sort the results returned in ascending or descending order. The ORDER BY command sorts the records in ascending order by default.
- You need to use the DESC keyword to sort the records in descending order.

```
SELECT * FROM table_name  
ORDER BY column1 DESC;
```

- Ordering Data Example:

```
SELECT * FROM employees  
ORDER BY last_name, first_name DESC;
```

Using WHERE, IN and BETWEEN keywords

```
SELECT * FROM employees  
WHERE first_name = 'John';
```

```
SELECT * FROM albums  
WHERE genre IN ('pop', 'soul');
```

```
SELECT * FROM albums  
WHERE released BETWEEN 1975 AND 1985;
```

Modifying Values

- The UPDATE statement is used to modify the existing rows in a table.
- To use the UPDATE statement you:
 - Choose the table where the row you want to change is located.
 - Set the new value(s) for the desired column(s).
 - Choose which of the rows you want to update using the WHERE statement. NB: If you omit this, all rows in the table will change.

```
UPDATE table_name  
  SET column1 = value1, column2 = value2, ...  
  WHERE condition;
```


Modifying Values Example

customer_id	first_name	last_name	address	city
1	Maria	Anderson	23 York Street	New York
2	Jackson	Peters	124 River Road	Berlin
3	Thomas	Hardy	455 Hanover Square	London
4	Kelly	Martins	55 Loop Street	Cape Town

```
UPDATE customers
  SET address = '78 Oak St', city= 'Los Angeles'
  WHERE customer_id = 1;
```

Removing Rows

- Removing a row is a simple process. All you need to do is select the right table and row that you want to remove.
- The DELETE statement is used to remove existing rows from a table.

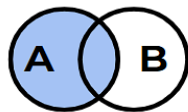
```
DELETE FROM table_name  
WHERE condition;
```

- Removing Row/s Example:

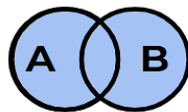
```
DELETE FROM customers  
WHERE customer_id = 4;
```

Accessing Multiple Tables

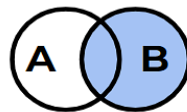
- **INNER JOIN** - Records match in both tables
- **LEFT JOIN** - All values in A, and matching values in B
- **FULL OUTER JOIN** - All values in both tables



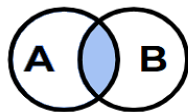
```
SELECT *  
FROM A  
LEFT JOIN B  
ON A.id = B.id
```



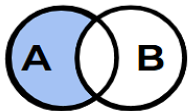
```
SELECT *  
FROM A  
FULL OUTER JOIN B  
ON A.id = B.id
```



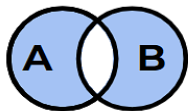
```
SELECT *  
FROM A  
RIGHT JOIN B  
ON A.id = B.id
```



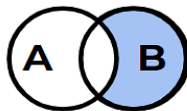
```
SELECT *  
FROM A  
INNER JOIN B  
ON A.id = B.id
```



```
SELECT *  
FROM A  
LEFT JOIN B  
ON A.id = B.id  
WHERE B.id IS NULL
```



```
SELECT *  
FROM A  
FULL OUTER JOIN B  
ON A.id = B.id  
WHERE A.id IS NULL  
OR B.id IS NULL
```



```
SELECT *  
FROM A  
RIGHT JOIN B  
ON A.id = B.id  
WHERE A.id IS NULL
```

Removing Tables

- The DROP TABLE statement is used to remove every trace of a table in a database.

```
DROP TABLE table_name;
```

```
;
```

- Removing Table Example:

```
DROP TABLE customers;
```

SQL vs SQLite

- ❖ SQLite is a form of SQL.
- ❖ SQL can be expressed in many ways:
 - SQLite
 - MySQL
 - PostgreSQL
- ❖ “UK” English vs. “US” English

SQLite

- ❖ Native to Python (Yay! No pip installations!)
- ❖ Self-Contained
 - Easy to port (Moving Database files)
- ❖ Serverless
 - Doesn't require client-server architecture. Works directly with files.
- ❖ Transactional
 - Atomic, Consistent, Isolated and Durable (ACID).
 - Ensures data integrity.

Basic SQLite Syntax ...

```
cursor.execute("INSERT INTO student(name, grade)  
VALUES(?,?)", (name1, grade1))  
db.commit()
```

```
students_ = [(name1, grade1), (name2, grade2), (name3, grade3)]  
cursor.executemany("INSERT INTO student(name, grade) VALUES(?,?)",  
students_)  
db.commit()
```


CoGrammar

Questions around Sequences



CoGrammar

Thank you for joining