Welcome to this CoGrammar session:

Connecting Python and SQL

The session will start shortly...

Questions? Drop them in the chat.
We'll have dedicated moderators
answering questions.

CoGrammar

Software Engineering Session Housekeeping

- The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly. **(Fundamental British Values: Mutual Respect and Tolerance)**
- 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 Academic Sessions. You can submit these questions here: Questions

Software Engineering Session 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

Skills Bootcamp
8-Week Progression Overview

✓ Criterion 3: Course Progress

- ***Completion:*** All mandatory tasks, including Build Your Brand and resubmissions by study period end
- *Interview Invitation:* Within 4 weeks post-course
- *Guided Learning Hours:* Minimum of 112 hours by support end date (10.5 hours average, each week)

✓ Criterion 4: Demonstrating Employability

- *Final Job or Apprenticeship Outcome:* Document within 12 weeks post-graduation
- *Relevance:* Progression to employment or related opportunity

CoGrammar

Learning Outcomes

- Recall the importance of connecting Python with SQL for data manipulation and analysis.
- Explain the basic concepts of database connectivity in Python.
- Implement Python scripts to interact with SQL databases.
- Execute SQL queries from Python and retrieve data.

Polls

- *Refer to the polls section to vote for you option.*

1. **What is the purpose of connecting Python with SQL?**
    a. Data visualisation
    b. Data manipulation and analysis
    c. Front-end development
    d. Server administration

Polls

- *Refer to the polls section to vote for you option.*

2. What is a SQL database commonly used for?

a. Storing programming code

b. Managing web server configurations

c. Managing structured data

d. Running Python scripts

CoGrammar

CoGrammar

SKILLS FOR LIFE

SKILLS BOOTCAMPS

Department for Education

Connecting Python and SQL

May 2024

Understanding Database Connectivity

- Database connectivity refers to the ability of an application or software system to establish a connection and interact with a database management system (DBMS) to perform various operations such as data retrieval, insertion, update, and deletion.

Importance of Database Connectivity

- **Data Access:** Database connectivity enables applications to access and retrieve data stored in databases.
- This is crucial for data-driven applications that rely on accessing and analysing large volumes of structured or unstructured data to perform various functions.

CoGrammar

Importance of Database Connectivity

- **Data Manipulation:** With database connectivity, applications can manipulate data within the database, such as inserting new records, updating existing ones, or deleting unwanted data.
- This allows for dynamic management and maintenance of the dataset.

Importance of Database Connectivity

- **Real-Time Updates:** Many data-driven applications require real-time updates to reflect changes in the underlying data.
- Database connectivity facilitates the synchronisation of application data with the database, ensuring that users always have access to the most current information.

CoGrammar

Importance of Database Connectivity

- **Scalability:** Data-driven applications often need to scale to handle increasing data volumes and user loads.
- Enables applications to connect to scalable database systems that can efficiently manage large datasets and handle concurrent user requests.

Importance of Database Connectivity

- **Security:** Database connectivity allows applications to implement security measures such as authentication, authorisation, and encryption to ensure the confidentiality, integrity, and availability of data.
- This is essential for protecting sensitive information from unauthorised access or malicious activities.

Importance of Database Connectivity

- **Integration:** Many applications need to integrate with multiple data sources and systems to aggregate and analyse data from different sources.
- Enables seamless integration between the application and various database systems, allowing for efficient data exchange and interoperability.

CoGrammar

Importance of Database Connectivity

- **Performance Optimisation:** Database connectivity allows applications to optimise performance by executing queries directly on the database server, leveraging the database's processing power and indexing capabilities.
- This reduces the need for data transfer between the application and the database, improving overall performance and response times.

CoGrammar

Database Connectivity: Benefits of Using Python

- Using Python for database connectivity offers numerous benefits, including:
  - ease of use,
  - extensive library support,
  - cross-platform compatibility,
  - integration with data analysis and machine learning,
  - strong community support
  - rapid development capabilities.
- These advantages make Python a compelling choice for building database-driven applications across a wide range of domains and industries.

Database Interaction: SQLite

- SQLite is a lightweight, self-contained SQL database engine that requires minimal setup and configuration. It is often used for smaller-scale projects or applications where simplicity and ease of use are prioritised.

CoGrammar

SQLite: Key Features

- **Zero Configuration:** SQLite databases are self-contained, meaning they require no external server or setup. They are simply files that can be accessed by the application directly.
- **Single File:** The entire database is stored in a single file, making it easy to distribute and manage.
- **SQL Support:** SQLite supports standard SQL syntax for querying and manipulating data, making it compatible with existing SQL-based applications and tools.

Database Interaction: SQLAlchemy

- SQLAlchemy is a powerful SQL toolkit and Object-Relational Mapping (ORM) library for Python. It provides a high-level interface for interacting with SQL databases, abstracting away many of the complexities involved in database interaction.

SQLAlchemy: Key Features

- **ORM Support:** SQLAlchemy allows developers to define database models using Python classes, which are then mapped to database tables.
- This enables seamless interaction with the database using Python objects.

CoGrammar

SQLAlchemy: Key Features

- **SQL Expression Language:** SQLAlchemy provides a SQL expression language that allows developers to construct SQL queries using Pythonic syntax.
- This makes it easier to write complex queries and perform database operations.

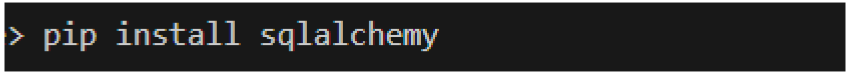
CoGrammar

SQLAlchemy: Key Features

- **Cross-Database Compatibility:** SQLAlchemy supports multiple SQL database engines, allowing applications to switch between different databases without changing the code significantly.

How to use SQLAlchemy

- To use SQLAlchemy in Python, you need to install the library using pip.



```
> pip install sqlalchemy
```

- Further details on SQLAlchemy is beyond the scope of this lesson. Please feel free to do some research on how to implement SQLAlchemy.

CoGrammar

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

CoGrammar

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.

SQL: 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

SQL: Creating a 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,
        ...
        );
```

SQL: 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),
	);
```

CoGrammar

SQL: More Keywords

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

CoGrammar

SQL: Inserting Records

- There are two ways to write the INSERT 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,…);
```

CoGrammar

SQL: INSERT 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');
```

CoGrammar

SQL: 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;
```

CoGrammar

SQL: SELECT, WHERE 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;
```

- To select columns from specific records only:

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

CoGrammar

SQL: 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.
- Records are 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;
```

SQL: Modifying Data

- The UPDATE statement is used to modify rows in a table.
    - 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 the WHERE, all rows in the table will change.

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

SQL: UPDATE Example

- From the table below

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

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

CoGrammar

SQL: Removing Records

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

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

- Removing Row/s Example:

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

CoGrammar

SQL: Special Operations (Used in Conditional 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

CoGrammar

Using IN and BETWEEN keywords

- IN Example:

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

- BETWEEN Example:

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

CoGrammar

SQL: Aggregate Functions (Used when specifying columns)

- 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

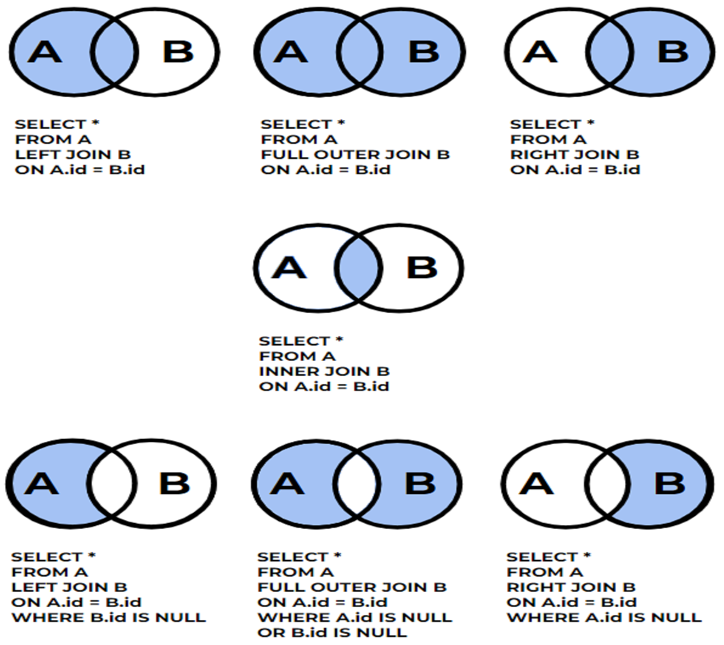
- COUNT Example:

```
SELECT COUNT(student_id) FROM students;
```

CoGrammar

SQL: 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
```

CoGrammar

SQL: Removing Tables

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

```
DROP TABLE table_name;
```

- DROP TABLE Example:

```
DROP TABLE customers;
```

CoGrammar

SQL vs SQLite

- SQLite is a form of SQL.
- SQL can be expressed in many ways:
  - SQLite
  - MySQL
  - PostgreSQL
- Like “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.

CoGrammar

SQLite Syntax

```
import sqlite3

db = sqlite3.connect('data/student_db')
cursor = db.cursor()

cursor.execute('''
    CREATE TABLE student(id INTEGER PRIMARY KEY, name TEXT,
                grade INTEGER)
''')

db.commit()
```

CoGrammar

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

Let’s take a short break

CoGrammar

Let's get coding!

CoGrammar

Polls

- *Refer to the polls section to vote for you option.*

1. **How can Python be used to interact with SQL databases?**
    a. By executing SQL queries and fetching data
    b. By managing database server configurations
    c. By generating HTML reports from database tables
    d. By visualising database schemas

Polls

- *Refer to the polls section to vote for you option.*

2. What is the purpose of the sqlite library in Python?

    a. For web development

    b. For database connectivity

    c. For data visualisation

    d. For machine learning tasks

CoGrammar

Questions and Answers

CoGrammar

Summary

- Database connectivity refers to an application's ability to connect and interact with a database management system (DBMS) to perform various operations such as data retrieval, insertion, update, and deletion.
- Python provides several libraries and frameworks for database connectivity, including built-in modules like sqlite3 for SQLite databases and third-party libraries like SQLAlchemy for interacting with various SQL databases.

Summary

- SQL (Structured Query Language) is a standard language for interacting with relational databases.
- SQL queries are used to perform operations on databases, such as querying data, inserting new records, updating existing records, and deleting records.
- Common SQL operations include SELECT (retrieve data), INSERT (add new data), UPDATE (modify existing data), and DELETE (remove data).

CoGrammar

Summary

- Python integrates seamlessly with databases through libraries and modules that provide APIs for executing SQL queries and interacting with databases.
- Libraries like **sqlite3** provide a simple and direct interface for interacting with SQLite databases, while ORM libraries like **SQLAlchemy** offer higher-level abstractions and object-oriented approaches for database interaction.

Encouragement

- Continue to explore advanced topics in Python-SQL connectivity. This can greatly enhance your skills as a developer and open up a world of possibilities for building sophisticated, data-driven applications.
- Many web frameworks, such as Django and Flask, provide built-in support for database connectivity and ORM. Learning how to integrate Python-SQL connectivity with web frameworks enables developers to build dynamic web applications with database backend.

Homework

- Create an application for managing student information using a SQL database. The application will allow users to perform tasks such as data entry, retrieval, and analysis related to student records.
- Key Features:
    - *Database Connectivity:* Establish a connection to a SQL database (e.g., SQLite) using appropriate libraries.
    - *Student Registration* and *Student Information Retrieva*l
    - *Data Analysis:* Such as calculating the average age of students.
    - *Data Modification*: Update or delete existing student.
    - *User Interface:* The application will have a user-friendly terminal layout.

CoGrammar

Thank you for attending

SKILLS FOR LIFE
SKILLS BOOTCAMPS

Department for Education

CoGrammar