SQL PART 3: INTEGRATING WITH OTHER TOOLS

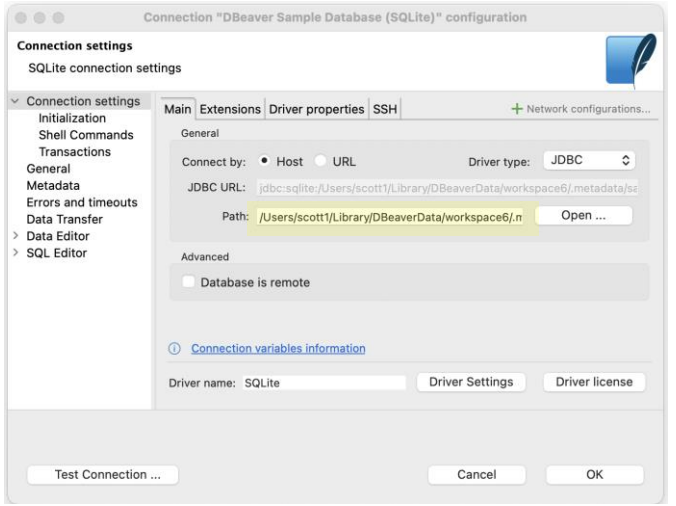
Working with SQLite in Python

CONNECTING TO AN SQLITE DATABASE WITH PYTHON

1. Import sqlite3:

```
import sqlite3
```

- This module provides a lightweight, built-in database engine that is part of Python's standard library.
- It allows you to create, connect to, and interact with SQLite databases.



CTL-click on DBeaver Sample Database (SQLite)→ Edit Connection

Copy/Paste to Collab directory

2. Connect to the Database:

```
# Use the full path to the database file found in DBeaver
conn = sqlite3.connect('Chinook.db')
```

3. Create a Cursor:

```
# Create a cursor object to execute SQL queries
cursor = conn.cursor()
```

- cursor(): The cursor object allows us to execute SQL queries and retrieve data from the database.
- It acts as a control structure to interact with the database.

BYU

CREATING AND INSERTING INTO A TABLE

- This SQL statement creates a table named employees if it doesn't already exist. The table has four columns:
- id: An integer that serves as the primary key, which means it uniquely identifies each row.
- name: A text field to store the employee's name.
- age: An integer to store the employee's age.
- department: A text field to store the employee's department.
- The IF NOT EXISTS clause ensures that the table is only created if it doesn't already exist, avoiding errors if it already does.

```
# Create a table (if it doesn't exist)
cursor.execute('''
    CREATE TABLE IF NOT EXISTS employees (
        id INTEGER PRIMARY KEY,
        name TEXT,
        age INTEGER,
        department TEXT
    )
''')

# Insert data into the table
cursor.execute("INSERT INTO employees (name, age, department) VALUES ('John Doe', 30, 'Sales')")
cursor.execute("INSERT INTO employees (name, age, department) VALUES ('Jane Smith', 25, 'Marketing')")
cursor.execute("INSERT INTO employees (name, age, department) VALUES ('David Lee', 35, 'IT')")

# Commit the changes to the database
conn.commit()
```

SQL INSERT Statement: These queries insert three employee records into the employees table. Each record contains the name, age, and department values.

commit(): This command saves any changes made during the session (like the data inserted above) to the database. Without committing, the changes would not be permanent.

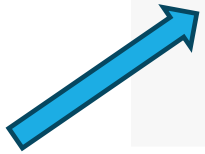
BYU

SELECTING DATA FROM A TABLE

- SELECT * FROM employees: This SQL query retrieves all the rows and columns from the **employees** table.
- fetchall(): This fetches all the results of the query as a list of tuples. Each tuple represents a row in the table.

```
# Select all data from the table
cursor.execute("SELECT * FROM employees")
rows = cursor.fetchall()

# Print the data
for row in rows:
    print(row)
```



This loop iterates over the rows list and prints each row (a tuple) representing an employee's data.

```
(1, 'John Doe', 30, 'Sales')
(2, 'Jane Smith', 25, 'Marketing')
(3, 'David Lee', 35, 'IT')
(4, 'John Doe', 30, 'Sales')
(5, 'Jane Smith', 25, 'Marketing')
(6, 'David Lee', 35, 'IT')
```

BYU

FILTERING DATA FROM A TABLE & CLOSING THE CONNECTION

- **SQL Query with** WHERE **Clause**: This query retrieves only the rows where the age is greater than 30, filtering the results based on this condition.
- This loop prints out only the employees whose age is greater than 30. It provides filtered data based on the condition.

```
# Select employees older than 30
cursor.execute("SELECT * FROM employees WHERE age > 30")
rows = cursor.fetchall()

# Print the selected data
print("\nEmployees older than 30:")
for row in rows:
    print(row)
```

```
Employees older than 30:
(3, 'David Lee', 35, 'IT')
(6, 'David Lee', 35, 'IT')
```

- close(): Closes the connection to the database. It's a good practice to close the connection after all queries are completed to release resources.

```
[ ]  # Close the connection
     conn.close()
```