SQL Database Comprehensive Guide

A channel can have multiple videos (One-to-Many relationship).

A channel can have multiple playlists (One-to-Many relationship).

A video can have multiple comments (One-to-Many relationship).

Here's the complete code:

Step 1: Define Table Schemas and Data

```
-- Drop existing tables to avoid conflicts
DROP TABLE IF EXISTS Comments, Videos, Playlists, Channels;

-- Create Channels table
CREATE TABLE Channels (
    channel_id INT PRIMARY KEY,
    channel_name VARCHAR(255),
    description TEXT,
    total_videos INT,
    subscribers BIGINT
);

-- Create Videos table
CREATE TABLE Videos (
    video_id INT PRIMARY KEY,
    channel_id INT,
    title VARCHAR(255),
    views INT,
    likes INT,
    FOREIGN KEY (channel_id) REFERENCES Channels(channel_id)
);

-- Create Playlists table
CREATE TABLE Playlists (
    playlist_id INT PRIMARY KEY,
    channel_id INT,
    title VARCHAR(255),
    FOREIGN KEY (channel_id) REFERENCES Channels(channel_id)
);

-- Create Comments table
CREATE TABLE Comments (
    comment_id INT PRIMARY KEY,
    video_id INT,
    user_name VARCHAR(255),
    comment_text TEXT,
    FOREIGN KEY (video_id) REFERENCES Videos(video_id)
);
```

Step 2: Insert Data

```
-- Insert data into Channels table
INSERT INTO Channels (channel_id, channel_name, description, total_videos, subscribers) VALUE
S
(1, 'TechTalk', 'Technology and gadget reviews', 120, 1300000),
(2, 'DailyVlogs', 'Daily life vlogs and travel', 250, 800000),
```

```
(3, 'CookingWithLove', 'Delicious recipes and cooking tips', 300, 500000),
(4, 'FitnessFirst', 'Workouts and fitness tips', 220, 750000),
(5, 'TravelWithMe', 'Travel guides and tips', 180, 650000),
(6, 'GameOn', 'Gaming reviews and walkthroughs', 150, 900000),
(7, 'MusicMania', 'Music reviews and tutorials', 110, 400000),
(8, 'AutoWorld', 'Automobile reviews and news', 200, 850000),
(9, 'FashionFrenzy', 'Fashion and beauty tips', 170, 550000),
(10, 'HistoryBuff', 'Historical documentaries and discussions', 95, 300000);

-- Insert data into Videos table
INSERT INTO Videos (video_id, channel_id, title, views, likes) VALUES
(1, 1, 'Latest Smartphone Review', 10000, 500),
(2, 2, 'A Day in My Life', 8000, 300),
(3, 3, 'Easy Pasta Recipe', 5000, 250),
(4, 4, 'Morning Yoga Routine', 12000, 800),
(5, 5, 'Exploring Paris', 15000, 1000),
(6, 6, 'Epic Game Review', 20000, 1500),
(7, 7, 'Guitar Tutorial', 9000, 600),
(8, 8, 'Latest Car Models', 18000, 1300),
(9, 9, 'Summer Fashion Trends', 7000, 400),
(10, 10, 'World War II Documentary', 22000, 1900),
(11, 1, 'Laptop Review', 11000, 550),
(12, 2, 'Camping Trip Vlog', 9000, 350),
(13, 3, 'Healthy Salad Recipe', 4000, 220),
(14, 4, 'Cardio Workout', 14000, 850),
(15, 5, 'Visiting London', 16000, 1100),
(16, 6, 'New Game Trailer', 25000, 2000),
(17, 7, 'Piano Tutorial', 9500, 620),
(18, 8, 'Classic Car Review', 19000, 1350),
(19, 9, 'Winter Fashion Trends', 8000, 450),
(20, 10, 'Ancient Egypt Documentary', 23000, 1950),
(21, 1, 'Tablet Review', 10500, 530);

-- Insert data into Playlists table
INSERT INTO Playlists (playlist_id, channel_id, title) VALUES
(1, 1, 'Smartphone Reviews'),
(2, 2, 'Travel Vlogs'),
(3, 3, 'Quick Recipes'),
(4, 4, 'Fitness Routines'),
(5, 5, 'Travel Guides'),
(6, 6, 'Game Walkthroughs'),
(7, 7, 'Music Tutorials'),
(8, 8, 'Car Reviews'),
(9, 9, 'Beauty Tips'),
(10, 10, 'History Lessons'),
(11, 1, 'Laptop Reviews'),
(12, 2, 'Nature Vlogs'),
(13, 3, 'Healthy Recipes'),
(14, 4, 'Cardio Workouts'),
(15, 5, 'City Guides'),
(16, 6, 'Game Reviews'),
(17, 7, 'Piano Tutorials'),
(18, 8, 'Classic Car Reviews'),
(19, 9, 'Winter Fashion Tips'),
(20, 10, 'Ancient History');

-- Insert data into Comments table
INSERT INTO Comments (comment_id, video_id, user_name, comment_text) VALUES
```

```
(1, 1, 'TechLover', 'Great review!'),
(2, 2, 'VlogFan', 'Loved this video!'),
(3, 3, 'ChefMaster', 'Thanks for the recipe!'),
(4, 4, 'YogaEnthusiast', 'This routine is amazing!'),
(5, 5, 'Traveler123', 'Paris is beautiful!'),
(6, 6, 'GamerDude', 'This game is awesome!'),
(7, 7, 'MusicLover', 'Great tutorial!'),
(8, 8, 'CarFanatic', 'Loved the car review!'),
(9, 9, 'Fashionista', 'Great fashion tips!'),
(10, 10, 'HistoryBuff', 'Very informative!'),
(11, 11, 'TechFan', 'Loved the laptop review!'),
(12, 12, 'NatureLover', 'Great camping trip vlog!'),
(13, 13, 'HealthyEater', 'Awesome salad recipe!'),
(14, 14, 'FitnessGuru', 'Great cardio workout!'),
(15, 15, 'Traveler456', 'London looks amazing!'),
(16, 16, 'GamerGirl', 'Can’t wait for this game!'),
(17, 17, 'PianoPro', 'Very helpful piano tutorial!'),
(18, 18, 'CarEnthusiast', 'Loved the classic car review!'),
(19, 19, 'FashionGuru', 'Great winter fashion tips!'),
(20, 20, 'HistoryBuff2', 'Fascinating documentary on Egypt!'),
(21, 21, 'TechLover2', 'Tablet review was spot on!');
```

Step 3: Set Python Program to Connect to PostgreSQL and Perform Tasks

First install psycopg2 module from PyCharm terminal by typing:

pip install psycopg2

Then type the following lines:

```
import psycopg2

connection = psycopg2.connect(user='postgres',
                              password='tc7@3*G5h9Bx%W%',
                              host='localhost',
                              port='5432',
                              database='Test')
cursor = connection.cursor()

# Count of rows of each table
tables = ['Channels', 'Videos', 'Playlists', 'Comments']
for table in tables:
    cursor.execute(f"SELECT COUNT(*) FROM {table}")
    count = cursor.fetchone()[0]
    print(f"Count of rows in {table}: {count}")

# Show a sample of 3 rows from each table
tables = ['Channels', 'Videos', 'Playlists', 'Comments']
for table in tables:
    cursor.execute(f"SELECT * FROM {table} LIMIT 3")
    rows = cursor.fetchall()
    print(f"Sample rows from {table}:")
    for row in rows:
        print(row)

# Inner Join example (Videos and Channels)
print("Inner Join Result between Videos and Channels:")
cursor.execute("""
    SELECT Videos.video_id, Videos.title, Channels.channel_name
```

```
    FROM Videos
    INNER JOIN Channels ON Videos.channel_id = Channels.channel_id
""")
inner_join_result = cursor.fetchall()
for row in inner_join_result:
    print(row)

cursor.close()
connection.close()
```

Explanation:

Count of rows of each table

```
cursor = connection.cursor()
tables = ['Channels', 'Videos', 'Playlists', 'Comments']
for table in tables:
    cursor.execute(f"SELECT COUNT(*) FROM {table}")
    count = cursor.fetchone()[0]
    print(f"Count of rows in {table}: {count}")
```

Setting Up the Cursor (`cursor = connection.cursor()`):

- Imagine the database as a big storage room. The `connection` is like having a key to enter that room.
- The `cursor` acts like a flashlight you can use to navigate and retrieve information from the room (database).

Code Breakdown

1. **List of Tables**:

   ```
   tables = ['Channels', 'Videos', 'Playlists', 'Comments']
   ```

   This line creates a list named `tables` that contains the names of the tables you want to count the rows for. A list in Python is a way to store multiple items in a single variable.

2. **For Loop**:

   ```
   for table in tables:
   ```

   This line starts a `for` loop. A `for` loop in Python is used to iterate over a sequence (in this case, the list `tables`). The loop will go through each item in the list one by one.

3. **SQL Query Execution**:

   ```
   cursor.execute(f"SELECT COUNT(*) FROM {table}")
   ```

   Inside the loop, this line executes an SQL query. The `cursor.execute()` method runs a command in the database. The command here is `f"SELECT COUNT(*) FROM {table}"`. The `f` before the quotation marks indicates an f-string, which is a way to format strings in Python. It allows you to insert the value of the variable `table` directly into the string. For example, when `table` is `'Channels'`, the command becomes `"SELECT COUNT(*) FROM Channels"`.

   The SQL command `SELECT COUNT(*) FROM table_name` counts the number of rows in the specified table.

4. **Fetching the Result**:

   ```
   count = cursor.fetchone()[0]
   ```

   After executing the SQL query, this line retrieves the result. `cursor.fetchone()` fetches the next row of a query result set, returning a single sequence, or `None` when no more data is available. `[0]` extracts the first element of the fetched result, which is the count of rows in the table.

5. **Printing the Result**:

```
print(f"Count of rows in {table}: {count}")
```

This line prints out the result in a readable format. Again, using an f-string, it inserts the name of the current table and the count of rows into the string and prints it. For example, if `table` is `'Channels'` and `count` is `10`, it will print: `Count of rows in Channels: 10`.

Show a sample of 3 rows from each table

```
tables = ['Channels', 'Videos', 'Playlists', 'Comments']
for table in tables:
    cursor.execute(f"SELECT * FROM {table} LIMIT 3")
    rows = cursor.fetchall()
    print(f"Sample rows from {table}:")
    for row in rows:
        print(row)
```

Purpose of the Code

This code snippet is designed to connect to a database, retrieve a sample of 3 rows from each of the specified tables (`Channels`, `Videos`, `Playlists`, `Comments`), and print out these rows.

Code Breakdown

1. **List of Tables**:

```
tables = ['Channels', 'Videos', 'Playlists', 'Comments']
```

This line creates a list named `tables` that contains the names of the tables you want to get sample rows from. A list in Python is a way to store multiple items in a single variable.

2. **For Loop**:

```
for table in tables:
```

This line starts a `for` loop. A `for` loop in Python is used to iterate over a sequence (in this case, the list `tables`). The loop will go through each item in the list one by one.

3. **SQL Query Execution**:

```
cursor.execute(f"SELECT * FROM {table} LIMIT 3")
```

Inside the loop, this line executes an SQL query. The `cursor.execute()` method runs a command in the database. The command here is `f"SELECT * FROM {table} LIMIT 3"`. The `f` before the quotation marks indicates an f-string, which is a way to format strings in Python. It allows you to insert the value of the variable `table` directly into the string. For example, when `table` is `'Channels'`, the command becomes `"SELECT * FROM Channels LIMIT 3"`.

The SQL command `SELECT * FROM table_name LIMIT 3` retrieves up to 3 rows from the specified table.

4. **Fetching the Result**:

```
rows = cursor.fetchall()
```

After executing the SQL query, this line retrieves the result. `cursor.fetchall()` fetches all (remaining) rows of a query result, returning a list of tuples. Each tuple represents a row from the table.

5. **Printing the Result**:

```
print(f"Sample rows from {table}:")
for row in rows:
    print(row)
```

These lines print out the result in a readable format:

- The `print(f"Sample rows from {table}:")` line prints a header for the table's sample rows. Using an f-string, it inserts the name of the current table into the string and prints it. For example, if `table` is `'Channels'`, it will print: `Sample rows from Channels:`.
- The `for row in rows:` line starts another `for` loop that goes through each row in the `rows` list.
- The `print(row)` line prints each row in the table. A row is represented as a tuple, which contains the values of that row.

Join Method:

```
# Inner Join
print("Inner Join Result:")
cursor.execute("""
    SELECT Videos.video_id, Videos.title, Channels.channel_name
    FROM Videos
    INNER JOIN Channels ON Videos.channel_id = Channels.channel_id
""")
inner_join_result = cursor.fetchall()
for row in inner_join_result:
    print(row)
```

Purpose of the Code

This code snippet is designed to join two tables in a database (`Videos` and `Channels`), retrieve some specific information from both tables, and print out the result.

Code Breakdown

1. **Executing the SQL Query**:

```
cursor.execute("""
    SELECT Videos.video_id, Videos.title, Channels.channel_name
    FROM Videos
    JOIN Channels ON Videos.channel_id = Channels.channel_id
""")
```

- `cursor.execute(...)`: This method runs a command in the database. Here, it executes an SQL query.
- `""" ... """`: Triple quotes are used to write multi-line strings in Python. This makes the query more readable.
- `SELECT Videos.video_id, Videos.title, Channels.channel_name`: This part of the query specifies the columns we want to retrieve. We want the `video_id` and `title` from the `Videos` table and the `channel_name` from the `Channels` table.
- `FROM Videos`: This indicates the main table we are selecting data from, which is the `Videos` table.
- `JOIN Channels ON Videos.channel_id = Channels.channel_id`: This part of the query joins the `Channels` table to the `Videos` table. The `ON` clause specifies that we should join rows where the `channel_id` in the `Videos` table matches the `channel_id` in the `Channels` table.

In simpler terms, this query combines rows from the `Videos` table with rows from the `Channels` table where they share the same `channel_id`.

2. **Fetching the Result**:

```
inner_join_result = cursor.fetchall()
```

- `cursor.fetchall()`: This method retrieves all the rows returned by the SQL query. It returns a list of tuples, where each tuple represents a row in the result.

3. **Printing the Result**:

```
for row in join_result:
    print(row)
```

- `for row in join_result:`: This starts a `for` loop that iterates over each row in the `join_result` list.
- `print(row)`: This prints each row. Each row is a tuple containing the `video_id` and `title` from the `Videos` table and the `channel_name` from the `Channels` table.

Example Output

Suppose we have the following data in the `Videos` and `Channels` tables:

- `Videos` table:

```
+----------+------------+-----------+-------+
| video_id | channel_id | title     | views |
+----------+------------+-----------+-------+
| 1        | 1          | Video1    | 100   |
| 2        | 2          | Video2    | 200   |
| 3        | 1          | Video3    | 150   |
+----------+------------+-----------+-------+
```

- `Channels` table:

```
+------------+--------------+---------------------------+-------+
| channel_id | channel_name | description               | subs  |
+------------+--------------+---------------------------+-------+
| 1          | Channel1     | Channel 1 Description     | 1000  |
| 2          | Channel2     | Channel 2 Description     | 2000  |
+------------+--------------+---------------------------+-------+
```

The output of the code will be:

```
Join result between Videos and Channels:
(1, 'Video1', 'Channel1')
(2, 'Video2', 'Channel2')
(3, 'Video3', 'Channel1')
```

Explanation of the Output

- `(1, 'Video1', 'Channel1')`: This means the video with `video_id` 1 has the title 'Video1' and belongs to 'Channel1'.
- `(2, 'Video2', 'Channel2')`: This means the video with `video_id` 2 has the title 'Video2' and belongs to 'Channel2'.
- `(3, 'Video3', 'Channel1')`: This means the video with `video_id` 3 has the title 'Video3' and belongs to 'Channel1'.

The join operation successfully combines information from both tables where they share the same `channel_id`.

Code Breakdown

- `cursor.close()`: Closes the cursor to free up resources and prevent errors.
- `connection.close()`: Closes the connection to the database to free up resources, avoid limits, and maintain security.

Always remember to close your cursors and connections when you're done using them. This is a crucial step in managing database resources effectively.