Python Course **WT 24/25**

Lecture: Marcus Rockel & Prof. Dr. Eva Lütkebohmert-Holtz, Tutorial: Felix Häschel
https://www.finance.uni-freiburg.de/

Tutorial 8

Exercise 1 (SQL Queries - music.db)**.**
This set of exercises is designed to help you practice fundamental SQL operations. You will work on grouping, joining, and filtering data to gain a solid foundation for more advanced SQL concepts. The tasks are focused on exploring relationships between tables and performing basic aggregations.
Write the appropriate SQL query to obtain the desired results for each task:

1. Find the total number of tracks for each album.
2. List the top 3 genres with the highest number of tracks.
3. Find the average length of tracks for each genre.
4. List the albums with more than 10 tracks.
5. Find the total sales for each album.
6. List the customers who have purchased tracks from more than 5 different genres.
7. Find the most popular genre in terms of total sales using a CTE.
8. List the top 5 customers who have spent the most money on music.
9. Find the average number of tracks per album for each artist.
10. List the tracks that have been purchased more than once.

Solution.

```sql
-- Exercise 1: music.db

-- 1. Find the total number of tracks for each album
SELECT al.AlbumId, al.Title AS AlbumName, COUNT(*) AS TotalTracks
FROM tracks t
JOIN albums al USING (AlbumId)
GROUP BY al.AlbumId, al.Title;

-- 2. List the top 3 genres with the highest number of tracks
SELECT g.GenreId, g.Name AS GenreName, COUNT(*) AS TotalTracks
FROM tracks t
JOIN genres g USING (GenreId)
GROUP BY g.GenreId, g.Name
ORDER BY TotalTracks DESC
LIMIT 3;
```

```
-- 3. Find the average length of tracks for each genre
SELECT g.GenreId, g.Name AS GenreName, AVG(t.Milliseconds) AS AverageTrackLength
FROM tracks t
JOIN genres g USING (GenreId)
GROUP BY g.GenreId, g.Name;

-- 4. List the albums with more than 10 tracks
SELECT al.AlbumId, al.Title AS AlbumName, COUNT(*) AS TotalTracks
FROM tracks t
JOIN albums al USING (AlbumId)
GROUP BY al.AlbumId, al.Title
HAVING COUNT(*) > 10;

-- 5. Find the total sales for each album
SELECT al.AlbumId, al.Title AS AlbumName, SUM(ii.UnitPrice * ii.Quantity) AS
↪  TotalSales
FROM invoice_items ii
JOIN tracks t USING (TrackId)
JOIN albums al USING (AlbumId)
GROUP BY al.AlbumId, al.Title;

-- 6. List the customers who have purchased tracks from more than 5 different
↪  genres
SELECT c.CustomerId, c.FirstName || ' ' || c.LastName AS CustomerName,
↪  COUNT(DISTINCT g.GenreId) AS GenreCount
FROM customers c
JOIN invoices i USING (CustomerId)
JOIN invoice_items ii USING (InvoiceId)
JOIN tracks t USING (TrackId)
JOIN genres g USING (GenreId)
GROUP BY c.CustomerId, c.FirstName, c.LastName
HAVING COUNT(DISTINCT g.GenreId) > 5;

-- 7. Find the most popular genre in terms of total sales using a CTE
WITH GenreSales AS (
    SELECT g.GenreId, g.Name AS GenreName, SUM(ii.UnitPrice * ii.Quantity) AS
    ↪  TotalSales
    FROM tracks t
    JOIN genres g USING (GenreId)
    JOIN invoice_items ii USING (TrackId)
    GROUP BY g.GenreId, g.Name
)
SELECT GenreId, GenreName, TotalSales
FROM GenreSales
ORDER BY TotalSales DESC
LIMIT 1;
```

```
-- 8. List the top 5 customers who have spent the most money on music
SELECT c.CustomerId, c.FirstName || ' ' || c.LastName AS CustomerName,
↪  SUM(ii.UnitPrice * ii.Quantity) AS TotalSpent
FROM customers c
JOIN invoices i USING (CustomerId)
JOIN invoice_items ii USING (InvoiceId)
GROUP BY c.CustomerId, c.FirstName, c.LastName
ORDER BY TotalSpent DESC
LIMIT 5;

-- 9. Find the average number of tracks per album for each artist
SELECT a.ArtistId, a.Name AS ArtistName, AVG(AlbumTracks.TotalTracks) AS
↪  AverageTracksPerAlbum
FROM artists a
JOIN albums al USING (ArtistId)
JOIN (
    SELECT AlbumId, COUNT(*) AS TotalTracks
    FROM tracks
    GROUP BY AlbumId
) AS AlbumTracks ON al.AlbumId = AlbumTracks.AlbumId
GROUP BY a.ArtistId, a.Name;

-- 10. List the tracks that have been purchased more than once
SELECT t.TrackId, t.Name AS TrackName, COUNT(ii.InvoiceId) AS PurchaseCount
FROM tracks t
JOIN invoice_items ii USING (TrackId)
GROUP BY t.TrackId, t.Name
HAVING COUNT(ii.InvoiceId) > 1;
```

Exercise 2 (DDL: Creating a New Table)**.**
In the given music database, create a new table named `play_histories` to store track play events. The table should have the following columns:

- `PlayHistoryId` (INTEGER) as the primary key
- `TrackId` (INTEGER) referencing the `tracks` table
- `PlayDate` (TEXT) to store the date and time of the play event
- `Device` (TEXT) to store the name/type of the device used to play the track

Ensure the foreign key constraint is properly defined.

Solution.

```
CREATE TABLE IF NOT EXISTS play_histories (
    PlayHistoryId INTEGER PRIMARY KEY,
    TrackId INTEGER,
    PlayDate TEXT,
    Device TEXT,
    FOREIGN KEY (TrackId) REFERENCES tracks(TrackId)
);
```

Exercise 3 (DML: Inserting and Updating Data)**.**
Insert a new entry into the `play_histories` table for a track with `TrackId` 1, played on 2022-01-01 at 10:00 AM on a device named "Phone". Then, update the device name to "Tablet" for the same play event.

Solution.

```sql
INSERT INTO play_histories (TrackId, PlayDate, Device)
VALUES (1, '2022-01-01 10:00:00', 'Phone');

UPDATE play_histories
SET Device = 'Tablet'
WHERE TrackId = 1 AND PlayDate = '2022-01-01 10:00:00';
```

Exercise 4 (TCL: Ensuring Data Consistency with Transactions)**.**
Write a sequence of SQL statements for the music database that:

1. Begins a transaction.
2. Inserts two new customers into the `customers` table.
3. Commits the transaction if both inserts succeed.
4. If any of the inserts fail, the transaction should be rolled back (no manual action needed, but the logic should be set up so that if an error occurs, you can issue a `ROLLBACK`).

Make sure the operations are either all performed successfully or all aborted.

Solution.

```sql
BEGIN TRANSACTION;

INSERT INTO customers (FirstName, LastName, Company, Address, City, State, Country,
↪  PostalCode, Phone, Fax, Email, SupportRepId)
VALUES ('Alice', 'Smith', 'Acme Corp', '123 Main St', 'Metropolis', 'NA', 'USA', '12345',
↪  '555-1234', NULL, 'alice@acme.com', 3);

INSERT INTO customers (FirstName, LastName, Company, Address, City, State, Country,
↪  PostalCode, Phone, Fax, Email, SupportRepId)
VALUES ('Bob', 'Jones', 'Acme Corp', '456 Elm St', 'Metropolis', 'NA', 'USA', '12345',
↪  '555-5678', NULL, 'bob@acme.com', 3);

COMMIT;
```