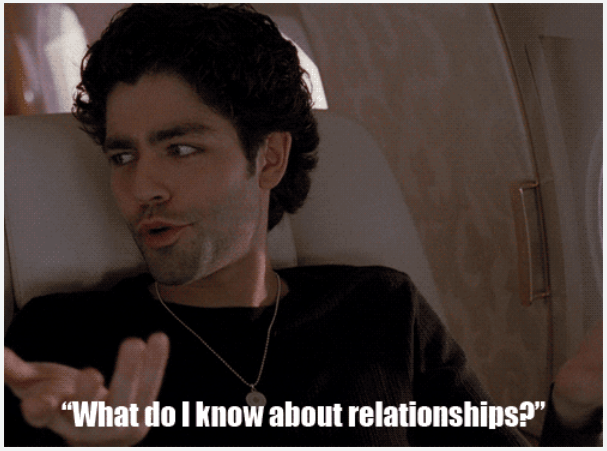
TECH TALENT SOUTH

SQL Part 3: Relational Databases

Creating Relationships in SQL

In past lectures we have learned how to execute all of the basic CRUD operations on databases using SQL.

Now we are going to learn how to create relationships between tables (how to implement a relational database) in SQL.



TECH TALENT SOUTH

Review: Relational Database

A relational database is a collection of data items with ***pre-defined*** relationships between them.

- Items are organized as a set of tables
- Tables have columns (properties) and rows (records)
- Primary Keys are unique identifiers that distinguish individual records (rows) in a table
- Foreign Keys (primary keys from other tables) link tables within a database

Joins

JOIN is used to combine results across several tables.

There are several scenarios in which this might occur:

- if a user wants to access several tables from a statement
- if a user wanted to combine rows in order to obtain data using the SELECT statement.

Combination of the tables depends on the similar fields the tables have.

Joins

SQL join is comprised of five types;

- Inner join
- Left outer join
- Right outer join
- Full outer join
- Cross join.

TECH TALENT SOUTH

Joins

Suppose you want to return the album_id and SUM(length) from the songs table and order them by album_id. You would write:

```
SELECT
        album_id,
        SUM(length) AS Minutes
FROM songs GROUP BY album_id;
```

In this query:
songs is the base or 'left' table.

(That's the table name that appears after 'FROM')

This result is fine, but...

TECH TALENT SOUTH

Joins

Wouldn't it be nice if we could return the **Album Name** that is associated with the **Album ID**, instead of that pretty meaningless album_id value.

However, album_name is in the **albums** table and we've been working from the **songs** table!

TECH TALENT SOUTH

JOINS

An SQL **JOIN** is going to allow us to do just that!

JOIN brings together several related tables.

```
1 SELECT albums.id, albums.name, SUM(songs.length) AS Minutes
2 FROM songs
3 INNER JOIN albums ON songs.album_id = albums.id
4 GROUP BY albums.id;
```

The songs table has a foreign key (album_id) to the primary key (id) of the albums table.

Since the two tables are linked, we can explicitly bring them together by using an INNER JOIN.

TECH TALENT SOUTH

JOINS

In this case, we are telling the database to give us all the rows where the foreign key of songs (albums_id) matches the primary key (id) of albums.

```
1 SELECT albums.id, albums.name, SUM(songs.length) AS Minutes
2 FROM songs
3 INNER JOIN albums ON songs.album_id = albums.id
4 GROUP BY albums.id;
```

INNER JOIN: returns rows when there is a match in both tables

TECH TALENT SOUTH

JOINS

LEFT JOIN: A **LEFT JOIN** will return all the rows from the left table, even if there are no matches in the right table.

```
SELECT
        albums.name,
        songs.name,
        songs.length AS Minutes
FROM albums
LEFT JOIN songs ON albums.id = songs.album_id
ORDER BY albums.name, Minutes;
```

Here, you can see that we are still able to display the album name, but we are returning all of the individual songs on the album (799 records).

TECH TALENT SOUTH

JOINS SUMMARY

In brief, if you are selecting from one table, but you want or need a column from another linked to the first, use a JOIN, since it can bring columns together from distinct tables.

An INNER JOIN will return every record where the join statement is true in both tables. A LEFT JOIN will return everything selected from the base table, even if there are no matches between the two tables.

USING JOINS

You can use multiple JOIN clauses in the same query:

```
SELECT
    artists.name AS Artist,
    songs.name AS Song,
    songs.length AS Minutes,
        albums.name AS Album,
    albums.label AS Label,
    albums.year_released AS Released
FROM albums
INNER JOIN songs ON albums.id = songs.album_id
INNER JOIN artists ON albums.artist_id = artists.id
GROUP BY albums.name
ORDER BY artists.name;
```

TECH TALENT SOUTH

USING JOINS

You can also JOIN to a dataset formed by a subselect. Take a look:

```
SELECT
        albums.name,
        songSubSelect.name,
        songSubSelect.length AS Minutes
FROM albums
INNER JOIN
(
        SELECT * FROM songs
) songSubSelect ON albums.id = songSubSelect.album_id;
```

You can JOIN multiple times to the same table, so long as you distinguish each JOIN with an alias.

TECH TALENT SOUTH

SET OPERATIONS

JOIN uses logical relationships between tables to return data from those tables.

A set operation is different in that it combines two or more datasets into a single result (or dataset).

TECH TALENT SOUTH

UNION & UNION ALL

UNION ALL stacks datasets on top of each other.

The only rule is that both datasets must have the same number and type of columns. Moreover, the columns in the top dataset are the columns for the result dataset.

```
SELECT id, name, created_at FROM songs --a dataset

UNION ALL

SELECT id, name, updated_at FROM albums --another dataset
```

TECH TALENT SOUTH

UNION & UNION ALL

UNION (without the ALL) will do the same thing *except* it will eliminate any row in the bottom dataset that is a duplicate of a row in the top dataset.

```
SELECT * FROM songs

UNION

SELECT * FROM songs;
```

...we will only return the actual number of records in the songs table (i.e., 799).

But, UNION ALL would return twice that amount (i.e., 1598).

TECH TALENT SOUTH

PRACTICE PROBLEM

Explain why it does or does not matter whether UNION or UNION ALL is used in the following query.

```
SELECT id, created_at FROM songs

UNION --or UNION ALL

SELECT album_id, updated_at FROM songs;
```

TECH TALENT SOUTH

INTERSECT

INTERSECT is similar to UNION and UNION ALL in that INTERSECT SELECT statements must be of the same number and data type.

However, INTERSECT does not stack datasets. Rather, it returns the intersection of two or more datasets.

Here is an example:

```
SELECT album_id, name FROM songs

INTERSECT

SELECT id, name FROM albums;
```

TECH TALENT SOUTH

INTERSECT

Let's run SELECT COUNT on album_id to see how many albums share a name with one of the included songs

```
SELECT COUNT(album_id) FROM
(
    SELECT album_id, name FROM songs

    INTERSECT

    SELECT id, name FROM albums;
);
```

TECH TALENT SOUTH

INTERSECT

Sometimes, of course, there are no columns in common, and nothing is returned.

```
SELECT album_id, name, created_at FROM songs

INTERSECT

SELECT id, name, created_at FROM albums;
```

EXCEPT

Unlike INTERSECT, EXCEPT returns all records the first SELECT statement that are not in the second.

Basically, it looks at the first dataset and eliminates any of its records that match what gets returned in the second dataset.

Like UNION and INTERSECT, the columns in the SELECT statements must be of the same number and data type.

TECH
TALENT
SOUTH

EXCEPT

Here is an example:

```
SELECT name
FROM songs
WHERE album_id < 65

EXCEPT

SELECT name
FROM songs
WHERE name = 'Ironic';
```

TECH TALENT SOUTH

PRACTICE PROBLEM

Which simpler query can more efficiently get the same result as the SELECT statement just shown?

Use COUNT to ensure your answer returns the same number of results.

TECH TALENT SOUTH

TEMP TABLES

SQL allows you to create temporary tables, which are stored in an accessible 'temp' database. It is similar to creating an actual table:

TECH TALENT SOUTH

TEMP TABLES

```sql
CREATE TEMPORARY TABLE MusicNames
(
  MusicNameID INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
  ArtistName VARCHAR NOT NULL,
  SongName VARCHAR NOT NULL,
  AlbumName VARCHAR NOT NULL,
  GenreName VARCHAR NOT NULL
);

INSERT INTO MusicNames
(
  ArtistName,
  SongName,
  AlbumName,
  GenreName
)
SELECT
    art.name,
        s.name,
    al.name,
    g.name
FROM albums al
INNER JOIN songs s ON al.id = s.album_id
INNER JOIN genres g ON al.genre_id = g.id
INNER JOIN artists art ON al.artist_id = art.id;
--Notice that I did not use 'AS' before the alias.
--Yes, that is allowed!

SELECT *
FROM MusicNames mn
INNER JOIN genres ON mn.GenreName = genres.name;
--select whatever you want!
```

TECH TALENT SOUTH

TEMP TABLES

To get rid of a TEMP table, run the following command:

```
DROP TABLE [tableName];
```

In our case:

```
DROP TABLE MusicNames;
```

KEYWORD: DISTINCT

DISTINCT ensures you don't get any repeated values in columns that might have more than one of the same value.

The basic syntax of DISTINCT keyword is as follows:

```
SELECT DISTINCT column 1, column 2
    FROM table name
    WHERE (condition)
```

For Example:

```
SELECT DISTINCt album_id FROM songs; --71
--vs
SELECT album_id FROM songs; -- 799
```

TECH TALENT SOUTH

SORTING

Finally, you'll likely want to sort the results you get from queries.

The records or rows in a table are usually not ordered, therefore you end up viewing the rows randomly. SQL sorting allows you to arrange the rows as per your preference.

This is enabled by the **ORDER BY** and **GROUP BY** clauses.

TECH TALENT SOUTH

SORTING: ORDER BY

ORDER BY sorts results either in ascending or descending order. Different types of sorting include the following:

```
1 -- Sorting by one column
2 SELECT column
3     FROM table name
4     ORDER BY sort_column [ASC | DESC]
```

```
1 -- Sorting by multiple columns
2 SELECT column
3     FROM table
4     ORDER BY column 1, column 2, column N [ASC | DESC]
```

```
1 -- Sorting by relative column positions
2 SELECT column
3     FROM table
4     ORDER BY sort_num1 [ASC | DESC]; sort_num2 [ASC | DESC]; sort_num N [ASC | DESC]
5
```

TECH TALENT SOUTH

SORTING: ORDER BY

For Example:

```
1 SELECT * from albums ORDER BY artist_id, name ASC;
```

Let's try a more complex query:

```
1 SELECT albums.name AS Album, songs.name AS Song, artists.name AS Artist
2 FROM songs
3 INNER JOIN albums ON songs.album_id = albums.id
4 INNER JOIN artists ON albums.artist_id = artists.id
5 ORDER BY Album, Song;
```

TECH TALENT SOUTH

SORTING: GROUP BY

GROUP BY organizes similar data into clusters.

It is used before making the ORDER BY query and after using the WHERE Clause.

A simple syntax of this clause is as follows;

```
SELECT column 1, column 2
   FROM table name
   WHERE (condition)
   GROUP BY column 1, column 2
   ORDER BY column 1, column 2
```

TECH TALENT SOUTH

SORTING: GROUP BY

```
1 SELECT songs.name AS Song, albums.name AS Album, artists.name AS Artist
2 FROM songs
3 INNER JOIN albums ON songs.album_id = albums.id
4 INNER JOIN artists ON albums.artist_id = artists.id
5 GROUP BY Album
6  ORDER BY Album, Song
```

GROUP BY does remove duplicate rows based on the column values -- but, group by should not be relied on to remove duplicates - instead use DISTINCT to remove duplicates.

TECH TALENT SOUTH

Challenge

1. UPDATE the favorite table (don't forget to set UpdatedAt to CURRENT_TIMESTAMP!) so that every record but two have a favorite song (SongID), genre (GenreID), artist (ArtistID), and album (AlbumID).
2. Starting from the User table, get the name of every User and that user's favorite song, genre, artist, and album. This will involve some JOINs!
3. Starting from the albums table, SELECT every album name and its respective ID that are longer than 5 minutes (this will involve a JOIN).

TECH TALENT SOUTH