
MONTANA
STATE UNIVERSITY

Aggregation

Summarizing Data In SQL

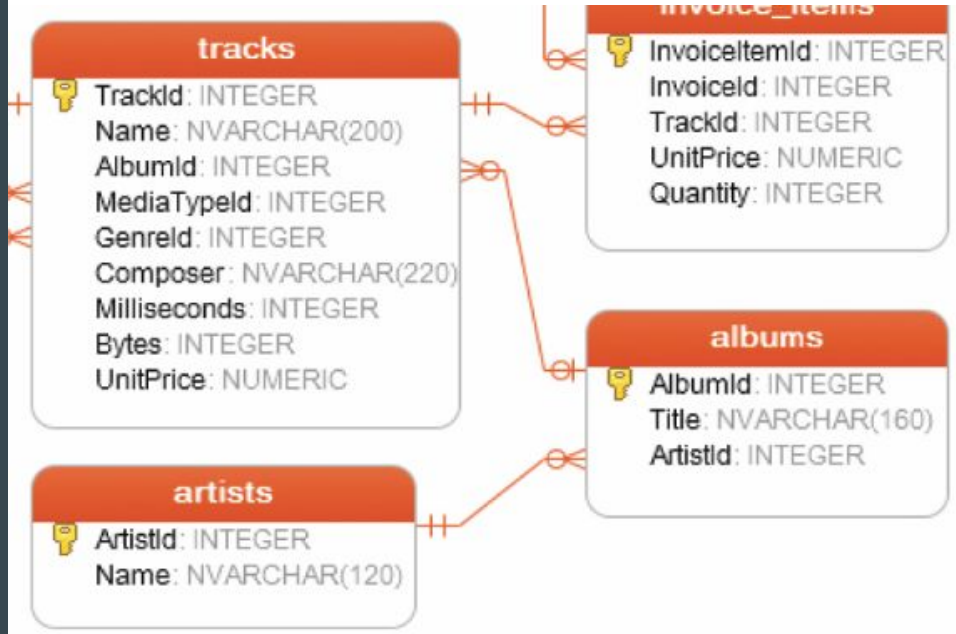
Last Lecture

- In the last lecture we learned about JOINS, which allowed you to correlate one table with another via a condition (typically a foreign key)
- In this lecture, we are going to discuss how to “roll data up” into useful summary data, using the GROUP BY clause

```
SELECT name
FROM tracks
JOIN albums ON tracks.AlbumId = albums.AlbumId
WHERE albums.Title = "Machine Head"
```

Tracks & Albums

- We will be focusing on the following three tables
  - Tracks
  - Albums
  - Artists



GROUP BY

- The general form of the GROUP BY clause is:

  SELECT *columns*
  FROM *table*
  GROUP BY *columns*

```sql
-- Select all AlbumID FKs from the tracks table
SELECT AlbumID
FROM tracks
GROUP BY AlbumID;
```

GROUP BY

- This query will select all data in the tracks table and, for each AlbumID, group the results into a single resulting row
- Not a very interesting grouping, but this does return all distinct AlbumIDs in the tracks table

```sql
-- Select all AlbumID FKs from the tracks table
SELECT AlbumID
FROM tracks
GROUP BY AlbumID;
```

DISTINCT

- Speaking of distinct album IDs, there is also a DISTINCT operator that can be used
- If you were really just after distinct AlbumIDs, this would be clearer

```
-- Select all AlbumID FKs from the tracks table
SELECT DISTINCT AlbumID
FROM tracks;
```

GROUP BY

- So, what is GROUP BY for then?
- The typical use case for GROUP BY is for rolling data up with *Aggregate Functions*

```
-- Select all AlbumID FKs from the tracks table
SELECT AlbumID, COUNT(*)
FROM tracks
GROUP BY AlbumID
```

GROUP BY

- Here is an example that uses the COUNT() aggregate function
- This query will aggregate all rows rows with the same AlbumID, and include a count of the number of rows

```sql
-- Select all AlbumID FKs from the tracks table
SELECT AlbumID, COUNT(*)
FROM tracks
GROUP BY AlbumID
```

GROUP BY

- Often you will want to give a nice name to the aggregated column, by using an alias
- This makes it easier to work with in the resulting data

```sql
-- Select all AlbumID FKs from the tracks table
SELECT AlbumID, COUNT(*) as TrackCount
FROM tracks
GROUP BY AlbumID
```

GROUP BY

- Group By can be combined with Joins to give even better results
- Note that we had to fully qualify tracks.AlbumID because this column is in both the tracks and albums table
- This makes AlbumID by itself *ambiguous*

```sql
-- Select all AlbumID FKs from the tracks table
SELECT tracks.AlbumID, Title, COUNT(*) as TrackCount
FROM tracks
JOIN albums on tracks.AlbumId = albums.AlbumId
GROUP BY tracks.AlbumID;
```

GROUP BY

- This new query now gives us something really cool: the count of the number of tracks on each album!
- This is a legitimate report-tier query that a business might ask for


Output | Select all AlbumID F...from the tracks table

347 rows

	AlbumId	Title	1	...
1	156	...And Justice For All		9
2	257	20th Century Masters - The Millennium…		12
3	296	A Copland Celebration, Vol. I		1
4	94	A Matter of Life and Death		11
5	95	A Real Dead One		12
6	96	A Real Live One		11
7	285	A Soprano Inspired		1
8	139	A TempestadeTempestade Ou O Livro Dos…		15
9	203	A-Sides		17
10	160	Ace Of Spades		15
11	232	Achtung Baby		12

GROUP BY

- Other Aggregation Functions
  - AVG - average
  - MAX - maximum value
  - MIN - minimum value
  - SUM - summed value
- What is the total runtime of albums?

```sql
-- Calculate run time of albums by summing the tracks
SELECT tracks.AlbumID, Title,
       SUM(tracks.Milliseconds) as Milliseconds
FROM tracks
JOIN albums on tracks.AlbumId = albums.AlbumId
GROUP BY tracks.AlbumID;
```

GROUP BY

- What if we wanted to extend this out to find out the total runtime of music by artists?
- Add another JOIN and update the GROUP BY clause

```sql
-- Calculate run time of artists by summing the tracks
SELECT artists.Name,
       SUM(tracks.Milliseconds) as Milliseconds
FROM tracks
JOIN albums on tracks.AlbumId = albums.AlbumId
JOIN artists on albums.ArtistId = artists.ArtistId
GROUP BY albums.ArtistId;
```

GROUP BY

- What if we want to see the number of tracks and albums as well?
- Attempt 1, add counts of TrackID and AlbumID

```sql
-- Calculate run time of artists by summing the tracks
SELECT artists.Name,
       COUNT(tracks.TrackId) as Tracks,
       COUNT(albums.AlbumId) as Albums,
       SUM(tracks.Milliseconds) as Milliseconds
FROM tracks
JOIN albums on tracks.AlbumId = albums.AlbumId
JOIN artists on albums.ArtistId = artists.ArtistId
GROUP BY albums.ArtistId;
```

GROUP BY

- What if we want to see the number of tracks and albums as well?
- Attempt 1, add counts of TrackID and AlbumID
- Oops...

	Name	Tracks	Albums	Milliseconds
1	AC/DC	18	18	4853674
2	Accept	4	4	1200650
3	Aerosmith	15	15	4411709
4	Alanis Morissette	13	13	3450925
5	Alice In Chains	12	12	3249365
6	Antônio Carlos Jobim	31	31	7128385
7	Apocalyptica	8	8	2671407
8	Audioslave	40	40	10655588
9	BackBeat	12	12	1615722
10	Billy Cobham	8	8	2680524
11	Black Label Society	18	18	5507674
12	Black Sabbath	17	17	4896722

GROUP BY

- The problem here is that when we GROUP BY, we get a row for each unique track/album/artist combination
- There are just as many AlbumIds as TrackIds

```sql
-- Simple Join to show the rows
SELECT TrackId, albums.AlbumId, artists.ArtistId
FROM tracks
         JOIN albums on tracks.AlbumId = albums.AlbumId
         JOIN artists on albums.ArtistId = artists.ArtistId;
```

	TrackId	AlbumId	ArtistId
1	1	1	1
2	6	1	1
3	7	1	1
4	8	1	1
5	9	1	1
6	10	1	1
7	11	1	1
8	12	1	1
9	13	1	1
10	14	1	1
11	15	4	1
12	16	4	1

GROUP BY

- DISTINCT to the rescue!
- Attempt 2

```sql
-- Calculate run time of artists by summing the tracks
SELECT artists.Name,
       COUNT(tracks.TrackId) as Tracks,
       COUNT(DISTINCT albums.AlbumId) as Albums,
       SUM(tracks.Milliseconds) as Milliseconds
FROM tracks
         JOIN albums on tracks.AlbumId = albums.AlbumId
         JOIN artists on albums.ArtistId = artists.ArtistId
GROUP BY albums.ArtistId;
```

GROUP BY

- DISTINCT to the rescue!
- Attempt 2
- Yeah kids, now we are cookin’ with *gasoline*

	Name	Tracks	Albums	Milliseconds
1	AC/DC	18	2	4853674
2	Accept	4	2	1200650
3	Aerosmith	15	1	4411709
4	Alanis Morissette	13	1	3450925
5	Alice In Chains	12	1	3249365
6	Antônio Carlos Jobim	31	2	7128385
7	Apocalyptica	8	1	2671407
8	Audioslave	40	3	10655588
9	BackBeat	12	1	1615722
10	Billy Cobham	8	1	2680524
11	Black Label Society	18	2	5507674

GROUP BY

- What if we wanted this information for all artists having more than 10 tracks?
- We can use the HAVING clause

```
SELECT artists.Name,
       COUNT(tracks.TrackId) as Tracks,
       COUNT(DISTINCT albums.AlbumId) as Albums,
       SUM(tracks.Milliseconds) as Milliseconds
FROM tracks
         JOIN albums on tracks.AlbumId = albums.AlbumId
         JOIN artists on albums.ArtistId = artists.ArtistId
GROUP BY albums.ArtistId
HAVING Tracks >= 10;
```

GROUP BY

- The HAVING clause is similar to the WHERE clause, but it applies to the *aggregated data*
- All predicates in the HAVING clause should work only with aggregated columns
  - SQLite doesn't enforce this
  - Not sure why

```
SELECT artists.Name,
       COUNT(tracks.TrackId) as Tracks,
       COUNT(DISTINCT albums.AlbumId) as Albums,
       SUM(tracks.Milliseconds) as Milliseconds
FROM tracks
         JOIN albums on tracks.AlbumId = albums.AlbumId
         JOIN artists on albums.ArtistId = artists.ArtistId
GROUP BY albums.ArtistId
HAVING Tracks >= 10;
```

GROUP BY

- Show me all artists who have tracks that start with an A, and the count of tracks and albums that these tracks are on, and the total runtime of those tracks
- First attempt… nope.

```sql
SELECT artists.Name,
       COUNT(tracks.TrackId) as Tracks,
       COUNT(DISTINCT albums.AlbumId) as Albums,
       SUM(tracks.Milliseconds) as Milliseconds
FROM tracks
         JOIN albums on tracks.AlbumId = albums.AlbumId
         JOIN artists on albums.ArtistId = artists.ArtistId
GROUP BY albums.ArtistId
HAVING Tracks >= 10 AND tracks.Name LIKE "A%";
```

GROUP BY

- You might be tempted to put this in the HAVING clause and, unfortunately, SQLite allows this
  - Not quite sure what it means
- But SQLite warns you: this should be in the WHERE clause instead

```
SELECT artists.Name,
       COUNT(tracks.TrackId) as Tracks,
       COUNT(DISTINCT albums.AlbumId) as Albums,
       SUM(tracks.Milliseconds) as Milliseconds
FROM tracks
         JOIN albums on tracks.AlbumId = albums.AlbumId
         JOIN artists on albums.ArtistId = artists.ArtistId
WHERE tracks.Name LIKE "A%"
GROUP BY albums.ArtistId
HAVING Tracks > 2;
```

GROUP BY

- Note that the WHERE clause is applied first, before aggregation
- Then the data is aggregated
- Then the HAVING clause is applied

```
SELECT artists.Name,
       COUNT(tracks.TrackId) as Tracks,
       COUNT(DISTINCT albums.AlbumId) as Albums,
       SUM(tracks.Milliseconds) as Milliseconds
FROM tracks
         JOIN albums on tracks.AlbumId = albums.AlbumId
         JOIN artists on albums.ArtistId = artists.ArtistId
WHERE tracks.Name LIKE "A%"
GROUP BY albums.ArtistId
HAVING Tracks > 2;
```

GROUP BY

- Let’s go back to this query
- What if we wanted to include the average runtime of tracks in this query?

```sql
-- Calculate run time of artists by summing the tracks
SELECT artists.Name,
       COUNT(tracks.TrackId) as Tracks,
       COUNT(DISTINCT albums.AlbumId) as Albums,
       SUM(tracks.Milliseconds) as Milliseconds
FROM tracks
         JOIN albums on tracks.AlbumId = albums.AlbumId
         JOIN artists on albums.ArtistId = artists.ArtistId
GROUP BY albums.ArtistId;
```

Aggregation Summary

- OK, so, we looked at the GROUP BY functionality in SQL, for creating aggregate queries
- We saw how you can join across one or more tables to generate more useful queries
- We discussed how you can use the HAVING clause to filter your aggregated data
- And we talked about how the WHERE clause can still be used to filter out data *before* aggregation occurs
- Pretty cool stuff!


MONTANA
STATE UNIVERSITY