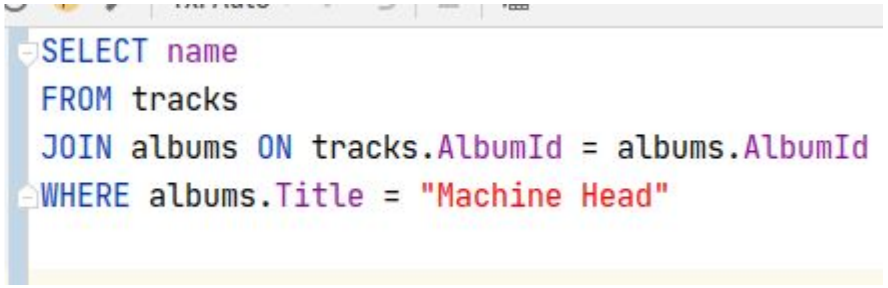
MONTANA
STATE UNIVERSITY

Joins

Correlating Tables with Foreign Keys

Last Lecture

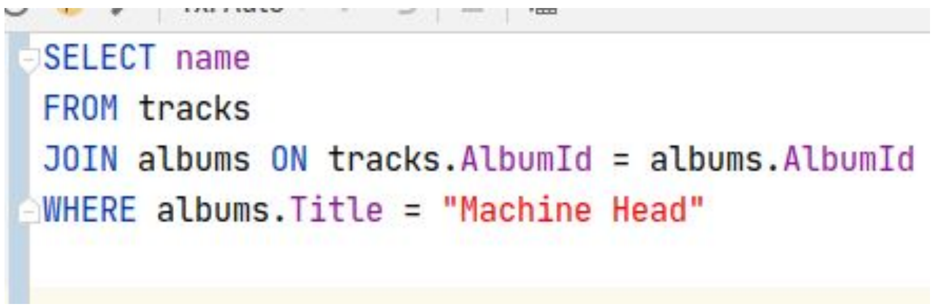
- Recall in the last lecture we learned about sub-queries
- Sub-queries could be be used to answer questions like *“Give me the name of all tracks on the album named ‘Machine Head’”*



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

Joins

- While Sub-Queries are useful and used in production environments, this is a common operation that is typically and more efficiently achieved with JOINS

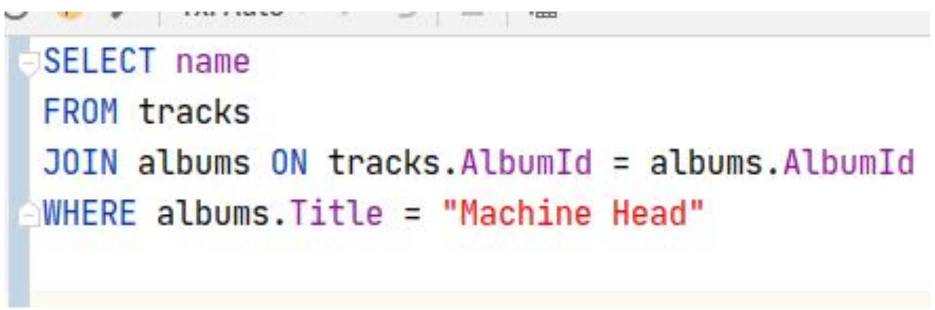


```sql
SELECT name
FROM tracks
JOIN albums ON tracks.AlbumId = albums.AlbumId
WHERE albums.Title = "Machine Head"
```

Joins

- The basic join syntax is

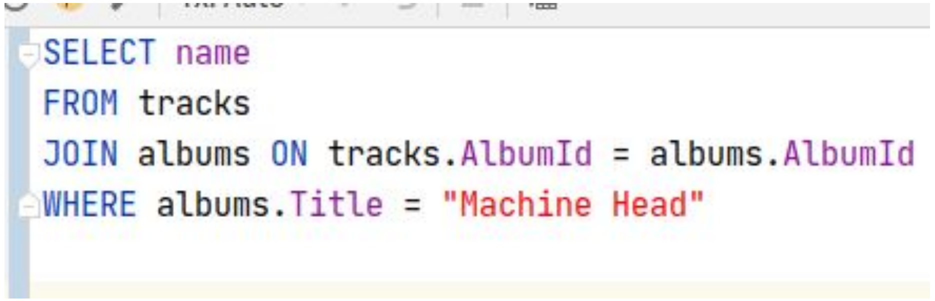
  SELECT <cols>
  FROM <table name>
  JOIN <table name> ON <condition>
  WHERE <predicates>



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

Joins

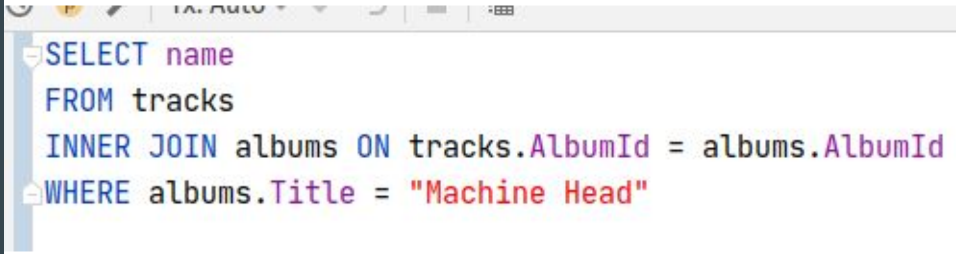
- This example shows what is called an *Inner Join*
- An inner join returns all rows for which the condition in the join clause is true



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

Joins

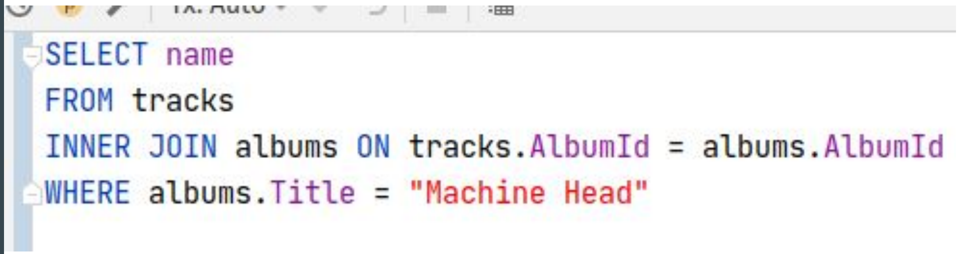
- Can also be written out explicitly with an INNER keyword
- Inner joins are the most common sort of join you will work with



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

Joins

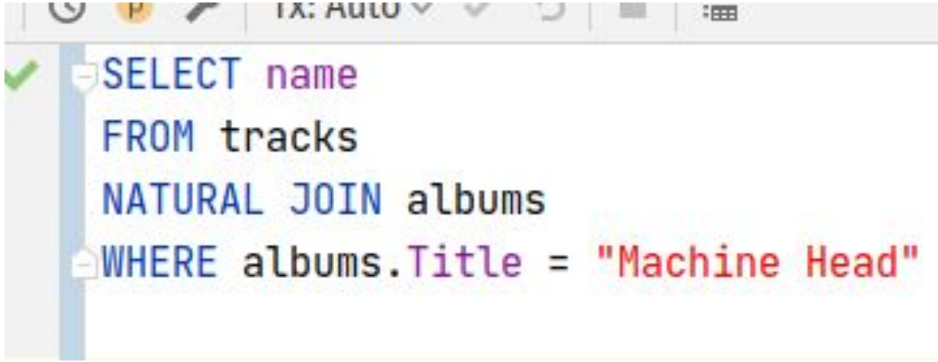
- Note that the join condition uses the Foreign Key in tracks and the key in albums
- The join condition is usually an equality condition, but it doesn't have to be



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

Natural Joins

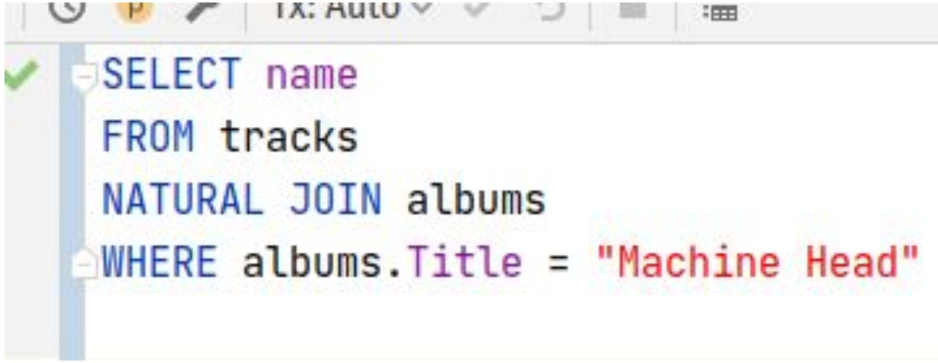
- It is often the case that the columns in one table line up with the columns on another table
- FKs typically have the same name as the referred table
- If this is the case, you can use a natural join and omit the equality expression



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

Natural Joins

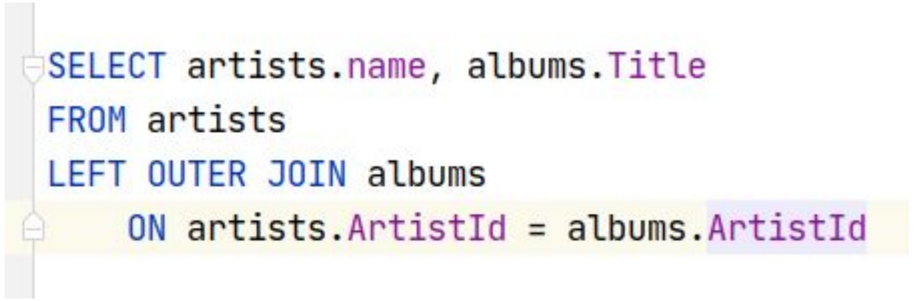
- A little cleaner
- Rare in practice
  - Why?



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

Outer Join

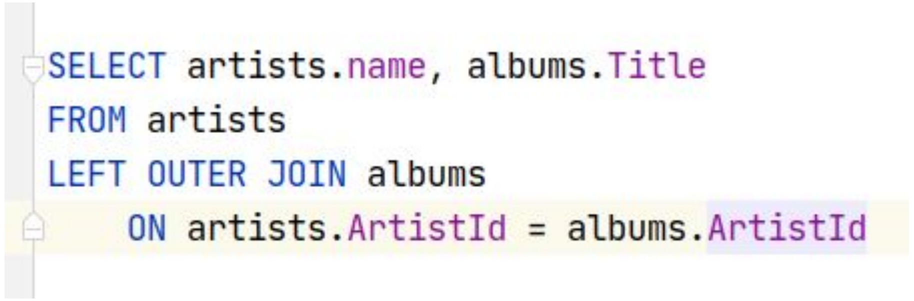
- Inner joins are fairly intuitive
- But sometimes you want ALL the rows of a table, even if there isn't a match in the joined table
- To accomplish this we need to use OUTER JOINs



```
SELECT artists.name, albums.Title
FROM artists
LEFT OUTER JOIN albums
    ON artists.ArtistId = albums.ArtistId
```

Outer Join

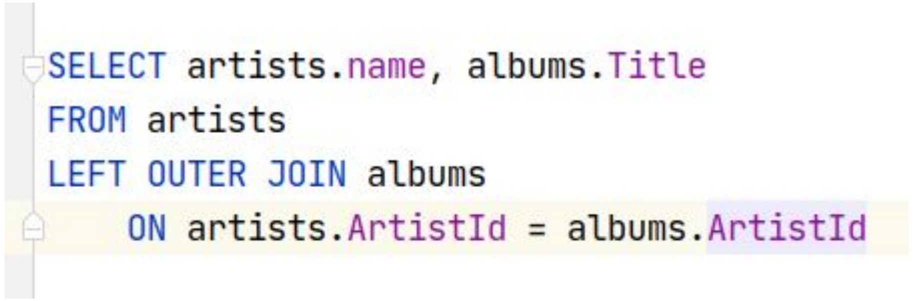
- Note that I am selecting values from both the artists as well as the albums table
- This query will return all artist/album combinations *as well as artists who have no albums, with the album value set to null*



```
SELECT artists.name, albums.Title
FROM artists
LEFT OUTER JOIN albums
    ON artists.ArtistId = albums.ArtistId
```

Outer Join

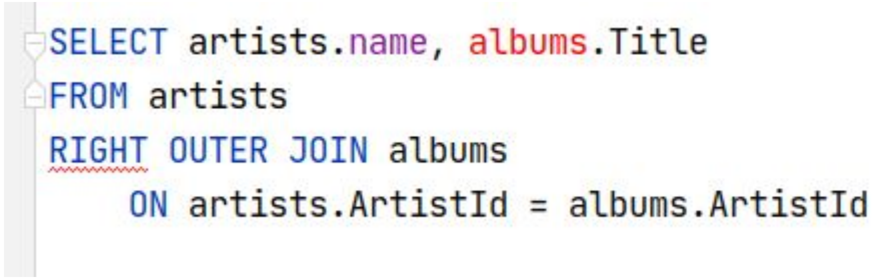
- If I changed this to an inner join, artists who had no albums would be excluded from the results



```
SELECT artists.name, albums.Title
FROM artists
LEFT OUTER JOIN albums
    ON artists.ArtistId = albums.ArtistId
```

Outer Join

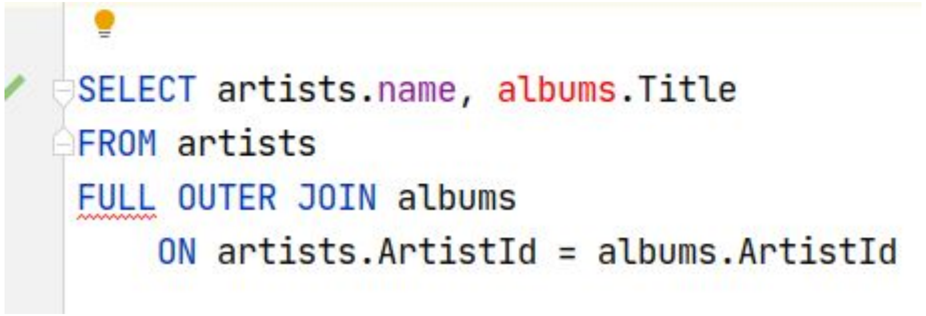
- We could change this to a RIGHT OUTER JOIN to include nulls from the right table (in this case albums)
- The artists table is considered “left” of the albums table
- Hence LEFT OUTER JOIN



```
SELECT artists.name, albums.Title
FROM artists
RIGHT OUTER JOIN albums
    ON artists.ArtistId = albums.ArtistId
```

Outer Join

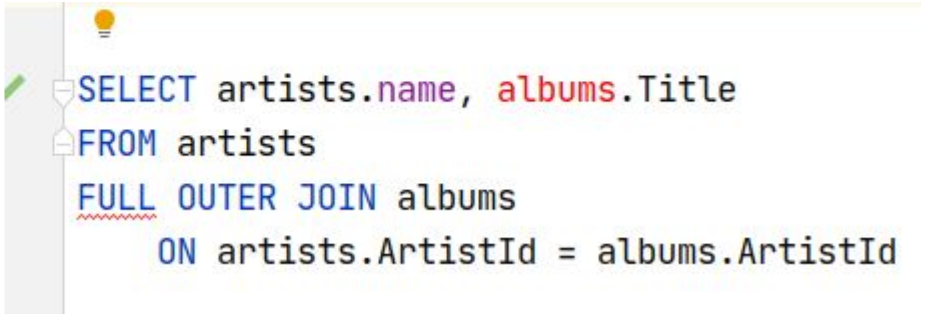
- And, finally, we could use a FULL OUTER JOIN to include null values from both tables



```
SELECT artists.name, albums.Title
FROM artists
FULL OUTER JOIN albums
    ON artists.ArtistId = albums.ArtistId
```

Outer Join

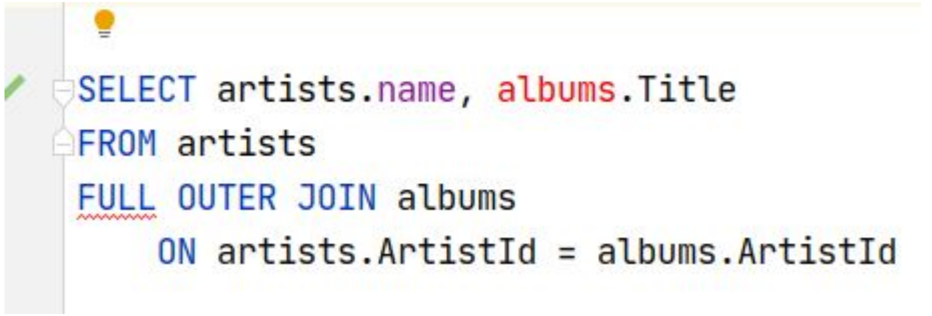
- As you might have noticed, SQLite does not support Right Outer Joins or Full Outer Joins
- In practice, Outer Joins are rare, and Right and Full outer joins are even rarer



```
SELECT artists.name, albums.Title
FROM artists
FULL OUTER JOIN albums
    ON artists.ArtistId = albums.ArtistId
```

Self Joins

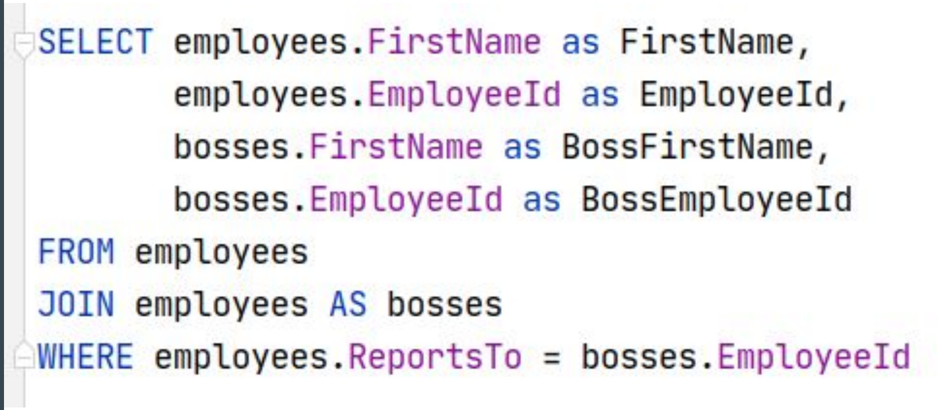
- Chinook DB has a self-referential table in it
- Employees have a reference to their boss via the ReportTo foriegn key
- How do we deal with that?



```
SELECT artists.name, albums.Title
FROM artists
FULL OUTER JOIN albums
    ON artists.ArtistId = albums.ArtistId
```

Self Joins

- It's pretty much the same technique using a JOIN
- However, you need to *alias* the table to a new name
- And you need to disambiguate the column names
- Both of these use the "as" syntax

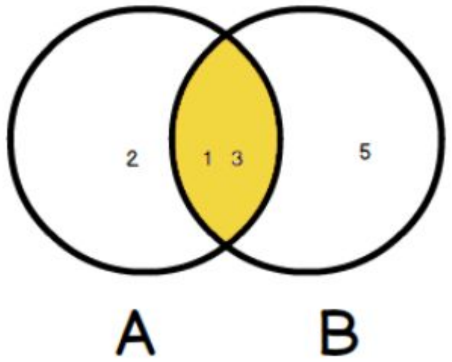


```
SELECT employees.FirstName as FirstName,
       employees.EmployeeId as EmployeeId,
       bosses.FirstName as BossFirstName,
       bosses.EmployeeId as BossEmployeeId
FROM employees
JOIN employees AS bosses
WHERE employees.ReportsTo = bosses.EmployeeId
```

Joins As Venn Diagrams

- One way to think about the types of joins is to use a ven diagram
- Inner Joins are the *intersection* of the two relational sets

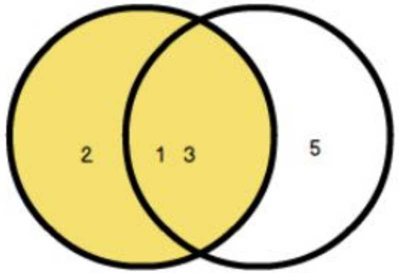
The following diagram illustrates the `INNER JOIN` clause:



Joins As Venn Diagrams

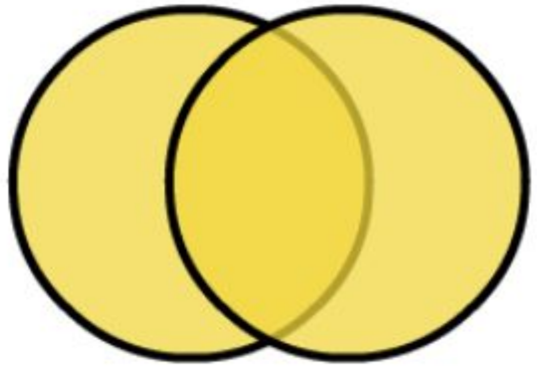
- Left Joins are all the left table and whatever matches in the right table

The following Venn Diagram illustrates the `LEFT JOIN` clause.



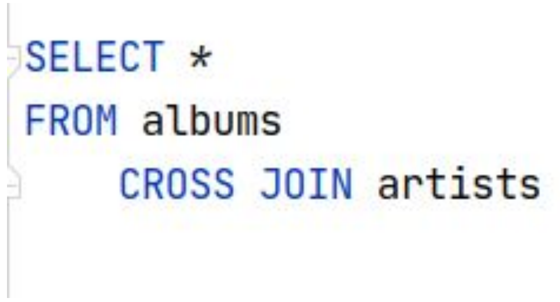
Joins As Venn Diagrams

- And Full Outer Joins include all rows from all tables
- I personally find it easier to just think in terms of nulls in the join condition:
    - Inner - ignore nulls
    - Outer - keep nulls



More Exotic Joins

- There are some more exotic joins, such as the cross join
- This creates all possible combinations of album rows and artist rows
- I have never used it, but I can imagine uses for it

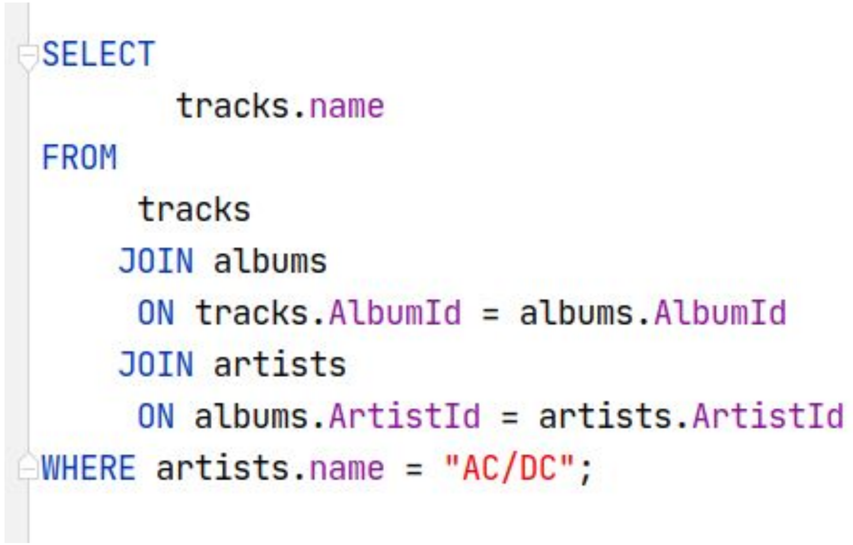


```
SELECT *
FROM albums
    CROSS JOIN artists
```

Joining Multiple Tables

- Sometimes you need to join across multiple tables to get what you want

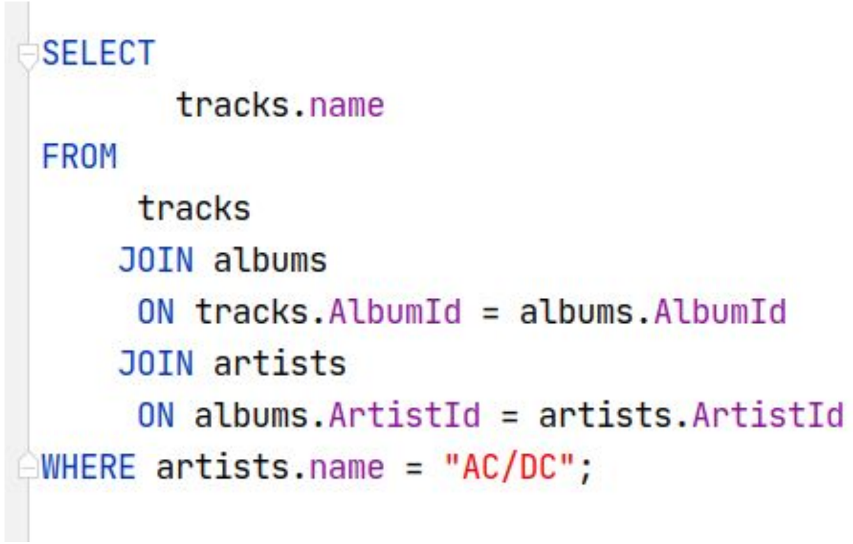
  "Give me the name of all tracks by AC/DC"



```
SELECT
        tracks.name
FROM
      tracks
     JOIN albums
      ON tracks.AlbumId = albums.AlbumId
     JOIN artists
      ON albums.ArtistId = artists.ArtistId
WHERE artists.name = "AC/DC";
```

Joining Multiple Tables

- Column naming can get tricky when you are joining multiple table
- You may have to use aliasing to get exactly what you want



```
SELECT
        tracks.name
FROM
      tracks
     JOIN albums
      ON tracks.AlbumId = albums.AlbumId
     JOIN artists
      ON albums.ArtistId = artists.ArtistId
WHERE artists.name = "AC/DC";
```

Joins Summary

- Joins are a way to correlate data across tables
- There are a few different types of joins
  - Inner joins are the most common
- A table with an FK to itself can be joined by *aliasing* the table in the join clause
- You can disambiguate table and column names using the AS keyword
- Joining across multiple tables involves multiple JOIN clauses

MONTANA
STATE UNIVERSITY