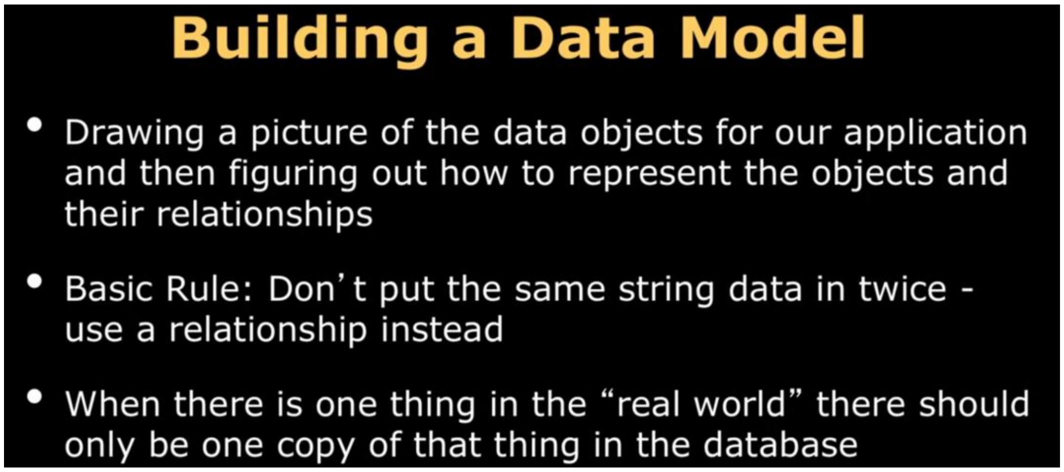
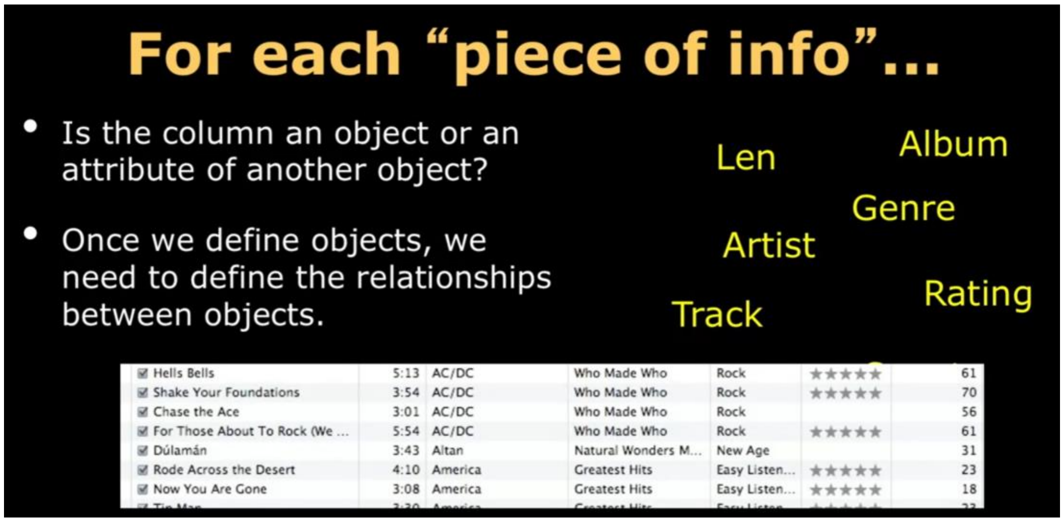
Relational Database

Relational Database Design



Building a Data Model

- Drawing a picture of the data objects for our application and then figuring out how to represent the objects and their relationships
- Basic Rule: Don't put the same string data in twice - use a relationship instead
- When there is one thing in the "real world" there should only be one copy of that thing in the database

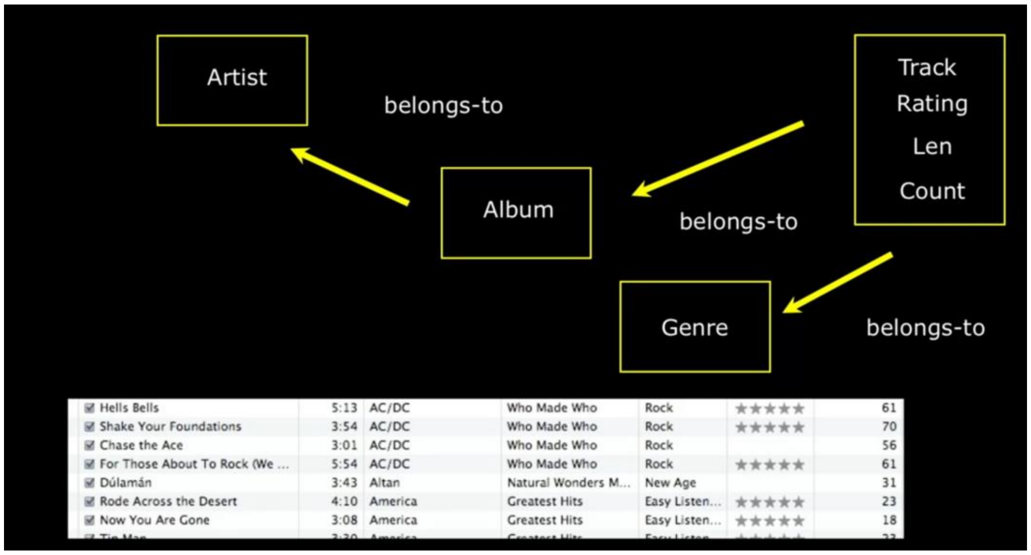


For each "piece of info"...

- Is the column an object or an attribute of another object?
- Once we define objects, we need to define the relationships between objects.

Len
Album
Genre
Artist
Rating
Track

Hells Bells	5:13	AC/DC	Who Made Who	Rock	★★★★★	61
Shake Your Foundations	3:54	AC/DC	Who Made Who	Rock	★★★★★	70
Chase the Ace	3:01	AC/DC	Who Made Who	Rock		56
For Those About To Rock (We ...	5:54	AC/DC	Who Made Who	Rock	★★★★★	61
Dúlamán	3:43	Altan	Natural Wonders M...	New Age		31
Rode Across the Desert	4:10	America	Greatest Hits	Easy Listen...	★★★★★	23
Now You Are Gone	3:08	America	Greatest Hits	Easy Listen...	★★★★★	18

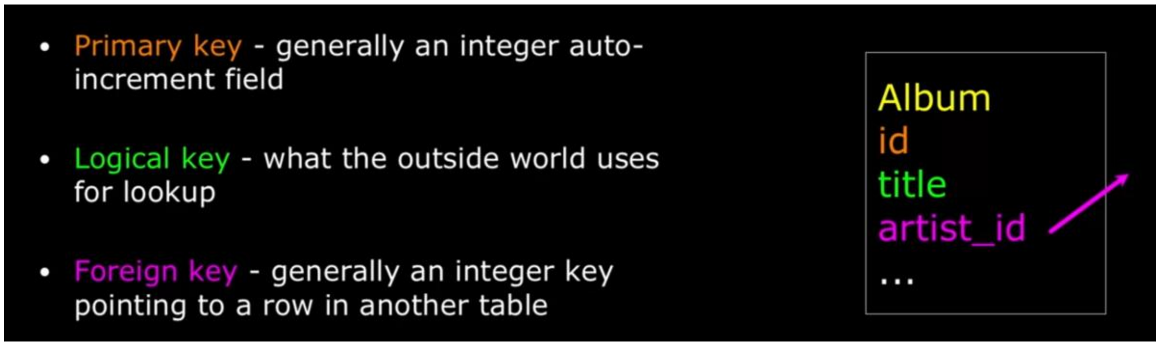


Hells Bells	5:13	AC/DC	Who Made Who	Rock	★★★★★	61
Shake Your Foundations	3:54	AC/DC	Who Made Who	Rock	★★★★★	70
Chase the Ace	3:01	AC/DC	Who Made Who	Rock		56
For Those About To Rock (We ...	5:54	AC/DC	Who Made Who	Rock	★★★★★	61
Dúlamán	3:43	Altan	Natural Wonders M...	New Age		31
Rode Across the Desert	4:10	America	Greatest Hits	Easy Listen...	★★★★★	23
Now You Are Gone	3:08	America	Greatest Hits	Easy Listen...	★★★★★	18

Keys in Relational database

There are three types of keys in relational database

1. **Primary key:** generally, an integer auto-increment field
2. **Logical key:** what the outside world uses for lookup
3. **Foreign key:** generally, an integer key pointing to a row in another table

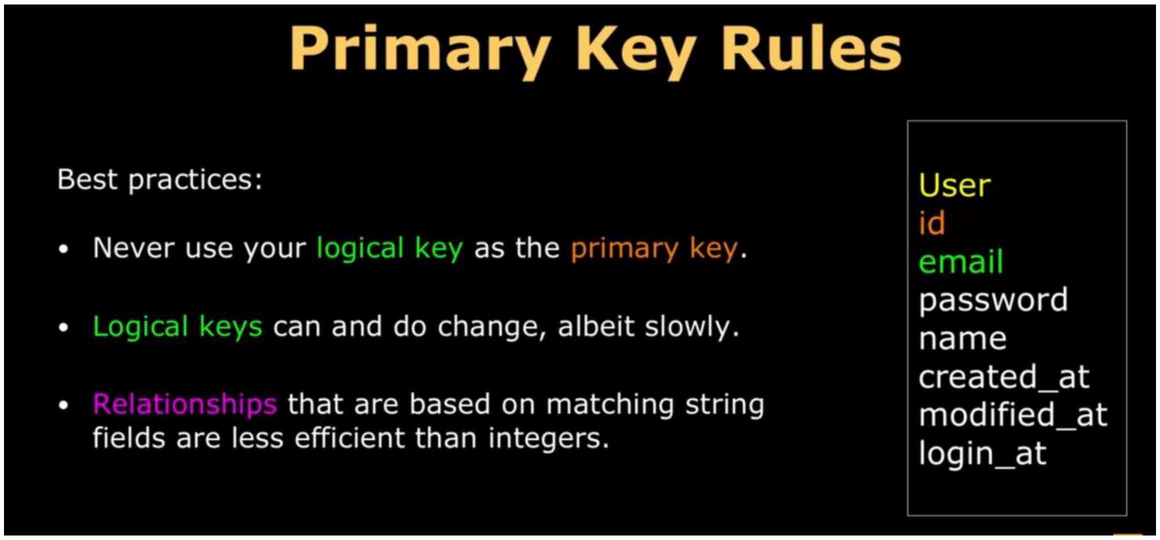


- Primary key - generally an integer auto-increment field
- Logical key - what the outside world uses for lookup
- Foreign key - generally an integer key pointing to a row in another table

Naming convention

1. ID are most commonly used primary key
2. For foreign key, use "tableName_fieldName" to name it

Primary key rules

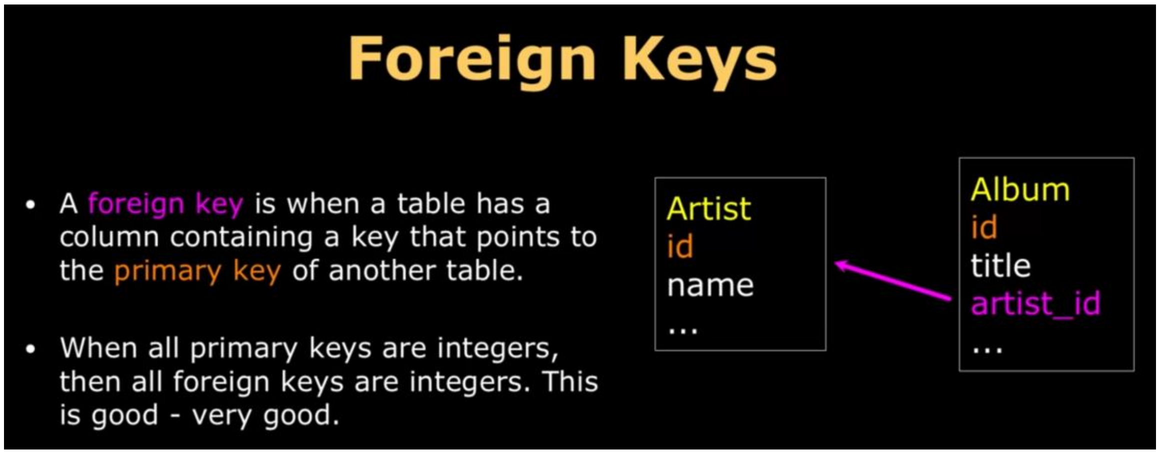


Primary Key Rules

Best practices:

- Never use your logical key as the primary key.
- Logical keys can and do change, albeit slowly.
- Relationships that are based on matching string fields are less efficient than integers.

Foreign key rules

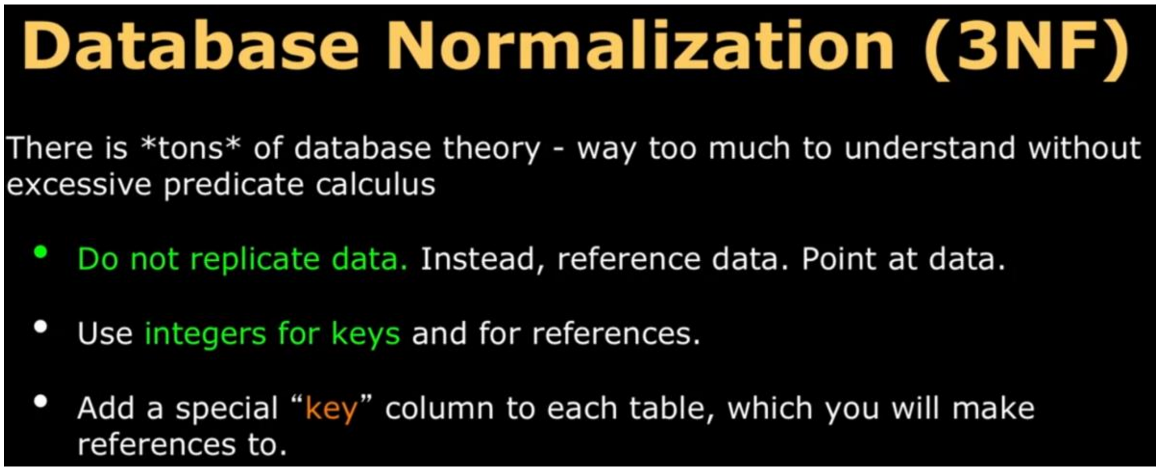


Foreign Keys

- A foreign key is when a table has a column containing a key that points to the primary key of another table.
- When all primary keys are integers, then all foreign keys are integers. This is good - very good.

Artist
id
name
...
Album
id
title
artist_id
...

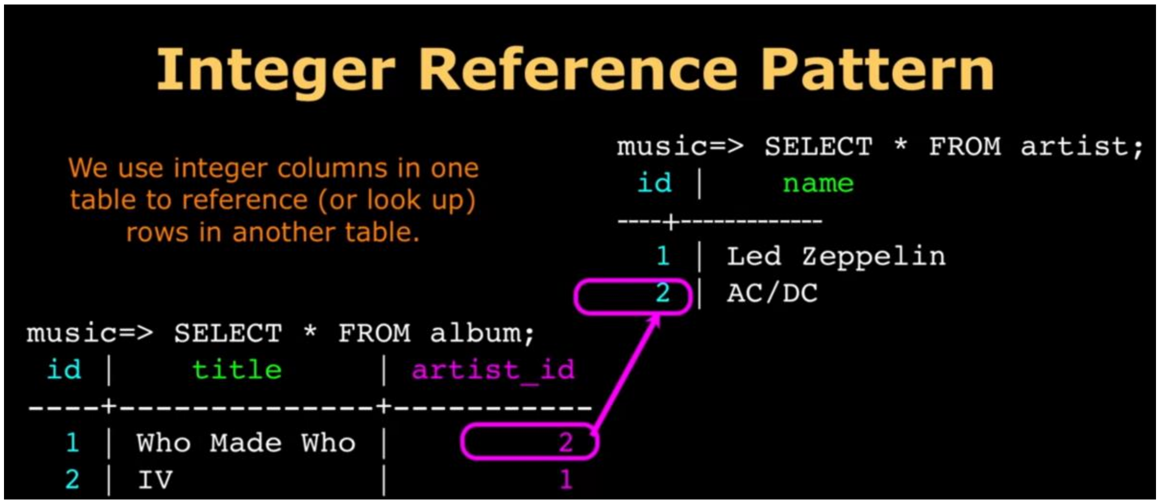
Database normalization



Database Normalization (3NF)

There is *tons* of database theory - way too much to understand without excessive predicate calculus

- Do not replicate data. Instead, reference data. Point at data.
- Use integers for keys and for references.
- Add a special “key” column to each table, which you will make references to.



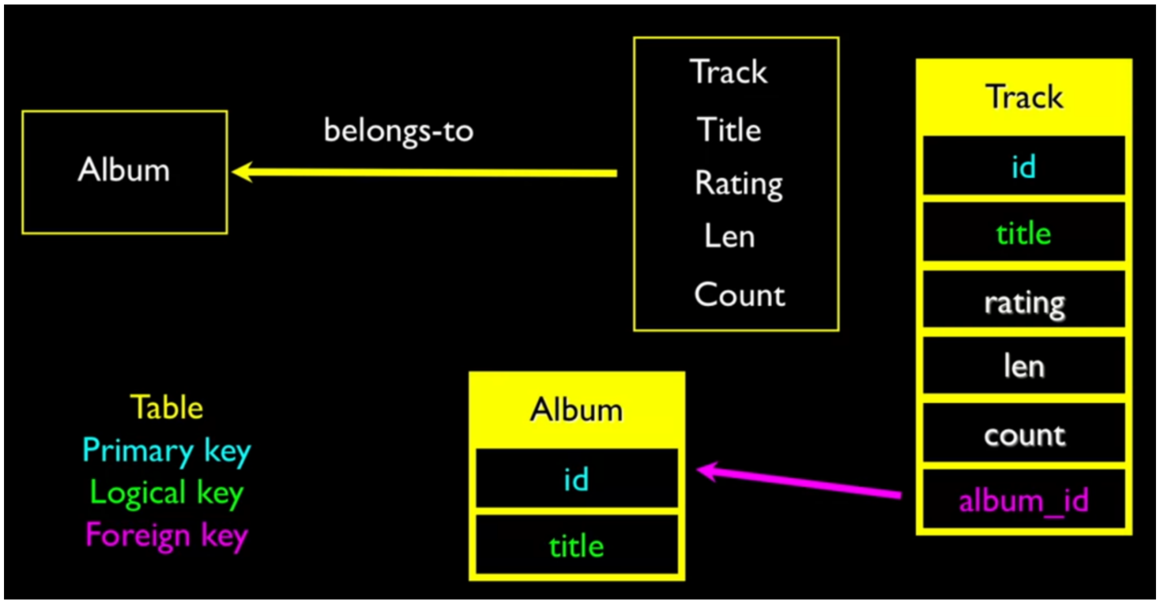
Integer Reference Pattern

We use integer columns in one table to reference (or look up) rows in another table.

```
music=> SELECT * FROM artist;
 id |     name
----+-------------
  1 | Led Zeppelin
  2 | AC/DC
```

```
music=> SELECT * FROM album;
 id |     title      | artist_id
----+----------------+-----------
  1 | Who Made Who   |         2
  2 | IV             |         1
```

For the relationship “belongs-to”, we use foreign key to represent it.

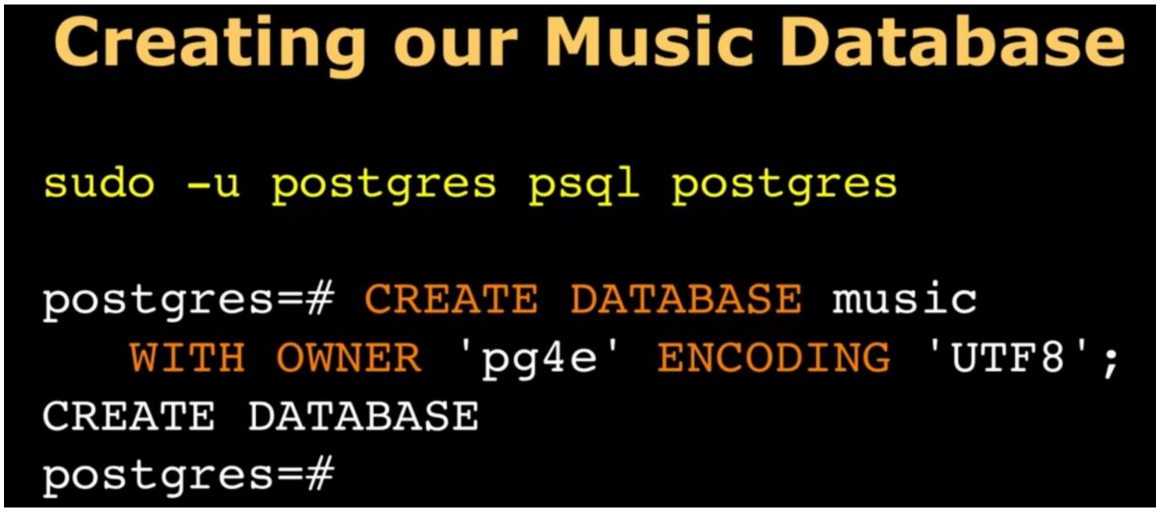


Examples

Creating our Music database

Sudo -u postgres psql postgres

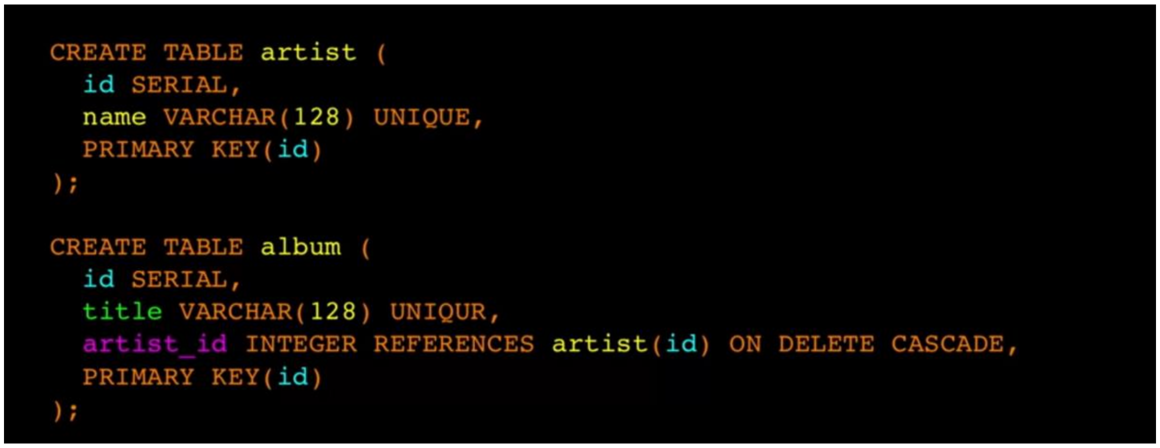
Postgres = # CREATE DATABASE music WITH OWNER ‘pg4e’ ENCOING ‘UTF8’;



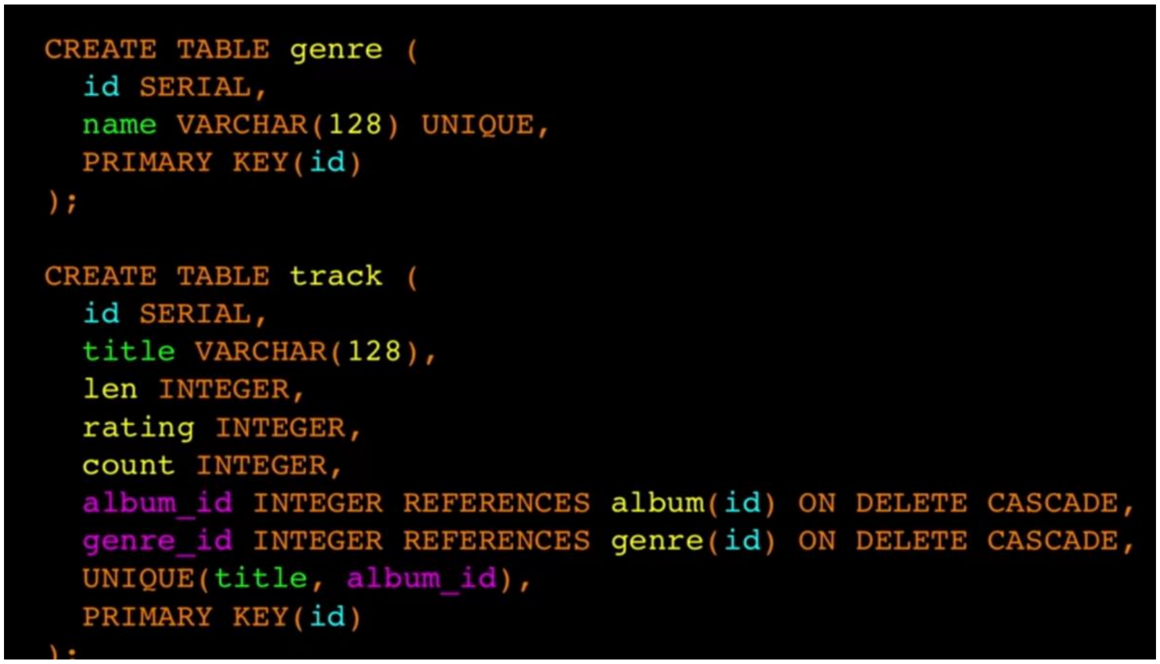
Creating our Music Database

```
sudo -u postgres psql postgres

postgres=# CREATE DATABASE music
   WITH OWNER 'pg4e' ENCODING 'UTF8';
CREATE DATABASE
postgres=#
```

Create tables

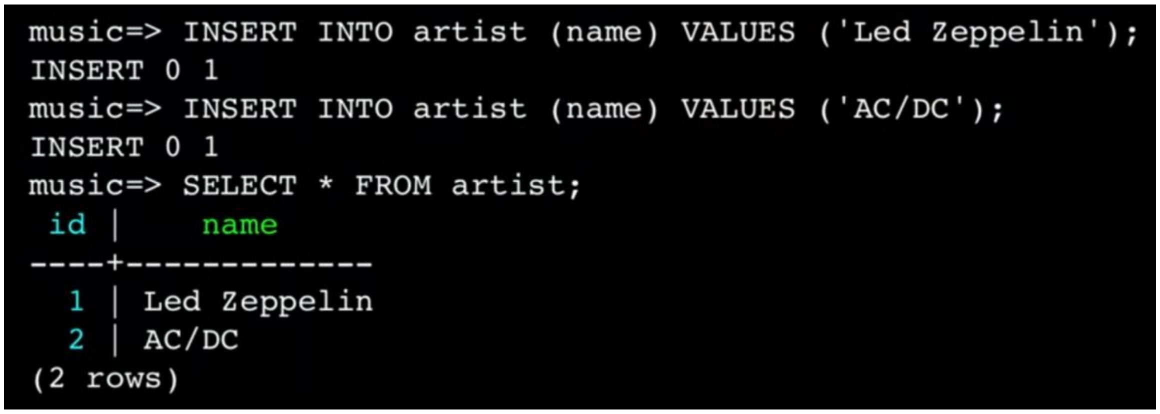


```
CREATE TABLE artist (
  id SERIAL,
  name VARCHAR(128) UNIQUE,
  PRIMARY KEY(id)
);

CREATE TABLE album (
  id SERIAL,
  title VARCHAR(128) UNIQUR,
  artist_id INTEGER REFERENCES artist(id) ON DELETE CASCADE,
  PRIMARY KEY(id)
);
```

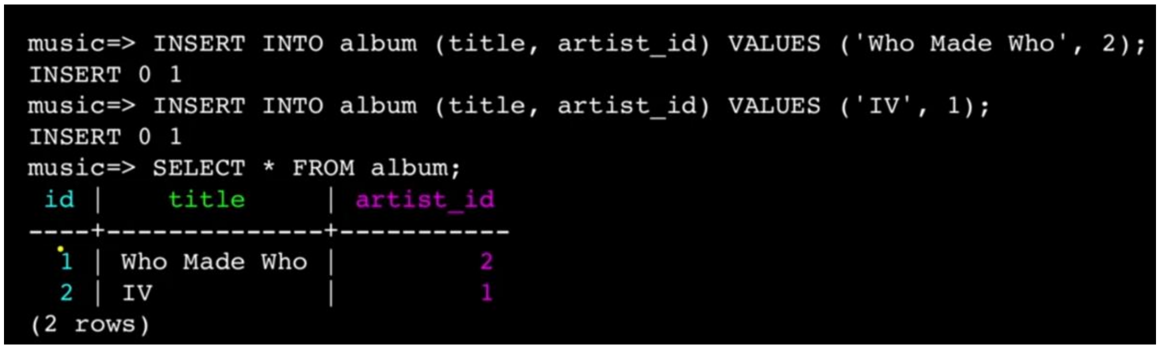


```
CREATE TABLE genre (
  id SERIAL,
  name VARCHAR(128) UNIQUE,
  PRIMARY KEY(id)
);

CREATE TABLE track (
  id SERIAL,
  title VARCHAR(128),
  len INTEGER,
  rating INTEGER,
  count INTEGER,
  album_id INTEGER REFERENCES album(id) ON DELETE CASCADE,
  genre_id INTEGER REFERENCES genre(id) ON DELETE CASCADE,
  UNIQUE(title, album_id),
  PRIMARY KEY(id)
```

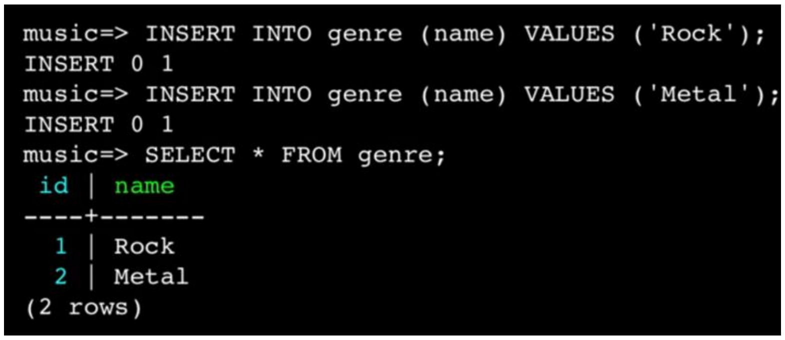
Insert data



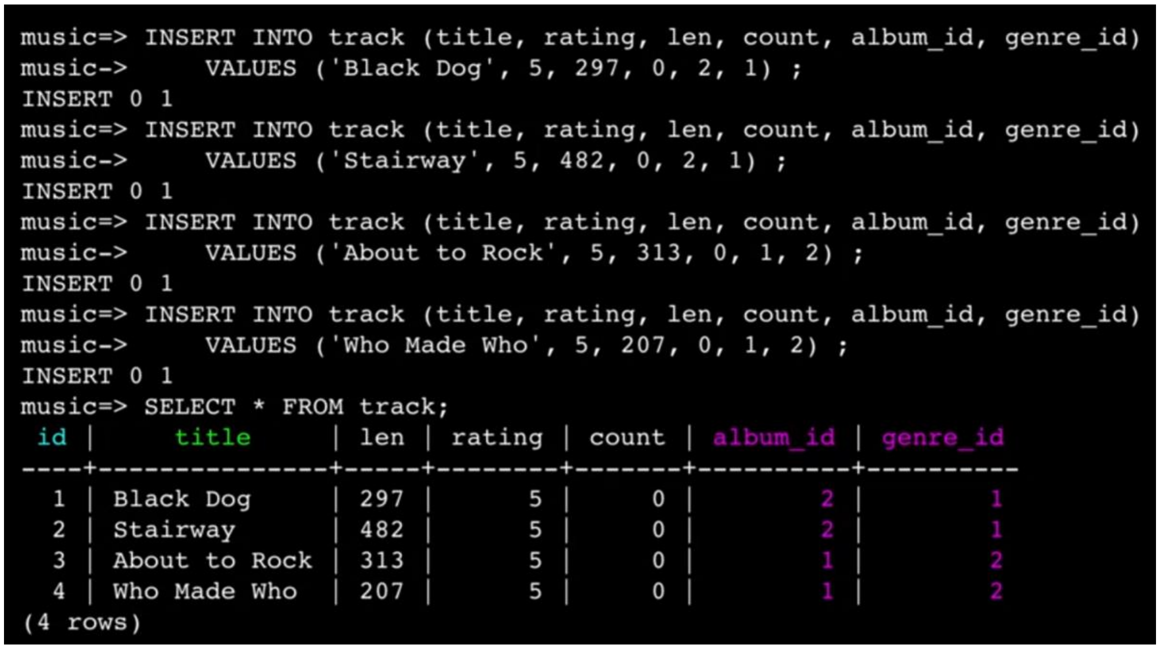
```
music=> INSERT INTO artist (name) VALUES ('Led Zeppelin');
INSERT 0 1
music=> INSERT INTO artist (name) VALUES ('AC/DC');
INSERT 0 1
music=> SELECT * FROM artist;
 id |     name
----+-------------
  1 | Led Zeppelin
  2 | AC/DC
(2 rows)
```



```
music=> INSERT INTO album (title, artist_id) VALUES ('Who Made Who', 2);
INSERT 0 1
music=> INSERT INTO album (title, artist_id) VALUES ('IV', 1);
INSERT 0 1
music=> SELECT * FROM album;
 id |     title      | artist_id
----+---------------+-----------
  1 | Who Made Who  |         2
  2 | IV            |         1
(2 rows)
```

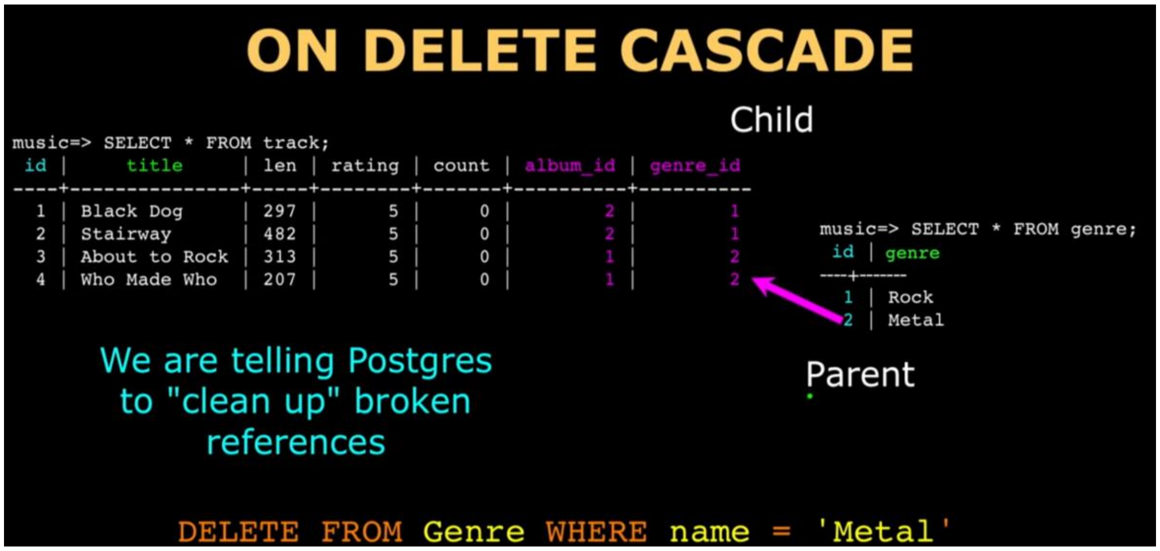


```
music=> INSERT INTO genre (name) VALUES ('Rock');
INSERT 0 1
music=> INSERT INTO genre (name) VALUES ('Metal');
INSERT 0 1
music=> SELECT * FROM genre;
 id | name
----+-------
  1 | Rock
  2 | Metal
(2 rows)
```



```
music=> INSERT INTO track (title, rating, len, count, album_id, genre_id)
music->     VALUES ('Black Dog', 5, 297, 0, 2, 1) ;
INSERT 0 1
music=> INSERT INTO track (title, rating, len, count, album_id, genre_id)
music->     VALUES ('Stairway', 5, 482, 0, 2, 1) ;
INSERT 0 1
music=> INSERT INTO track (title, rating, len, count, album_id, genre_id)
music->     VALUES ('About to Rock', 5, 313, 0, 1, 2) ;
INSERT 0 1
music=> INSERT INTO track (title, rating, len, count, album_id, genre_id)
music->     VALUES ('Who Made Who', 5, 207, 0, 1, 2) ;
INSERT 0 1
music=> SELECT * FROM track;
 id |     title      | len | rating | count | album_id | genre_id
----+----------------+-----+--------+-------+----------+----------
  1 | Black Dog      | 297 |      5 |     0 |        2 |        1
  2 | Stairway       | 482 |      5 |     0 |        2 |        1
  3 | About to Rock  | 313 |      5 |     0 |        1 |        2
  4 | Who Made Who   | 207 |      5 |     0 |        1 |        2
(4 rows)
```

ON DELETE CASCADE



ON DELETE CASCADE

Child

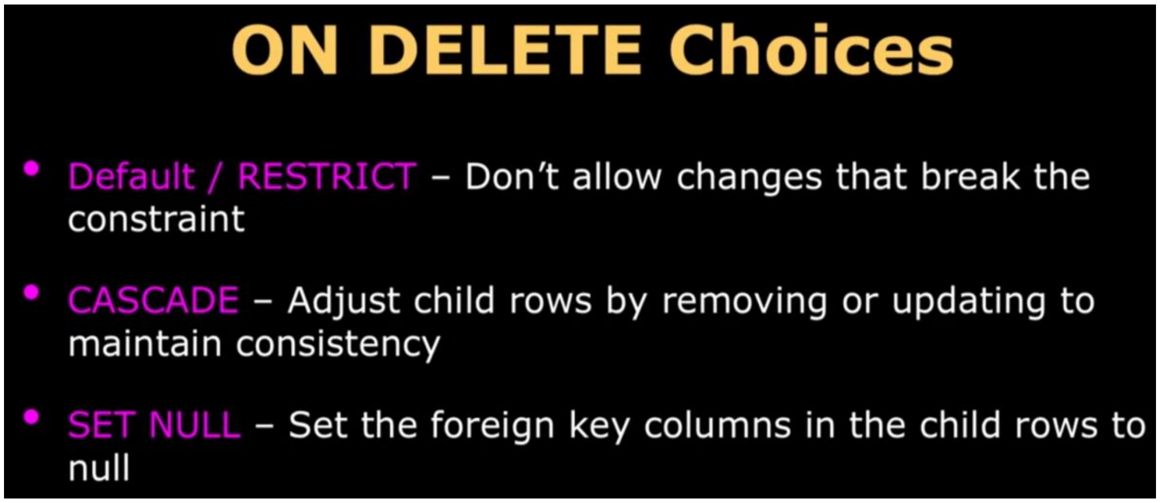
```
music=> SELECT * FROM track;
 id |     title      | len | rating | count | album_id | genre_id
----+----------------+-----+--------+-------+----------+----------
  1 | Black Dog      | 297 |      5 |     0 |        2 |        1
  2 | Stairway       | 482 |      5 |     0 |        2 |        1
  3 | About to Rock  | 313 |      5 |     0 |        1 |        2
  4 | Who Made Who   | 207 |      5 |     0 |        1 |        2
```

```
music=> SELECT * FROM genre;
 id | genre
----+-------
  1 | Rock
  2 | Metal
```

Parent

We are telling Postgres to "clean up" broken references

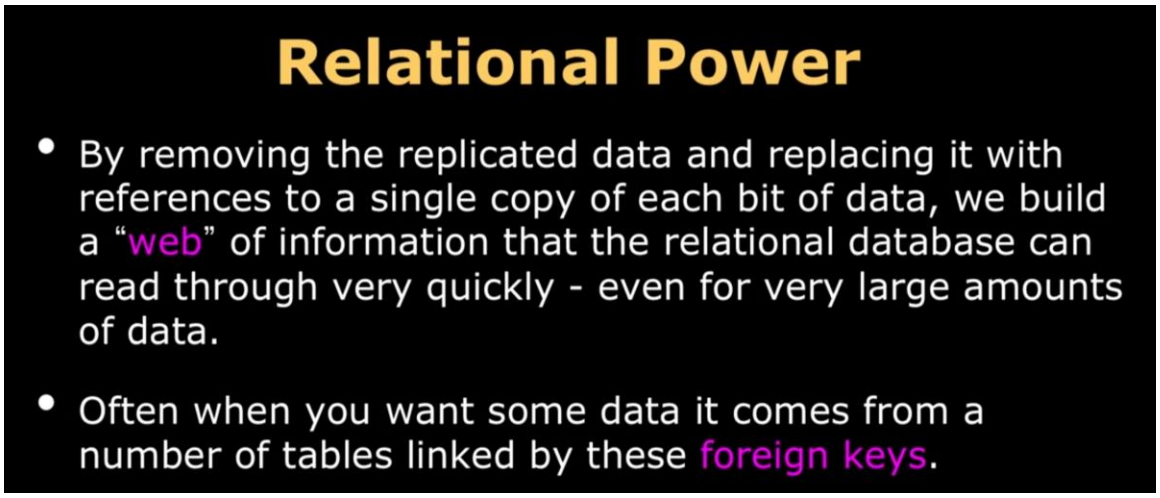
```
DELETE FROM Genre WHERE name = 'Metal'
```



ON DELETE Choices

- Default / RESTRICT – Don't allow changes that break the constraint
- CASCADE – Adjust child rows by removing or updating to maintain consistency
- SET NULL – Set the foreign key columns in the child rows to null

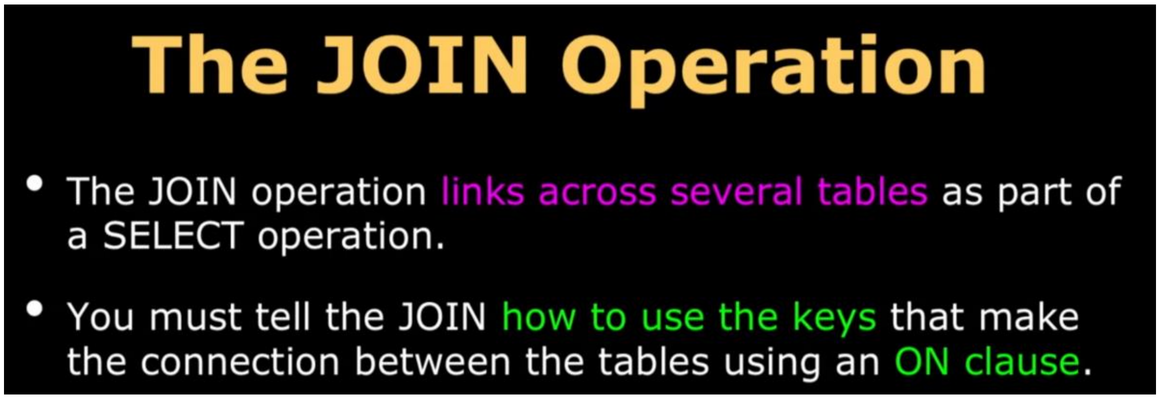
Using JOIN Across Tables



Relational Power

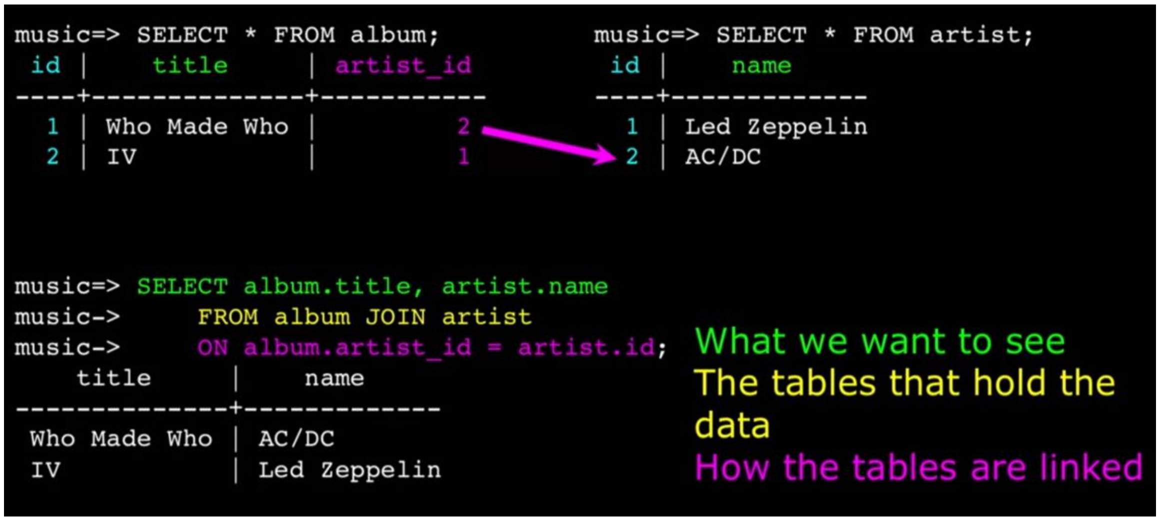
- By removing the replicated data and replacing it with references to a single copy of each bit of data, we build a "web" of information that the relational database can read through very quickly - even for very large amounts of data.
- Often when you want some data it comes from a number of tables linked by these foreign keys.

JOIN



The JOIN Operation

- The JOIN operation links across several tables as part of a SELECT operation.
- You must tell the JOIN how to use the keys that make the connection between the tables using an ON clause.

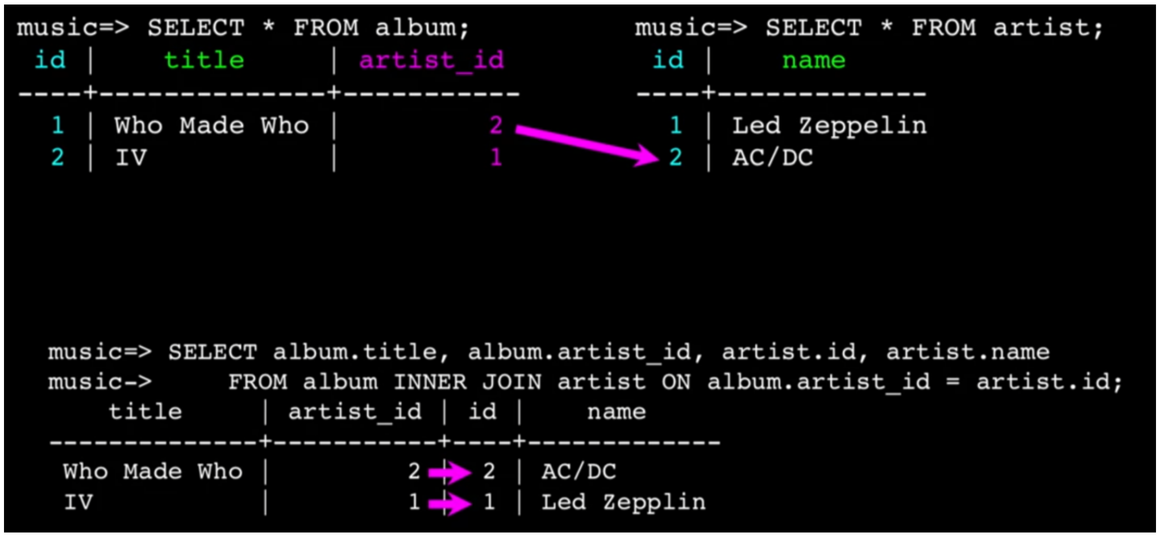


```
music=> SELECT * FROM album;            music=> SELECT * FROM artist;
 id |     title      | artist_id         id |     name
----+----------------+-----------       ----+-------------
  1 | Who Made Who   |         2          1 | Led Zeppelin
  2 | IV             |         1          2 | AC/DC


music=> SELECT album.title, artist.name
music->     FROM album JOIN artist
music->     ON album.artist_id = artist.id;
    title       |     name
----------------+-------------
 Who Made Who   | AC/DC
 IV             | Led Zeppelin
```

What we want to see
The tables that hold the data
How the tables are linked

INNER JOIN

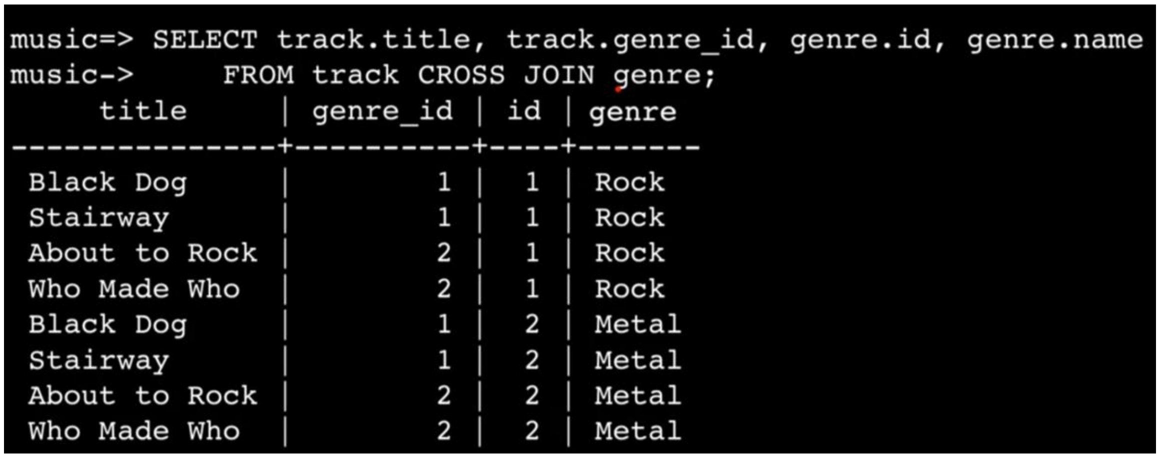
Join when it matches



```
music=> SELECT * FROM album;            music=> SELECT * FROM artist;
 id |     title      | artist_id         id |     name
----+----------------+-----------       ----+-------------
  1 | Who Made Who   |         2          1 | Led Zeppelin
  2 | IV             |         1          2 | AC/DC


music=> SELECT album.title, album.artist_id, artist.id, artist.name
music->     FROM album INNER JOIN artist ON album.artist_id = artist.id;
    title       | artist_id | id |     name
----------------+-----------+----+-------------
 Who Made Who   |         2    2 | AC/DC
 IV             |         1    1 | Led Zepplin
```

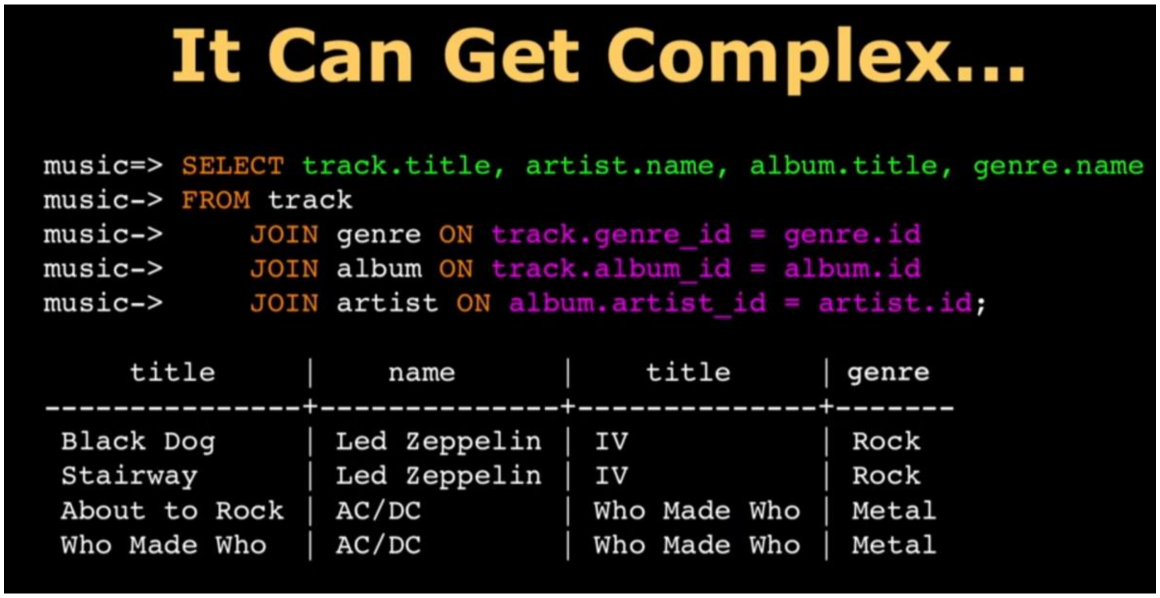
CROSS JOIN

Join every possible combination



```
music=> SELECT track.title, track.genre_id, genre.id, genre.name
music->     FROM track CROSS JOIN genre;
     title     | genre_id | id | genre
---------------+----------+----+-------
 Black Dog     |        1 |  1 | Rock
 Stairway      |        1 |  1 | Rock
 About to Rock |        2 |  1 | Rock
 Who Made Who  |        2 |  1 | Rock
 Black Dog     |        1 |  2 | Metal
 Stairway      |        1 |  2 | Metal
 About to Rock |        2 |  2 | Metal
 Who Made Who  |        2 |  2 | Metal
```

Complex example



It Can Get Complex...

```
music=> SELECT track.title, artist.name, album.title, genre.name
music-> FROM track
music->     JOIN genre ON track.genre_id = genre.id
music->     JOIN album ON track.album_id = album.id
music->     JOIN artist ON album.artist_id = artist.id;

     title     |     name     |    title     | genre
---------------+--------------+--------------+-------
 Black Dog     | Led Zeppelin | IV           | Rock
 Stairway      | Led Zeppelin | IV           | Rock
 About to Rock | AC/DC        | Who Made Who | Metal
 Who Made Who  | AC/DC        | Who Made Who | Metal
```