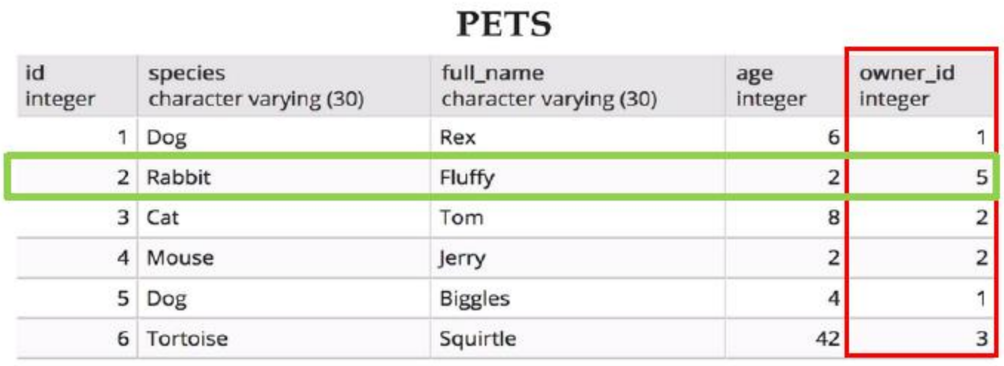
Relational Database

These are by far the most popular type of databases around today, the most popular databases such as Oracle, MySQL or SQL server, these are all relational databases. PostgreSQL is also a relational database.

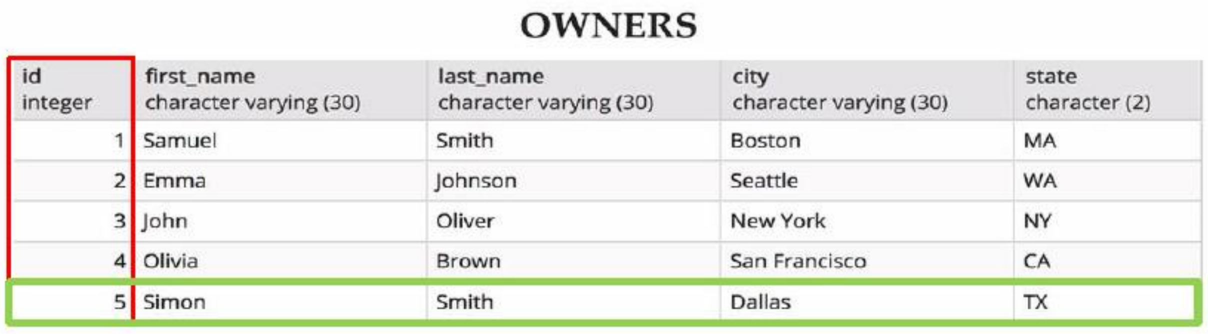
In a relational database, data is stored in tables which are referred to as relations. The tables are just like tables in excel. They contain columns and rows of data these tables can be linked together in relationships.

It is these relationships between different tables in the database that makes relational databases so powerful they allow us to identify a piece of data in relation to another piece of data within the database. For example, there are two tables as below:



PETS

id integer	species character varying (30)	full_name character varying (30)	age integer	owner_id integer
1	Dog	Rex	6	1
2	Rabbit	Fluffy	2	5
3	Cat	Tom	8	2
4	Mouse	Jerry	2	2
5	Dog	Biggles	4	1
6	Tortoise	Squirtle	42	3



OWNERS

id integer	first_name character varying (30)	last_name character varying (30)	city character varying (30)	state character (2)
1	Samuel	Smith	Boston	MA
2	Emma	Johnson	Seattle	WA
3	John	Oliver	New York	NY
4	Olivia	Brown	San Francisco	CA
5	Simon	Smith	Dallas	TX

These two tables are linked and are set to be in a database relationship from these two columns. We can match the data and see who owns which pets.

Relational Database Management Systems (RDBMS)

When people talk about databases they are normally talking about relational database management systems, the popular databases such as Oracle, MySQL, SQL Server and PostgreSQL are actually relational database management systems or RDBMS. RDBMS allows us to interact with the relational database they provide tools and a gooey or graphical user interface to interact with the database PostgreSQL is the relational database management system we will be using for this course and it is the most powerful and popular open source database in the world.

SQL stands for Structured Query Language and it is the language used to talk or interact with relational databases. We can write queries using SQL to create tables within the database as well as insert and modify data and even retrieve data from a database and much more as well.
The syntax for SQL is very similar across the different database systems as well. What we learn in this course won't just apply to PostgreSQL, but can also be used for MySQL, Oracle and other relational databases.

Three kinds of keys

Primary key: generally an integer auto-increment field

Foreign key: a foreign key is a field (or collection of fields) in one table, that refers to the PRIMARY KEY in another table.

Logical key: logical keys are anything that define relationships between data and tables. For example, a logical key could be a **pet_species**.

Primary key rules

A primary key is the column that contains values that are uniquely identified in each row in a table.

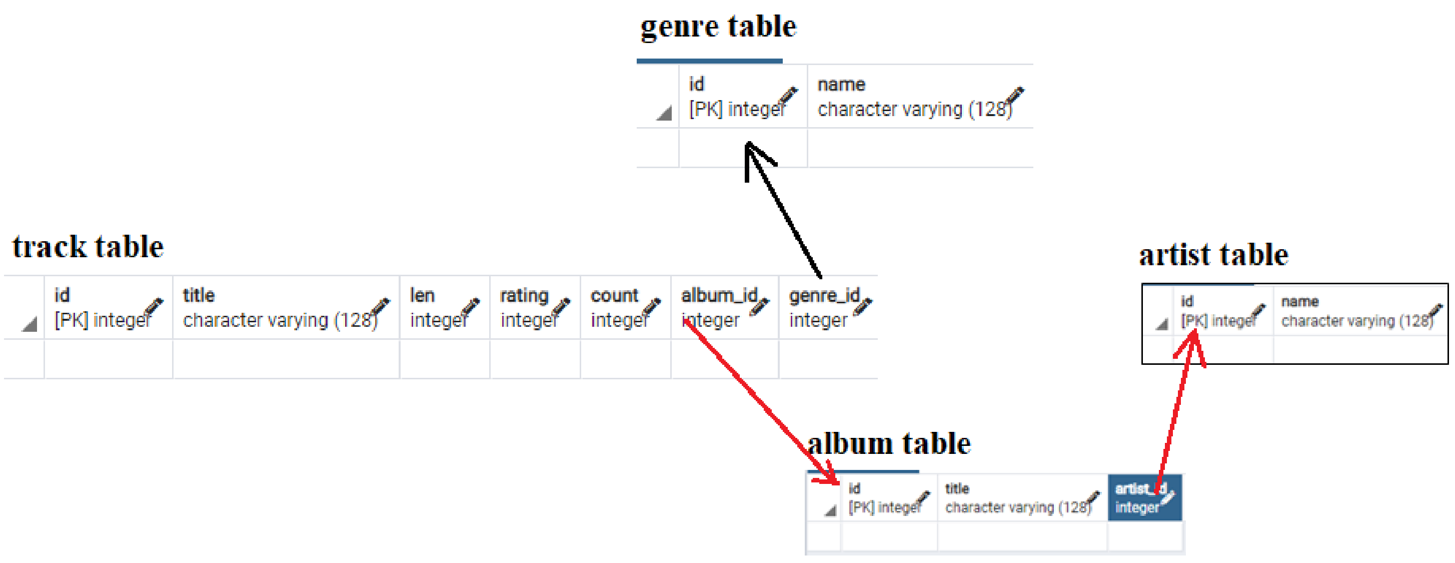
Best practices:

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

Building tables

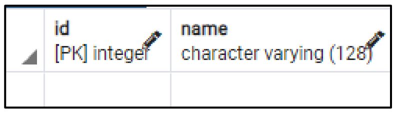
We can create a relationship between two tables by REFERENCES to the next table. REFERENCES refers to a foreign key. We have to create the tables first in order to make it into foreign key.

Example) Create a schema of the following database with the following primary and foreign keys:



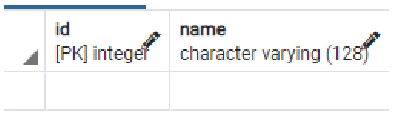
Let create artist and the genre table with the following values:

artist table



id [PK] integer	name character varying (128)

genre table



id [PK] integer	name character varying (128)

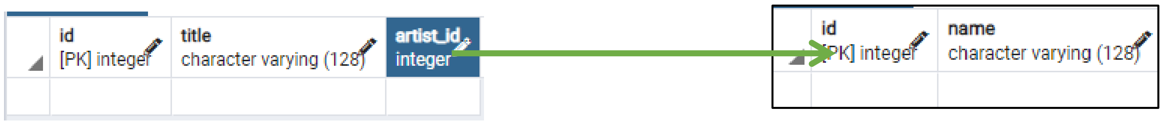
In order to make a connection, between the album and the artist table, we use the REFERENCES keyword. The `references` keyword is used to define which table and column is used in a foreign key relationship.

Syntax:

Name_Foreign_key key_value **REFERENCES** name_foreign_table(primary_key_name);

For our schema, let us say that we want to link **album** table to the **artist** table:

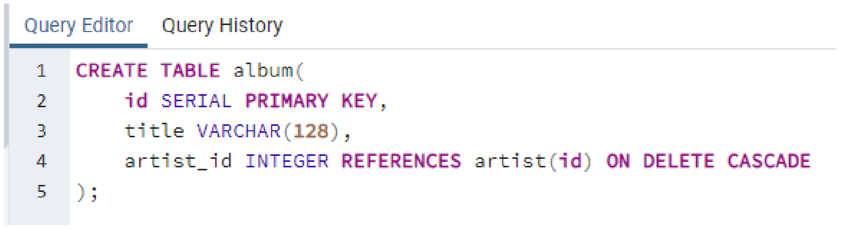
album table **artist** table



id [PK] integer	title character varying (128)	artist_id integer

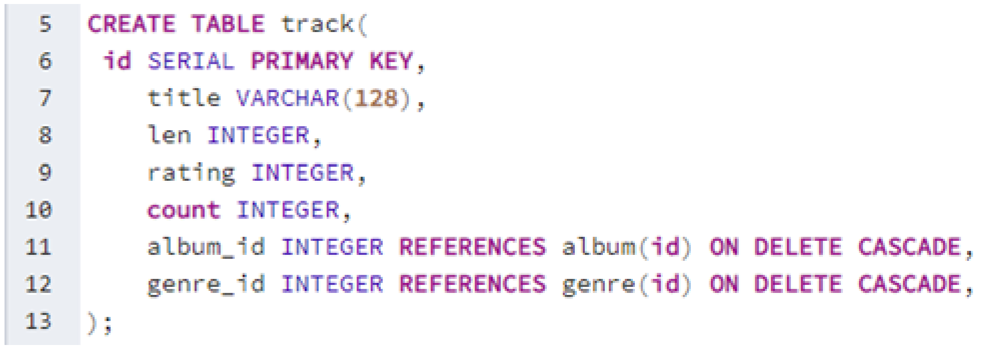
id [PK] integer	name character varying (128)

for this, after creating the table artist, we can create the album table and have a foreign key to artist table:



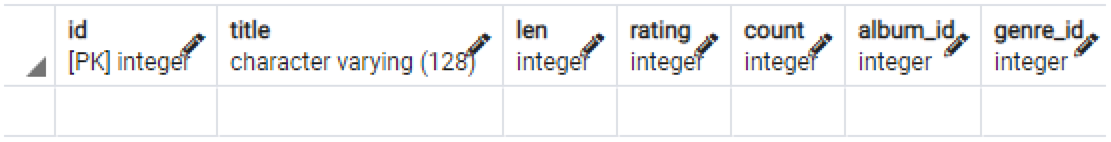
```
CREATE TABLE album(
    id SERIAL PRIMARY KEY,
    title VARCHAR(128),
    artist_id INTEGER REFERENCES artist(id) ON DELETE CASCADE
);
```

Now, let us create a **track** table with a foreign key to **album** and **genre** table



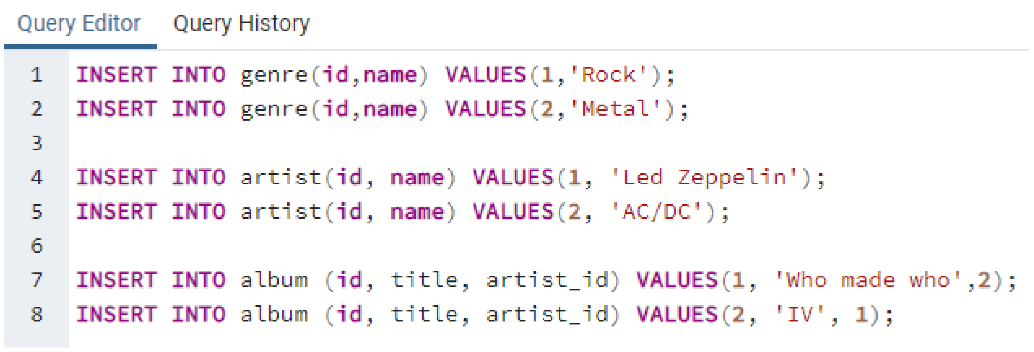
```
CREATE TABLE track(
 id SERIAL PRIMARY KEY,
    title VARCHAR(128),
    len INTEGER,
    rating INTEGER,
    count INTEGER,
    album_id INTEGER REFERENCES album(id) ON DELETE CASCADE,
    genre_id INTEGER REFERENCES genre(id) ON DELETE CASCADE,
);
```

Refreshing the **track** table we have:



	id [PK] integer	title character varying (128)	len integer	rating integer	count integer	album_id integer	genre_id integer

INSERTING DATA



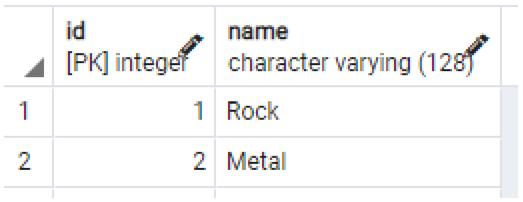
```
INSERT INTO genre(id,name) VALUES(1,'Rock');
INSERT INTO genre(id,name) VALUES(2,'Metal');

INSERT INTO artist(id, name) VALUES(1, 'Led Zeppelin');
INSERT INTO artist(id, name) VALUES(2, 'AC/DC');

INSERT INTO album (id, title, artist_id) VALUES(1, 'Who made who',2);
INSERT INTO album (id, title, artist_id) VALUES(2, 'IV', 1);
```

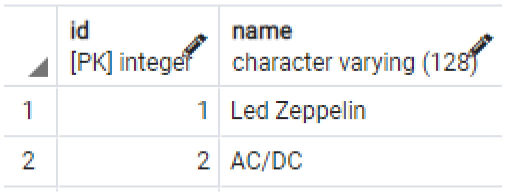
Refreshing our tables, we have:

genre table



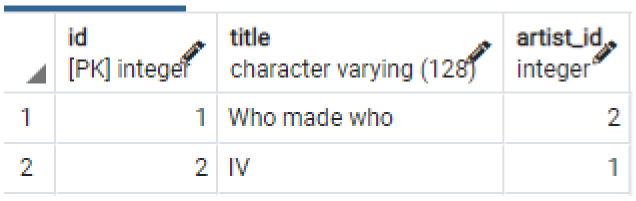
	id [PK] integer	name character varying (128)
1	1	Rock
2	2	Metal

artist table



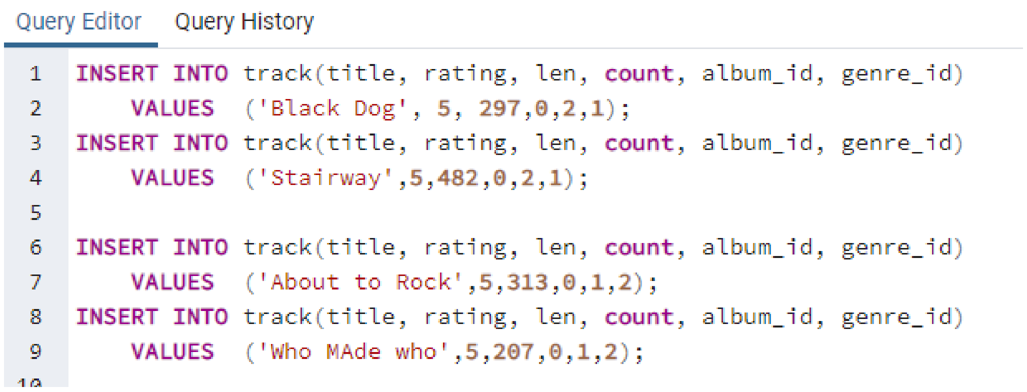
	id [PK] integer	name character varying (128)
1	1	Led Zeppelin
2	2	AC/DC

album table



	id [PK] integer	title character varying (128)	artist_id integer
1	1	Who made who	2
2	2	IV	1

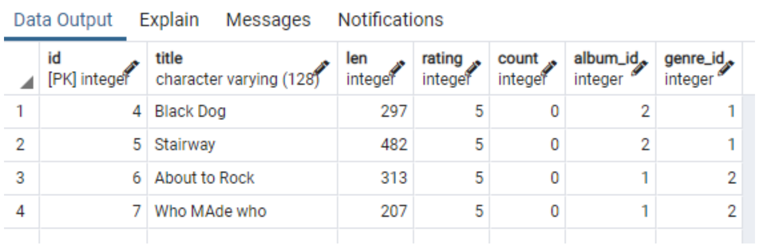
Now, let us insert some values to **track** album



```
INSERT INTO track(title, rating, len, count, album_id, genre_id)
    VALUES  ('Black Dog', 5, 297,0,2,1);
INSERT INTO track(title, rating, len, count, album_id, genre_id)
    VALUES  ('Stairway',5,482,0,2,1);

INSERT INTO track(title, rating, len, count, album_id, genre_id)
    VALUES  ('About to Rock',5,313,0,1,2);
INSERT INTO track(title, rating, len, count, album_id, genre_id)
    VALUES  ('Who MAde who',5,207,0,1,2);
```

Refreshing the **track** table



	id [PK] integer	title character varying (128)	len integer	rating integer	count integer	album_id integer	genre_id integer
1	4	Black Dog	297	5	0	2	1
2	5	Stairway	482	5	0	2	1
3	6	About to Rock	313	5	0	1	2
4	7	Who MAde who	207	5	0	1	2

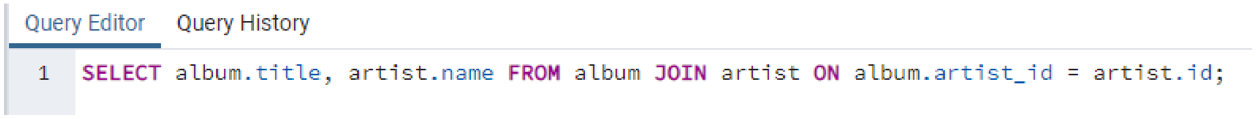
JOIN clause

The SQL JOIN clause is used to combine records from two or more tables in a database. A JOIN is a means for combining fields from two tables by using values common to each.

Syntax:

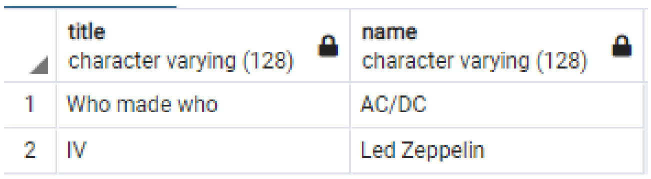
SELECT table_colum_name, table_column_name **FROM** table_name **JOIN** foreign_table_name **ON** table_foreign_key_name = foreign_tabel_primary_key_name

For example, we can display the name of the album title with the artist name in one table using JOIN as:



```
SELECT album.title, artist.name FROM album JOIN artist ON album.artist_id = artist.id;
```

The resulting table will be:



	title character varying (128)	name character varying (128)
1	Who made who	AC/DC
2	IV	Led Zeppelin