SQL Joins

Tracking information that isn't a single value.

In relationship to other data.

Let's go to the movies

```
pgcli SuncoastMovies
```

Primary Key

- Relating information between multiple tables.
- Need a way to uniquely identify a row of data in a table.
- Known as the table's `PRIMARY KEY`.
- Uniquely identifies the row and also cannot repeat.

Primary Key

- You might have an existing column that you feel uniquely identify the row.
- You might think that the movie's title would uniquely identify the movie. However, we know that sometimes a movie's title changes during the production, or even uses the name of a movie that has existed in the past.
- So if we used the title to uniquely identify it we would run into many issues.

SERIAL Data Type

- Databases provide their own way of supplying a unique value for each row in the database.
- In `Postgres` we call this the SERIAL column.
- A `SERIAL` data type will begin at `1` and increase for each new row.
- Values are never reused or repeated.

Defining a Primary Key for our Movies

- Use `Id` column name. This is a common pattern.
- Define the data type as: `SERIAL`.
- Denote that this column is part of the `PRIMARY KEY`

```
       Data Type
       |
       |      Mark this column as part of the PRIMARY KEY
       |      |
       |      |
       v      v---------v
"Id"  SERIAL PRIMARY KEY,
CREATE TABLE "Movies" (
  "Id"                   SERIAL PRIMARY KEY,
  "Title"                TEXT NOT NULL,
  "PrimaryDirector"      TEXT,
  "YearReleased"         INT,
  "Genre"                TEXT
);
```

Add some movies

```sql
INSERT INTO "Movies" ("Title",  "PrimaryDirector", "YearReleased", "Genre")
VALUES ('The Lost World', 'Steven Spielberg', 1997, 'sci-fi');

INSERT INTO "Movies" ("Title",  "PrimaryDirector", "YearReleased", "Genre")
VALUES ('Pirates of the Caribbean: The Curse of the Black Pearl', 'Gore Verbinski', 2003, 'fantasy');

INSERT INTO "Movies" ("Title",  "PrimaryDirector", "YearReleased", "Genre")
VALUES ('Harry Potter and Goblet of Fire', 'Mike Newell', 2005, 'fantasy');

INSERT INTO "Movies" ("Title",  "PrimaryDirector", "YearReleased", "Genre")
VALUES ('The Hobbit: An Unexpected Journey', 'Peter Jackson', 2012, 'fantasy');
```

Foreign Keys

- In order to keep track of the rating for any given movie we will add a single table, named `Ratings` that will store the name of the rating.
- Since we also want to uniquely identify the ratings, we'll ensure this table also has a serial primary key.

Ratings

```
CREATE TABLE "Ratings" (
  "Id" SERIAL PRIMARY KEY,
  "Description" TEXT
);
```

Let's insert some ratings:

```sql
INSERT INTO "Ratings" ("Description") VALUES ('G');
INSERT INTO "Ratings" ("Description") VALUES ('PG');
INSERT INTO "Ratings" ("Description") VALUES ('PG-13');
INSERT INTO "Ratings" ("Description") VALUES ('R');
```

Actors

- Let's also add a table to keep information about our actors.
- For this table we want to know the full name of the actor and their birthday.
- We'll also create an `Id` that is a `PRIMARY KEY` and is `SERIAL`.

```
CREATE TABLE "Actors" (
  "Id"         SERIAL PRIMARY KEY,
  "FullName" TEXT NOT NULL,
  "Birthday" DATE
);
```

Actors

```
INSERT INTO "Actors" ("FullName", "Birthday")
VALUES ('Orlando Bloom', '1977-01-13');

INSERT INTO "Actors" ("FullName", "Birthday")
VALUES ('Warwick Davis', '1970-02-03');

INSERT INTO "Actors" ("FullName", "Birthday")
VALUES ('Martin Freeman', '1971-09-08');
```

Relationships

```
+----------------------------+          +-----------------------+
|         Movies             |          |       Ratings         |
|                            |          |                       |
| Id                  SERIAL |          |                       |
| Title               TEXT   |          |   Id          SERIAL  |
| PrimaryDirector     TEXT   |          |   Description TEXT    |
| YearReleased        INT    |          |                       |
| Genre               TEXT   |          +-----------------------+
+----------------------------+
```

Relationship

Let's add a new column to our `Movies` to indicate *WHICH* rating is associated to each row representing a movie.

```
ALTER TABLE "Movies" ADD COLUMN "RatingId" INTEGER NULL REFERENCES "Ratings" ("Id");
```

- `RatingId` is an integer since it matches the `SERIAL` which we are going to relate to.
- `NULL` indicates that we are allowed to have no value.
- Next we indicate that this is a foreign key (we are *relating* this table) to the `Ratings` table.
- We also specify the column in the other table, in this case `Id` in `Ratings`, we mean to match.

```
+----------------------------+                  +-----------------------+
|         Movies             |                  |       Ratings         |
|                            |                  |                       |
| Id                  SERIAL |        +--------->   Id           SERIAL |
| Title               TEXT   |        |     one |   Description  TEXT   |
| PrimaryDirector     TEXT   |        |         |                       |
| YearReleased        INT    |        |         |                       |
| Genre               TEXT   | many   |         +-----------------------+
| RatingId            INT    <--------+
|                            |
+----------------------------+
```

Now we can specify the `RatingId` associated to each movie when we insert the
movie.

```
UPDATE "Movies" SET "RatingId" = 2 WHERE "Id" IN (10);
UPDATE "Movies" SET "RatingId" = 3 WHERE "Id" IN (1, 2, 3, 4, 5, 6, 7, 8, 9 );
UPDATE "Movies" SET "RatingId" = 4 WHERE "Id" IN (11, 12, 13 );
```

Joining tables

So now that we have these two tables, how do we *join* them together so that we can retrieve information about movies and their ratings or get information about a rating and the associated movies.

Query movies and also get their rating

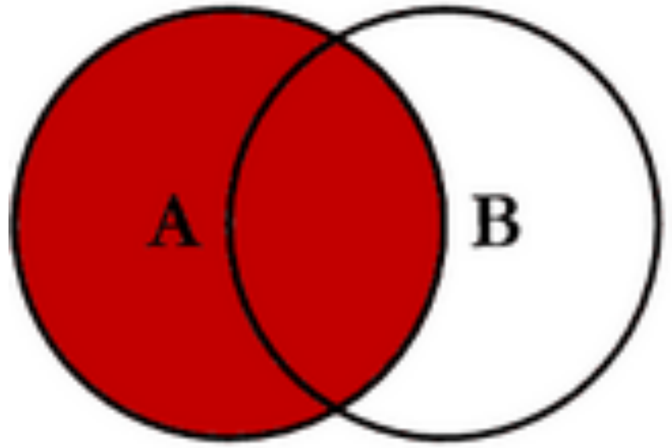
```
SELECT *
FROM "Movies"
JOIN "Ratings" ON "Movies"."RatingId" = "Ratings"."Id";
```

Query for all movies that are "R", adding a WHERE clause

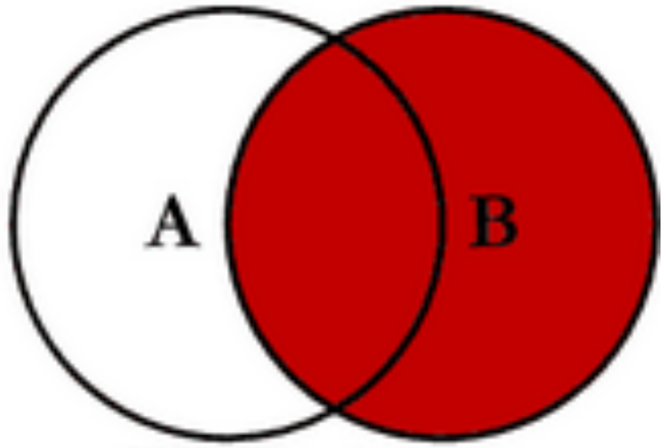
```
SELECT *
FROM "Movies"
JOIN "Ratings" ON "Movies"."RatingId" = "Ratings"."Id"
WHERE "Ratings"."Description" = 'R';
```

- This query will give us movies and their ratings.
- But only for movies that have a `RatingId` that matches an `Id` from the ratings table.
- That is, any `movie` with a `null` value for `RatingId` (or a value that doesn't match an `id`) will not be in the results.

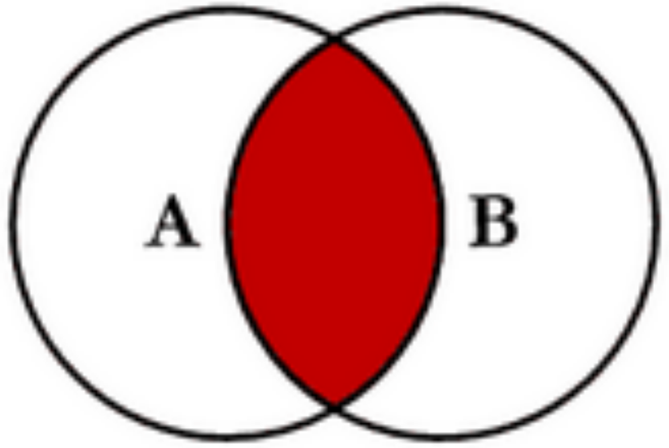
SQL JOINS



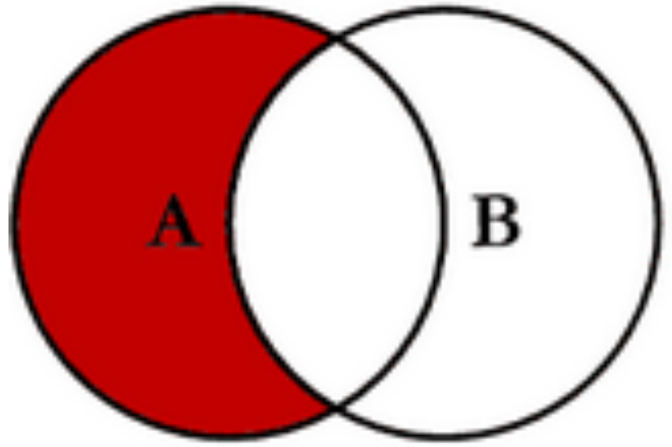
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



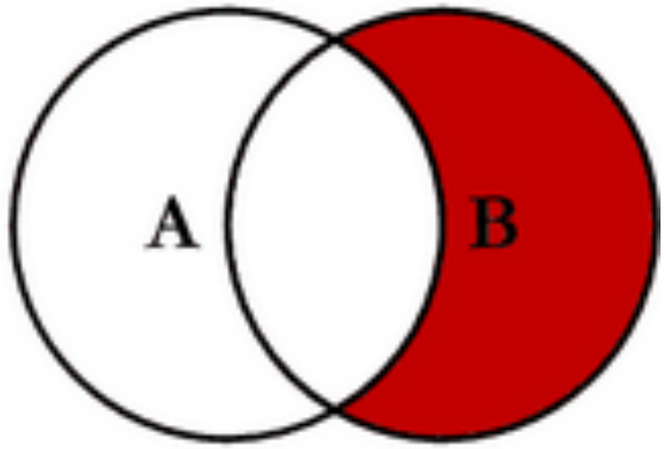
```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```



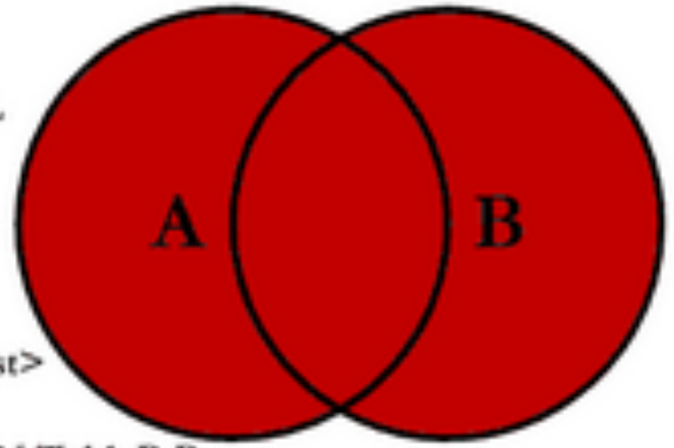
```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```



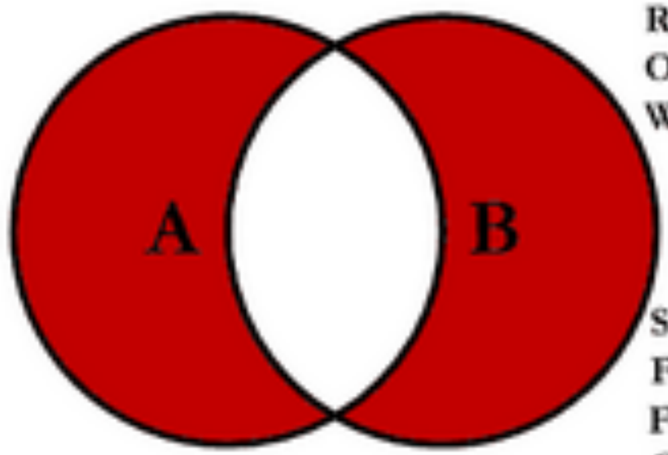
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```



```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```



```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```



```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

Query all the movies and include ratings when possible

```
SELECT *
FROM "Movies"
LEFT JOIN "Ratings" ON "Movies"."RatingId" = "Ratings"."Id";
```

Many to Many

```
+--------------------------------+           +---------------------------+
|            Movies              |           |         Ratings           |
|                                |           |                           |
|    Id                  SERIAL  |           |     Id          SERIAL    |
|    Title               TEXT    |many    one|     Description TEXT      |
|    PrimaryDirector     TEXT    +-----------+                           |
|    YearReleased        INT     |           +---------------------------+
|    Genre               TEXT    |
|    RatingId            INT     |
|                                |
+------------+-------------------+
             |
             | many
             |
             |
             |                   +-------------------------+
             |                   |        ACTORS           |
             |                   |                         |
             |              many |    Id          SERIAL   |
             +-------------------+    FullName    TEXT     |
                                 |    Birthday    DATE     |
                                 |                         |
                                 +-------------------------+
```

- In the case of a *many-to-many* relationship we cannot place the foreign keys on either of the tables.
- In this case we need a third table, commonly referred to as a *join table* to store the relationships.
- In this table, we will place two foreign keys, one to the left (movies) and the other to the right (actors).
- We attempt to name this table based on the relationship between the two tables.

```sql
CREATE TABLE "Roles" (
  "Id"         SERIAL PRIMARY KEY,
  "MovieId"    INTEGER REFERENCES "Movies" ("Id"),
  "ActorId"    INTEGER REFERENCES "Actors" ("Id")
);
```

```
+--------------------------------+           +---------------------------+
|            Movies              |           |         Ratings           |
|                                |           |                           |
|    Id                  SERIAL  |           |     Id           SERIAL   |
|    Title               TEXT    | many  one |     Description  TEXT     |
|    PrimaryDirector     TEXT    <----------->                           |
|    YearReleased        INT     |           +---------------------------+
|    Genre               TEXT    |
|    RatingId            INT     |
|                                |
+-------------^------------------+
              | one
              |
              |
              |
              |
              | many
      +-------v---------------+               +-------------------------+
      |        Roles          |               |          Actors         |
      |                       | many      one |                         |
      |   Id       SERIAL     <--------------->    Id          SERIAL   |
      |   MovieId  INT        |               |    FullName    TEXT     |
      |   ActorId  INT        |               |    Birthday    DATE     |
      |                       |               |                         |
      +-----------------------+               +-------------------------+
```

Insert some Roles

Query for the casts and actors

```
SELECT "Movies"."Title", "Actors"."FullName"
FROM "Movies"
JOIN "Roles" ON "Roles"."MovieId" = "Movies"."Id"
JOIN "Actors" ON "Actors"."Id" = "Roles"."ActorId";
```

Adding information to the join table.

- What if we wanted to capture the name of the character the actor played?
- It can't go on the `Movies` table since it isn't distinct to a movie. It can't go on the `Actors` table since it isn't unique to that either.
- The correct place here is to place that column on the `Roles` table.

Updating the Roles table

Let's call this new column `CharacterName` and add it to the `Roles` table.

```
ALTER TABLE "Roles" ADD COLUMN "CharacterName" TEXT NULL;
```

Now that we have done that, we can add in a few character names. In order to know what rows to update, let's add the `Roles.Id` to our query above.

```sql
SELECT "Roles"."Id", "Movies"."Title", "Actors"."FullName", "Roles"."CharacterName"
FROM "Movies"
JOIN "Roles" ON "Roles"."MovieId" = "Movies"."Id"
JOIN "Actors" ON "Actors"."Id" = "Roles"."ActorId";
```

Now let's update the roles for all of our actors

```sql
-- Orlando Bloom played Will Turner in Pirates (ID 1)
UPDATE "Roles" SET "CharacterName" = 'Will Turner' WHERE "Id" IN (1);

-- Orlando Bloom played Legolas in the Lord of the Rings movies
UPDATE "Roles" SET "CharacterName" = 'Legolas' WHERE "Id" IN (2,3,4);
```

Rerun the query

```sql
SELECT "Roles"."Id", "Movies"."Title", "Actors"."FullName", "Roles"."CharacterName"
FROM "Movies"
JOIN "Roles" ON "Roles"."MovieId" = "Movies"."Id"
JOIN "Actors" ON "Actors"."Id" = "Roles"."ActorId";
```

If we started with a more detailed ERD we could have avoided the alter table statements for adding our relationships