Multi-table queries with JOINs

Up to now, we've been working with a single table, but entity data in the real world is often broken down into pieces and stored across multiple orthogonal tables using a process known as *normalization*[1].

Database normalization

Database normalization is useful because it minimizes duplicate data in any single table, and allows for data in the database to grow independently of each other (ie. Types of car engines can grow independent of each type of car). As a trade-off, queries get slightly more complex since they have to be able to find data from different parts of the database, and performance issues can arise when working with many large tables.

In order to answer questions about an entity that has data spanning multiple tables in a normalized database, we need to learn how to write a query that can combine all that data and pull out exactly the information we need.

Multi-table queries with JOINs

Tables that share information about a single entity need to have a *primary key* that identifies that entity *uniquely* across the database. One common primary key type is an auto-incrementing integer (because they are space efficient), but it can also be a string, hashed value, so long as it is unique.

Using the **JOIN** clause in a query, we can combine row data across two separate tables using this unique key. The first of the joins that we will introduce is the **INNER JOIN**.

Select query with INNER JOIN on multiple tables

```
SELECT column, another_table_column, ...
FROM mytable
INNER JOIN another_table
    ON mytable.id = another_table.id
WHERE condition(s)
ORDER BY column, ... ASC/DESC
LIMIT num_limit OFFSET num_offset;
```

The **INNER JOIN** is a process that matches rows from the first table and the second table which have the same key (as defined by the **ON** constraint) to create a result row with the combined columns from both tables. After the tables are joined, the other clauses we learned previously are then applied.

Did you know?
You might see queries where the **INNER JOIN** is written simply as a **JOIN**. These two are equivalent, but we will continue to refer to these joins as inner-joins because they make the query easier to read once you start using other types of joins, which will be introduced in the following lesson.

Exercise

We've added a new table to the Pixar database so that you can try practicing some joins. The **BoxOffice** table stores information about the ratings and sales of each particular Pixar movie, and the **Movie_id** column in that table corresponds with the **Id** column in the **Movies** table 1-to-1. Try and solve the tasks below using the **INNER JOIN** introduced above.

Exercise — Tasks

1. Find the domestic and international sales for each movie
2. Show the sales numbers for each movie that did better internationally rather than domestically
3. List all the movies by their ratings in descending order