Learning Objectives

Learners will be able to...

- Identify relations between tables based on ER-diagrams
- Recognize INNER JOIN
- Understand LEFT/RIGHT JOIN
- Classify usage of FULL OUTER JOIN

info

Make Sure You Know

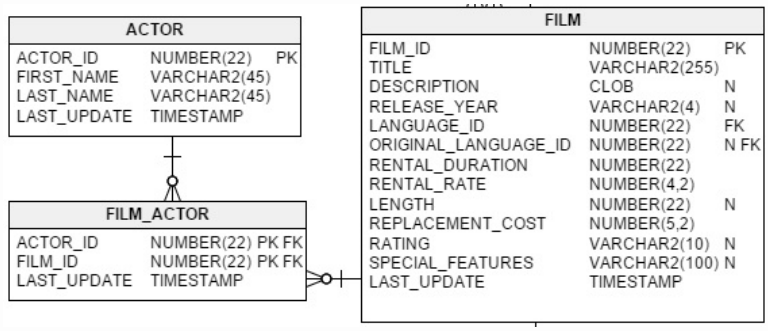
Before starting the assignment, learners should have a basic understanding of relational databases and ER-diagrams. Learners should also have a good grasp of SQL syntax and be comfortable writing SQL queries.

Graphic display of relationship

You already know how to read the graphical structure of a single table.

In databases, tables are not standalone entities. They are interconnected through links, which is why this type of database is called relational.

In this picture you see an example of a graphical display of the links between them.

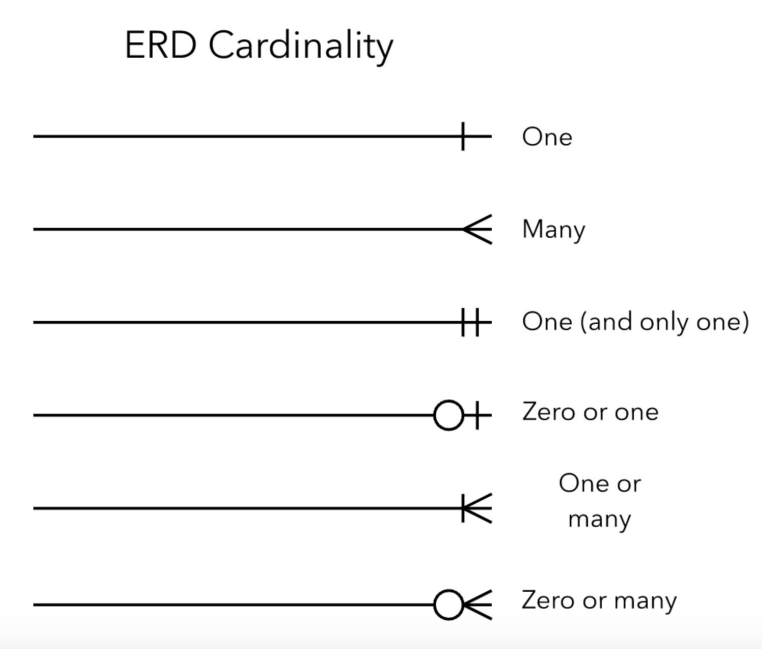


A picture of the sakila database with a graphical display of the links between them. It shows the primary and foreign keys in each table. There are three tables displayed

Picture Source

A relationship is represented by a line connecting the primary keys of one table to the foreign keys of another table.

Different symbols are used to depict various types of relationships, which may vary depending on the visualization system. The most commonly used symbols are as follows:



The ERD Cardinality graphic display that shows the relationships between tables.

Picture Source

Therefore, this example illustrates a common one-to-many relationship scenario. In this case, an actor can participate in multiple films, and a film can feature multiple actors. This allows us to obtain comprehensive information about the cast of a film without duplicating data.

info

Mandatory and optional relationship

Pay attention to the white dot near the link - it means that the table will not necessarily have a value with this identifier. For example, there may be a film with no actors, or an actor may not have appeared in any of the rental discs.

In addition, you may come across another notation for an optional relationship - a dotted line.

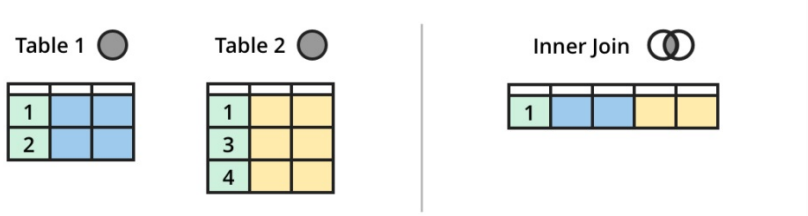
Join types

SQL uses JOIN statements of various types to join tables.

Here are the different types of the JOINs in SQL:

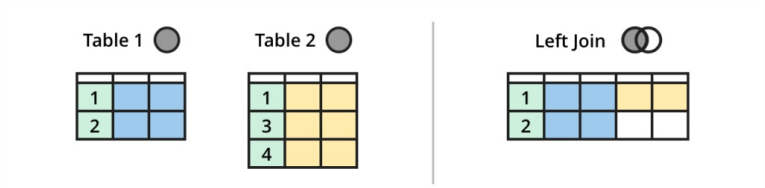
INNER JOIN

INNER JOIN: Returns records that have matching values in both tables



LEFT/RIGHT OUTER JOIN

LEFT OUTER JOIN: Returns all records from the left table, and the matched records from the right table



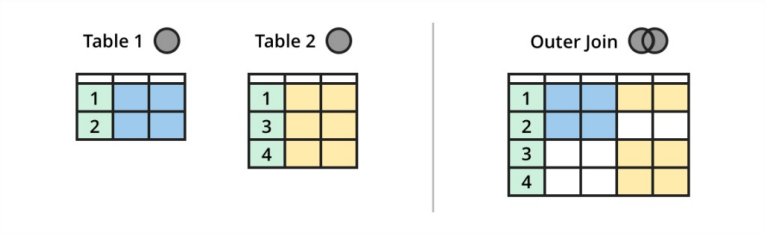
Picture of two tables and then an image of the tables Left Joined

RIGHT OUTER JOIN: Returns all records from the right table, and the matched records from the left table

> info
>
> ## RIGHT OUTER JOIN
>
> Right outer join is symmetric to left outer join: you could achieve it by swapping table names during join.

FULL OUTER JOIN

FULL OUTER JOIN: Returns all records when there is a match in either left or right table



Picture of two tables and then an image of the tables with a FULL Outer Joined

INNER JOIN

Let's get a list of actors who starred in a particular movie. To do this, we will need to use the INNER JOIN operator.

The general syntax of the operator is as follows:

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
```

The description above shows the join of two tables, but in our case we need to join three already, which is also supported by the language.

```
SELECT actor.first_name, actor.last_name, film.title,
        film.film_id
FROM film
INNER JOIN film_actor
ON film.film_id = film_actor.film_id
INNER JOIN actor
ON film_actor.actor_id = actor.actor_id
WHERE film.film_id = 940;
```

We used **INNER JOIN** statement twice: first we join `film` with `film_actor` table and on second join - `film_actor` to `actor` table. The condition of joining is specified with ON statement: `ON film.film_id = film_actor.film_id` and `ON film_actor.actor_id = actor.actor_id`.

The result of this query will be the following table.

```
 first_name | last_name |      title      | film_id
------------+-----------+-----------------+---------
 GARY       | PHOENIX   | VICTORY ACADEMY |     940
 KENNETH    | PESCI     | VICTORY ACADEMY |     940
 DEBBIE     | AKROYD    | VICTORY ACADEMY |     940
 CUBA       | BIRCH     | VICTORY ACADEMY |     940
 MERYL      | ALLEN     | VICTORY ACADEMY |     940
(5 rows)
```

info

INNER JOIN vs JOIN

In SQL, two types of **JOIN** operator notation are used: INNER JOIN - full notation. You can use the short form - **JOIN**. In both cases, the result will be the same.

Very often in such queries you can see the previously studied alias operator - AS. The main thing in this case is not to get confused about which table which alias belongs to.

```
SELECT a.first_name, a.last_name, f.title, f.film_id
FROM film AS f
INNER JOIN film_actor AS fa
ON f.film_id = fa.film_id
INNER JOIN actor AS a
ON fa.actor_id = a.actor_id
WHERE f.film_id = 940;
```

And of course, in these queries, you can use all the previously studied functions.

Earlier, we considered the minimum, maximum and amount of payments by customers and displayed a user ID that was incomprehensible to us. Now, with the help of the join operator, we can display their name!

```
SELECT MIN(amount), MAX(amount), SUM(amount),
        customer.first_name || ' ' || customer.last_name AS
        customer
FROM payment
JOIN customer
ON payment.customer_id = customer.customer_id
GROUP BY customer.customer_id
LIMIT 10;
```

RIGHT and LEFT OUTER JOIN

LEFT OUTER JOIN is the second most commonly used type of JOIN operation in SQL. The initial portion of the data generated by this operation resembles the result of an INNER JOIN, and could even be identical if there are no unmatched entries in the left table.
However, SQL handles the left table (the first one) quite differently. For each row in the first (or LEFT) table that doesn't find a match, SQL will still include this row in the resulting table, but will fill in NULL values for the columns corresponding to the other table.

Let's imagine that we want to write a query that will return the namesakes of our users with actors.

```
SELECT c.customer_id, c.first_name, c.last_name, a.actor_id,
       a.first_name, a.last_name
FROM customer c
JOIN actor a
ON c.last_name = a.last_name
ORDER BY c.last_name
LIMIT 10;
```

But this query has a problem, it will return data only for those customers for which there is a match in the table of actors. To get all users we need to use left join.

```
SELECT c.customer_id, c.first_name, c.last_name, a.actor_id,
       a.first_name, a.last_name
FROM customer c
LEFT JOIN actor a
ON c.last_name = a.last_name
ORDER BY c.last_name
LIMIT 10;
```

Note that at the beginning of this query, there are 5 additional rows with 3 empty right columns. This is exactly the data that we missed when we used a inner join.

You can get the same result by using a right join and swapping the tables. For this reason, the right join is not often used.

```
SELECT c.customer_id, c.first_name, c.last_name, a.actor_id,
        a.first_name, a.last_name
FROM actor a
RIGHT JOIN customer c
ON a.last_name = c.last_name
ORDER BY c.last_name
LIMIT 10;
```

info

Missed OUTER word

You may have noticed that there is no word OUTER in the queries. This is due to the fact that in the presence of the words LEFT, RIGHT, FULL, the word OUTER can be omitted.

But if we add the word right in the initial request, then we will get a list of all actors who have (and do not have) namesakes among customers.

```
SELECT c.customer_id, c.first_name, c.last_name, a.actor_id,
        a.first_name, a.last_name
FROM customer c
RIGHT JOIN actor a
ON c.last_name = a.last_name
ORDER BY a.last_name
LIMIT 10;
```

FULL OUTER JOIN

FULL OUTER JOIN creates the result-set by combining results of both LEFT JOIN and RIGHT JOIN. The result-set will contain all the rows from both tables. For the rows for which there is no matching, the result-set will contain NULL values.

```
SELECT c.customer_id, c.first_name, c.last_name, a.actor_id,
        a.first_name, a.last_name
FROM customer c
FULL JOIN actor a
ON c.last_name = a.last_name
ORDER BY c.last_name
LIMIT 10;
```

The difference in queries will be especially noticeable if we compare the number of rows returned.

Query	Rows count
SELECT COUNT(*) FROM actor a **JOIN** customer c ON a.last_name = c.last_name;	43
SELECT COUNT(*) FROM actor a **LEFT JOIN** customer c ON a.last_name = c.last_name;	200
SELECT COUNT(*) FROM actor a **RIGHT JOIN** customer c ON a.last_name = c.last_name;	620
SELECT COUNT(*) FROM actor a **FULL JOIN** customer c ON a.last_name = c.last_name;	777

FULL OUTER JOIN is not the most convenient operator and is best used only on small tables. Large tables when using FULL OUTER JOIN can grow even more, and this will complicate the work with data.