IRON HACK

DATA ANALYSIS BOOTCAMP

SQL

SQL

SQL is the single most requested language for positions of Data Analyst and Data Scientist.
Ahead of Python, ahead of R, ahead of Excel.

Why do you think that is?

SQL

Structured **Q**uery **L**anguage is a standard database language used to create, manage and query data from relational databases.

We need to clarify some of these words today

- Database
- Relational
- Query

DATABASES

Organized collection of data structured in ways that impose some rules on the data according to needs, e.g.

- Size
- Accuracy
- Security
- Redundancy
- Accessibility
- Governance

We can do a comparison with Excel to understand these needs

TYPES OF DATABASES

OPERATIONAL VS ANALYTICAL

Operational databases are the backbone of many companies, organizations, and institutions throughout the world today. This type of database is primarily used to **collect, modify, and maintain data** on a day-to-day basis. The type of **data stored is dynamic**, meaning that it changes constantly and always reflects up-to-the-minute information.

Analytical database stores and tracks **historical and time-dependent data**. It is a valuable asset for tracking trends, viewing statistical data over a long period, or making tactical or strategic business projections. The type of **data stored is static**, meaning that the data is never (or very rarely) modified, although new data might often be added.

WHAT IS A RELATIONAL DATABASE

A relational database is a database structured to recognise relations between stored items of information.

- Relational databases are built of tables, which establish the relation between **tuples (records or rows)** and **attributes (fields or columns).**

- Each table can be related to each other using the concepts of **keys**

RDBMS

A **relational database management system** (RDBMS) is a software application program you use to create, maintain, modify, and manipulate a relational database.

SQL is the *de facto* universal language to work with data from relational databases so naturally, most RDBMS support SQL. SQL is more than just a means for extracting knowledge from data. It's also a language for defining the structures that hold data so we can organize relationships in the data.

MySQL is one of many RDBMS that support the use of SQL. We will use a graphical user interface to work with MySQL called **MySQL workbench**.

ANATOMY OF A DATABASE

ALL THE COMPONENTS WE WILL TALK ABOUT

- Tables
- Columns
- Rows

- Keys
- Relationships
- Views

TABLES, COLUMNS, ROWS

TABLES

Tables are the **main structures in the database**.

Each table always represents a **single, specific subject**. The order of rows and columns within a table is of absolutely no importance. Every table should contain at least one column—known as a **primary key**—that uniquely identifies each of its rows.

The subject that a given table represents is usually **either an object or an event**. When the **subject is an object**, the table represents something that is tangible, such as a person, place, or thing. (i.e.: Pilots, products, machines, students, buildings, and equipment) When the **subject is an event**, the table represents something that occurs at a given point in time and has characteristics you wish to record. (i.e.: judicial hearings, distributions of funds, lab test results, and geological surveys)

COLUMNS - ATTRIBUTES OF THE OBJECT

Columns represent a **characteristic of the subject of the table** to which it belongs.

ROWS - ENTRIES OF THE OBJECT

A row represents a **unique instance of the subject of a table**. It is composed of the entire set of columns in a table, regardless of whether or not the columns contain any values.

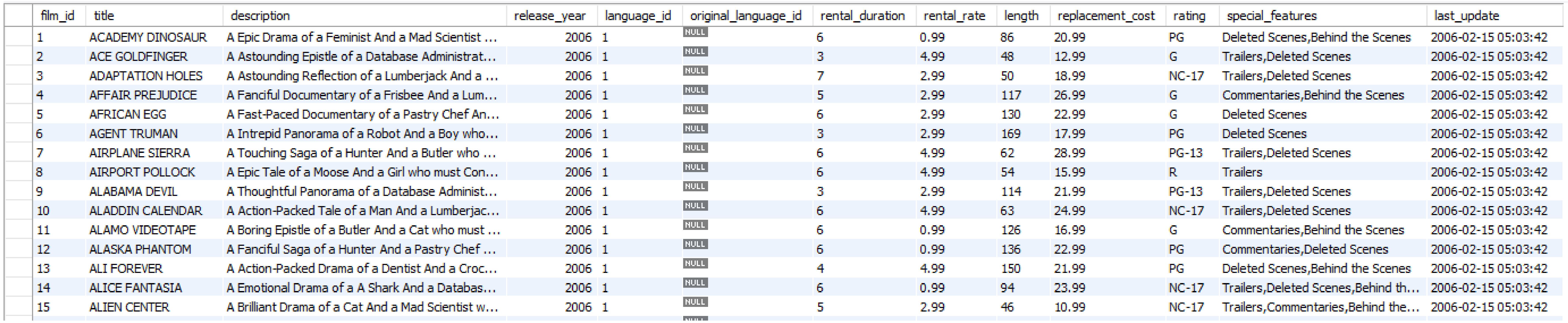
DATA TYPES

WHAT TYPES CAN THE SINGLE PIECE OF DATA BE?

- **Integers** (smallint, int, bigint)
- **Char**
- **Varchar** (variable length char)
- **Text** (unlimited length)
- **Serial (auto incrementing:** smallserial, serial, bigserial)
- **Fixed/floating point** (numeric real double precision)
- **Date** (timestamp, date, time, interval)

SAMPLE - SAKILA DATABASE

THE FILMS TABLE



film_id	title	description	release_year	language_id	original_language_id	rental_duration	rental_rate	length	replacement_cost	rating	special_features	last_update
1	ACADEMY DINOSAUR	A Epic Drama of a Feminist And a Mad Scientist ...	2006	1	NULL	6	0.99	86	20.99	PG	Deleted Scenes,Behind the Scenes	2006-02-15 05:03:42
2	ACE GOLDFINGER	A Astounding Epistle of a Database Administrat...	2006	1	NULL	3	4.99	48	12.99	G	Trailers,Deleted Scenes	2006-02-15 05:03:42
3	ADAPTATION HOLES	A Astounding Reflection of a Lumberjack And a ...	2006	1	NULL	7	2.99	50	18.99	NC-17	Trailers,Deleted Scenes	2006-02-15 05:03:42
4	AFFAIR PREJUDICE	A Fanciful Documentary of a Frisbee And a Lum...	2006	1	NULL	5	2.99	117	26.99	G	Commentaries,Behind the Scenes	2006-02-15 05:03:42
5	AFRICAN EGG	A Fast-Paced Documentary of a Pastry Chef An...	2006	1	NULL	6	2.99	130	22.99	G	Deleted Scenes	2006-02-15 05:03:42
6	AGENT TRUMAN	A Intrepid Panorama of a Robot And a Boy who...	2006	1	NULL	3	2.99	169	17.99	PG	Deleted Scenes	2006-02-15 05:03:42
7	AIRPLANE SIERRA	A Touching Saga of a Hunter And a Butler who ...	2006	1	NULL	6	4.99	62	28.99	PG-13	Trailers,Deleted Scenes	2006-02-15 05:03:42
8	AIRPORT POLLOCK	A Epic Tale of a Moose And a Girl who must Con...	2006	1	NULL	6	4.99	54	15.99	R	Trailers	2006-02-15 05:03:42
9	ALABAMA DEVIL	A Thoughtful Panorama of a Database Administ...	2006	1	NULL	3	2.99	114	21.99	PG-13	Trailers,Deleted Scenes	2006-02-15 05:03:42
10	ALADDIN CALENDAR	A Action-Packed Tale of a Man And a Lumberjac...	2006	1	NULL	6	4.99	63	24.99	NC-17	Trailers,Deleted Scenes	2006-02-15 05:03:42
11	ALAMO VIDEOTAPE	A Boring Epistle of a Butler And a Cat who must ...	2006	1	NULL	6	0.99	126	16.99	G	Commentaries,Behind the Scenes	2006-02-15 05:03:42
12	ALASKA PHANTOM	A Fanciful Saga of a Hunter And a Pastry Chef ...	2006	1	NULL	6	0.99	136	22.99	PG	Commentaries,Deleted Scenes	2006-02-15 05:03:42
13	ALI FOREVER	A Action-Packed Drama of a Dentist And a Croc...	2006	1	NULL	4	4.99	150	21.99	PG	Deleted Scenes,Behind the Scenes	2006-02-15 05:03:42
14	ALICE FANTASIA	A Emotional Drama of a A Shark And a Databas...	2006	1	NULL	6	0.99	94	23.99	NC-17	Trailers,Deleted Scenes,Behind th...	2006-02-15 05:03:42
15	ALIEN CENTER	A Brilliant Drama of a Cat And a Mad Scientist w...	2006	1	NULL	5	2.99	46	10.99	NC-17	Trailers,Commentaries,Behind the...	2006-02-15 05:03:42

QUERIES I

- Retrieving information from the database
- You have to specify which table(s) you want information for
- What is the criteria for the rows to be returned? any filters?
- How to manipulate the resulting fields?

SELECT STATEMENT

```
SELECT * FROM sakila.film;

SELECT title, description, rating  FROM sakila.film;

SELECT DISTINCT rental_duration
FROM sakila.film;

SELECT DISTINCT rental_duration, language_id
FROM sakila.film;
```

ORDER BY

```
SELECT title, rental_rate, length
FROM sakila.film
ORDER BY length DESC;
```

```
SELECT title, rental_rate, length
FROM sakila.film
ORDER BY length DESC, rental_rate DESC;
```

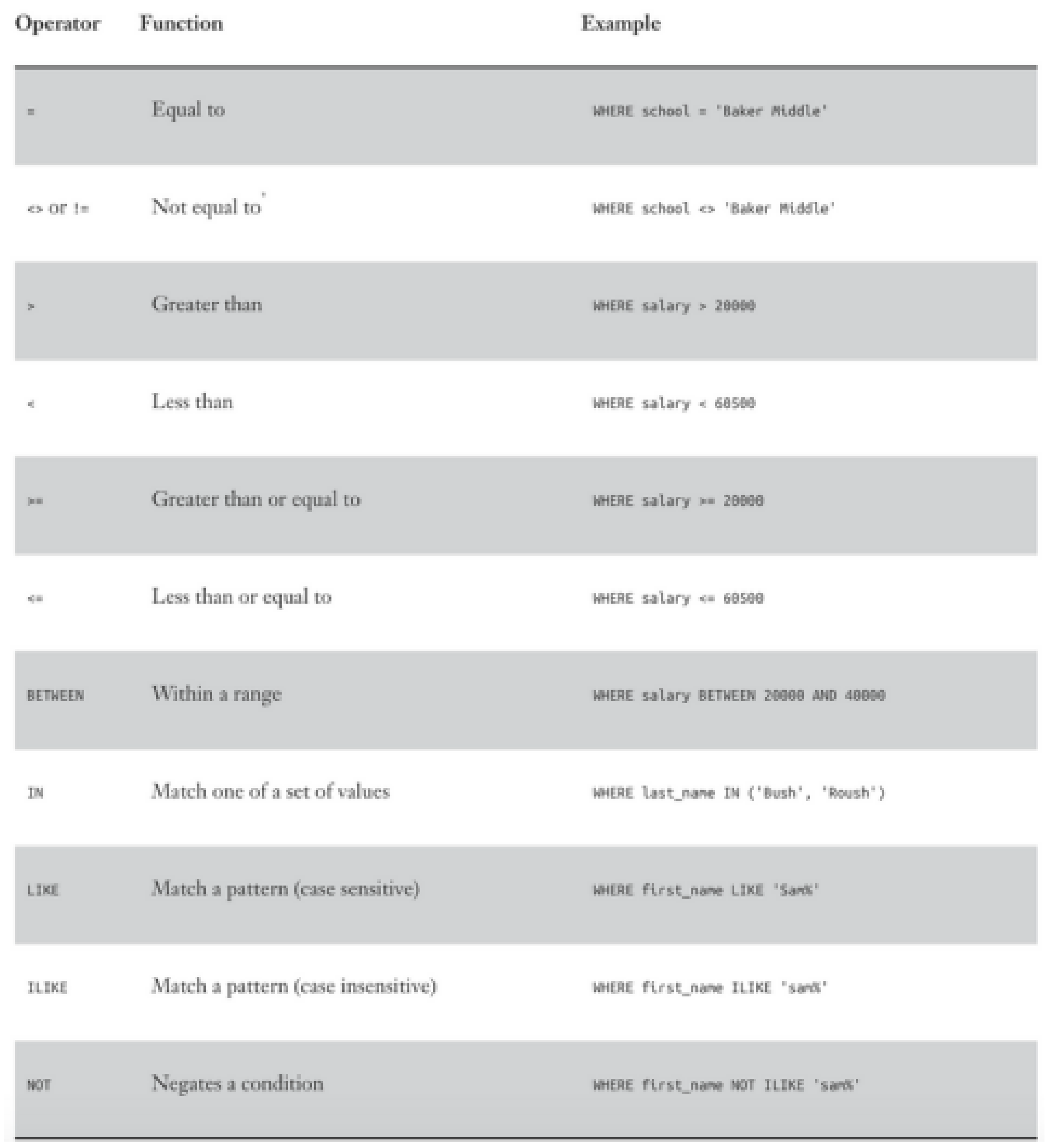
ALIASING and COMPUTATIONS

```
SELECT title, rental_rate AS cost, length
FROM sakila.film;

SELECT title, rental_rate/length AS price_per_min
FROM sakila.film
ORDER BY price_per_min ASC;

SELECT CONCAT(title,', rating:',rating) AS descriptor
FROM sakila.film;
```

https://www.w3schools.com/sql/sql_ref_sqlserver.asp

WHERE - HOW TO RETRIEVE DATA FROM SPECIFIC ROWS

```
SELECT *
FROM sakila.film
WHERE rental_duration = 6;
```



Operator	Function	Example
=	Equal to	WHERE school = 'Baker Middle'
<> or !=	Not equal to*	WHERE school <> 'Baker Middle'
>	Greater than	WHERE salary > 20000
<	Less than	WHERE salary < 60500
>=	Greater than or equal to	WHERE salary >= 20000
<=	Less than or equal to	WHERE salary <= 60500
BETWEEN	Within a range	WHERE salary BETWEEN 20000 AND 40000
IN	Match one of a set of values	WHERE last_name IN ('Bush', 'Roush')
LIKE	Match a pattern (case sensitive)	WHERE first_name LIKE 'Sam%'
ILIKE	Match a pattern (case insensitive)	WHERE first_name ILIKE 'sam%'
NOT	Negates a condition	WHERE first_name NOT ILIKE 'sam%'

AGGREGATIONS AND GROUP BY

```
SELECT COUNT(*),MAX(rental_duration),AVG(replacement_cost),AVG(rental_duration)
FROM sakila.film;
```

```
SELECT rating, COUNT(*), AVG(rental_rate)
FROM sakila.film
GROUP BY rating;
```

```
SELECT rating, rental_duration, COUNT(*), AVG(rental_rate)
FROM sakila.film
GROUP BY rating, rental_duration;
```

RELATIONSHIPS

RELATIONSHIPS BETWEEN OBJECTS\TABLES

If rows in a given table can be **associated** in some way with rows in another table, the tables are said to have a relationship between them.

There are three types of relationships:

- 1 to 1

- 1 to many

- many to many

Keys are a special table construct designed to denote relationships between tables

KEYS

PRIMARY VS FOREIGN

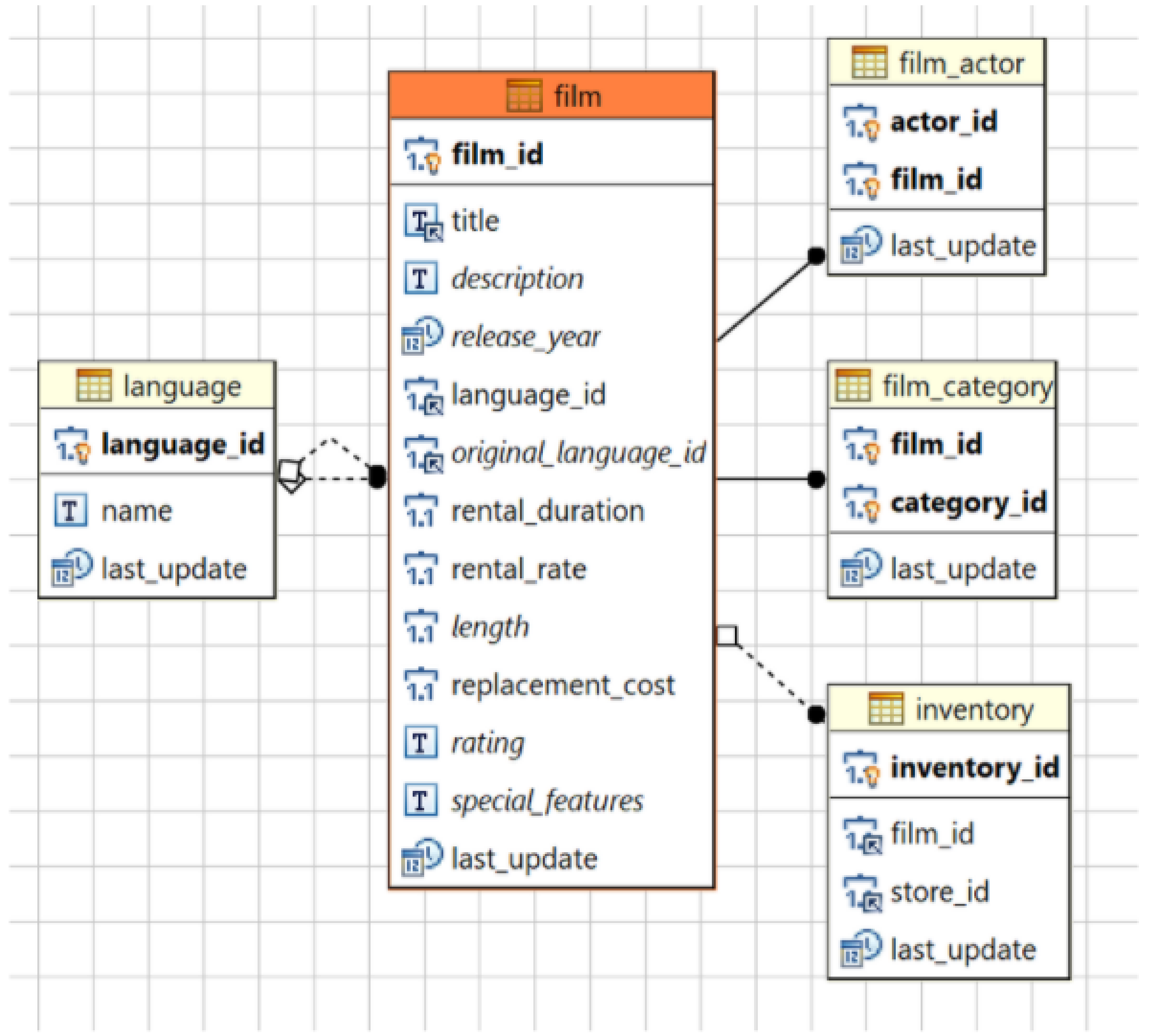
Every table in your database should have a primary key. A primary key consists of one column that **uniquely identifies** each row within a table.

The primary key:
- identifies a specific row throughout the entire database,
- its column identifies a given table throughout the entire database.
- enforce table-level integrity and help establish relationships with other tables.

When you determine that a pair of tables has a relationship to each other, you typically establish the relationship by taking a copy of the primary key from the first table and inserting it into the second table, where it becomes a **foreign key**.

RELATIONAL DATABASE SCHEMAS

- Entity Relational Diagram (ERD)
- Shows content of each table and connections between tables within a database schema
- Primary Key and Foreign Key visual here



ONE-TO-ONE RELATIONSHIP

A SINGLE ROW IN THE FIRST TABLE IS RELATED TO ONLY ONE ROW IN THE SECOND TABLE

A pair of tables has a one-to-one relationship when a single row in the first table is related to only one row in the second table, and a single row in the second table is related to only one row in the first table.

In this type of relationship, one table is referred to as the **primary table**, and the other is referred to as the **secondary table**. You establish this relationship by taking the primary key of the primary table and inserting it into the secondary table, where it becomes a **foreign key**. This is a special type of relationship because **in nearly all cases the foreign key also acts as the primary key of the secondary table.**

(e.g. citizen id and passport nr. Notice not all citizens have a passport)

ONE-TO-MANY RELATIONSHIP

ONE ROW IN ONE TABLE HAS A RELATIONSHIP WITH MULTIPLE ROWS IN ANOTHER TABLE

When a pair of tables has a one-to-many relationship, a single row in the first table can be related to many rows in the second table, but a single row in the second table can be related to only one row in the first table.

This relationship is established by taking the primary key of the table on the “one” side and inserting it into the table on the “many” side, where it becomes a foreign key. In this case **the second table has also its own primary key**.

(e.g. city and country - world database)

MANY-TO-MANY RELATIONSHIP

MANY ROWS IN ONE TABLE HAVE A RELATIONSHIP WITH MULTIPLE ROWS IN ANOTHER TABLE

A pair of tables is in a many-to-many relationship when a single row in the first table can be related to many rows in the second table, and a single row in the second table can be related to many rows in the first table.

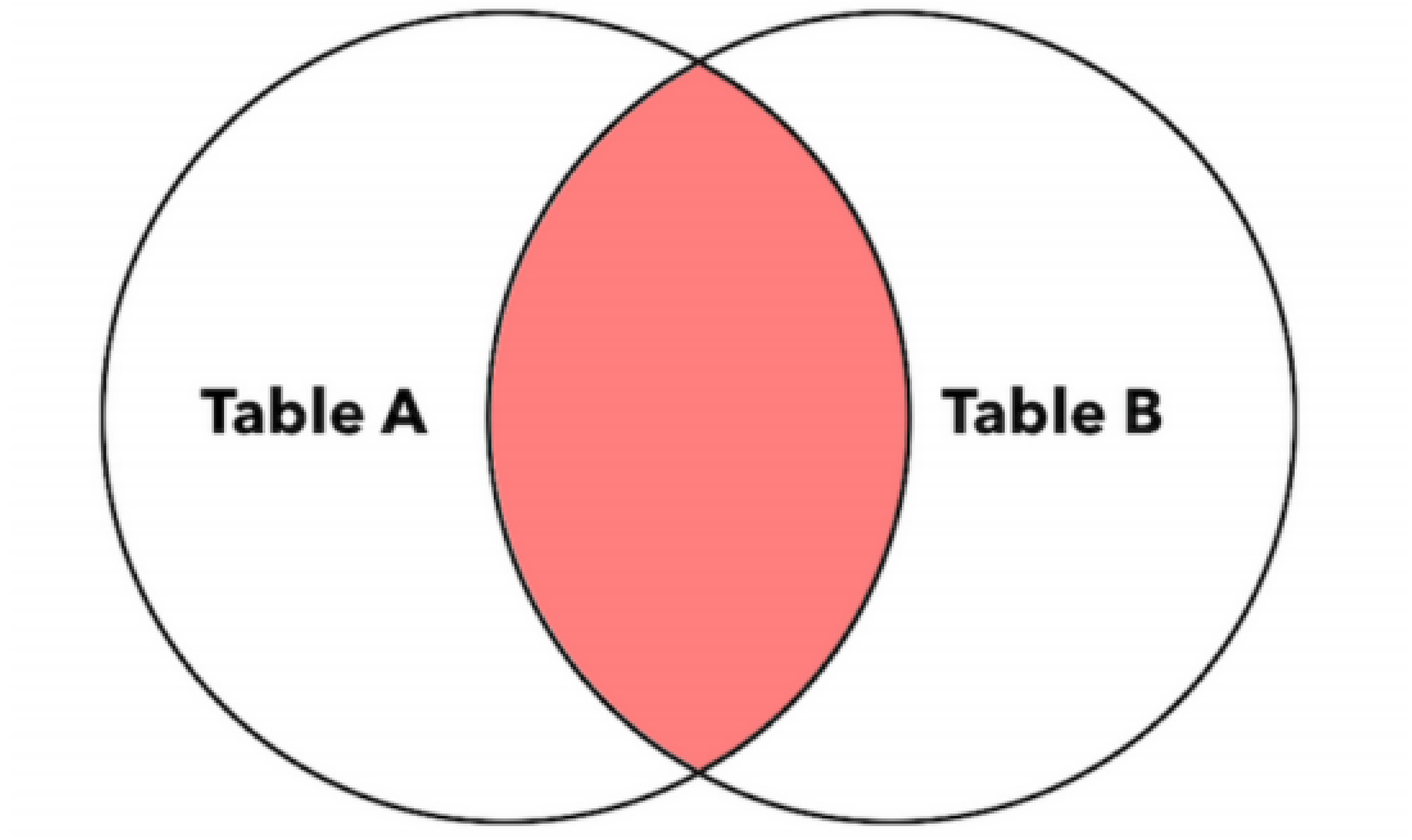
To establish this relationship properly, you typically create what is known as a **linking table**. This table provides an easy way to associate rows from one table with those of the other and will help to ensure that you have no problems adding, deleting, or modifying any related data. You define a linking table by taking a copy of **the primary key of each table in the relationship and using them to form the structure of the new table**. These columns actually serve two distinct roles: Together they form the composite primary key of the linking table, and separately they each serve as a foreign key.

(e.g. actor and film - sakila)

INNER JOIN

RETURNS ROWS FROM BOTH TABLES WHERE MATCHING VALUES ARE FOUND IN THE JOINED COLUMNS OF BOTH TABLES.

```
SELECT *
FROM Table_A
INNER JOIN Table_B
ON Table_A.Key = Table_B.Key
```

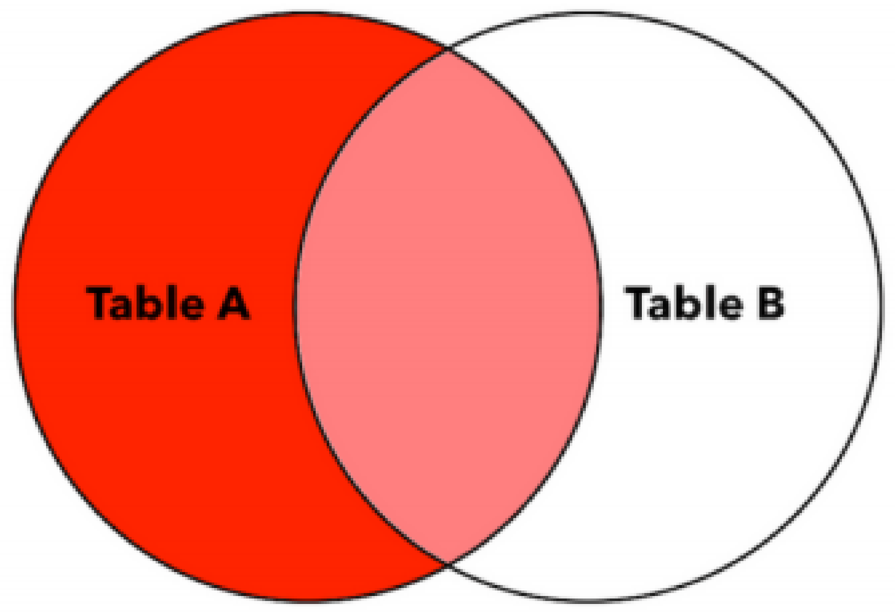


```
SELECT world.country.Name AS Country_Name,
world.city.Name AS City_Name,
world.country.GNP/world.city.Population AS GNP_per_capita
FROM world.country
INNER JOIN world.city
ON world.country.Code = world.city.CountryCode;
```

LEFT JOIN

RETURNS EVERY ROW FROM THE LEFT TABLE PLUS ROWS THAT MATCH VALUES IN THE JOINED COLUMN FROM THE RIGHT TABLE.

```
SELECT *
FROM Table_A
LEFT JOIN Table_B
ON Table_A.Key = Table_B.Key
```



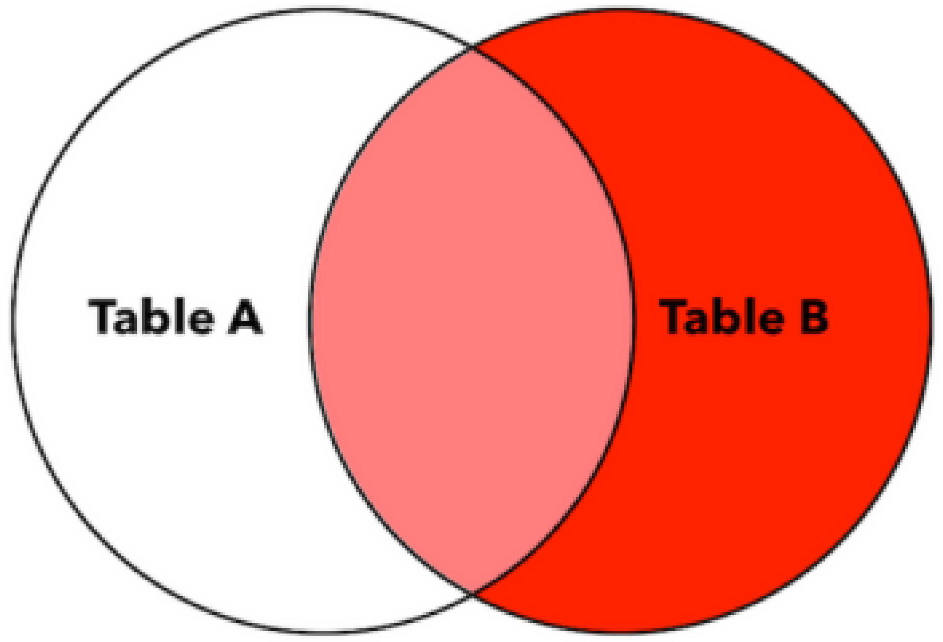
If there's no match for a value in the left table, the query result contains NULL values for the right table columns.

RIGHT JOIN

RETURNS EVERY ROW FROM THE RIGHT TABLE PLUS ROWS THAT MATCH THE KEY VALUES IN THE KEY COLUMN FROM THE LEFT TABLE

```
SELECT *
FROM Table_A
RIGHT JOIN Table_B
ON Table_A.Key = Table_B.Key
```

If there's no match for a value in the right table, the query result contains NULL values for the left table columns.

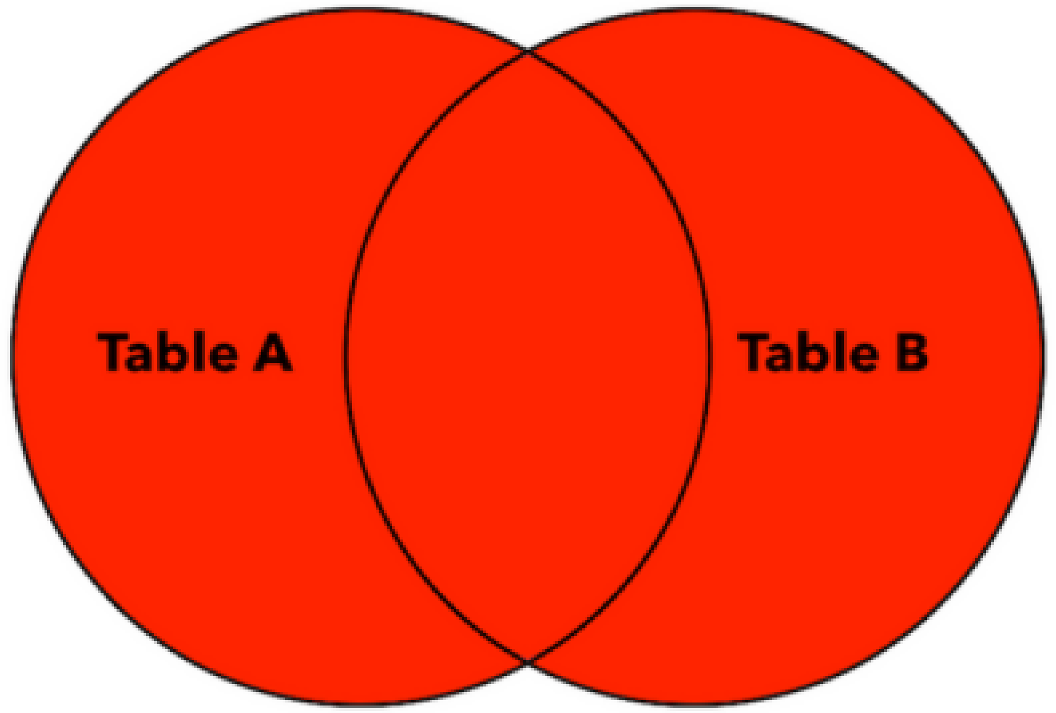


FULL OUTER JOIN

RETURNS EVERY ROW FROM BOTH TABLES AND MATCHES ROWS; THEN JOINS THE ROWS WHERE VALUES IN THE JOINED COLUMNS MATCH.

empty row for the other table.

```
SELECT *
FROM Table_A
OUTER JOIN Table_B
ON Table_A.Key = Table_B.Key
```



If there’s no match for a value in either the left or right table, the query result contains an

SUBQUERIES

As the name implies, subqueries are queries nested inside another query.

You can use a subquery in the place of any table of the main query, since SELECT statements themselves return “tables”

This is often a solution used for quick analysis since subqueries can easily become very complex

SUBQUERIES

Example: find the names of the actors which starred in movies with lengths higher or equal to the average length of all the movies

SUBQUERIES

Example: find the names of the actors which starred in movies with lengths higher or equal to the average length of all the movies

```
SELECT AVG(length) AS average FROM sakila.film;
```

SUBQUERIES

Example: find the names of the actors which starred in movies with lengths higher or equal to the average length of all the movies

```
SELECT *
FROM sakila.film
WHERE film.length >= (SELECT AVG(length) AS average FROM film);
```

SUBQUERIES

Example: find the names of the actors which starred in movies with lengths higher or equal to the average length of all the movies

```
SELECT film_id
FROM sakila.film
WHERE length > (SELECT AVG(length) AS average FROM film)
```

SUBQUERIES

Example: find the names of the actors which starred in movies with lengths higher or equal to the average length of all the movies

```
SELECT DISTINCT actor_id
FROM sakila.film_actor
INNER JOIN (SELECT film_id FROM sakila.film WHERE length >= (SELECT AVG(length) AS average
FROM film)) AS selected_films_id
ON sakila.film_actor.film_id = selected_films_id.film_id;
```

SUBQUERIES

Example: find the names of the actors which starred in movies with lengths higher or equal to the average length of all the movies

```
SELECT *
FROM sakila.actor
INNER JOIN (
SELECT DISTINCT actor_id
FROM sakila.film_actor
INNER JOIN (SELECT film_id FROM sakila.film WHERE length >= (SELECT AVG(length) AS average
FROM film)) AS selected_films_id
ON sakila.film_actor.film_id = selected_films_id.film_id) AS selected_actors
ON sakila.actor.actor_id = selected_actors.actor_id;
```

TEMPORARY TABLES

COMPLEXITY BREEDS ERROR

Subqueries can easily become very complex and therefore hard to debug and get wrong

Temporary Tables can be reused and stored for a whole session

They are destroyed after the session, no actual database space is used

TEMPORARY TABLES

```
CREATE TEMPORARY TABLE sakila.selected_films
SELECT film_id
FROM sakila.film
WHERE length > (SELECT AVG(length) AS average FROM film);


CREATE TEMPORARY TABLE sakila.selected_actors
SELECT DISTINCT actor_id
FROM sakila.film_actor
INNER JOIN sakila.selected_films
ON sakila.film_actor.film_id = selected_films.film_id;

SELECT first_name, last_name
FROM sakila.actor
INNER JOIN sakila.selected_actors
ON sakila.actor.actor_id = selected_actors.actor_id;
```