# Que es SQL?

- SQL: `Structured Query Language`
- Es un lenguaje de consultas estructuradas
- Diseñado para administrar y consultar bases de datos
- SQL es un estandar desde 1987 (`tiene una especificación`)

# Motores de bases de datos

- **mySQL**
- **postgreSQL**
- **SQLite**

Dentro de cada motor, podemos tener `bases de datos`

Cada base de datos, contiene `tablas`

# Tipos de bases de datos

## Relacionales o SQL

- Existen relaciones que evitan la duplicidad de los datos , por eso existen `relaciones` entre las tablas generalmente con **IDs**

- Cada tabla o entidad tiene `atributos`

#### Contras

- Hacer `muchas consultas` para recuperar la info que es necesaria

#### Pros

- `Ocupan menos lugar` porque no repiten los datos
- `Coherencia` en la información almacenada

## No relacionales o No SQL

- Colecciones de documentos que se pueden relacionar pero es más costoso

#### Contras

- Normalizar los datos es problemático
  - Que el cambio realizado llegue a todas las referencias
- Updates en todos los documentos
- Integridad de los datos no está garantizada

#### Pros

- Consultas `más rápidas` y especialmente las que son grandes

# Herramientas y programas

### Programas

- URL: https://dev.mysql.com/downloads/
- MySQL Community Downloads --> **MySQL Workbench** (macOS)
- MySQL Community --> **MySQL Community Installer de 430MB** (Windows)

### Herramientas para bases de dastos

- **DBngin** (recomendando macOS)
  - URL: https://dbngin.com/
  - Detecta bases, se puede iniciar MySQL, postGre, Redis
- **SQLBolt**
  - Para practicar bases de datos y sintaxis con ejercicios interactivos
  - URL: https://sqlbolt.com/

# Primeros pasos con SQL (Ejercicios de SQLBolt)

- **IMPORTANTE**

  - **NO** es CASE SENSITIVE

- **RECOMENDABLE**

  - Comandos en `MAYÚSCULAS`
  - Nombre de columnas o tablas en `minúsculas`

  ## SQLBolt Lessons

  ### Lesson 1 : SELECT Queries

  Utilizando la siguiente tabla como ejemplo:

| id | Title | Director | Year | Length (minutes) |
|---|---|---|---|---|
| 1 | Toy Story | John Lasseter | 1995 | 81 |
| 2 | A Bug's Life | John Lasseter | 1998 | 95 |
| 3 | Toy Story 2 | John Lasseter | 1999 | 93 |
| 4 | Monsters, Inc. | Pete Docter | 2001 | 92 |
| 5 | Finding Nemo | Andrew Stanton | 2003 | 107 |
| 6 | The Incredibles | Brad Bird | 2004 | 116 |
| 7 | Cars | John Lasseter | 2006 | 117 |

| id | Title | Director | Year | Length (minutes) |
|---|---|---|---|---|
| 8 | Ratatouille | Brad Bird | 2007 | 115 |
| 9 | WALL-E | Andrew Stanton | 2008 | 104 |
| 10 | Up | Pete Docter | 2009 | 101 |

### Queries

```
SELECT * FROM movies
```

Trae `todas` las columnas de la tabla `movies`

```
SELECT director FROM movies
```

Trae la columna `director` de la tabla `movies`

```
SELECT director, titles FROM movies
```

Trae las columnas `director` y `titles` de la tabla `movies`

---

## Lesson 2: Queries con CONSTRAINTS

Permiten filtrar los resultados de consultas con el operador `WHERE`

```
SELECT column, another_column, …
FROM mytable WHERE condition
AND/OR another_condition
AND/OR …;
```

### Operadores para **data numérica**:

| Operator | Condition | SQL Example |
|---|---|---|
| =, !=, <, <=, >, >= | Standard numerical operators | `col_name != 4` |
| BETWEEN … AND … | Number is within range of two values (inclusive) | `col_name BETWEEN 1.5 AND 10.5` |
| NOT BETWEEN … AND … | Number is not within range of two values (inclusive) | `col_name NOT BETWEEN 1 AND 10` |
| IN (…) | Number exists in a list | `col_name IN (2, 4, 6)` |

| Operator | Condition | SQL Example |
|---|---|---|
| NOT IN (…) | Number does not exist in a list | `col_name NOT IN (1, 3, 5)` |

Utilizando la tabla anterior como ejemplo:

```
SELECT * FROM movies WHERE id = 6;
```

Trae todo de la tabla `movies` que tenga el ID = 6

```
SELECT * FROM movies WHERE year BETWEEN 2000 AND 2010;
```

Trae todo de la tabla `movies` que tenga el `year` entre el 2000 y el 2010

```
SELECT * FROM movies WHERE year NOT BETWEEN 2000 AND 2010;
```

Trae todo de la tabla `movies` que **NO** tenga el `year` entre el 2000 y el 2010

```
SELECT * FROM movies WHERE id BETWEEN 1 AND 5;
```

Trae todo de la tabla `movies` que tenga el `id` entre el 1 y el 5, es decir los 5 primeros ya que el ID es `autoincremental`.

---

## Lesson 3: Queries con CONSTRAINTS parte 2

Operadores para **data de texto**:

| Operator | Condition | Example |
|---|---|---|
| = | Case sensitive exact string comparison (notice the single equals) | `col_name = "abc"` |
| != or <> | Case sensitive exact string inequality comparison | `col_name != "abcd"` |
| LIKE | Case insensitive exact string comparison | `col_name LIKE "ABC"` |
| NOT LIKE | Case insensitive exact string inequality comparison | `col_name NOT LIKE "ABCD"` |
| % | Used anywhere in a string to match a sequence of zero or more characters (only with LIKE or NOT LIKE) | `col_name LIKE "%AT%"` (matches "AT", "ATTIC", "CAT" or even "BATS") |

| Operator | Condition | Example |
|---|---|---|
| _ | Used anywhere in a string to match a single character (only with LIKE or NOT LIKE) | `col_name LIKE "AN_"` (matches "AND", but not "AN") |
| IN (…) | String exists in a list | `col_name IN ("A", "B", "C")` |
| NOT IN (…) | String does not exist in a list | `col_name NOT IN ("D", "E", "F")` |

A raiz de la siguiente tabla:

| id | Title | Director | Year | Length (minutes) |
|---|---|---|---|---|
| 1 | Toy Story | John Lasseter | 1995 | 81 |
| 2 | A Bug's Life | John Lasseter | 1998 | 95 |
| 3 | Toy Story 2 | John Lasseter | 1999 | 93 |
| 4 | Monsters, Inc. | Pete Docter | 2001 | 92 |
| 5 | Finding Nemo | Andrew Stanton | 2003 | 107 |
| 6 | The Incredibles | Brad Bird | 2004 | 116 |
| 7 | Cars | John Lasseter | 2006 | 117 |
| 8 | Ratatouille | Brad Bird | 2007 | 115 |
| 9 | WALL-E | Andrew Stanton | 2008 | 104 |
| 10 | Up | Pete Docter | 2009 | 101 |
| 11 | Toy Story 3 | Lee Unkrich | 2010 | 103 |
| 12 | Cars 2 | John Lasseter | 2011 | 120 |
| 13 | Brave | Brenda Chapman | 2012 | 102 |
| 14 | Monsters University | Dan Scanlon | 2013 | 110 |
| 87 | WALL-G | Brenda Chapman | 2042 | 97 |

### Queries

```
SELECT title FROM movies WHERE title LIKE "Toy Story%";
```

- Trae las filas de la columna `title` que contengan el texto `"Toy Story"`.
- El `%` indica que luego hay mas caracteres.

```
SELECT title FROM movies WHERE director LIKE "John Lasseter"
```

- Trae las filas de la columna `title` en donde `director` sea `"John Lasseter"`.
- El `LIKE` permite buscar por coincidencia exacta del texto **(NO ES CASE SENSITIVE)**.

```
SELECT title FROM movies WHERE director NOT LIKE "John Lasseter"
```

- Trae las filas de la columna `title` en donde `director` **NO** sea `"John Lasseter"`.
- El `LIKE` permite buscar por coincidencia exacta del texto **(NO ES CASE SENSITIVE)**.

```
SELECT title FROM movies WHERE title LIKE "WALL-%"
```

- Trae las filas de la columna `title` en donde contenga "WALL-"
- El `%` indica que luego hay mas caracteres.

## Lesson 4: Filtrar y ordenar resultados de consultas

### Operador `DISTINCT` para **resultados únicos y no duplicados**:

```
SELECT DISTINCT column, another_column, …
FROM mytable
WHERE condition(s);
```

### Operador `ORDER BY` para **ordenar resultados en ascedente o descendente**:

```
SELECT column, another_column, …
FROM mytable
WHERE condition(s)
ORDER BY column ASC/DESC;
```

### Operadores `LIMIT` y `OFFSET` para **limitar y saltear resultados**:

```
SELECT column, another_column, …
FROM mytable
WHERE condition(s)
ORDER BY column ASC/DESC
LIMIT num_limit OFFSET num_offset;
```

### A raiz de la siguiente tabla:

| id | Title | Director | Year | Length (minutes) |
|---|---|---|---|---|
| 1 | Monsters, Inc. | Pete Docter | 2001 | 92 |
| 2 | Up | Pete Docter | 2009 | 101 |
| 3 | The Incredibles | Brad Bird | 2004 | 116 |
| 4 | Toy Story 3 | Lee Unkrich | 2010 | 103 |
| 5 | A Bug's Life | John Lasseter | 1998 | 95 |
| 6 | Toy Story 2 | John Lasseter | 1999 | 93 |
| 7 | Brave | Brenda Chapman | 2012 | 102 |
| 8 | Monsters University | Dan Scanlon | 2013 | 110 |
| 9 | WALL-E | Andrew Stanton | 2008 | 104 |
| 10 | Toy Story | John Lasseter | 1995 | 81 |
| 11 | Cars | John Lasseter | 2006 | 117 |
| 12 | Finding Nemo | Andrew Stanton | 2003 | 107 |
| 13 | Ratatouille | Brad Bird | 2007 | 115 |
| 14 | Cars 2 | John Lasseter | 2011 | 120 |

### Queries

```
SELECT DISTINCT director FROM movies ORDER BY director ASC;
```

- Trae las filas de la columna `director` ordenadas alfabéticamente y sin repetirse.
- Por defector, ORDER BY es ASC (ascedente). Puede o no ponerse el ASC.

```
 SELECT * FROM movies ORDER BY year DESC LIMIT 4
```

- Trae todas las filas ordenadas por la columna `year` en orden descendente (por mas recientes)
- `LIMIT` permite traer una cantidad fija, en este caso 4

```
SELECT * FROM movies ORDER BY title ASC LIMIT 5
```

- Trae todas las filas ordenadas por la columna `title` en orden ascedente (alfabéticamente)
- `LIMIT` permite traer una cantidad fija, en este caso 5

```
SELECT * FROM movies ORDER BY title ASC LIMIT 5 OFFSET 5
```

- Trae todas las filas ordenadas por la columna `title` en orden ascedente (alfabéticamente)
- `LIMIT` permite traer una cantidad fija, en este caso 5
- `OFFSET` permite traer las **PROXIMAS** filas, en este caso salteando las 5 primeras

---

## SQL Review: Simple SELECT Queries

### Poner en práctica la query SELECT con problemas reales

```
SELECT column, another_column, …
FROM mytable
WHERE condition(s)
ORDER BY column ASC/DESC
LIMIT num_limit OFFSET num_offset;
```

### Considerando la siguiente tabla de países de Norte América:

| City | Country | Population | Latitude | Longitude |
|---|---|---|---|---|
| Guadalajara | Mexico | 1,500,800 | 20.659699 | -103.349609 |
| Toronto | Canada | 2,795,060 | 43.653226 | -79.383184 |
| Houston | United States | 2,195,914 | 29.760427 | -95.369803 |
| New York | United States | 8,405,837 | 40.712784 | -74.005941 |
| Philadelphia | United States | 1,553,165 | 39.952584 | -75.165222 |
| Havana | Cuba | 2,106,146 | 23.054070 | -82.345189 |
| Mexico City | Mexico | 8,555,500 | 19.432608 | -99.133208 |
| Phoenix | United States | 1,513,367 | 33.448377 | -112.074037 |
| Los Angeles | United States | 3,884,307 | 34.052234 | -118.243685 |
| Ecatepec de Morelos | Mexico | 1,742,000 | 19.601841 | -99.050674 |
| Montreal | Canada | 1,717,767 | 45.501689 | -73.567256 |
| Chicago | United States | 2,718,782 | 41.878114 | -87.629798 |

### Queries

#### 1 - List all the Canadian cities and their populations

```
SELECT * FROM north_american_cities WHERE country = "Canada"
```

**2- Order all the cities in the United States by their latitude from north to south**

```
SELECT * FROM north_american_cities
WHERE country = "United States"
ORDER BY latitude DESC;
```

- Aqui tambien funciona `WHERE country LIKE "united states"` pero es una query MENOS eficiente porque debe buscar más

**3- List all the cities west of Chicago, ordered from west to east**

```
  SELECT city, longitude
  FROM north_american_cities
  WHERE longitude < -87.629798
  ORDER BY longitude ASC;
```

**4- List the two largest cities in Mexico (by population)**

```
  SELECT city, population
  FROM north_american_cities
  WHERE country LIKE "Mexico"
  ORDER BY population DESC
  LIMIT 2;
```

**5- List the third and fourth largest cities (by population) in the United States and their population**

```
  SELECT city, population
  FROM north_american_cities
  WHERE country LIKE "United States"
  ORDER BY population DESC
  LIMIT 2
  OFFSET 2;
```

---

## Lesson 6: Multi-table queries with JOINs

Operador `INNER JOIN` para **unir filas de la primera tabla y de la segunda tabla que tienen la misma clave**:

SELECT queries 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;
```

Considerando la primer tabla (Movies):

| id | Title | Director | Year | Length (minutes) |
|---|---|---|---|---|
| 1 | Toy Story | John Lasseter | 1995 | 81 |
| 2 | A Bug's Life | John Lasseter | 1998 | 95 |
| 3 | Toy Story 2 | John Lasseter | 1999 | 93 |
| 4 | Monsters, Inc. | Pete Docter | 2001 | 92 |
| 5 | Finding Nemo | Andrew Stanton | 2003 | 107 |
| 6 | The Incredibles | Brad Bird | 2004 | 116 |
| 7 | Cars | John Lasseter | 2006 | 117 |
| 8 | Ratatouille | Brad Bird | 2007 | 115 |
| 9 | WALL-E | Andrew Stanton | 2008 | 104 |
| 10 | Up | Pete Docter | 2009 | 101 |
| 11 | Toy Story 3 | Lee Unkrich | 2010 | 103 |
| 12 | Cars 2 | John Lasseter | 2011 | 120 |
| 13 | Brave | Brenda Chapman | 2012 | 102 |
| 14 | Monsters University | Dan Scanlon | 2013 | 110 |

Considerando la segunda tabla (Boxoffice):

| Movie ID | Rating | Domestic Sales ($) | International Sales ($) |
|---|---|---|---|
| 5 | 8.2 | 380,843,261 | 555,900,000 |
| 14 | 7.4 | 268,492,764 | 475,066,843 |
| 8 | 8.0 | 206,445,654 | 417,277,164 |
| 12 | 6.4 | 191,452,396 | 368,400,000 |
| 3 | 7.9 | 245,852,179 | 239,163,000 |

| Movie ID | Rating | Domestic Sales ($) | International Sales ($) |
|---|---|---|---|
| 6 | 8.0 | 261,441,092 | 370,001,000 |
| 9 | 8.5 | 223,808,164 | 297,503,696 |
| 11 | 8.4 | 415,004,880 | 648,167,031 |
| 1 | 8.3 | 191,796,233 | 170,162,503 |
| 7 | 7.2 | 244,082,982 | 217,900,167 |
| 10 | 8.3 | 293,004,164 | 438,338,580 |
| 4 | 8.1 | 289,916,256 | 272,900,000 |
| 2 | 7.2 | 162,798,565 | 200,600,000 |
| 13 | 7.2 | 237,283,207 | 301,700,000 |

### Queries

**1 - Find the domestic and international sales for each movie**

```
SELECT title, domestic_sales, international_sales
FROM movies
INNER JOIN boxoffice
ON id = movie_id;
```

**2 - Show the sales numbers for each movie that did better internationally rather than domestically**

```
SELECT title, domestic_sales, international_sales
FROM movies
INNER JOIN boxoffice
ON id = movie_id
WHERE international_sales > domestic_sales;
```

**3 - List all the movies by their ratings in descending order**

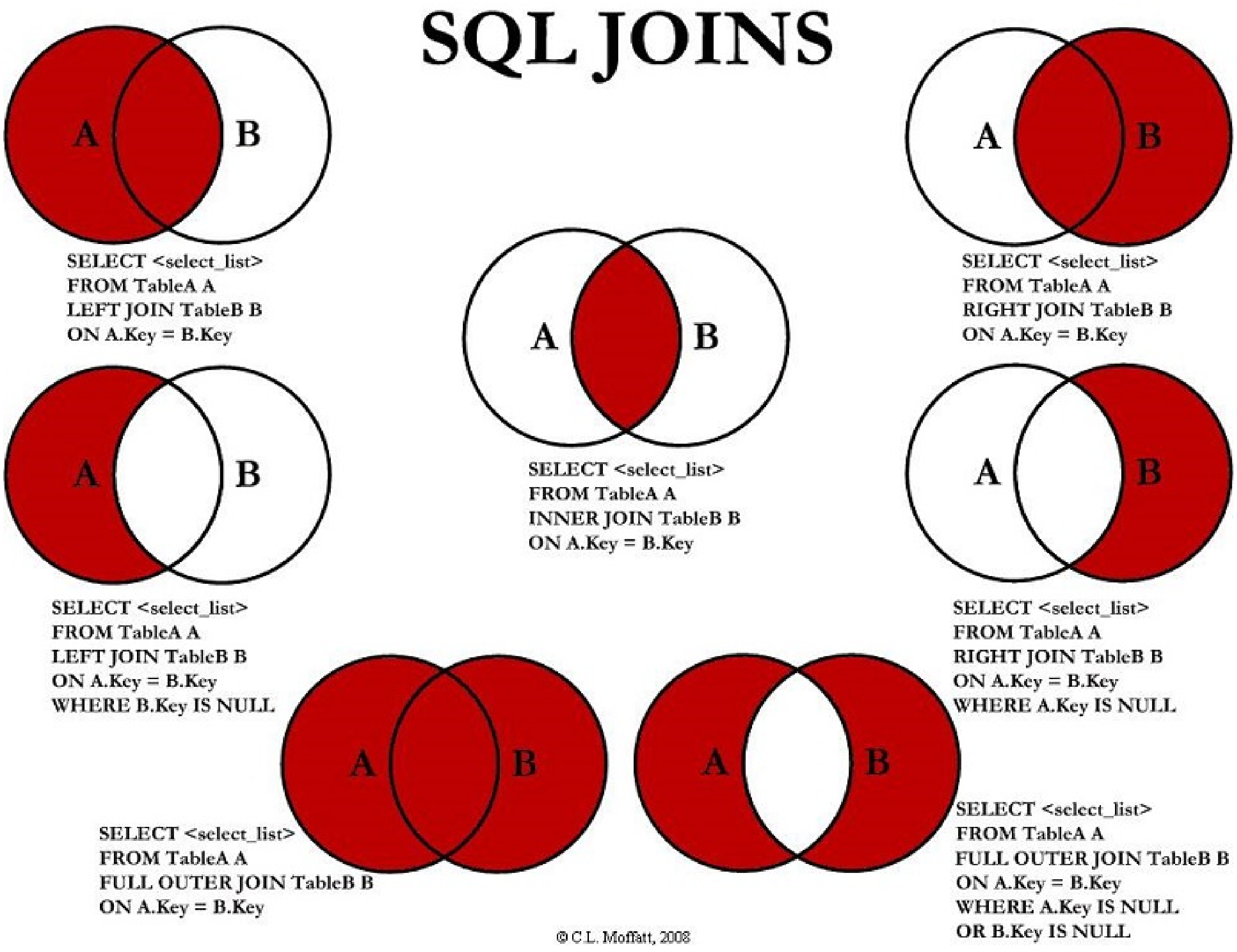
```
SELECT title, domestic_sales, international_sales
FROM movies
INNER JOIN boxoffice
ON id = movie_id
ORDER BY rating DESC;
```

---

## Lesson 7: OUTER JOINs

Si las 2 tablas en cuestion tienen data asimetrica, se utliza `LEFT JOIN`, `RIGHT JOIN` o `FULL JOIN`

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



Considerando la primer tabla (Buildings):

| Building Name | Capacity |
|---|---|
| 1e | 24 |
| 1w | 32 |
| 2e | 16 |
| 2w | 20 |

Considerando la segunda tabla (Employees):

| Role | Name | Building | Years Employed |
|---|---|---|---|
| Engineer | Becky A. | 1e | 4 |
| Engineer | Dan B. | 1e | 2 |
| Engineer | Sharon F. | 1e | 6 |
| Engineer | Dan M. | 1e | 4 |
| Engineer | Malcom S. | 1e | 1 |
| Artist | Tylar S. | 2w | 2 |
| Artist | Sherman D. | 2w | 8 |
| Artist | Jakob J. | 2w | 6 |
| Artist | Lillia A. | 2w | 7 |
| Artist | Brandon J. | 2w | 7 |
| Manager | Scott K. | 1e | 9 |
| Manager | Shirlee M. | 1e | 3 |
| Manager | Daria O. | 2w | 6 |

## Queries

#### 1- Find the list of all buildings that have employees

```
SELECT DISTINCT building FROM employees;
```

- No hizo falta hacer ningun `JOIN`. Atento a esto! Siempre que se pueda, evitarlos.

#### 2- Find the list of all buildings and their capacity

```
SELECT * FROM buildings;
```

- No hizo falta hacer ningun `JOIN`.

#### 3- List all buildings and the distinct employee roles in each building (including empty buildings)

```
SELECT DISTINCT building_name, role
FROM buildings
LEFT JOIN employees
ON building_name = building;
```

## Lesson 8: A short note on NULLs

Una alternativa a los valores NULL en su base de datos es tener valores predeterminados apropiados para el tipo de datos, como 0 para datos numéricos, cadenas vacías para datos de texto, etc.

Pero si su base de datos necesita almacenar datos incompletos, entonces los valores NULL pueden ser apropiados si los valores predeterminados distorsionarán el análisis posterior (por ejemplo, al tomar promedios de datos numéricos).

```
SELECT column, another_column, …
FROM mytable
WHERE column IS/IS NOT NULL
AND/OR another_condition
AND/OR …;
```

Considerando la primer tabla (Buildings):

| Building Name | Capacity |
|---|---|
| 1e | 24 |
| 1w | 32 |
| 2e | 16 |
| 2w | 20 |

Considerando la segunda tabla (Employees):

| Role | Name | Building | Years Employed |
|---|---|---|---|
| Engineer | Becky A. | 1e | 4 |
| Engineer | Dan B. | 1e | 2 |
| Engineer | Sharon F. | 1e | 6 |
| Engineer | Dan M. | 1e | 4 |
| Engineer | Malcom S. | 1e | 1 |
| Artist | Tylar S. | 2w | 2 |
| Artist | Sherman D. | 2w | 8 |
| Artist | Jakob J. | 2w | 6 |
| Artist | Lillia A. | 2w | 7 |
| Artist | Brandon J. | 2w | 7 |

| Role | Name | Building | Years Employed |
|---|---|---|---|
| Manager | Scott K. | 1e | 9 |
| Manager | Shirlee M. | 1e | 3 |
| Manager | Daria O. | 2w | 6 |
| Engineer | Yancy I. | | 0 |
| Artist | Oliver P. | | 0 |

## Queries

#### 1 - Find the name and role of all employees who have not been assigned to a building

```
SELECT name, role
FROM employees
WHERE building IS NULL;
```

#### 2 - Find the names of the buildings that hold no employees

```
SELECT DISTINCT building_name
FROM buildings
  LEFT JOIN employees
    ON building_name = building
WHERE role IS NULL;
```