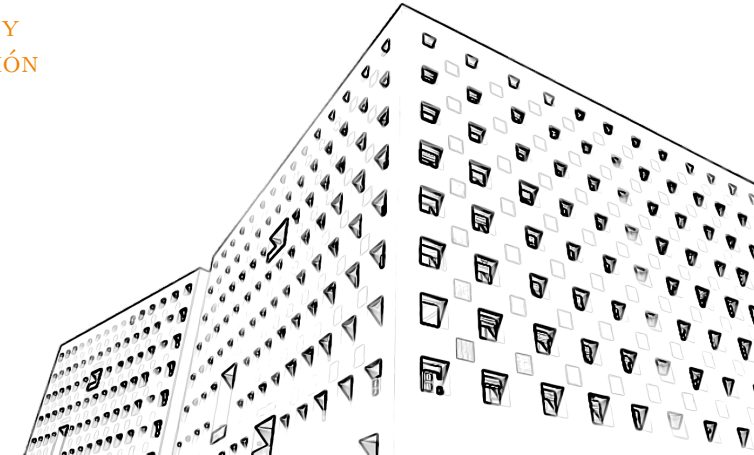
Universidad Autónoma Metropolitana
Unidad Cuajimalpa

Licenciatura en Sistemas y Tecnologías de Información

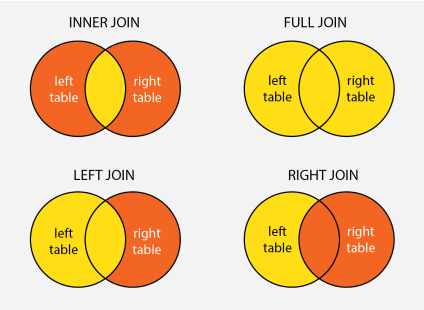
Bases de datos

Dr. Guillermo Monroy

Trimestre 25-P



Casa abierta al tiempo



Objetivo de la presentación

1. Entender el concepto de ”joins” en bases de datos
2. Aprender a aplicar los ”joins” en la validación de bases de datos

- Definición de Consultas JOIN
- Tipos de JOINS
- Ejemplo de tablas

Consultas JOIN

Las consultas JOIN en SQL se utilizan para combinar filas de dos o más tablas basándose en una relación entre ellas. Permiten recuperar datos relacionados de múltiples tablas en una sola consulta.

Importancia de los JOINS

Los JOINS son fundamentales para trabajar con bases de datos relacionales, ya que permiten:

- Combinar datos de múltiples tablas de manera eficiente.
- Realizar consultas complejas que involucren relaciones entre diferentes entidades.
- Mantener la integridad referencial al relacionar datos.

Casa abierta al tiempo

Tipos de JOINS

Existen varios tipos de JOINS en SQL, cada uno con un propósito específico:

- INNER JOIN
- LEFT JOIN (o LEFT OUTER JOIN)
- RIGHT JOIN (o RIGHT OUTER JOIN)
- FULL JOIN (o FULL OUTER JOIN)
- CROSS JOIN
- SELF JOIN

Uso de JOINS

Los JOINS se utilizan para:

- Combinar datos de diferentes tablas basándose en una condición de coincidencia.
- Extraer información relacionada de múltiples tablas en una sola consulta.
- Facilitar la normalización de bases de datos al evitar la redundancia de datos.

- Definición de Consultas JOIN
- Tipos de JOINS
- Ejemplo de tablas

Vamos a usar dos tablas de ejemplo:

```sql
-- Tabla de Empleados
CREATE TABLE empleados (
    id INT PRIMARY KEY,
    nombre VARCHAR(50),
    departamento_id INT
);

-- Tabla de Departamentos
CREATE TABLE departamentos (
    id INT PRIMARY KEY,
    nombre VARCHAR(50)
);

```

```sql
-- Datos de ejemplo
INSERT INTO empleados VALUES
(1, 'Ana', 1),
(2, 'Luis', 2),
(3, 'Marta', NULL),
(4, 'Carlos', 3);

INSERT INTO departamentos VALUES
(1, 'Recursos Humanos'),
(2, 'TI'),
(4, 'Marketing');
```

INNER JOIN

2 Tipos de JOINS

Devuelve solo las filas que contienen coincidencias en ambas tablas.

```
1 SELECT e.nombre AS empleado, d.nombre AS departamento
2 FROM empleados e
3 INNER JOIN departamentos d ON e.departamento_id = d.id;
4
5
```

LEFT JOIN (LEFT OUTER JOIN)

2 Tipos de JOINS

Devuelve todas las filas de la tabla izquierda y las coincidentes de la derecha. Si no hay coincidencia, pone NULL.

```
1 SELECT e.nombre AS empleado, d.nombre AS departamento
2 FROM empleados e
3 LEFT JOIN departamentos d ON e.departamento_id = d.id;
4
```

RIGHT JOIN (RIGHT OUTER JOIN)

Devuelve todas las filas de la tabla derecha y las coincidentes de la izquierda. Si no hay coincidencia, pone NULL.

```
SELECT e.nombre AS empleado, d.nombre AS departamento
FROM empleados e
RIGHT JOIN departamentos d ON e.departamento_id = d.id;


```

FULL JOIN (FULL OUTER JOIN)

Devuelve todas las filas de ambas tablas, coincidan o no. Donde no hay coincidencias, pone NULL.

```
1 SELECT e.nombre AS empleado, d.nombre AS departamento
2 FROM empleados e
3 FULL JOIN departamentos d ON e.departamento_id = d.id;
4
5
6
```

CROSS JOIN

2 Tipos de JOINS

Hace el producto cartesiano entre dos tablas: combina cada fila de una tabla con todas las filas de la otra.

```
1 SELECT e.nombre AS empleado, d.nombre AS departamento
2 FROM empleados e
3 CROSS JOIN departamentos d;
4
5
6
7
```

SELF JOIN

2 Tipos de JOINS

Se usa para hacer un join de una tabla consigo misma.
Ejemplo: jerarquía de empleados (supervisor-subordinado)

```sql
-- Nueva tabla empleados con jefe_id
CREATE TABLE empleados (
    id INT PRIMARY KEY,
    nombre VARCHAR(50),
    jefe_id INT
);

INSERT INTO empleados VALUES
(1, 'Ana', NULL),
(2, 'Luis', 1),
(3, 'Marta', 1),
(4, 'Carlos', 2);

-- Consulta self join
SELECT e.nombre AS empleado, j.nombre AS jefe
FROM empleados e
LEFT JOIN empleados j ON e.jefe_id = j.id;
```

- Definición de Consultas JOIN
- Tipos de JOINS
- Ejemplo de tablas

Ejemplo de tablas

3 Ejemplo de tablas

- Imaginemos dos tablas: **empleados** y **departamentos**.
- La tabla **empleados** contiene información sobre los empleados y su departamento.
- La tabla **departamentos** contiene información sobre los departamentos de la empresa.

Objetivo

Aprenderemos a usar diferentes tipos de joins para combinar estas tablas y obtener información útil.

Estructura de las tablas

3 Ejemplo de tablas

- Tabla **empleados**:
  - id: Identificador único del empleado
  - nombre: Nombre del empleado
  - departamento_id: Identificador del departamento al que pertenece el empleado
- Tabla **departamentos**:
  - id: Identificador único del departamento
  - nombre: Nombre del departamento

Relación

La relación entre las tablas se establece a través del campo **departamento_ id** en la tabla **empleados**, que hace referencia al campo **id** en la tabla **departamentos**.

Ejemplo de consultas JOIN

3 Ejemplo de tablas

- Vamos a realizar consultas JOIN utilizando las tablas **empleados** y **departamentos**.
- Estas consultas nos permitirán obtener información combinada de ambas tablas.

Objetivo de las consultas

Aprenderemos a usar diferentes tipos de joins para:

- Obtener empleados y sus departamentos.
- Identificar empleados sin departamento.
- Ver todos los departamentos, incluso aquellos sin empleados.

SQL JOIN del ejemplo

3 Ejemplo de tablas

Código SQL para crear las tablas

```sql
-- Tabla de Empleados
CREATE TABLE empleados (
    id INT PRIMARY KEY,
    nombre VARCHAR(50),
    departamento_id INT
);
-- Tabla de Departamentos
CREATE TABLE departamentos (
    id INT PRIMARY KEY,
    nombre VARCHAR(50)
);

```

SQL JOIN del ejemplo

3 Ejemplo de tablas

Código SQL para insertar datos

```sql
-- Datos de ejemplo
INSERT INTO empleados VALUES
(1, 'Ana', 1),
(2, 'Luis', 2),
(3, 'Marta', NULL),
(4, 'Carlos', 3);

INSERT INTO departamentos VALUES
(1, 'Recursos Humanos'),
(2, 'TI'),
(4, 'Marketing');

```

Vamos a usar dos tablas de ejemplo:

```sql
-- Tabla de Empleados
CREATE TABLE empleados (
    id INT PRIMARY KEY,
    nombre VARCHAR(50),
    departamento_id INT
);

-- Tabla de Departamentos
CREATE TABLE departamentos (
    id INT PRIMARY KEY,
    nombre VARCHAR(50)
);

```

Estructura de las tablas

3 Ejemplo de tablas

Estructura de las tablas

- Tabla **empleados**:
  - id: Identificador único del empleado
  - nombre: Nombre del empleado
  - departamento_id: Identificador del departamento al que pertenece el empleado
- Tabla **departamentos**:
  - id: Identificador único del departamento
  - nombre: Nombre del departamento

Relación

La relación entre las tablas se establece a través del campo **departamento_ id** en la tabla **empleados**, que hace referencia al campo **id** en la tabla **departamentos**.

Código SQL JOIN

3 Ejemplo de tablas

Código SQL para crear las tablas

```sql
-- Tabla de Empleados
CREATE TABLE empleados (
    id INT PRIMARY KEY,
    nombre VARCHAR(50),
    departamento_id INT
);
-- Tabla de Departamentos
CREATE TABLE departamentos (
    id INT PRIMARY KEY,
    nombre VARCHAR(50)
);

```

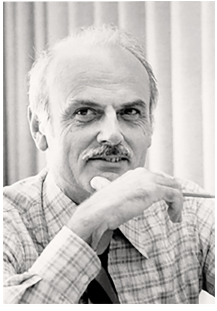
Código SQL para insertar datos

```sql
-- Datos de ejemplo
INSERT INTO empleados VALUES
(1, 'Ana', 1),
```

¡Gracias por su atención!

3 Ejemplo de tablas

- Preguntas o comentarios
- Contacto: Dr. Guillermo Monroy



Edgar F. Codd, creador del modelo relacional de bases de datos

Dr. Guillermo Monroy
Universidad Autonoma Metropolina, Unidad Cuajimalpa.

Temporary page!

LaTeX was unable to guess the total number of pages correctly. As there was so
unprocessed data that should have been added to the final page this extra page
has been added to receive it.
If you rerun the document (without altering it) this surplus page will go away,
because LaTeX now knows how many pages to expect for this document.