

Tema 10: Introducción a SQL

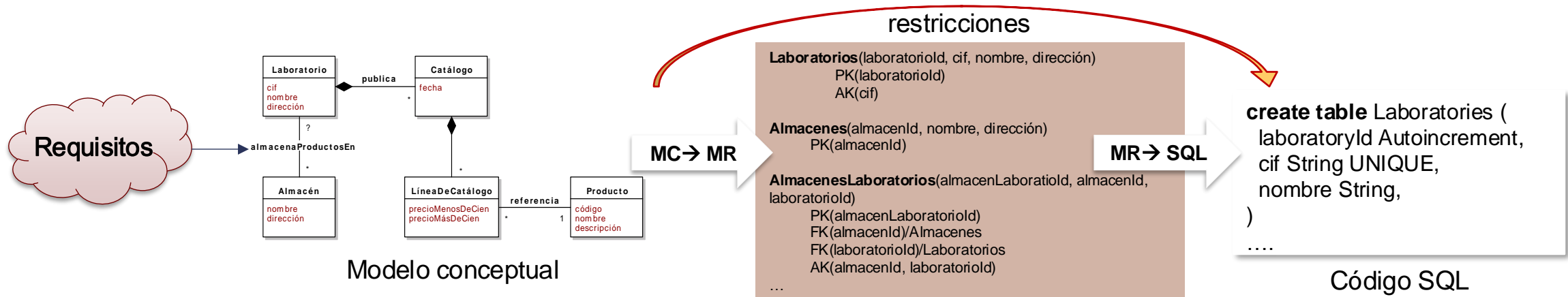
Introducción a la Ingeniería del Software y los Sistemas de Información I
Ingeniería Informática – Tecnologías Informáticas
Departamento de Lenguajes y Sistemas Informáticos



1. Introducción
2. Lenguaje de definición de datos (DDL)
3. Lenguaje de manipulación de datos (DML)
4. Lenguaje de consulta de datos (DQL)
5. Consultas complejas

Trazabilidad de modelos

- A partir del modelo conceptual se puede obtener un modelo relacional que se implementa posteriormente en SQL para obtener el esquema de la base de datos.

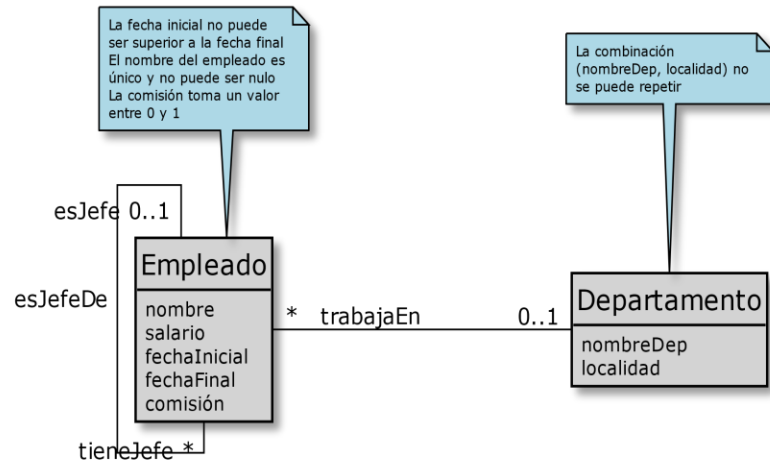


- SQL (*Structured Query Language*) es el lenguaje **estándar** para definir, manipular y consultar bases de datos relacionales.
- Se puede distinguir:
 - DDL (*Data Definition Language*): gestión del esquema de la base de datos (creación, modificación y borrado de tablas, claves, etc.). CREATE, ALTER, DROP
 - DML (*Data Manipulation Language*): gestión de los datos. INSERT, UPDATE, DELETE
 - DCL (*Data Control Language*): Control de acceso y permisos. GRANT y REVOKE
 - DQL (*Data Query Language*): Gestión de consultas. SELECT
 - TCL (*Transaction Control Language*): gestión de transacciones. COMMIT, ROLLBACK, TRANSACTION

Tutoriales SQL

- ◆ <http://www.w3schools.com/sql/>
- ◆ <https://www.tutorialspoint.com/sql/index.htm>
- ◆ <https://mariadb.com/kb/en/library/basic-sql-statements/>
- ◆ https://www.hcoe.edu.np/uploads/attachments/r96oytechsac_gzi4.pdf

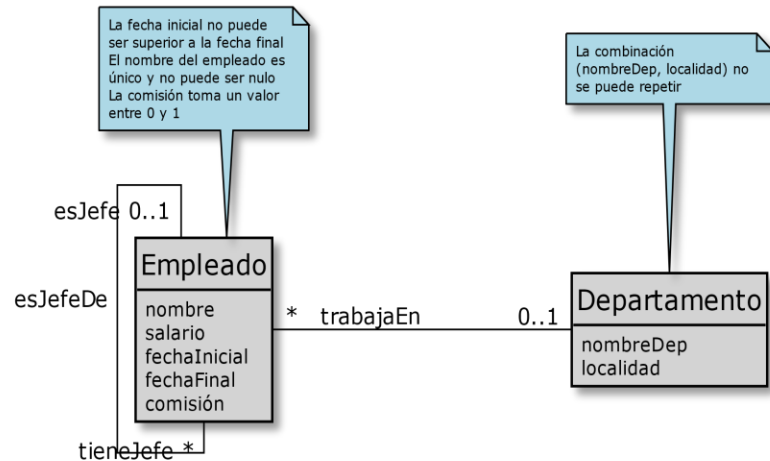
Creación de tablas



Departamentos(departamentoid, nombreDep, localidad)
PK(departamentoid)
AK(nombreDep, localidad)

```
CREATE TABLE Departamentos (  
    departamentoId INT NOT NULL AUTO_INCREMENT,  
    nombreDep VARCHAR(32),  
    localidad VARCHAR(64),  
    PRIMARY KEY(departamentoId),  
    UNIQUE(nombreDep, localidad)  
);
```

Creación de tablas



Empleados(empleadold, departamentold, jefeld, nombre, salario, fechaInicial, fechaFinal, comision)
PK(empleadold)
FK(departamentold)/Departamentos
FK(jefeld)/Empleados

Creación de tablas

Empleados(empleadoid, departamentoid, jefeid, nombre, salario, fechaInicial, fechaFinal, comision)

PK(empleadoid)

FK(departamentoid)/Departamentos

FK(jefeid)/Empleados



```
CREATE TABLE Empleados (  
    empleadoId INT NOT NULL AUTO_INCREMENT,  
    departamentoId INT,  
    jefe INT,  
    nombre VARCHAR(64) NOT NULL,  
    salario DECIMAL(6,2) DEFAULT 2000.00,  
    fechaInicial DATE,  
    fechaFinal DATE,  
    comision DOUBLE,  
    PRIMARY KEY(empleadoId),  
    FOREIGN KEY(departamentoId)  
        REFERENCES Departamentos(departamentoId)  
        ON DELETE SET NULL,  
    FOREIGN KEY(jefe)  
        REFERENCES Empleados(empleadoId),  
    UNIQUE(nombre),  
    CHECK (comision >=0 AND comision <=1),  
    CONSTRAINT fecha CHECK (fechaInicial < fechaFinal)  
);
```


Claves

- Claves primarias: PRIMARY KEY(tablaId)
- Claves alternativas: UNIQUE(atributo) o UNIQUE(atributo1, atributo2...)
- Claves ajenas:

FOREIGN KEY(atributo) **REFERENCES** OtraTabla(otraTablaId)

ON DELETE:

- **RESTRICT** (por defecto)
- **CASCADE**
- **SET NULL**
- **SET DEFAULT**

ON UPDATE: ...

Integridad referencial

The allowed actions for `ON DELETE` and `ON UPDATE` are:

- `RESTRICT` : The change on the parent table is prevented. The statement terminates with a 1451 error (`SQLSTATE '2300'`). This is the default behavior for both `ON DELETE` and `ON UPDATE` .
- `NO ACTION` : Synonym for `RESTRICT` .
- `CASCADE` : The change is allowed and propagates on the child table. For example, if a parent row is deleted, the child row is also deleted; if a parent row's ID changes, the child row's ID will also change.
- `SET NULL` : The change is allowed, and the child row's foreign key columns are set to `NULL` .
- `SET DEFAULT` : Only worked with PBXT. Similar to `SET NULL` , but the foreign key columns were set to their default values. If default values do not exist, an error is produced.

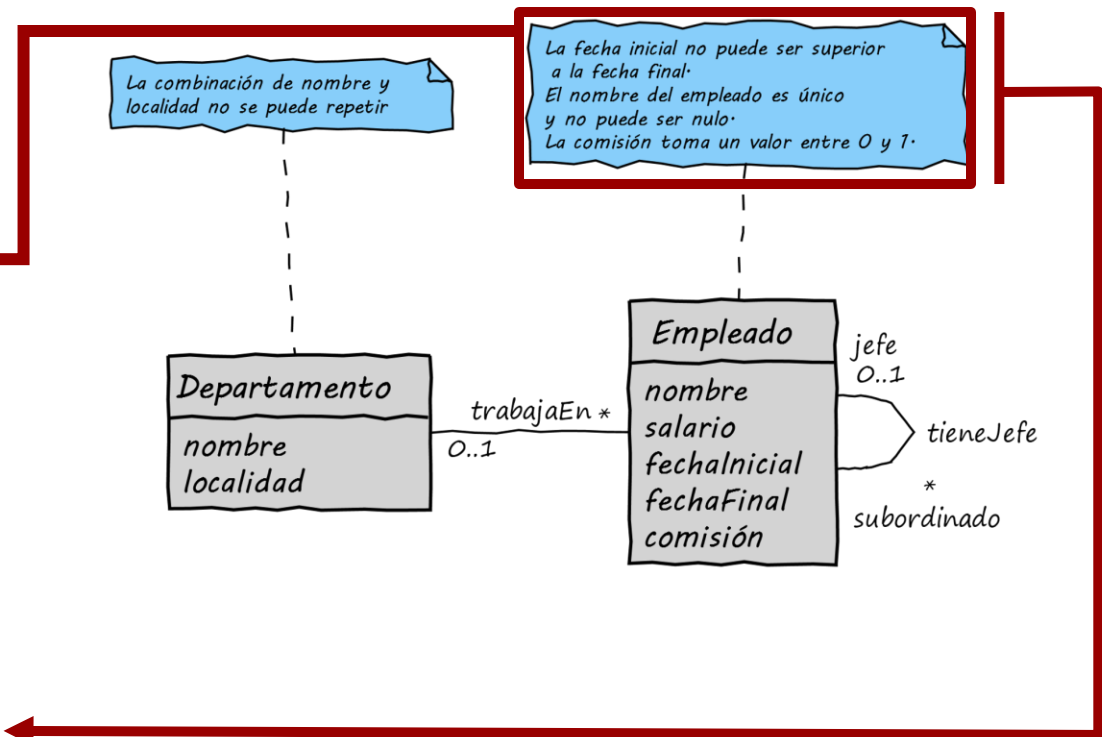
<https://mariadb.com/kb/en/library/foreign-keys/>

Reglas de negocio

- **atributo Tipo DEFAULT(valor)**
Define cuál será el *valor* que se le asigne al *atributo* cuando no se especifique ninguno.
- **CHECK** (expresión_lógica)
Define una restricción que deben cumplir los valores de uno o varios atributos.
- **CONSTRAINT nombre CHECK** (expresión_lógica)
Similar al anterior, dando un nombre a la restricción que aparecerá en caso de error.

Reglas de negocio

```
CREATE TABLE Empleados (  
    empleadoId INT NOT NULL AUTO_INCREMENT,  
    departamentoId INT,  
    jefe INT,  
    nombre VARCHAR(64) NOT NULL,  
    salario DECIMAL(6,2) DEFAULT 2000.00,  
    fechaInicial DATE,  
    fechaFinal DATE,  
    comision DOUBLE,  
    PRIMARY KEY(empleadoId),  
    FOREIGN KEY(departamentoId)  
        REFERENCES Departamentos(departamentoId)  
        ON DELETE SET NULL,  
    FOREIGN KEY(jefe)  
        REFERENCES Empleados(empleadoId),  
    UNIQUE(nombre),  
    CHECK (comision >=0 AND comision <=1),  
    CONSTRAINT fecha CHECK (fechaInicial < fechaFinal)  
);
```



Tipos de datos

Numéricos:

- ♦ TINYINT, **BOOLEAN**, SMALLINT, MEDIUMINT, **INT**, BIGINT, **DECIMAL**, FLOAT, DOUBLE, BIT...

Cadenas:

- ♦ **CHAR**, **VARCHAR**, TINYTEXT, **TEXT**, MEDIUMTEXT, LONGTEXT, **ENUM**...

Binarios:

- ♦ BINARY, VARBINARY, TINYBLOB, **BLOB**, MEDIUMBLOB, LONGBLOB...

Fechas:

- ♦ **DATE**, **TIME**, **DATETIME**, **YEAR**...

Geometrías:

- ♦ POINT, LINESTRING, POLYGON, MULTIPOINT...

<https://mariadb.com/kb/en/library/data-types/>

Inserción de datos

INSERT INTO NombreTabla (nombreAtributo1, nombreAtributo2, ... nombreAtributoN)
VALUES (valorAtributo1, valorAtributo2, ..., valorAtributoN);

Ejemplo:

```
INSERT INTO Departamentos (nombreDep, localidad)
VALUES ('Informática', 'Sevilla');
```

Actualización de datos

UPDATE NombreTabla SET nombreAtributo= valorAtributo
WHERE condición;

Ejemplo:

```
UPDATE Empleados SET salario='2500.00' WHERE empleadoId=1;
```

Borrado de datos

DELETE FROM NombreTabla
WHERE condición;

Ejemplo:

```
DELETE FROM Departamentos WHERE departamentoId=1;
```

* No te olvides de poner el Where en el Delete From

Lenguaje de consulta de datos (DQL)

```
SELECT < lista de columnas >  
FROM   < T1, T2,.. ,Tn >  
WHERE  < condición >
```

$\Pi_{columnas}$

```
SELECT nombre, salario  
FROM Empleados  
WHERE salario < 2000;
```

$\Pi_{nombre,salario}(\sigma_{salario < 2000}(Empleados))$

```
SELECT *  
FROM Empleados  
WHERE salario < 2000;
```

$\sigma_{salario < 2000}(Empleados)$

emple...	depart...	jefe	nombre	salario	fechaInicial	fechaFinal	comision
1	1	(NULL)	Pedro	2.300,00	2017-09-15	(NULL)	0,2
2	1	(NULL)	José	2.500,00	2018-08-15	(NULL)	0,5
3	2	(NULL)	Lola	2.300,00	2018-08-15	(NULL)	0,3
4	1	1	Luis	1.300,00	2018-08-15	2018-11-15	0
5	1	1	Ana	1.300,00	2018-08-15	2018-11-15	0

nombre	salario
Luis	1.300,00
Ana	1.300,00

Lenguaje de consulta de datos (DQL)

```
SELECT
  [ALL | DISTINCT | DISTINCTROW]
  [HIGH_PRIORITY]
  [STRAIGHT_JOIN]
  [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
  [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
  select_expr [, select_expr ...]
  [ FROM table_references
    [WHERE where_condition]
    [GROUP BY {col_name | expr | position} [ASC | DESC], ... [WITH ROLLUP]]
    [HAVING where_condition]
    [ORDER BY {col_name | expr | position} [ASC | DESC], ...]
    [LIMIT {[offset,] row_count | row_count OFFSET offset}]
    procedure|[PROCEDURE procedure_name(argument_list)]
    [INTO OUTFILE 'file_name' [CHARACTER SET charset_name] [export_options]

  INTO DUMPFILE 'file_name' INTO var_name [, var_name] ]

  [[FOR UPDATE | LOCK IN SHARE MODE] [WAIT n | NOWAIT] ] ]

export_options:
  [{FIELDS | COLUMNS}
    [TERMINATED BY 'string']
    [[OPTIONALLY] ENCLOSED BY 'char']
    [ESCAPED BY 'char']
  ]
  [LINES
    [STARTING BY 'string']
    [TERMINATED BY 'string']
  ]
]
```

- Los operadores del Álgebra Relacional siempre devuelven relaciones, es decir, **sin tuplas repetidas**.
- No ocurre lo mismo en SQL:

```
SELECT ALL fechaInicial, fechaFinal  
FROM Empleados;
```

fechaInicial	fechaFinal
2017-09-15	(NULL)
2018-08-15	2019-08-15
2018-08-15	(NULL)
2018-08-15	2018-11-15
2018-08-15	2018-11-15

```
SELECT DISTINCT fechaInicial,  
fechaFinal  
FROM Empleados;
```

fechaInicial	fechaFinal
2017-09-15	(NULL)
2018-08-15	2019-08-15
2018-08-15	(NULL)
2018-08-15	2018-11-15

(*) SELECT ALL equivale a SELECT en SQL.

La cláusula **WHERE** puede estar formada por:

- Una combinación de AND, OR y NOT
- Operador EXISTS
- Operador IN
- Operadores ALL, ANY o SOME
- Operadores BETWEEN, UNIQUE, TOP, IS NULL, LIKE

Cláusula BETWEEN:

```
SELECT DISTINCT nombre, salario
FROM Empleados
WHERE salario >=2000 AND salario <=3000;
```

```
SELECT DISTINCT nombre, salario
FROM Empleados
WHERE salario BETWEEN 2000 AND 3000;
```

Cláusula IN:

```
SELECT DISTINCT nombre, salario
FROM Empleados
WHERE salario IN (1000,2500,3000);
```

Cláusula **LIKE**:

- Compara cadenas de caracteres. Permite utilizar **expresiones regulares**.
- Una **expresión regular** es una cadena con caracteres especiales que define un patrón a buscar en la cadena dada.
- Los caracteres especiales en SQL para definir expresiones regulares son:
 - representa un único carácter
 - % representa cualquier cadena de caracteres

```
/* Empleados con una 'o' en la segunda posición de su nombre  
o que son jefes */  
SELECT *  
FROM Empleados  
WHERE nombre LIKE '_o%' OR jefe IS NULL;
```

Cláusula **ORDER BY**:

```
SELECT *  
FROM Empleados  
ORDER BY departamentoId, nombre;
```

- Por defecto ordena de forma ascendente, el orden se especifica con las palabras reservadas **ASC** y **DESC** junto a cada atributo.
- En Álgebra Relacional no existe equivalente por manejar conjuntos.

Producto cartesiano:

```
SELECT *  
FROM Empleados, Departamentos;
```

emple...	depar...	jefe	nom...	salario	fechaInicial	fechaFinal	co...	depa...	nombreDep	locali...
5	1	1	Ana	1.300,00	2018-08-15	2018-11-15	0	3	Arte	Cádiz
5	1	1	Ana	1.300,00	2018-08-15	2018-11-15	0	1	Historia	(NULL)
5	1	1	Ana	1.300,00	2018-08-15	2018-11-15	0	2	Informática	Sevilla
2	1	(NULL)	José	2.500,00	2018-08-15	2019-08-15	0,5	3	Arte	Cádiz
2	1	(NULL)	José	2.500,00	2018-08-15	2019-08-15	0,5	1	Historia	(NULL)
2	1	(NULL)	José	2.500,00	2018-08-15	2019-08-15	0,5	2	Informática	Sevilla
3	2	(NULL)	Lola	2.300,00	2018-08-15	(NULL)	0,3	3	Arte	Cádiz
3	2	(NULL)	Lola	2.300,00	2018-08-15	(NULL)	0,3	1	Historia	(NULL)
3	2	(NULL)	Lola	2.300,00	2018-08-15	(NULL)	0,3	2	Informática	Sevilla
4	1	1	Luis	1.300,00	2018-08-15	2018-11-15	0	3	Arte	Cádiz
4	1	1	Luis	1.300,00	2018-08-15	2018-11-15	0	1	Historia	(NULL)
4	1	1	Luis	1.300,00	2018-08-15	2018-11-15	0	2	Informática	Sevilla
1	1	(NULL)	Pedro	2.500,00	2017-09-15	(NULL)	0,2	3	Arte	Cádiz
1	1	(NULL)	Pedro	2.500,00	2017-09-15	(NULL)	0,2	1	Historia	(NULL)
1	1	(NULL)	Pedro	2.500,00	2017-09-15	(NULL)	0,2	2	Informática	Sevilla

Empleados × Departamentos

Natural Join:

```
SELECT nombre, salario, fechaInicial, nombreDep  
FROM Empleados NATURAL JOIN Departamentos;
```

```
SELECT nombre, salario, fechaInicial, nombreDep  
FROM Empleados E, Departamentos D  
WHERE E.departamentoId=D.departamentoId;
```

 nombre	salario	fechaInicial	 nombreDep
Pedro	2.500,00	2017-09-15	Historia
José	2.500,00	2018-08-15	Historia
Lola	2.300,00	2018-08-15	Informática
Luis	1.300,00	2018-08-15	Historia
Ana	1.300,00	2018-08-15	Historia

Resultado \leftarrow *Empleados* \bowtie *Departamentos*

Left/Right Join

Devuelve todas las filas de la tabla izquierda/derecha y todas las filas que cumplen de la tabla de la derecha/izquierda que cumplen la condición.

```
UPDATE Empleados SET departamentoId=NULL WHERE empleadoId=5;

SELECT nombre, salario, fechaInicial, nombreDep
FROM Empleados E
LEFT/RIGHT JOIN Departamentos D
ON E.departamentoId=D.departamentoId;
```

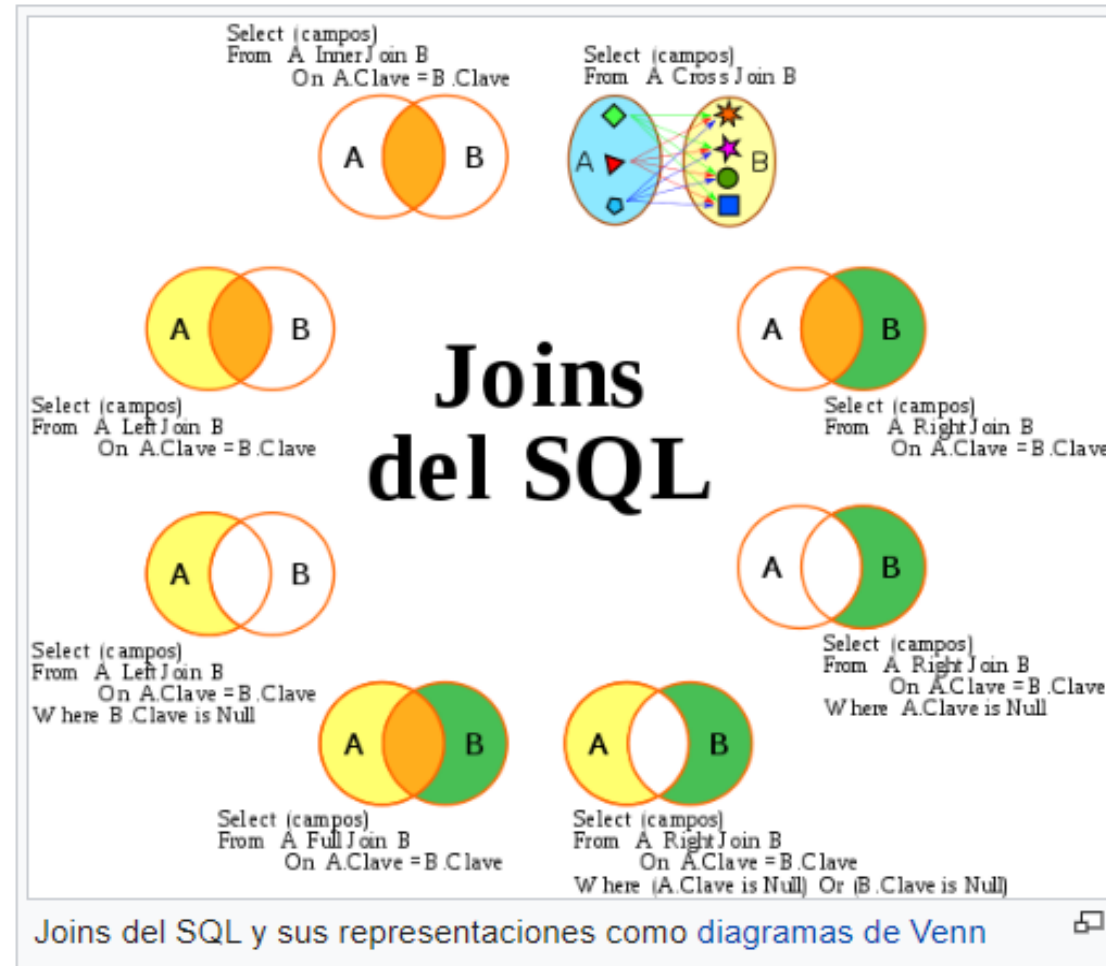
Left Join

 nombre	salario	fechaInicial	 nombreDep
Pedro	2.500,00	2017-09-15	Historia
José	2.500,00	2018-08-15	Historia
Lola	2.300,00	2018-08-15	Informática
Luis	1.300,00	2018-08-15	Historia
Ana	1.300,00	2018-08-15	(NULL)

Right Join

 nombre	salario	fechaInicial	 nombreDep
Pedro	2.500,00	2017-09-15	Historia
José	2.500,00	2018-08-15	Historia
Lola	2.300,00	2018-08-15	Informática
Luis	1.300,00	2018-08-15	Historia
(NULL)	(NULL)	(NULL)	Arte

La familia de los Join



Unión e intersección

```
SELECT *  
FROM Empleados E  
WHERE E.salario < 1000  
UNION  
SELECT *  
FROM Empleados E  
WHERE E.salario > 2000;
```

$R \cup S$



```
SELECT *  
FROM Empleados E  
WHERE E.salario < 1000  
INTERSECT  
SELECT *  
FROM Empleados E  
WHERE E.salario > 2000;
```

$R \cap S$

Cláusula Exists:

```
/* Departamentos sin Empleados */  
SELECT *  
FROM Departamentos D  
WHERE NOT EXISTS (  
    SELECT * FROM Empleados E  
    WHERE D.departamentoId=E.departamentoId  
);  
  
/* Departamentos con Empleados */  
SELECT *  
FROM Departamentos D  
WHERE EXISTS (  
    SELECT * FROM Empleados E  
    WHERE D.departamentoId=E.departamentoId  
);
```

 departamentoId	 nombreDep	localidad
3	Arte	Cádiz

 departamentoId	 nombreDep	localidad
1	Historia	(NULL)
2	Informática	Sevilla

Funciones de agregación

- **COUNT** devuelve el número de filas o valores especificados en una consulta.
- **SUM, MAX, MIN, AVG** se aplican a un conjunto o multiconjunto de valores numéricos y devuelven, respectivamente, la suma, el valor máximo, el mínimo y el promedio de dichos valores.
- Estas funciones se pueden usar con la cláusula **SELECT**.
- Ejemplo:

```
/* Estadísticas salarios de los empleados */  
SELECT COUNT(*), MIN(salario), MAX(salario), AVG(salario), SUM(salario)  
FROM Empleados;
```

COUNT(*)	MIN(salario)	MAX(salario)	AVG(salario)	SUM(salario)
5	1.300,00	2.500,00	1.980,000000	9.900,00

$\gamma^{count(*),min(salario),max(salario),avg(salario),sum(salario)}(Empleados)$

Cláusula GROUP BY

- Agrupar las tuplas que tienen el mismo valor para ciertos atributos.
- Permite aplicar las funciones de agregación (sum, max, min, avg, count, etc.) a cada uno de dichos grupos.
- Los atributos de agrupación pueden aparecer en la cláusula **SELECT**.
- Es el equivalente al operador de Álgebra Relacional: $\gamma_G^F(R)$
- Ejemplo:

```
SELECT departamentoId,  
       COUNT(*) ,  
       AVG(salario) salarioMedio,  
       AVG(salario * (1+comision))  
       salarioConComision,  
       SUM(salario) gastoSalarios  
FROM Empleados  
GROUP BY departamentoId;
```

$\gamma_{departamentoId}^{departamentoId, count(*), avg(salario) \dots} (Empleados)$

Cláusula **HAVING**

- Especifica una condición sobre el grupo de tuplas asociado a cada valor de los atributos de agrupación (clases de equivalencia).
- **Sólo los grupos que cumplan la condición entrarán en el resultado de la consulta.**
- **Primero se filtran las filas mediante **WHERE**, luego se agrupan, y luego se filtran los grupos mediante **HAVING**.**

```
SELECT departamentoId,  
       COUNT(*),  
       AVG(salario) salarioMedio,  
       AVG(salario * (1+comision))  
       salarioConComision,  
       SUM(salario) gastoSalarios  
FROM Empleados  
GROUP BY departamentoId HAVING  
COUNT(*)>1;
```

```
SELECT * FROM (  
    SELECT departamentoId,  
    COUNT(*) numEmpleados,  
    AVG(salario) salarioMedio,  
    AVG(salario * (1+comision))  
    salarioConComision,  
    SUM(salario) gastoSalarios  
    FROM Empleados  
    GROUP BY departamentoId  
    ) Estadistica  
WHERE numEmpleados>1;
```

$$\sigma_{count(*)>1}(\gamma \mid \mid departamentoId^{departamentoid, count(*), avg(salario) \dots (Empleados)})$$

Cláusulas **ALL** y **ANY**

- Permite comparar un valor individual v (nombre de atributo) con un conjunto de valores V (consulta anidada).
- Por ejemplo:

```
SELECT * FROM Empleados
WHERE salario >
ALL (SELECT AVG(salario)
      FROM Empleados
      GROUP BY departamentoId);
```

```
SELECT * FROM Empleados
WHERE salario >
ANY (SELECT AVG(salario)
      FROM Empleados
      GROUP BY departamentoId);
```

Cláusulas ALL y ANY

```
/* Departamento con más empleados */  
/* Opción 1 */  
SELECT departamentoId FROM Empleados  
GROUP BY departamentoId HAVING COUNT(*) >= ALL  
    ( SELECT COUNT(*)  
      FROM Empleados  
      GROUP BY departamentoId );  
  
/* Opción 2 */  
SELECT departamentoId FROM Empleados  
GROUP BY departamentoId HAVING COUNT(*) =  
    ( SELECT MAX(total) FROM  
      ( SELECT COUNT(*) AS total  
        FROM Empleados  
        GROUP BY departamentoId  
      ) NumEmpleados );
```

Vistas

- Son tablas virtuales creadas en base al resultado de una consulta.
- Pueden optimizar el espacio de almacenamiento y el tiempo de CPU.
- Una vez creadas, se utilizan de forma análoga a una tabla.

```
/* Vista con las estadísticas de los Empleados por Departamento */  
CREATE OR REPLACE VIEW EstadísticasEmpleados AS  
SELECT departamentoId,  
        COUNT(*) AS numEmpleados,  
        AVG(salario) salarioMedio,  
        SUM(salario) gastoSalarios  
FROM Empleados  
GROUP BY departamentoId;  
  
/* Número de empleados que tiene el departamento con más empleados */  
SELECT MAX(numEmpleados)  
FROM EstadísticasEmpleados;
```

Inyección SQL

¡Ojo con incorporar datos suministrados por el usuario directamente en una sentencia SQL!

```
query = "INSERT INTO Multas  
VALUES (" + matricula + ")"
```

Sanitize



Tema 10: Introducción a SQL

Introducción a la Ingeniería del Software y los Sistemas de Información I
Ingeniería Informática – Tecnologías Informáticas
Departamento de Lenguajes y Sistemas Informáticos

