T10 - Introducción a SQL

Departamento de Lenguajes y Sistemas Informáticos
Universidad de Sevilla

Objetivos


Departamento
Lenguajes
Valor
Salario
Complejas
Contenido
Consulta
Datos
Select
Definición
Introducción
Claves
Mariadb
Consultas
Lenguaje
Nombre
Insert
Valores
Manipulación

Contenido

Contenido

Introducción a SQL

- SQL (Structured Query Language) es el lenguaje **estándar** para definir, manipular y consultar bases de datos relacionales.
- Puede ser utilizado en lenguajes de programación de propósito general como Java, C, Python o bien en lenguajes específicos del fabricante (p.ej. PL/SQL en Oracle, Transact SQL en MS SQLServer).
- 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

Introducción a SQL

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/r96oytechsacgzi4.pdf

Introducción a SQL

Año	Nombre	Alias habitual	Comentarios
1986	SQL-86	SQL-87*	Primera estandarización ANSI (X3.135-1986).
1989	SQL-89	SQL1	Revisión menor de SQL-86.
1992	SQL-92	SQL2	Gran revisión: base del SQL relacional clásico moderno.
1999	SQL:1999	SQL3	Añade PSM (procedimientos), triggers, recursividad (`WITH RECURSIVE`), tipos/objetos, etc.
2003	SQL:2003	—	SQL/XML, secuencias y columnas identidad (autonuméricas), mejoras analíticas.
2006	SQL:2006	—	Consolidación y ampliación de SQL/XML.
2008	SQL:2008	—	Refinamientos y extensiones de lenguaje (sin ruptura grande).
2011	SQL:2011	—	Soporte temporal (periodos de aplicación y de sistema).
2016	SQL:2016	—	JSON en el estándar, row pattern recognition (`MATCH_RECOGNIZE`) y más analítica.
2023	SQL:2023	—	Revisión vigente de la familia ISO/IEC 9075 (actualiza múltiples partes).

> Fuente: https://www.iso.org/standard/76584.html

Ejemplo: Modelo Conceptual y Relacional

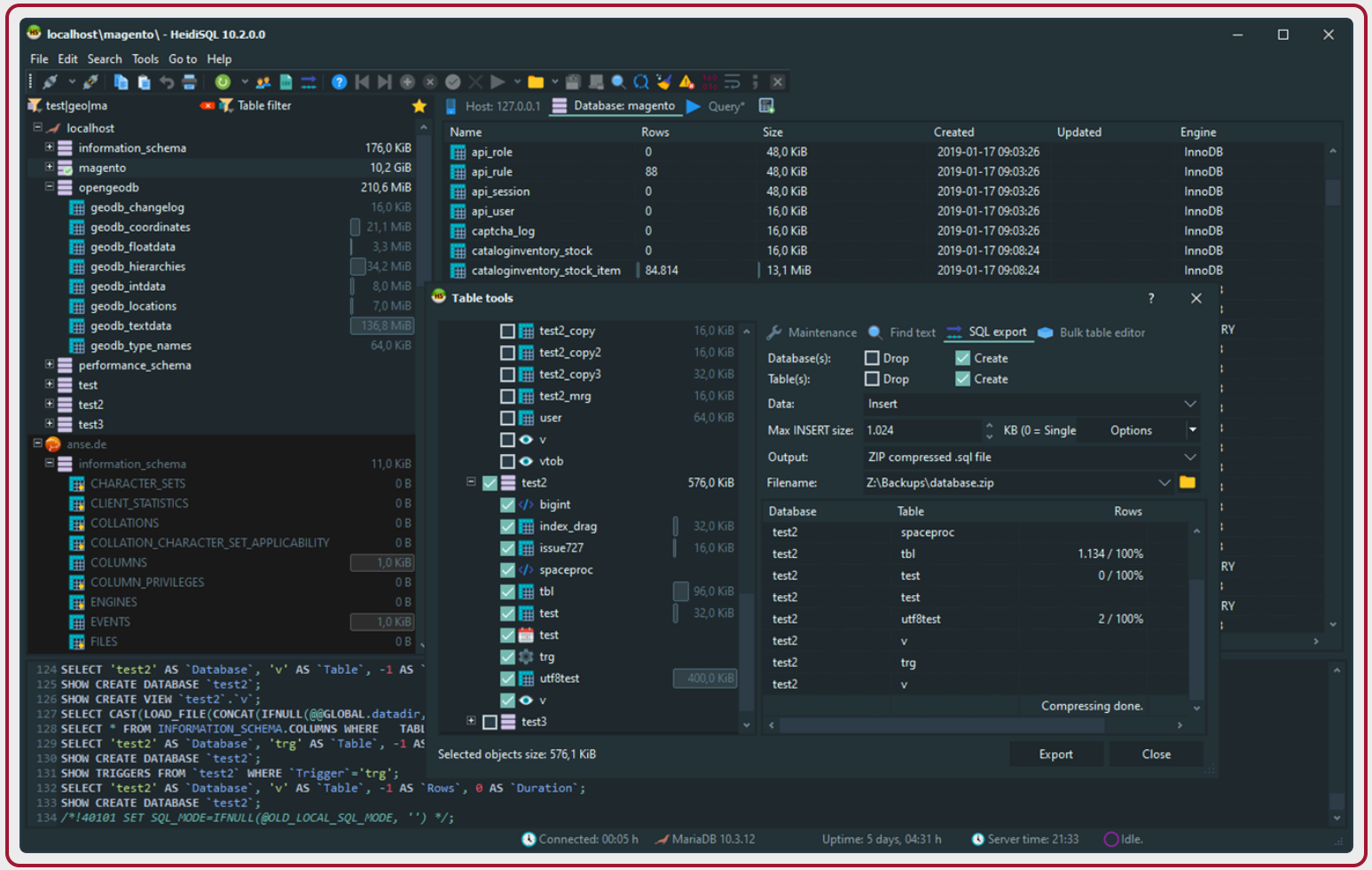

```
-- Intensión relacional
Departamentos = {departamentoId, nombreDep, localidad}
  PK(departamentoId)
  AK(nombreDep, localidad)

Empleados = {empleadoId, departamentoId, jefeId,
   nombre, salario, fechaInicial, fechaFinal, comision}
  PK(empleadoId)
  FK(departamentoId)/Departamentos
  FK(jefeId)/Empleados
```

Transformación del MR a SQL

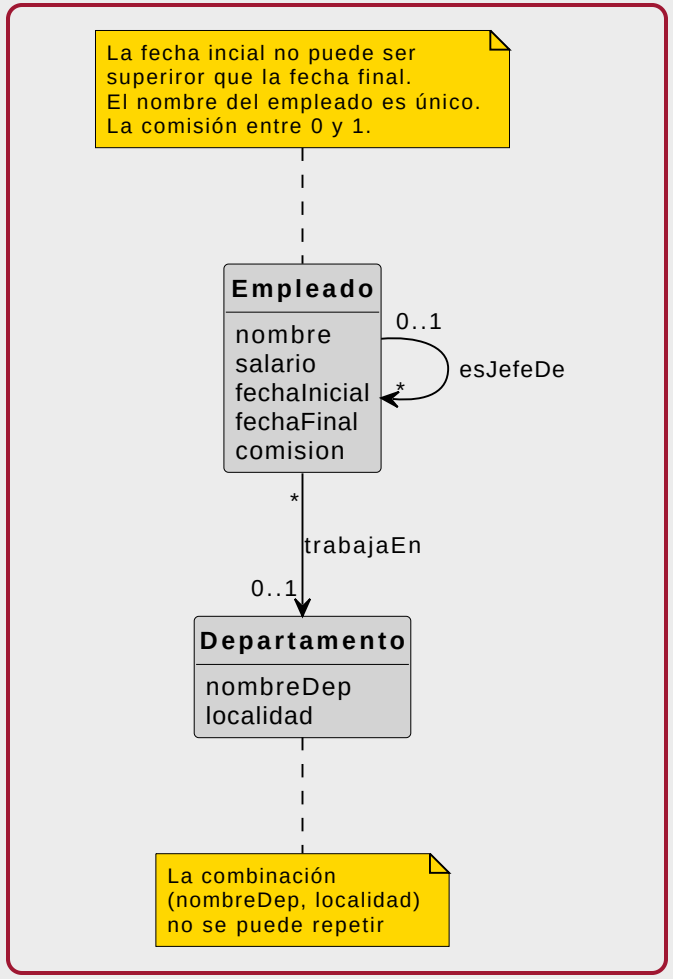
- Se creará una tabla por cada relación del modelo relacional.
- Será necesario la definición de claves primarias y ajenas.
- El resto de restricciones se definen mediante:
  - CHECK <condición>. Limita los valores a insertar.
  - UNIQUE. Para asegurar que no se repiten valores. (AK)
  - NOT NULL. No admite nulos.
  - En caso de restricciones que no se puedan definir mediante CHECK:
    - STORED PROCEDURE + TRIGGERS.

Contenido

HeidiSQL


localhost\magento\ - HeidiSQL 10.2.0.0
File Edit Search Tools Go to Help
Host: 127.0.0.1
Database: magento
Query*
Table tools
Maintenance
Find text
SQL export
Bulk table editor
Selected objects size: 576,1 KiB
Export
Close
Compressing done.
Connected: 00:05 h
MariaDB 10.3.12
Uptime: 5 days, 04:31 h
Server time: 21:33
Idle.

DDL – CREATE TABLE



Modelo relacional:

```
Departamentos = { departamentoId, nombreDep, localidad }
    PK(departamentoId)
    AK(nombreDep, localidad)
```

Modelo tecnológico (SQL):

```sql
CREATE TABLE departments (
  department_id INT NOT NULL AUTO_INCREMENT,
  department_name VARCHAR(32),
  city VARCHAR(64),
  PRIMARY KEY (department_id),
  UNIQUE (department_name, city)
);
```

DDL – CREATE TABLE


```
Empleados = { empleadoId, departamentoId, jefeId,
   nombre, salario, fechaInicial, fechaFinal, comision }
    PK(empleadoId)
    FK(departamentoId) / Departamentos
    FK(jefeId) / Empleados
    AK(nombre)
```

```
CREATE TABLE employees (
  employee_id INT NOT NULL AUTO_INCREMENT,
  department_id INT,
  manager_id INT,
  employee_name VARCHAR(64) NOT NULL,
  salary DECIMAL(6,2) DEFAULT 2000.00,
  start_date DATE,
  end_date DATE,
  commission DOUBLE,
  PRIMARY KEY (employee_id),
  FOREIGN KEY (department_id) REFERENCES departments(department_id)
    ON DELETE SET NULL,
  FOREIGN KEY (manager_id) REFERENCES employees(employee_id),
  UNIQUE (employee_name),
  CHECK (commission >= 0 AND commission <= 1),
  CONSTRAINT ck_employee_dates CHECK (start_date < end_date)
);
```

DDL - 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: ...
```

DDL – 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/

DDL – Reglas de negocio

`atributo Tipo DEFAULT(valor)` : Define cuál será el 'valor' que se le asigne al 'atributo' cuando no se especifique ningún valor.

`CHECK (expr)` : Define una restricción que deben cumplir los valores de uno o varios atributos.

`CONSTRAINT nombre CHECK (expr)` : Similar al anterior, dando un nombre a la restricción que aparecerá en caso de error.

DDL - Tipos de datos

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

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…

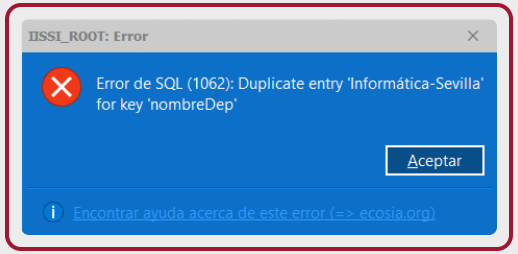
Fechas: DATE, TIME, DATETIME, YEAR…

Geometrías: POINT, LINESTRING, POLYGON, MULTIPOINT…

Contenido

DML – INSERT INTO

```sql
-- Tres valores correctos
INSERT INTO Departamentos (nombreDep, localidad) VALUES ('Historia', NULL);
INSERT INTO Departamentos (nombreDep, localidad) VALUES ('Informática', 'Sevilla');
INSERT INTO Departamentos (nombreDep, localidad) VALUES ('Arte', 'Cádiz');
-- Valores repetidos
-- Esta tupla se inserta dos veces !!
INSERT INTO Departamentos (nombreDep, localidad) VALUES ('Historia', NULL);
-- Esta no
INSERT INTO Departamentos (nombreDep, localidad) VALUES ('Informática', 'Sevilla');
```



departamentoId	nombreDep	localidad
1	Historia	(NULL)
2	Informática	Sevilla
3	Arte	Cádiz
4	Historia	(NULL)

DML – INSERT INTO

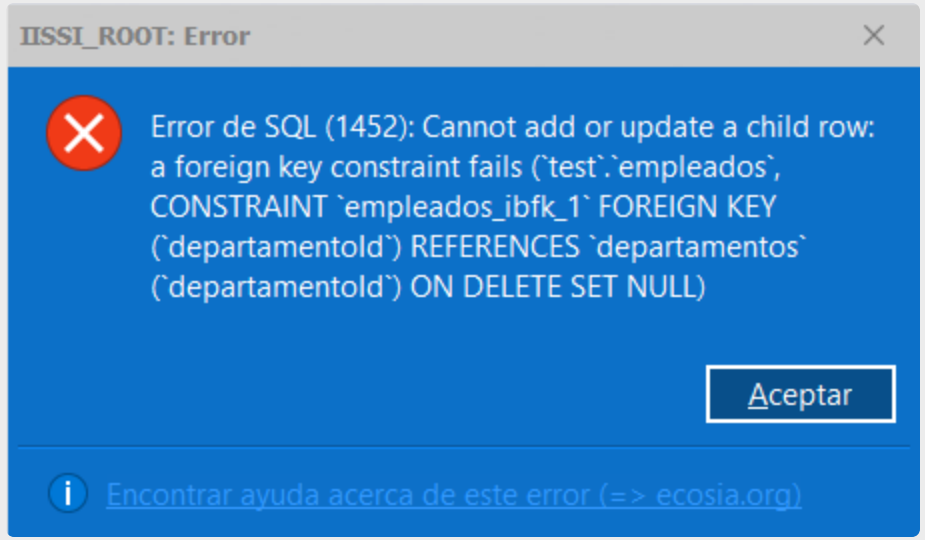
```sql
-- Pedro de Historia
INSERT INTO Empleados(departamentoId, jefe, nombre, salario, fechaInicial, fechaFinal, comision)
   VALUES (1, NULL, 'Pedro', 2300.00, '2017-09-15', NULL, 0.2);
-- José de Historia
INSERT INTO Empleados(departamentoId, jefe, nombre, salario, fechaInicial, fechaFinal, comision)
   VALUES (1, NULL, 'José', 2500.00, '2018-08-15', NULL, 0.5);
-- Lola de Informática
INSERT INTO Empleados(departamentoId, jefe, nombre, salario, fechaInicial, fechaFinal, comision)
   VALUES (2, NULL, 'Lola', 2300.00, '2018-08-15', NULL, 0.3);
-- Luis trabajó para Pedro 3 meses
INSERT INTO Empleados(departamentoId, jefe, nombre, salario, fechaInicial, fechaFinal, comision)
   VALUES (1, 1, 'Luis', 1300.00, '2018-08-15', '2018-11-15', 0);
-- Ana trabajó para Pedro 3 meses
INSERT INTO Empleados(departamentoId, jefe, nombre, salario, fechaInicial, fechaFinal, comision)
   VALUES (1, 1, 'Ana', 1300.00, '2018-08-15', '2018-11-15', 0);
```

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

DML – INSERT INTO

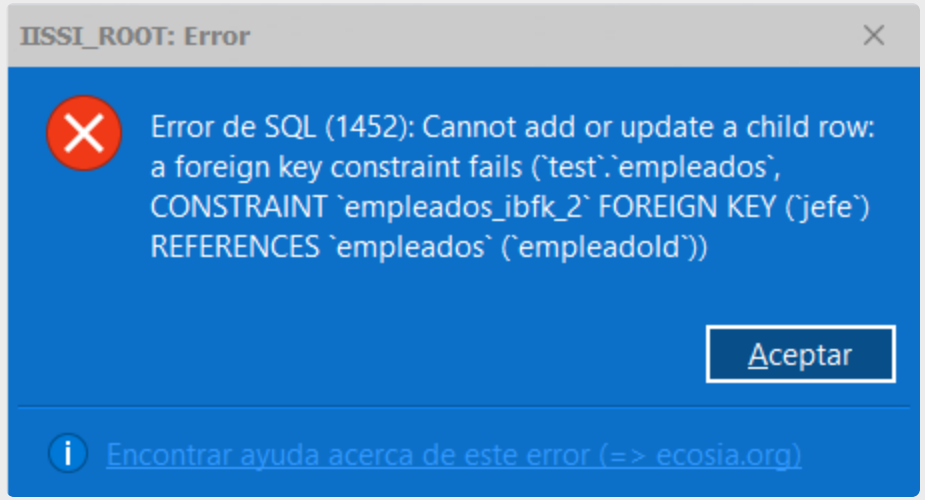
```sql
-- Manuel de Electrónica
INSERT INTO Empleados(departamentoId, jefe, nombre,
   salario, fechaInicial, fechaFinal, comision)
VALUES (5, null, 'Manuel',
   2300.00, '2017-08-15', NULL, 0.6);
```

El departamendo 5 no existe




```sql
-- Francisco trabajo para Yolanda
INSERT INTO Empleados(departamentoId, jefe, nombre,
   salario, fechaInicial, fechaFinal, comision)
VALUES (1, 23, 'Francisco',
   1300.00, '2017-08-15', '2017-11-15', 0);
```

El empleado 23 no existe



DML - UPDATE

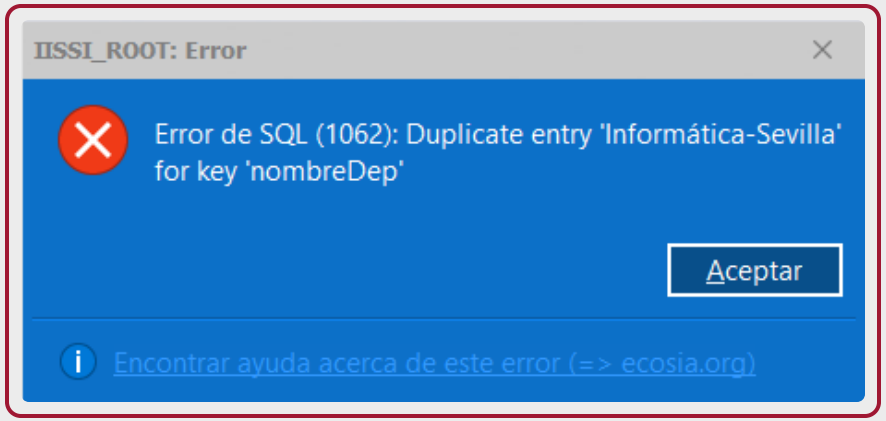
```sql
-- Subir el sueldo de Pedro
UPDATE Empleados SET salario='2500.00' WHERE  empleadoId=1;
-- Despedir a José
UPDATE Empleados SET fechaFinal='2019-08-15' WHERE empleadoId=2;
```

empleadoId	departamentoId	jefe	nombre	salario	fechaInicial	fechaFinal	comision
1	1	(NULL)	Pedro	2.500,00	2017-09-15	(NULL)	0,2
2	1	(NULL)	José	2.500,00	2018-08-15	2019-08-15	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

DML - UPDATE

departamentoId	nombreDep	localidad
1	Historia	(NULL)
2	Informática	Sevilla
3	Arte	Cádiz
4	Informática	Cádiz

```
UPDATE Departamentos SET localidad='Sevilla' WHERE  departamentoId=4;
```



DML - DELETE

departamentoId	nombreDep	localidad
1	Historia	(NULL)
2	Informática	Sevilla
3	Arte	Cádiz
4	Historia	(NULL)

```
CREATE TABLE Empleados ...
FOREIGN KEY(departamentoId) REFERENCES Departamentos(departamentoId)
   ON DELETE SET NULL,
FOREIGN KEY(jefe) REFERENCES Empleados(empleadoId)
...
-- Borrar departamento de Historia
DELETE FROM Departamentos WHERE  departamentoId=1;
```

departamentoId	nombreDep	localidad
2	Informática	Sevilla
3	Arte	Cádiz
4	Informática	Cádiz

empleadoId	departamentoId	jefe	nombre	salario	fechaInicial	fechaFinal	comision
1	(NULL)	(NULL)	Pedro	2.500,00	2017-09-15	(NULL)	0,2
2	(NULL)	(NULL)	José	2.500,00	2018-08-15	2019-08-15	0,5
3	2	(NULL)	Lola	2.300,00	2018-08-15	(NULL)	0,3
4	(NULL)	1	Luis	1.300,00	2018-08-15	2018-11-15	0
5	(NULL)	1	Ana	1.300,00	2018-08-15	2018-11-15	0

DML - DELETE

departamentoId	nombreDep	localidad
1	Historia	(NULL)
2	Informática	Sevilla
3	Arte	Cádiz
4	Historia	(NULL)

```
FOREIGN KEY(departamentoId) REFERENCES Departamentos(departamentoId)
   ON DELETE CASCADE,
FOREIGN KEY(jefe) REFERENCES Empleados(empleadoId)
   ON DELETE CASCADE,
...
-- Borrar departamento de Historia
DELETE FROM Departamentos WHERE  departamentoId=1;
```

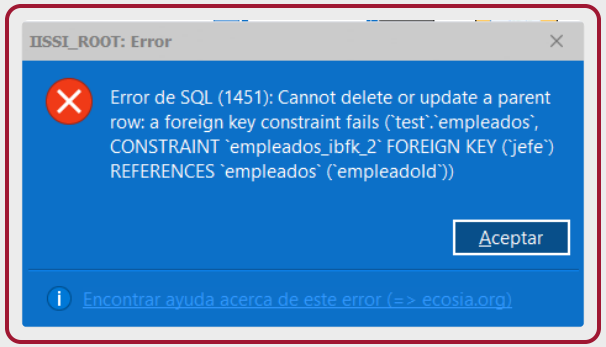
departamentoId	nombreDep	localidad
2	Informática	Sevilla
3	Arte	Cádiz
4	Informática	Cádiz

empleadoId	departamentoId	jefe	nombre	salario	fechaInicial	fechaFinal	comision
3	2	(NULL)	Lola	2.300,00	2018-08-15	(NULL)	0,3

DML - DELETE

```
FOREIGN KEY(departamentoId) REFERENCES Departamentos(departamentoId)
   ON DELETE CASCADE,
FOREIGN KEY(jefe) REFERENCES Empleados(empleadoId),
-- Borrar departamento de Historia
DELETE FROM Departamentos WHERE  departamentoId=1;
```

departamentoId	nombreDep	localidad
2	Informática	Sevilla
3	Arte	Cádiz
4	Informática	Cádiz



Inyecciones SQL

¡Ojo con insertar texto suministrado por el usuario directamente en una sentencia SQL!

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

Contenido

DQL – SELECT

$$\prod_{columnas} \left(\sigma_{condición} (T_1 \times T_2 \times \cdots \times T_n) \right)$$

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

$$\prod_{nombre,salario} \left(\sigma_{salario<2000} (Empleados) \right)$$

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

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

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

DQL – SELECT DISTINCT

```
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

DQL – SELECT

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`

DQL - SELECT

```
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']
    ]
```

> https://mariadb.com/kb/en/library/select/

DQL – SELECT (renombrado)

Para compactar las expresiones en AR podemos usar el operador de renombrado ρ, por ejemplo:

$$\rho_{E(eId,dId,jId,n,s,fi,ff,c)}(Empleados)$$

$$\rho_{D(dId,n,l)}(Departamentos)$$

Una vez renombradas las relaciones se puede hacer referencia al nombre de los empleados en lugar de $Empleados.nombre$ podemos escribir $E.n$.

DQL – SELECT (BETWEEN)

$$\prod_{n,s}\left(\sigma_{s\geq 2000 \wedge s\leq 3000}(E)\right)$$

```sql
-- Con DISTINCT nos aseguramos que no  haya duplicados en el resultado
SELECT DISTINCT nombre, salario
FROM Empleados
WHERE salario >=2000 AND salario <=3000;
-- equivalente
SELECT DISTINCT nombre, salario
FROM Empleados
WHERE salario BETWEEN 2000 AND 3000
;
```

nombre	salario
Pedro	2.500,00
José	2.500,00
Lola	2.300,00

DQL – SELECT (IN)

Permite comparar un valor individual v (un nombre de atributo) con un conjunto de valores V (generalmente una consulta anidada). Devuelve TRUE si v es uno de los elementos de V.

$$\prod_{n,s} \left(\sigma_{s=1000 \vee s=2000 \vee s=3000} (E) \right)$$

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

nombre	salario
Pedro	2.500,00
José	2.500,00

DQL – SELECT (LIKE)

Para comparar cadenas de caracteres se utiliza el operador de comparación LIKE.
Las cadenas parciales se especifican mediante los caracteres reservados % y _:

- % representa cualquier cadena de caracteres
- _ representa un único carácter

DQL – SELECT (LIKE)

```sql
-- 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;
```

empleadoId	departamentoId	jefe	nombre	salario	fechaInicial	fechaFinal	comision
1	1	(NULL)	Pedro	2.500,00	2017-09-15	(NULL)	0,2
2	1	(NULL)	José	2.500,00	2018-08-15	2019-08-15	0,5
3	2	(NULL)	Lola	2.300,00	2018-08-15	(NULL)	0,3

En AR no hay expresión equivalente

DQL – SELECT (ORDER BY)

```
-- Ordena por departamento descendente y a
-- igual departamento por nombre del empleado alfabético

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

empleadoId	departamentoId	jefe	nombre	salario	fechaInicial	fechaFinal	comision
5	1	1	Ana	1.300,00	2018-08-15	2018-11-15	0
2	1	(NULL)	José	2.500,00	2018-08-15	2019-08-15	0,5
4	1	1	Luis	1.300,00	2018-08-15	2018-11-15	0
1	1	(NULL)	Pedro	2.500,00	2017-09-15	(NULL)	0,2
3	2	(NULL)	Lola	2.300,00	2018-08-15	(NULL)	0,3

En AR no se puede definir orden, son conjuntos

DQL – SELECT (producto cartesiano)

El producto cartesiano devuelve una nueva relación con todas las posibles combinaciones entre las tuplas de las relaciones involucradas.

```
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

$$Empleados \times Departamentos$$

DQL – SELECT (NATURAL JOIN)

$$\prod_{E.n,E.s,E.fi,D.n} \left(\sigma_{D.did=E.dId} (E \times D) \right)$$

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

$$\prod_{E.n,E.s,E.fi,D.n} (E \bowtie D)$$

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

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

DQL – SELECT (LEFT/RIGHT JOIN)

`T1 RIGHT/LEFT JOINT T2 ON <cond>` devuelve todas las filas de la tabla derecha/izquierda, y todas las filas que cumplen de la tabla de la izquierda/derecha 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

En AR no vamos a usar estos operadores

DQL – SELECT (UNION)

```
SELECT *
FROM Empleados E
   LEFT JOIN Departamentos D
   ON E.departamentoId=D.departamentoId
UNION
SELECT *
FROM Empleados E
  RIGHT JOIN Departamentos D
  ON E.departamentoId=D.departamentoId;
```

empleadoId	departamentoId	jefe	nombre	salario	fechaInicial	fechaFinal	comision	departamentoId	nombreDep	localidad
1	1	(NULL)	Pedro	2.500,00	2017-09-15	(NULL)	0,2	1	Historia	(NULL)
2	1	(NULL)	José	2.500,00	2018-08-15	2019-08-15	0,5	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	1	Historia	(NULL)
5	(NULL)	1	Ana	1.300,00	2018-08-15	2018-11-15	0	(NULL)	(NULL)	(NULL)
(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	3	Arte	Cádiz

El conjunto de tuplas resultado contiene el “join” de todos empleados (aunque no estén en un departamento) con todos los departamentos (aunque no tengan ningún empleado)

DQL – SELECT (EXISTS)

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

departamentoId	nombreDep	localidad
3	Arte	Cádiz

```sql
-- Departamentos con Empleados
SELECT *
FROM Departamentos D
WHERE EXISTS (
SELECT * FROM Empleados E
WHERE D.departamentoId=E.departamentoId
);
```

departamentoId	nombreDep	localidad
1	Historia	(NULL)
2	Informática	Sevilla

$$D \setminus \prod_{D.dId, D.n, D.l} (D \bowtie E)$$

Contenido

Consultas complejas

Están enfocadas a los niveles táctico y estratégico de un sistema de información.
Presentan los datos agrupados a partir de los registros individuales que corresponden a las operaciones diarias

Consultas complejas (Agregados)

Funciones agregadas

- **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** o con la cláusula **HAVING**.

Consultas complejas (`GROUP BY`)

`GROUP BY`

- Agrupa las tuplas que tienen el mismo valor para ciertos atributos
- Permite aplicar las funciones de agregación a cada uno de esos grupos ($count$, sum, max, min, avg, ...).
- Los atributos de agrupacion pueden aparecer en la cláusula `SELECT`.
- Equivalente en AR:

$$_{agrupador}\Upsilon^{agrupador, funciones}$$

Consultas complejas (Agregados)

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

$$\Upsilon_{count(*),min(s),avg(s),sum(s)}(E)$$

Esta consulta al no agrupar por ningún atributo las funciones de agregación se aplican a toda la columna.

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

Consultas complejas (GROUP BY)

Calcular para cada departamento el número de empleados, el salario medio de los mismos, el salario con las comisiones, y el gasto total en salarios

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

$$\underset{E.dId}{\Upsilon_{E.dId,count(*),avg(s),avg(s*(1+c)),sum(s)}}(E)$$

departamentoId	COUNT(*)	salarioMedio	salarioConComision	gastoSalarios
(NULL)	1	1.300,000000	1.300	1.300,00
1	3	2.100,000000	2.683,3333333333335	6.300,00
2	1	2.300,000000	2.990	2.300,00

Consultas complejas (HAVING)

`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`

Consultas complejas (HAVING)

Calcular para cada departamento con más de un empleado, el salario medio de los mismos, el salario con las comisiones, y el gasto total en salarios

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

departamentoId	COUNT(*)	salarioMedio	salarioConComision	gastoSalarios
1	3	2.100,000000	2.683,3333333333335	6.300,00

Consultas complejas (HAVING)

```sql
-- Consulta equivalente sin HAVING
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;
```

$$\sigma_{n>1}\left(\begin{array}{c}E.dId,\rho_{n}\,count(*),avg(s),avg(s*(1+c)),sum(s)\\ \Upsilon_{E.dId}\end{array}(E)\right)$$

Consultas complejas (`ALL`, `ANY`)

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

Ejemplo: empleados con salario mayor que el salario medio en todos los departamentos.

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

$$\sigma_{s>m}\left(\rho_{m}{}_{(max(avg(s)))}\left(\Upsilon_{E.dId}{}^{avg(s)}(E)\right)\right)$$

empleadoId	departamentoId	jefe	nombre	salario	fechaInicial	fechaFinal	comision
1	1	(NULL)	Pedro	2.500,00	2017-09-15	(NULL)	0,2
2	1	(NULL)	José	2.500,00	2018-08-15	2019-08-15	0,5

Consultas complejas (ALL, ANY)

```sql
-- 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    );
```

$$numMax \leftarrow \overset{max(count(*))}{\Upsilon} \left(\overset{count(*)}{\underset{E.dId}{\Upsilon}} (E) \right)$$

$$\underset{n=numMax}{\sigma} \left(\overset{\rho \, count(*)}{\underset{n}{\underset{E.dId}{\Upsilon}}} (E) \right)$$

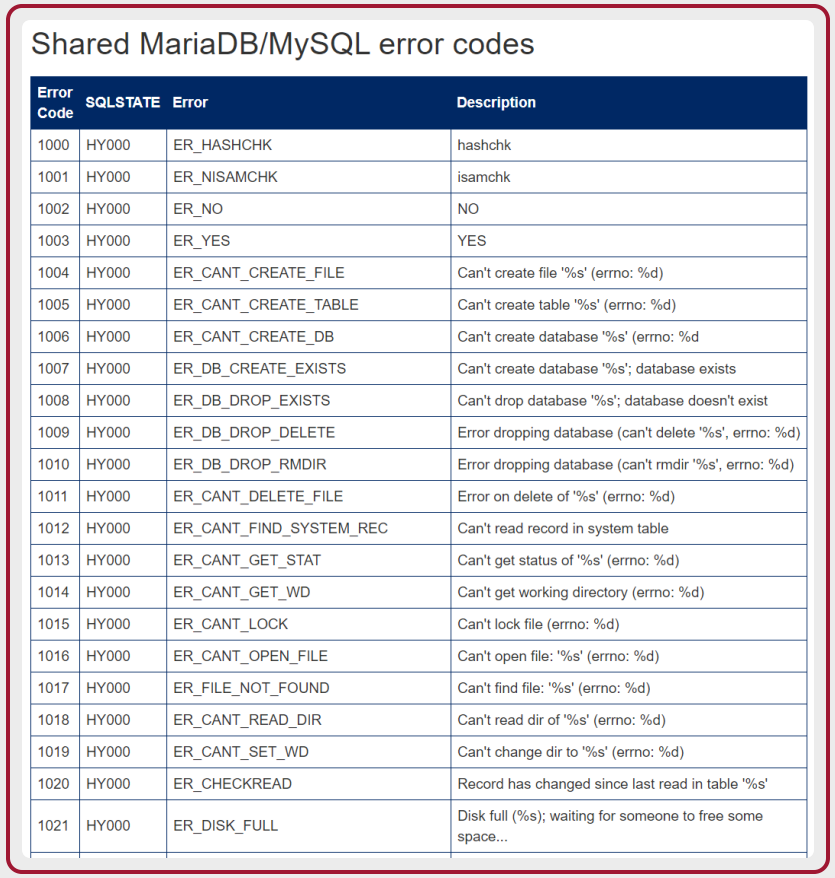
Consultas complejas. Views

`CREATE OR REPLACE VIEW [vista] AS [SELECT…]`

- Crea una nueva tabla con nombre “vista”, cuyo contenido es el de la consulta
- Mejora el tiempo de CPU
- Optimiza el espacio de almacenamiento

```
-- Vista con las estadísticas de los Empleados por Departamento
CREATE OR REPLACE VIEW EstadísticasEmpleados AS
SELECT departamentoId,
COUNT(*) AS numEmpleados,
AVG(salario) salarioMedio,
AVG(salario * (1+comision)) salarioConComision,
SUM(salario) gastoSalarios
FROM Empleados
GROUP BY departamentoId;
-- Número de empleados que tiene el departamento con más empleados
SELECT MAX(numEmpleados)
FROM EstadísticasEmpleados;
```

MariaDB Error Codes



Shared MariaDB/MySQL error codes

Error Code	SQLSTATE	Error	Description
1000	HY000	ER_HASHCHK	hashchk
1001	HY000	ER_NISAMCHK	isamchk
1002	HY000	ER_NO	NO
1003	HY000	ER_YES	YES
1004	HY000	ER_CANT_CREATE_FILE	Can't create file '%s' (errno: %d)
1005	HY000	ER_CANT_CREATE_TABLE	Can't create table '%s' (errno: %d)
1006	HY000	ER_CANT_CREATE_DB	Can't create database '%s' (errno: %d
1007	HY000	ER_DB_CREATE_EXISTS	Can't create database '%s'; database exists
1008	HY000	ER_DB_DROP_EXISTS	Can't drop database '%s'; database doesn't exist
1009	HY000	ER_DB_DROP_DELETE	Error dropping database (can't delete '%s', errno: %d)
1010	HY000	ER_DB_DROP_RMDIR	Error dropping database (can't rmdir '%s', errno: %d)
1011	HY000	ER_CANT_DELETE_FILE	Error on delete of '%s' (errno: %d)
1012	HY000	ER_CANT_FIND_SYSTEM_REC	Can't read record in system table
1013	HY000	ER_CANT_GET_STAT	Can't get status of '%s' (errno: %d)
1014	HY000	ER_CANT_GET_WD	Can't get working directory (errno: %d)
1015	HY000	ER_CANT_LOCK	Can't lock file (errno: %d)
1016	HY000	ER_CANT_OPEN_FILE	Can't open file: '%s' (errno: %d)
1017	HY000	ER_FILE_NOT_FOUND	Can't find file: '%s' (errno: %d)
1018	HY000	ER_CANT_READ_DIR	Can't read dir of '%s' (errno: %d)
1019	HY000	ER_CANT_SET_WD	Can't change dir to '%s' (errno: %d)
1020	HY000	ER_CHECKREAD	Record has changed since last read in table '%s'
1021	HY000	ER_DISK_FULL	Disk full (%s); waiting for someone to free some space...

https://mariadb.com/kb/en/library/mariadb-error-codes/

Retos HackerRank

https://www.hackerrank.com/domains/sql

Conclusiones

SQL (Structured Query Language) es el lenguaje estándar para gestionar bases de datos relacionales, permitiendo crear esquemas (DDL), manipular datos (DML) y controlar acceso (DCL). Su popularidad se debe a su sintaxis intuitiva basada en el lenguaje natural, siendo accesible para analistas y programadores.

El DDL (Data Definition Language) incluye CREATE, ALTER y DROP para definir estructuras: tablas, índices, vistas y restricciones. La transformación del modelo relacional a SQL es directa, con tipos de datos específicos para cada atributo y restricciones que garantizan integridad.

El DML (Data Manipulation Language) incluye SELECT (consultas), INSERT (inserción), UPDATE (actualización) y DELETE (eliminación). La cláusula WHERE filtra datos, JOIN combina tablas, y subconsultas permiten consultas complejas anidadas. El ORDER BY y GROUP BY organizan y agrupan resultados.

Las vistas ofrecen una capa de abstracción, presentando al usuario un subconjunto lógico de datos sin exponer la estructura interna. Los índices mejoran rendimiento de búsquedas a costa de almacenamiento y tiempo de actualización. Las restricciones (PRIMARY KEY, FOREIGN KEY, CHECK, UNIQUE) garantizan integridad.

SQL es únicamente de lectura y modificación; lógica procedural requiere lenguajes de extensión específicos (PL/SQL, T-SQL, PL/pgSQL). La optimización de consultas es un art que requiere comprensión de índices, planes de ejecución y estadísticas de la base de datos. Dominar SQL es fundamental para cualquier profesional que trabaje con datos.

Gracias

Universidad de Sevilla

Departamento de Lenguajes y Sistemas Informáticos