Realización de consultas

Tema 4

Objetivos

- Identificar herramientas y sentencias para realizar consultas.
- Identificar y crear consultas simples sobre una tabla.
- Identificar y crear consultas que generen valores resumen.
- Identificar y crear consultas con composiciones internas y externas.
- Identificar y crear subconsultas.
- Valorar las ventajas e inconvenientes de las distintas opciones válidas para realizar una consulta.

1. El lenguaje de manipulación de datos

- El sublenguaje de SQL que permite la consulta o manipulación de datos es el **DML** (Data Modification Language).

- Sentencias de DDL:
  - **SELECT**: para extraer información de la base de datos, ya sea de una tabla o de varias.
  - **INSERT**: para insertar uno o vario s registros en alguna tabla.
  - **DELETE**: para borrar registros de una tabla.
  - **UPDATE**: para modificar registros de una tabla.

- Cualquier ejecución de un comando en un SGBD se denomina **consulta**. Las consultas no solo son SELECT, sino también cualquier sentencia de tipo UPDATE, INSERT, CREATE, DROP, etc.

2. La sentencia SELECT

- La sentencia **SELECT** se utiliza para consultar información de determinadas tablas.
- Podemos tener:
  - Sentencias muy sencillas que muestran todos los registros de una tabla.
  - Consultas que obtienen información filtrada de múltiples tablas.
- Formato básico de una consulta:

SELECT [DISTINCT] select_exp [,select_expr]...[FROM tabla];

Select_expr:
Nombre_columna [AS alias]
| *
| expresión

2. La sentencia SELECT

- **Nombre_columna**: indica un nombre de columna. Se puede seleccionar de una tabla una serie de columnas, o todas (usando *), o una expresión algebraica compuesta por operadores, operandos y funciones.
- **DISTINCT**: fuerza a que solo se muestren los registros con valores distintos (se suprimen las repeticiones).

- **Ejemplos**: (Tabla vehiculos)

Campo	Tipo	Null	Clave	Default
matricula	VARCHAR(7)	NO	PRIMARIA	NULL
marca	VARCHAR(20)	YES		'Seat'
modelo	VARCHAR(20)	YES		NULL
precio	NUMERIC(7,2)	YES		NULL

2. La sentencia SELECT

- **Ejemplos**:

```
SELECT * FROM vehiculos;

SELECT matricula, modelo FROM vehiculos;

SELECT matricula, concat(marca,modelo) AS coche
FROM vehiculos;

SELECT matricula, modelo, 1+5
FROM vehiculos;

SELECT 1+6;

SELECT marca FROM vehiculos;

SELECT DISTINCT marca FROM vehiculos;
```

3. Filtros

- Los **filtros** son condiciones que cualquier gestor de base de datos interpreta para seleccionar registros y mostrarlos como resultado de la consulta.
- Para realizar filtros se usa la cláusula **WHERE**.
- Sintaxis de **SELECT** con filtros:

```
SELECT [DISTINCT] select_exp [,select_expr]…
[FROM tabla]
[WHERE filtro];
```

- **Filtro**: es una expresión que indica la condición o condiciones que deben satisfacer los registros para ser seleccionados.
- **Ejemplo**:

```
SELECT *
FROM vehiculos
WHERE marca='Seat';
```

3.1. Expresiones para filtros

- Los **filtros** se construyen mediante expresiones.
- Una **expresión** es una combinación de operadores, operandos y funciones que producen un resultado.
- **Elementos** que pueden formar parte de las expresiones:
  - **Operandos**: pueden ser constantes (3, 'hola'), variables (campo edad) u otras expresiones.
  - **Operadores aritméticos**: +, -, *, /, %
  - **Operadores relacionales**: <, >, <>, <=, >=, =. Devuelven 1 si el resultado es cierto y 0 si no lo es.
  - **Operadores lógicos**: AND, OR, NOT. Toman como operadores valores lógicos (en SQL 1 o 0).
  - **Paréntesis**: (). Para alterar la prioridad de los operadores.
  - **Funciones**: date_add, concat, left,… cada SGBD incorpora su propias funciones.

3.2. Construcciones de filtros

- **Ejemplos**: Construcción de filtros para una base de datos de jugadores de la liga americana de baloncesto (NBA). Tabla equipos.

Campo	Tipo	Null	Clave	Default
codigo	INT(11)	NO	PRIMARIA	NULL
nombre	VARCHAR(30)	YES		NULL
Procedencia	VARCHAR(20)	YES		NULL
Altura	VARCHAR(4)	YES		NULL
Peso	INT(11)	YES		NULL
Posicion	VARCHAR(5)	YES		NULL
Nombre_equipo	VARCHAR(20)	YES		NULL

3.2. Construcciones de filtros

- **Ejemplos**: Construcción de filtros para una base de datos de jugadores de la liga americana de baloncesto (NBA).

```
SELECT nombre
FROM jugadores
WHERE nombre_equipo='Lakers';


SELECT codigo, nombre, altura
FROM jugadores
WHERE nombre_equipo='Lakers' AND procedencia='Spain';


SELECT nombre, altura, procediencia
FROM jugadores
WHERE nombre_equipo='Lakers'
      AND (procedencia='Spain' OR procedencia='Slovenia');
```

3.3. Filtros con operador de pertenencia a conjuntos

- Se puede hacer uso del operador de pertenencia a conjuntos **IN**.
- Sintaxis:

  nombre_columna IN (value1, value2,…)
- Permite comprobar si una columna tiene valor igual que cualquier de los que están incluidos dentro del paréntesis.

- **Ejemplo**:

```
SELECT  nombre, altura, procedencia
FROM jugadores
WHERE nombre_equipo='Lakers' AND
      procedencia IN ('Spain', 'Slovenia', 'Serbia & Montenegro');
```

3.4. Filtros con operador de rango

- El operador de rango **BETWEEN** permite seleccionar los registros que estén incluidos en un rango.
- Sintaxis:

  Nombre_columna BETWEEN valor1 and valor2

- **Ejemplo**:

```
SELECT nombre, nombre_equipo, peso
FROM jugadores
WHERE peso BETWEEN 90 AND 150;
```

3.5. Filtros con test de valor nulo

- Los operadores **IS** e **IS NOT** permiten verificar si un campo es o no es nulo respectivamente.

- **Ejemplos**:

```
SELECT nombre, nombre_equipo
FROM jugadores
WHERE procedencia IS NULL;
```

```
SELECT nombre, nombre_equipo
FROM jugadores
WHERE procedencia IS NOT NULL;
```

3.6. Filtros con test de patrón

- Los **filtros con test de patrón** seleccionan los registros que cumplan una serie de características.
- **Caracteres comodines** para buscar una cadena de caracteres :
  - **%:** busca coincidencias de cualquier número de caracteres, incluso cero caracteres.
  - **_:** busca coincidencias de exactamente un carácter.
- **Ejemplos**:

```
SELECT *
FROM vehiculos
WHERE modelo like '%tdi%';

SELECT nombre, conferencia
FROM equipos
WHERE nombre LIKE 'R_ _ _ _ _ s';
```

4. Ordenación

- Para mostrar ordenados un conjunto de registros se utiliza la cláusula **ORDER BY** de la sentencia SELECT.
- **Ejemplo**:

```
SELECT [DISTINCT] select_expr [,select_expr] ...
[FROM tabla]
[WHERE filtro]
[ORDER BY {nombre_col | exp | posición} [ASC | DESC], ...]
```

- Permite ordenar el conjunto de resultados de forma ascendente (**ASC**) o descendente (**DESC**) por una o varias columnas.
- Por defecto se ordena de forma ascendente (**ASC**).

4. Ordenación

- **Ejemplos**: (tabla equipos)

Campo	Typo	Null	Clave	Default
nombre	VARCHAR(20)	NO	PRIMARIA	NULL
ciudad	VARCHAR(20)	YES		NULL
conferencia	VARCHAR(4)	YES		NULL
division	VARCHAR(9)	YES		NULL

```
SELECT nombre, division
FROM equipos
WHERE conferencia='West'
ORDER BY division ASC;

SELECT division, nombre
FROM equipos
WHERE conferencia='West'
ORDER BY division ASC, nombre DESC;
```

5. Consultas de resumen

- Se pueden generar consultas más complejas que resuman cierta información, extrayendo información calculada de varios conjuntos de registros.
- **Ejemplo**:

  SELECT COUNT(*) FROM vehiculos;

- **COUNT(*)**: función que tomo como entrada los registros de la tabla consultada y cuenta cuántos registros hay.
- Hay que hacer uso de las **funciones de columna**, que convierten un conjunto de registros en una información simple cuyo resultado es un cálculo.

5. Consultas de resumen

- **Funciones de columna:**

Función	Descripción
SUM(expresión)	Suma los valores indicados en el argumento
AVG(expresión)	Calcula la media de los valores
MIN(expresión)	Calcula el mínimo
MAX(expresión)	Calcula el máximo
COUNT(nbColumna)	Cuenta el número de valores de una columna (excepto los nulos)
COUNT(*)	Cuenta el número de valores de una fila (incluyendo los nulos)

5. Consultas de resumen

- **Ejemplos:**

```
SELECT MAX(peso)
FROM jugadores;

SELECT MIN(altura)
FROM jugadores;

SELECT COUNT(*)
FROM jugadores
WHERE nombre_equipo='Lakers';

SELECT AVG(peso)
FROM jugadores
WHERE nombre_equipo='Blazers';
```

5. Consultas de resumen

- Se pueden realizar **agrupaciones** de registros.
- **Agrupación de registros**: conjunto de registros que tienen una o varias columnas con el mismo valor.
- A un grupo de registros se le puede aplicar una función de columna.
- **Ejemplos**:

```
SELECT *
FROM vehiculos;

SELECT marca, COUNT(*)
FROM vehiculos
GROUP BY marca;
```

5. Consultas de resumen

- Sintaxis de **SELECT** con **GROUP BY**:

  SELECT [DISTINCT] select_expr [,select_expr] ...
  [FROM tabla]
  [WHERE filtro]
  [GROUP BY expr [, expr] ...]
  [ORDER BY {nombre_col | exp | posición} [ASC | DESC], ...]

- La sentencia **ORDER BY** debe ir después de la sentencia **GROUP BY**.
- Para cada agrupación se debe seleccionar también la columna por la cual se agrupa.
- Para mezclar funciones de columna y columnas de una tabla hay que escribir una cláusula **GROUP BY**.

5.1. Filtros de grupos

- Los filtros de grupos deben realizarse mediante el uso de la cláusula **HAVING**, ya que **WHERE** actúa antes de agrupar los registros.
- Sintaxis:

```
SELECT [DISTINCT] select_expr [,select_expr] ...
[FROM tabla]
[WHERE filtro]
[GROUP BY expr [, expr] ...]
[HAVING filtro_grupos]
[ORDER BY {nombre_col | exp | posición} [ASC | DESC], ...]
```

5.1. Filtros de grupos

- **Ejemplos:**

```
SELECT nombre_equipo, AVG(peso)
FROM jugadores
GROUP BY nombre_equipo
HAVING AVG(peso)>100
ORDER BY AVG(peso);

SELECT nombre_equipo, COUNT(*)
FROM jugadores
WHERE procedencia='Spain'
GROUP BY nombre_equipo
HAVING COUNT(*)>1;
```

6. Subconsultas

- Las **subconsultas** se utilizan para realizar filtrados con los datos de otra consulta.
- Estos filtros pueden ser aplicados tanto en la cláusula **WHERE** para filtrar registros como en la cláusula **HAVING** para filtrar grupos.

- **Ejemplos** :

```
SELECT nombre
FROM jugadores
WHERE nombre_equipo IN
      (SELECT nombre
      FROM equipos
      WHERE división = 'SouthWest');
```

6.1. Test de comparación

- Consiste en usar los operadores de comparación =, <=, >=, <>, > y < para comparar el valor producido con un valor único generado por una subconsulta.

- **Ejemplo**:

```
SELECT nombre
FROM jugadores
WHERE altura =
      (SELECT MAX(altura) FROM jugadores);
```

6.2. Test de pertenencia a conjunto

- Consiste en usar el operador **IN** para filtrar los registros cuya expresión coincida con algún valor producido por la subconsulta.

- **Ejemplo**:

```
SELECT division
FROM equipos
WHERE nombre IN
      (SELECT nombre_equipo
      FROM jugadores
      WHERE procedencia='Spain');
```

6.3. Test de existencia

- Permite filtrar los resultados de una consulta si existen filas en la subconsulta asociada, es decir, si la subconsulta genera un número de filas distinto de 0.
- Se utiliza el operador **EXISTS (o NOT EXISTS)**.
- Sintaxis:

  SELECT columnas

  FROM tabla

  WHERE EXISTS | NOT EXISTS (sbconsulta);

- **Ejemplo**:

```
SELECT nombre
FROM equipos
WHERE NOT EXISTS
      (SELECT nombre
      FROM jugadores
      WHERE equipos.nombre=jugadores.nombre_equipo AND
      procedencia='Spain');
```

6.4. Test cuantificados ALL Y ANY

- Sirven para calcular la relación entre una expresión y todos los registros de la subconsulta (**ALL**) o algunos de los registros de la subconsulta (**ANY**).

- **Ejemplos**:

```
SELECT nombre, peso
FROM jugadores
WHERE peso > ALL
      (SELECT peso
      FROM jugadores
      WHERE procedencia='Spain');

SELECT nombre, peso
FROM jugadores
WHERE posicion?='G' AND peso > ANY
      (SELECT peso
      FROM jugadores
      WHERE posicion='C');
```

(los bases que pesan más que cualquier pivot)

6.5. Subconsultas anidadas

- Se puede usar una subconsulta para filtrar los resultados de otra subconsulta (se anidan subconsultas).

- **Ejemplo**:

  Obtener el nombre de la ciudad donde juega el jugador más alto de la NBA.

  1. Obtener la altura del jugador más alto:

     X ⟸ **SELECT MAX(altura) FROM jugadores**

  2. Obtener el nombre del jugador, a través de la altura se localiza al jugado y por tanto, su equipo:

     Y ⟸ **SELECT nombre_equipo FROM jugadores WHERE altura=X**

  3. Obtener la ciudad:

     SELECT ciudad FROM equipos WHERE nombre=Y

6.5. Subconsultas anidadas

- **Ejemplo**:

  Obtener el nombre de la ciudad donde juega el jugador más alto de la NBA.

```
SELECT ciudad
FROM equipos
WHERE nombre=
      (SELECT nombre_equipo
      FROM jugadores
      WHERE altura=
                  (SELECT MAX(altura)
                  FROM jugadores));
```

7. Consultas multitabla

- Una consulta multitabla es aquella en la que se puede consultar información de más de una tabla.
- Se aprovechan los campos relacionados de las tablas para unirlas (join).

- Sintaxis:

SELECT [DISTINCT] select_expr [,select_expr] ...
[FROM referencias_tablas]
[WHERE filtro]
[GROUP BY expr [, expr] ...]
[HAVING filtro_grupos]
[ORDER BY {nombre_col | exp | posición} [ASC | DESC], ...]

Referencias_tablas:
Referencia_tabla [, referencia_tabla]...
| referencia_tabla [INNER | CROSS] JOIN referencia_tabla [ON condición]
| referencia_tabla LEFT [OUTER] JOIN referencia_tabla ON condición
| referencia_tabla RIGHT [OUTER] JOIN referencia_tabla ON condición

Referencia_tabla:
Nombre_tabla [[AS] alias]

7.1. Consultas multitabla SQL 1

- El producto cartesiano de dos tablas son todas las combinaciones de las filas de una tabla unidas a las filas de la otra tabla.

- **Ejemplo**: Una base de datos con dos tablas mascotas y propietarios.

SELECT * FROM propietarios;

Dni	Nombre
12345678T	José Pérez
44556677P	Marta López

SELECT * FROM animales;

Codigo	Nombre	Tipo	Propietario
1	Pipo	Gato	12345678T
2	Cuca	Gato	44556677P
3	Pepe	Perro	44556677P

7.1. Consultas multitabla SQL 1

- **Ejemplo**: Una base de datos con dos tablas mascotas y propietarios.

  SELECT * FROM animales, propietarios;

Codigo	Nombre	Tipo	Propietario	Dni	nombre
1	Pipo	Gato	12345678T	12345678T	José Pérez
2	Cuca	Gato	44556677P	12345678T	José Pérez
3	Pepe	Perro	44556677P	12345678T	José Pérez
1	Pipo	Gato	12345678T	44556677P	Marta López
2	Cuca	Gato	44556677P	44556677P	Marta López
3	Pepe	Perro	44556677P	44556677P	Marta López

Obtenemos el producto cartesiano de las tablas animales y propietarios

7.1. Consultas multitabla SQL 1

- **Ejemplo**: Aplicamos un filtro al producto cartesiano.

```
SELECT *
FROM animales, propietarios
WHERE propietarios.dni=animales.propietario;
```

Codigo	Nombre	Tipo	Propietario	Dni	nombre
1	Pipo	Gato	12345678T	12345678T	José Pérez
2	Cuca	Gato	44556677P	44556677P	Marta López
3	Pepe	Perro	44556677P	44556677P	Marta López

JOIN (de las tablas animales y propietarios): consiste en realizar un producto cartesiano de ambas tablas y un filtro por el campo relacionado (clave externa vs clave primaria).

JOIN = PRODUCTO CARTESIANO + FILTRO

7.2. Consultas multitabla SQL 2

- Otra sintaxis para los siguientes tipos de consultas multitabla:
  1. **Join Interna**:
     - De equivalencia (***INNER JOIN***)
     - Natural (***NATURAL JOIN***)
  2. **Producto Cartesiano** (***CROSS JOIN***)
  3. **Join Externa**:
     - De tabla derecha (***RIGHT OUTER JOIN***)
     - De tabla izquierda (***LEFT OUTER JOIN***)
     - Completa (***FULL OUTER JOIN***)

7.2.1. Composiciones internas. INNER JOIN

- Para expresar este tipo de composiciones se puede usar:
  - **INNER JOIN** (no tiene en cuenta los valores nulos)
  - **,** (como en SQL 1)

- **Ejemplo**:

```
SELECT *
FROM animales INNER JOIN propietarios
ON animales.propietario = propietarios.dni;
```

Codigo	Nombre	Tipo	Propietario	Dni	nombre
1	Pipo	Gato	12345678T	12345678T	José Pérez
2	Cuca	Gato	44556677P	44556677P	Marta López
3	Pepe	Perro	44556677P	44556677P	Marta López

7.2.2. Composiciones naturales. NATURAL JOIN

- **NATURAL JOIN** es una especialización de la **INNER JOIN**.
- Se comparan las columnas que tengan el mismo nombre en ambas tablas.
- La tabla resultante contiene solo una columna por cada par de columnas con el mismo nombre.

7.2.3. Producto cartesiano. CROSS JOIN

- **CROSS JOIN** devuelve el producto cartesiano de dos tablas.
- **Ejemplo**:

```
SELECT *
FROM  animales CROSS JOIN propietarios;
```

Codigo	Nombre	Tipo	Propietario	Dni	nombre
1	Pipo	Gato	12345678T	12345678T	José Pérez
2	Cuca	Gato	44556677P	12345678T	José Pérez
3	Pepe	Perro	44556677P	12345678T	José Pérez
1	Pipo	Gato	12345678T	44556677P	Marta López
2	Cuca	Gato	44556677P	44556677P	Marta López
3	Pepe	Perro	44556677P	44556677P	Marta López

7.2.4. Composiciones externas. OUTER JOIN

- Las tablas relacionadas no requieren que haya una equivalencia. El registro es seleccionado para ser mostrado aunque no haya otro registro que le corresponda.
- **OUTER JOIN** se subdivide dependiendo de la tabla a la cual se le admitirán los registros que no tienen correspondencia, ya sean de tabla izquierda, de tabla derecha o combinación completa.
- **LEFT OUTER JOIN:** los registros que admiten no tener correspondencia son los que aparecen en la tabla izquierda. (animales sin propietario).
- **RIGHT OUTER JOIN:** los registros que admiten no tener correspondencia son los que aparecen en la tabla derecha. (propietarios sin animales).
- **FULL OUTER JOIN:** admite registros sin correspondencia tanto para la tabla izquierda como para la derecha. (animales sin propietario y propietarios sin animales)

7.2.4. Composiciones externas. OUTER JOIN

- **Ejemplo LEFT OUTER JOIN:**

```
SELECT *
FROM animales LEFT OUTER JOIN propietarios
ON animales.propietario = propietarios.dni;
```

Codigo	Nombre	Tipo	Propietario	Dni	nombre
1	Pipo	Gato	12345678T	12345678T	José Pérez
2	Cuca	Gato	44556677P	44556677P	Marta López
3	Pepe	Perro	44556677P	44556677P	Marta López
4	Coco	Perro	NULL	NULL	NULL

Sentido: sacar todos los animales y si tienen relación, sacar sus propietarios, y si no tiene propietario, indicarlo con un valor nulo.

7.2.4. Composiciones externas. OUTER JOIN

- **Ejemplo RIGHT OUTER JOIN:**

```
SELECT *
FROM animales RIGHT OUTER JOIN propietarios
ON animales.propietario = propietarios.dni;
```

Codigo	Nombre	Tipo	Propietario	Dni	nombre
1	Pipo	Gato	12345678T	12345678T	José Pérez
NULL	NULL	NULL	NULL	22332233J	Matías Fernández
2	Cuca	Gato	44556677P	44556677P	Marta López
3	Pepe	Perro	44556677P	44556677P	Marta López

7.2.4. Composiciones externas. OUTER JOIN

- **Ejemplo FULL OUTER JOIN:**

```
SELECT *
FROM animales FULL OUTER JOIN propietarios
ON animales.propietario = propietarios.dni;
```

Codigo	Nombre	Tipo	Propietario	Dni	nombre
1	Pipo	Gato	12345678T	12345678T	José Pérez
2	Cuca	Gato	44556677P	44556677P	Marta López
3	Pepe	Perro	44556677P	44556677P	Marta López
4	Coco	Perro	NULL	NULL	NULL
NULL	NULL	NULL	NULL	22332233J	Matías Fernández

8. Consultas reflexivas

- Obtener información de relaciones reflexivas.

- **Ejemplo**: un informe de empleados donde junto a su nombre y apellidos aparezca el nombre y apellidos de su jefe.
  - Realizar un JOIN entre registros de la misma tabla.

```
SELECT CONCAT(emp.Nombre, ` `, emp.Apellido1) as Empleado
            CONCAT(jefe.Nombre, ` `, jefe.Apellido1) as Jefe
FROM Empleados emp INNER JOIN Empleados jefe
ON emp.CodigoEmpleado = jefe.CodigoJefe;
```

8. Consultas reflexivas

- **Ejemplo**: un informe de empleados donde junto a su nombre y apellidos aparezca el nombre y apellidos de su jefe.

Field	Type	Null	Key	Default
CodigoEmpleado	Int(11)	No	Primary	NULL
Nombre	Varchar(50)	No		NULL
Apellido1	Varchar(50)	No		NULL
Apellido2	Varchar(50)	Yes		NULL
Extension	Varchar(10)	No		NULL
Email	Varchar(100)	No		NULL
CodigoOficina	Varchar(10)	No		NULL
CodigoJefe	Int(11)	Yes		NULL
Puesto	Varchar(50)	Yes		NULL

8. Consultas reflexivas

- **Ejemplo**: un informe de empleados donde junto a su nombre y apellidos aparezca el nombre y apellidos de su jefe.

```
SELECT CONCAT(emp.Nombre, ` `, emp.Apellido1) as Empleado
            CONCAT(jefe.Nombre, ` `, jefe.Apellido1) as Jefe
FROM Empleados emp INNER JOIN Empleados jefe
ON emp.CodigoEmpleado = jefe.CodigoJefe;
```

Empleado	Jefe
Marcos Magaña	Rubén López
Rubén López	Alberto Soria
…	…
Alberto Soria	Kevin Fallmer
Kevin Fallmer	Julian Bellinelli
Kevin Fallmer	María Molina

9. Consultas con tablas derivadas

- **Las consultas con tablas derivadas** son aquellas que utilizan sentencias SELECT en cláusula FROM en lugar de nombres de tablas.

- **Ejemplo**:

```
SELECT *
FROM
    (SELECT CodigoEmpleado, Nombre
    FROM Empleados
    WHERE CodigoOficina='TAL_ES') as tabla_derivada;
```