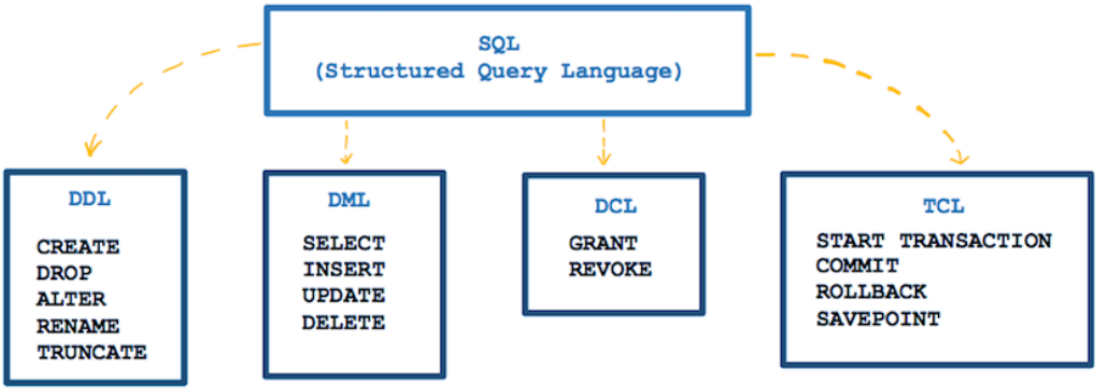
Realización de consultas SQL



1. Introducción.

En la fase de análisis hemos realizado la E.R.S. A partir de dicha especificación de requisitos, en la unidad anterior, aprendimos a realizar el diseño conceptual de una BD mediante el Modelo E/R y el Modelo E/R extendido respectivamente.

También vimos como hacer el modelo lógico mediante el modelo relacional que se obtenía a partir del modelo E/R y aprendimos a comprobar que dicho modelo estaba normalizado.

Siguiendo con el proceso de desarrollo, lo que debemos hacer ahora es pasar al diseño físico de la BD. Es decir, implementar la Base de Datos. Para ello, se programarán las diferentes tablas que constituirán la Base de Datos, se introducirán los datos y, más adelante, se construirán las consultas. Todo ello se hará programando en el lenguaje más extendido para la definición y manipulación de datos en SGBDR: SQL.

Las prácticas del módulo de Gestión Bases de Datos se van a realizar utilizando el Sistema de Gestión de Bases de Datos Relacional (RDBMS) ORACLE. Varias son las razones que justifican la impartición de estas prácticas utilizando ORACLE:

En primer lugar, ORACLE es un producto comercial ampliamente extendido y utilizado, que cuenta con una importante cuota de mercado dentro del mundo de las bases de datos, estando disponible para prácticamente la totalidad de plataformas posibles (Windows, MAC, UNIX, LINUX, …) con la ventaja de que las aplicaciones realizadas para una

plataforma concreta pueden ser portadas de forma automática a cualquiera de las otras plataformas.

ORACLE permite almacenar y manejar gran cantidad de información de forma rápida y segura, destacando además su valor educativo, ya que la herramienta que utiliza ORACLE para acceder a la base de datos es el lenguaje no procedural SQL, y este lenguaje implementa prácticamente toda la funcionalidad y características del modelo relacional teórico.

1.1. EL LENGUAJE SQL

1.1.1. Historia

El nacimiento del lenguaje SQL data de 1970 cuando E. F. Codd publica su libro: «Un modelo de datos relacional para grandes bancos de datos compartidos». Ese libro dictaría las directrices de las bases de datos relacionales. Apenas dos años después IBM (para quien trabajaba Codd) utiliza las directrices de Codd para crear el Standard English Query Language (Lenguaje Estándar Inglés para Consultas) al que se le llamó SEQUEL. Más adelante se le asignaron las siglas SQL (Standard Query Language, lenguaje estándar de consulta) aunque en inglés se siguen pronunciando secuel. En español se pronuncia esecuele.

En 1979 Oracle presentó la primera implementación comercial del lenguaje. Poco después se convertía en un estándar en el mundo de las bases de datos avalado por los organismos ISO y ANSI. En el año 1986 se toma como lenguaje estándar por ANSI de los SGBD relacionales. Un año después lo adopta ISO, lo que convierte a SQL en estándar mundial como lenguaje de bases de datos relacionales.

En 1989 aparece el estándar ISO (y ANSI) llamado SQL89 o SQL1. En 1992 aparece la nueva versión estándar de SQL (a día de hoy sigue siendo la más conocida) llamada SQL92. En 1999 se aprueba un nuevo SQL estándar que incorpora mejoras que incluyen triggers, procedimientos, funciones,… y otras características de las bases de datos objeto-relacionales; dicho estándar se conoce como SQL99. El último estándar es el del año 2011 (SQL2011) Elementos de SQL

SQL se basa en la Teoría Matemática del Álgebra Relacional. El lenguaje SQL consta de varios elementos:

Lenguaje de definición de datos (DDL): proporciona órdenes para definir, modificar o eliminar los distintos objetos de la base de datos (tablas, vistas, índices…).

Lenguaje de Manipulación de Datos (DML): proporciona órdenes para insertar, suprimir y modificar registros o filas de las tablas. También contempla la realización de consultas sobre la BD.

Lenguaje de Control de Datos (DCL): permite establecer derechos de acceso de los usuarios sobre los distintos objetos de la base de datos. Lo forman las instrucciones GRANT y REVOKE.

Oracle contempla además **sentencias para transacciones**. Administran las modificaciones creadas por las instrucciones DML. Lo forman las instrucciones ROLLBACK, COMMIT y SAVEPOINT.

1.1.2. Proceso de ejecución de sentencia SQL

El proceso de una instrucción SQL es el siguiente:

- Se analiza la instrucción. Para comprobar la sintaxis de la misma
- Si es correcta se valora si los metadatos de la misma son correctos. Se comprueba esto en el diccionario de datos.
- Si es correcta, se optimiza, a fin de consumir los mínimos recursos posibles.
- Se ejecuta la sentencia y se muestra el resultado al emisor de la misma.

1.1.3. Criterios de notación

La notación utilizada para la especificación de los comandos de SQL es la siguiente:

CRITERIOS DE NOTACIÓN

- Palabras clave de la sintaxis SQL en **MAYÚSCULAS**.
- Los corchetes **[ ]** indican **opcionalidad**.
- Las llaves **{ }** delimitan **alternativas** separadas por **|** de las que se debe elegir una.
- Los puntos suspensivos … indican **repetición** varias veces de la opción anterior.

1.1.4. Normas de escritura

- En SQL no se distingue entre mayúsculas y minúsculas.
- Da lo mismo como se escriba. El final de una instrucción o sentencia lo marca el signo de **punto y coma**.
- Las sentencias SQL (SELECT, INSERT, …) se pueden escribir en varias líneas siempre que las palabras no sean partidas.
- Los comentarios en el código SQL pueden ser de 2 tipos:
  - de bloque: comienzan por /* y terminan por */
  - de línea: comienzan por – y terminan en final de línea.

```
/*
  Esto es un comentario
  de varias líneas.

  Fin.
*/

-- Esto es un comentario de una línea
```

2. El lenguaje DML de SQL.Consultas

A lo largo de esta unidad nos centraremos en la cláusula **SELECT.** Sin duda es el comando más versátil del lenguaje SQL. El comando **SELECT** permite:

- Obtener datos de ciertas columnas de una tabla **(proyección).**
- Obtener registros (filas) de una tabla de acuerdo con ciertos criterios **(selección).**
- Mezclar datos de tablas diferentes **(asociación, join)**.

2.1. CONSULTAS

Para realizar consultas a una base de datos relacional hacemos uso de la sentencia SELECT. La sintaxis básica del comando SELECT es la siguiente:

```
SELECT * | {[ DISTINCT ] columna | expresión [[AS] alias ], ...}
FROM nombre_tabla;
```

Donde:

- *. El asterisco significa que se seleccionan todas las columnas.
- **DISTINCT**. Hace que no se muestren los valores duplicados.
- **columna**. Es el nombre de una columna de la tabla que se desea mostrar.
- **expresión**. Una expresión válida SQL.
- **alias**. Es un nombre que se le da a la cabecera de la columna en el resultado de esta instrucción.

Ejemplos:

```
-- Selección de todos los registros de la tabla CLIENTES
SELECT * FROM CLIENTES;

-- Selección de algunos campos de la tabla CLIENTES
SELECT nombre, apellido1, apellido2 FROM CLIENTES;
```

2.2. CÁLCULOS

2.2.1. Cálculos Aritméticos

Los operadores + (suma), - (resta), * (multiplicación) y / (división), se pueden utilizar para hacer cálculos en las consultas. Cuando se utilizan como expresión en una consulta SELECT, no modifican los datos originales.

Ejemplo:

```
-- Consulta con 3 columnas
SELECT nombre, precio, precio*1.16 FROM ARTICULOS;


-- Ponemos un alias a la tercera columna.
-- Las comillas dobles en el alias hacen que se respeten mayúsculas y minúsculas,
-- de otro modo siempre aparece en mayúsculas
SELECT nombre, precio, precio*1.16 AS "Precio + IVA" FROM ARTICULOS;
```

La **prioridad** de esos operadores es: tienen más prioridad la multiplicación y división, después la suma y la resta. En caso de igualdad de prioridad, se realiza primero la operación que esté más a la izquierda. Como es lógico se puede evitar cumplir esa prioridad usando paréntesis; el interior de los paréntesis es lo que se ejecuta primero. Cuando una expresión aritmética se calcula sobre valores NULL, el resultado de la expresión es siempre NULL.

2.2.2. Concatenación

El operador **||** es el de la concatenación. Sirve para unir textos.

Ejemplo:

```
SELECT tipo, modelo, tipo || '-' || modelo "Clave Pieza" FROM PIEZAS;
```

El resultado de esa consulta tendría esta estructura:

TIPO	MODELO	Clave Pieza
AR	6	AR-6
AR	7	AR-7
AR	8	AR-8
AR	9	AR-9
AR	12	AR-12
AR	15	AR-15
AR	20	AR-20
AR	21	AR-21
BI	10	BI-10
BI	20	BI-20
BI	22	BI-22
BI	24	BI-24

2.2.3. Condiciones

Se puede realizar consultas que restrinjan los datos de salida de las tablas. Para ello se utiliza la cláusula **WHERE.** Esta cláusula permite colocar una **condición** que han de cumplir todos los registros que queremos que se muestran. Las filas que no la cumplan no aparecerán en la ejecución de la consulta.

Nota

Con el comando **SELECT** indicamos las **columnas** que queremos que aparezcan en nuestra consulta. Con el comando **WHERE** indicamos las **filas** que queremos que aparezcan en nuestra consulta (serán las que cumplan las condiciones que especifiquemos detrás del WHERE).

Ejemplo:

```
-- Tipo y modelo de las piezas cuyo precio es mayor que 3
SELECT tipo, modelo
FROM PIEZAS
WHERE precio > 3;
```

2.2.3.1. Operadores de comparación

Los operadores de comparación que se pueden utilizar en la cláusula WHERE son:

Operador	Significado
>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que
=	Igual
<>	Distinto
!=	Distinto

Se pueden utilizar tanto para comparar números como para comparar textos y fechas. En el caso de los textos, las comparaciones se hacen en orden alfabético. Sólo que es un orden alfabético estricto. Es decir el orden de los caracteres en la tabla de códigos. Así la letra Ñ y las vocales acentuadas nunca quedan bien ordenadas ya que figuran con códigos más altos. Las mayúsculas figuran antes que las minúsculas (la letra "Z" es menor que la "a").

2.2.3.2. Operadores lógicos.

Operador	Significado
AND	Devuelve verdadero si las expresiones a su izquierda y derecha son ambas verdaderas
OR	Devuelve verdadero si cualquiera de las dos expresiones a izquierda y derecha del OR, son verdaderas
NOT	Invierte la lógica de la expresión que está a su derecha. Si era verdadera, mediante NOT pasa a ser falso.

Ejemplos:

```
-- Personas entre 25 y 50 años
SELECT nombre, apellidos
FROM PERSONAS
WHERE edad >= 25 AND edad <= 50;

-- Personas de más de 60 y menos de 20
SELECT nombre, apellidos
FROM PERSONAS
WHERE edad > 60 OR edad < 20;
```

2.2.3.3. BETWEEN

El operador BETWEEN nos permite obtener datos que se encuentren entre dos valores determinados (incluyendo los dos extremos).

Ejemplo:

```
-- Selección de las piezas cuyo precio está entre 3 y 8
-- (ambos valores incluidos)
SELECT tipo, modelo, precio FROM PIEZAS
WHERE precio BETWEEN 3 AND 8;
```

El operador NOT BETWEEN nos permite obtener los valores que son menores (estrictamente) que el más pequeño y mayores (estrictamente) que el más grande. Es decir, no incluye los extremos.

Ejemplo:

```
-- Selección de las piezas cuyo precio sea menor que 3 o mayor que 8
-- (los de precio 3 y precio 8 no estarán incluidos)
SELECT tipo, modelo, precio
FROM PIEZAS
WHERE precio NOT BETWEEN 3 AND 8;
```

2.2.3.4. IN

El operador IN nos permite obtener registros cuyos valores estén en una lista:

Ejemplo:

```
-- Selección de las piezas cuyo precio sea igual a 3, 5 u 8
SELECT tipo, modelo, precio
FROM PIEZAS
WHERE precio IN ( 3,5,8 );

-- Selección de las piezas cuyo precio no sea igual a 3, 5 u 8
SELECT tipo, modelo, precio
FROM PIEZAS
WHERE precio NOT IN ( 3,5,8 );
```

2.2.3.5. LIKE

El operador LIKE se usa sobre todo con textos, permite obtener registros cuyo valor en un campo cumpla una condición textual. LIKE utiliza una cadena que puede contener estos símbolos:

Símbolo	Significado
%	Una serie cualquiera de caracteres
_	Un carácter cualquiera

Ejemplos:

```
-- Selección el nombre de las personas que empiezan por A
SELECT nombre
FROM PERSONAS
WHERE nombre LIKE 'A%';
```

```
-- Selección el nombre y los apellidos de las personas
–cuyo primer apellido sea Jiménez, Giménez, Ximénez
SELECT nombre, apellido1, apellido2
FROM PERSONAS
WHERE apellido1 LIKE '_iménez';
```

2.2.3.6. IS NULL

La cláusula IS NULL devuelve “verdadero” si una expresión contiene un nulo, y “Falso” en caso contrario. La cláusula IS NOT NULL devuelve “verdadero” si una expresión NO contiene un nulo, y “Falso” en caso contrario.

Ejemplos:

```
-- Devuelve el nombre y los apellidos de las personas que NO tienen teléfono
SELECT nombre, apellido1, apellido2
FROM PERSONAS
WHERE telefono IS NULL;

-- Devuelve el nombre y los apellidos de las personas que SÍ tienen teléfono
SELECT nombre, apellido1, apellido2
FROM PERSONAS
WHERE telefono IS NOT NULL;
```

2.2.3.7. Precedencia de operadores.

A veces las expresiones que se producen en los SELECT son muy extensas y es difícil saber qué parte de la expresión se evalúa primero, por ello se indica la siguiente tabla de precedencia:

Orden de precedencia	Operador
1	*(Multiplicar) / (dividir)
2	+ (Suma) - (Resta)
3	|| (Concatenación)
4	Comparaciones (>, <, !=, ...)
5	IS [NOT] NULL, [NOT]LIKE, IN
6	NOT
7	AND
8	OR

2.3. SUBCONSULTAS.

Se trata de una técnica que permite utilizar el resultado de una tabla SELECT en otra consulta SELECT. Permite solucionar problemas en los que el mismo dato aparece dos veces. La sintaxis es:

```
SELECT lista expresiones
FROM tablas
WHERE expresión OPERADOR
   ( SELECT lista expresiones
     FROM tablas );
```

Se puede colocar el SELECT dentro de las cláusulas WHERE, HAVING o FROM. El operador puede ser >,<,>=,<=,!=, = o IN.
Ejemplo:

```
-- Obtiene los empleados cuyas pagas sean inferiores a lo que gana Martina.
SELECT nombre_empleado, paga
FROM EMPLEADOS
WHERE paga < ( SELECT paga
           FROM EMPLEADOS
           WHERE nombre_empleado='Martina');
```

Lógicamente el resultado de la subconsulta debe incluir el campo que estamos analizando. Se pueden realizar esas subconsultas las veces que haga falta:

```
SELECT nombre_empleado, paga
FROM EMPLEADOS
WHERE paga < ( SELECT paga
           FROM EMPLEADOS
           WHERE nombre_empleado='Martina')
AND   paga > ( SELECT paga
           FROM EMPLEADOS
           WHERE nombre_empleado='Luis');
```

La última consulta obtiene los empleados cuyas pagas estén entre lo que gana Luis y lo que gana Martina. **Una subconsulta que utilice los valores >,<,>=,... tiene que devolver un único valor,** de otro modo ocurre un error. Pero a veces se utilizan consultas del tipo: mostrar el sueldo y nombre de los empleados cuyo sueldo supera al de cualquier empleado del departamento de ventas.

La subconsulta necesaria para ese resultado mostraría los sueldos del departamento de ventas. Pero no podremos utilizar un operador de comparación directamente ya que compararíamos un valor con muchos valores. La solución a esto es utilizar instrucciones especiales entre el operador y la consulta. Esas instrucciones son:

Instrucción	Significado
ANY	Compara con cualquier registro de la subconsulta. La instrucción es válida si hay un registro en la subconsulta que permite que la comparación sea cierta
ALL	Compara con todos los registros de la consulta. La instrucción resulta cierta si es cierta toda comparación con los registros de la subconsulta
IN	No usa comparador, ya que sirve para comprobar si un valor se encuentra en el resultado de la subconsulta
NOT IN	Comprueba si un valor no se encuentra en una subconsulta

Ejemplo:

```
-- Obtiene el empleado que más cobra.
SELECT nombre, sueldo
FROM EMPLEADOS
WHERE sueldo >= ALL ( SELECT sueldo
                      FROM EMPLEADOS );

-- Obtiene los nombres de los empleados cuyos DNI están en la tabla de directivos.
-- Es decir, obtendrá el nombre de los empleados que son directivos.
SELECT nombre, sueldo

FROM EMPLEADOS

WHERE DNI IN ( SELECT DNI

               FROM DIRECTIVOS );
```

2.4. ORDENACIÓN.

El orden inicial de los registros obtenidos por un SELECT guarda una relación con el orden en el que fueron introducidos. Para ordenar en base a criterios más interesantes, se utiliza la cláusula **ORDER BY.** En esa cláusula se coloca una lista de campos que indica la forma de ordenar. Se ordena primero por el primer campo de la lista, si hay coincidencias por el segundo, si ahí también las hay por el tercero, y así sucesivamente. Se puede colocar las palabras **ASC** O **DESC** (por defecto se toma ASC). Esas palabras significan en ascendente (de la A a la Z, de los números pequeños a los grandes) o en descendente (de la Z a la a, de los números grandes a los pequeños) respectivamente.

Sintaxis completa de SELECT:

```
SELECT * | {[DISTINCT] columna | expresión [[AS] alias], ... }
FROM nombre_tabla
[WHERE condición]
[ORDER BY columna1[{ASC|DESC}]][,columna2[{ASC|DESC}]]...;
```

```
-- Devuelve el nombre y los apellidos
-- de las personas que tienen teléfono, ordenados por
-- apellido1, luego por apellido2 y finalmente por nombre
SELECT nombre, apellido1, apellido2
FROM PERSONAS
WHERE telefono IS NOT NULL
ORDER BY apellido1, apellido2, nombre;
```

2.5. Alias

Los alias de SQL se utilizan para dar a una tabla, o a una columna de una tabla, un nombre temporal.

Los alias se utilizan a menudo para hacer que los nombres de las columnas sean más legibles.

Un alias solo existe mientras dure esa consulta.

Se crea un alias con la **AS** palabra clave.

```
-- Consulta con 3 columnas
SELECT nombre, precio, precio*1.16 FROM ARTICULOS;


-- Ponemos un alias a la tercera columna.
-- Las comillas dobles en el alias hacen que se respeten mayúsculas y minúsculas,
-- de otro modo siempre aparece en mayúsculas
SELECT nombre, precio, precio*1.16 AS "Precio + IVA" FROM ARTICULOS;
```