Introducción a SQL

Luis Valencia Cabrera (lvalencia@us.es),
David Orellana Martín (dorellana@us.es)

Research Group on Natural Computing
Departamento de Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla

23-10-2023, Bases de Datos

Índice

1. Introducción
2. Data Definition Language
3. Data Manipulation Language
4. Select
5. Group by
6. Joins
7. Otros
8. Bibliografía

Índice

1 Introducción
2 Data Definition Language
3 Data Manipulation Language
4 Select
5 Group by
6 Joins
7 Otros
8 Bibliografía

SQL

- SQL (Structured Query Language) es el **lenguaje estándar** de las bases de datos relacionales. Es un lenguaje declarativo que permite especificar diversos tipos de operaciones sobre estas.
- Es capaz de conjugar las operaciones del **álgebra** y el cálculo **relacional** con operadores adicionales, y *definir* así *consultas* para *recuperar* o *modificar* información de bases de datos, así como hacer cambios en ellas.
- Pero no sólo incluye consulta. SQL está compuesto por comandos, cláusulas, operadores y funciones de agregado. En conjunto, disponemos de instrucciones para **definir** (*crear* y *modificar* el esquema), **mantener** (*insertar*, *actualizar*, *eliminar*) y **consultar** BBDD relacionales.

Origen

- 1970: E.F. Codd (IBM) presenta el modelo relacional.
- Junto al modelo teórico, propuso el lenguaje DSL/Alpha.
- IBM creó versión simplificada: SQUARE.
- Más tade lanzaron SEQUEL[1], mejora de SQUARE.
- Finalmente, SEQUEL (Structured English Query Language) se renombró como SQL.
- Desde 1986 se han ido sucediendo distintas versiones del estándar ANSI/ISO SQL, ampliamente conocido como *Structured Query Language*.

[1] Donald D. Chamberlin and Raymond F. Boyce. 1974. **SEQUEL: A structured English query language**. In *Proceedings of the 1974 ACM SIGFIDET (now SIGMOD) workshop on Data description, access and control (SIGFIDET '74)*. ACM, New York, 249–264.

SGBD relacionales usando SQL

Sistemas comerciales de Gestión:

- Oracle Database de Oracle Corporation
- SQL Server de Microsoft
- DB2 Universal Database de IBM
- Sybase Adaptive Server de Sybase

SGBD relacionales usando SQL

Sistemas comerciales de Gestión:

- Oracle Database de Oracle Corporation
- SQL Server de Microsoft
- DB2 Universal Database de IBM
- Sybase Adaptive Server de Sybase
- ¿MS Access?

SGBD relacionales usando SQL

Sistemas comerciales de Gestión:

- Oracle Database de Oracle Corporation
- SQL Server de Microsoft
- DB2 Universal Database de IBM
- Sybase Adaptive Server de Sybase
- ¿MS Access? Cubre gran parte de la funcionalidad del núcleo del estándar, pero con variaciones en la sintaxis de algunos aspectos, y quedando otras partes del estándar no cubiertas.

SGBD relacionales usando SQL

Sistemas comerciales de Gestión:

- Oracle Database de Oracle Corporation
- SQL Server de Microsoft
- DB2 Universal Database de IBM
- Sybase Adaptive Server de Sybase
- ¿MS Access? Cubre gran parte de la funcionalidad del núcleo del estándar, pero con variaciones en la sintaxis de algunos aspectos, y quedando otras partes del estándar no cubiertas.

Sistemas de código abierto:

- *Mysql* $\rightarrow$ MariaDB
- PostgreSQL
- SQLite

Índice

1 Introducción
2 Data Definition Language
3 Data Manipulation Language
4 Select
5 Group by
6 Joins
7 Otros
8 Bibliografía

Comandos

Los tipos de comandos en SQL se agrupan en dos categorías o sub-lenguajes:

- **DDL** (Data Definition Language): permite definir el esquema de bases de datos, creando relaciones (tablas), campos e índices, o modificando las definiciones existentes.
- **DML** (Data Maniplation Language): permiten generar consultas para ordenar, filtrar y extraer datos de la base de datos, así como insertar, modificar y eliminar registros de las tablas.

Además está **DCL**, con comandos de control y seguridad de datos, que gobiernan los privilegios de los usuarios, los controles de acceso, ...

Un ejemplo de DDL: creación de esquemas

Palabra reservada CREATE

- El estándar SQL no dispone exactamente de sentencia de creación de base de datos como tal.
- En su lugar, dispone de una sentencia de creación de esquemas: `CREATE SCHEMA`. Con la creación de esquemas podemos agrupar un conjunto de elementos de la base de datos que son propiedad de un usuario.
- Conceptualmente, podemos entender que la BD contendrá tanto la definición de los datos como los registros, logs... El **esquema** únicamente establecerá la **definición de la base de datos**, plasmando el diseño lógico en el SGBD concreto especificando las relaciones (tablas) y otros objetos relacionados (como las vistas, por ejemplo.)

Creación de Bases de Datos

- No obstante, la mayoría de SGBD incorporan sentencias de creación de bases de datos como `CREATE DATABASE`. En el caso de MySQL, lo define como sinónimo de `CREATE SCHEMA`.
- Además de creación, dentro de DDL tenemos instrucciones para eliminación; por ejemplo, podemos borrar un esquema o una base de datos mediante `DROP SCHEMA`, e igualmente los SGBD suelen incorporar `DROP DATABASE`.

Juegos de caracteres

- La codificación de caracteres nos permite introducir símbolos propios del castellano que no corresponden a la codificación internacional (por ejemplo la *ñ*), aunque a nivel práctico lo desaconsejamos.
- En MySQL, podemos configurar el juego de caracteres para toda la base de datos; por ejemplo:

```
CREATE DATABASE libros character set utf8;
```

- También podemos hacer lo propio con un campo (columna), añadiendo el juego de caracteres a continuación del tipo:

```
descripcion VARCHAR(50) character set utf8
```

- Los juegos más comunes son latin1 y utf8

Comandos DDL

- `CREATE SCHEMA`/`DROP SCHEMA` (o bien `CREATE DATABASE`/`DROP DATABASE`), utilizados para crear/eliminar el esquema, la base de datos.
- `USE`, para activar la base de datos actual (`USE mibd;`)
- `CREATE TABLE`/`DROP TABLE`, utilizados para crear/eliminar tablas nuevas, respectivamente.
  - Por ejemplo `DROP TABLE 'ALUMNOS';`
- `ALTER TABLE` sirve para modificar tablas, agregando/eliminando campos o cambiando su definición
- Veremos a continuación algunos ejemplos
- Para una descripción detallada de las instrucciones SQL correspondientes se recomienda consultar este documento

Creación de una tabla

Definición general

```
CREATE TABLE tabla (
        definición de campos,
        claves y restricciones
);
```

Definición de campos

Tipos de datos

Definición:

```
nombre_campo tipo marcadores
```

Tipos:

- `CHAR(n)` Cadenas longitud fija hasta *n* caracteres.
- `VARCHAR(n)` Cadenas longitud variable hasta *n* caracteres.
- `INTEGER`, `BIGINT`, `SMALLINT`, . . . Enteros. . . .
- `REAL` Número real
- `DATE` Fechas. Están compuestas de: **`YEAR`**, **`MONTH`** y **`DAY`**.
- `TIME` Horas. Están compuestas de **`HOUR`**, **`MINUTE`** y **`SECOND`**.
- . . . y muchos más (varían según el SGBD).

Marcadores y restricciones

- `AUTO_INCREMENT` - Autonumérico, secuencial que va asignando el entero siguiente al máximo valor almacenado para el campo.
- `DEFAULT val` - Establece un valor por defecto al campo.
- `NOT NULL` - No puede contener valores nulos.
- `PRIMARY KEY` - El campo es la clave primaria no compuesta.
- `REFERENCES tabla [(campo)]` - Clave ajena simple.[2]
- `CHECK (cond.)` - El campo debe cumplir una condición.[3]

[2] Se ignora en MySQL, hay que usar la otra forma.
[3] Disponible dependiendo del SGBD y su versión

Claves compuestas y restricciones

- `PRIMARY KEY (campos)` - Clave primaria (compuesta o no).
- `FOREIGN KEY (campos) REFERENCES tabla [(campos)]` - Clave ajena (compuesta o no).
- `CHECK (cond.)` - El campo debe cumplir una condición.[4]

Tanto a las claves primarias y ajenas como a los chequeos de condición se les puede anteponer la partícula `CONSTRAINT nombre`, para nombrar la restricción.

[4] Disponible dependiendo del SGBD y su versión

Creación de una tabla

Ejemplo básico

```
CREATE TABLE PRODUCTOS (
  codigo_producto INTEGER AUTO_INCREMENT,
  nombre_producto VARCHAR(20) UNIQUE NOT NULL,
  tipo VARCHAR(20),
  descripcion VARCHAR(50),
  precio REAL DEFAULT 1.0,
  fabrica INTEGER DEFAULT NULL,
  PRIMARY KEY (codigo_producto),
  FOREIGN KEY (fabrica) REFERENCES FABRICAS(id_fabrica)
        ON DELETE RESTRICT ON UPDATE CASCADE,
  CONSTRAINT precio CHECK (precio>5)
);
```

Comandos ALTER

- Uso de ADD, MODIFY y DROP para campos.

```
ALTER TABLE t1 ADD proveedor VARCHAR(50);
ALTER TABLE t1 MODIFY tipo INTEGER;
ALTER TABLE t1 DROP descripcion;
```

- Adición de restricciones, claves, etc.[5]

```
ALTER TABLE t1 ADD CONSTRAINT FK1
        FOREIGN KEY (c1) REFERENCES t2;
```

[5] Más información en esta web.

Índice

1 Introducción
2 Data Definition Language
3 Data Manipulation Language
4 Select
5 Group by
6 Joins
7 Otros
8 Bibliografía

Comandos DML

SQL define cuatro sentencias de manipulación de datos principales:

- `INSERT`, para insertar registros en la base de datos
- `UPDATE` Encargado de modificar los valores de los campos indicados, en los registros que cumplan cierta condición
- `DELETE` Encargado de eliminar los registros de una tabla que cumplan una condición
- `SELECT` Encargado de consultar registros de la base de datos que satisfagan una condición determinada

Inserción de datos - INSERT

Podemos insertar valores en una tabla de grado n de dos formas:

- Sin especificar los nombres de las columnas, sólo los valores:

```
INSERT INTO nombre_tabla
VALUES (v1,v2,..., vn);
```

- Especificando el nombre de los $k \leq n$ atributos para los que insertamos valores

```
INSERT INTO nombre_tabla (col1, col2, ..., colk)
VALUES (v1, v2, ..., vk);
```

Inserción de datos - INSERT

Ejemplos

- Suponiendo que los campos son **codigo, nombre, tipo, precio** y **procedencia**, podemos hacer:

```
INSERT INTO productos
VALUES (1, 'Peras', 3, 0.75, 'Utrera');
```

  o bien...

```
INSERT INTO productos (codigo, precio)
VALUES (5, 0.60);
```

- En el segundo ejemplo, los valores no indicados explícitamente toman el valor `NULL`, o bien el valor por defecto si lo hemos definido mediante `DEFAULT`.
- Deben respetarse las restricciones establecidas (de entidad, referencial, condición de requerido...)

Actualización de datos - UPDATE

Podemos modificar valores en una tabla mediante:

```
UPDATE Nombre_tabla
SET col_1 = v_1 , col_2 = v_2 , . . . , col_n = v_n
WHERE condiciones ;
```

Esto actualizará a los valores $v_1, ..., v_n$ las columnas $col_1, ..., col_n$, respectivamente, para todos los registros que cumplan las condiciones dadas.

Eliminación de registros - DELETE

Podemos eliminar registros de una tabla mediante:

```
DELETE FROM Nombre_tabla
WHERE condiciones;
```

Esto eliminará los registros de la tabla que cumplan las condiciones dadas.

Modificadores

- El uso de `UPDATE`, `DELETE` y `SELECT` necesita el modificador `WHERE` que acabamos de ver, y que establece las condiciones que los registros a seleccionar deben cumplir.
- La partícula `FROM` de las operaciones de tipo `DELETE` establece la tabla de la que borrar los registros, y en operaciones de tipo `SELECT` indica la tabla/s de la/s que seleccionar los registros.
- Otros modificadores (cláusulas) nos permiten generar criterios para definir los datos a selecionar en operaciones de tipo `SELECT`:
  - `GROUP BY`: criterio para agrupar los registros seleccionados.
  - `HAVING`: establece condiciones sobre datos calculados para los grupos generados por `GROUP BY`.
  - `ORDER BY`: ordena los registros seleccionados según los campos indicados y su orden ascendente o descendente.

Operadores lógicos

Estos operadores permiten formar condiciones más complejas a partir de otras pasadas como operandos. Así, dadas tres condiciones (expresiones lógicas) *a*, *b* y *c*:

- *a* `AND` *b*: conjunción, evalúa las condiciones *a* y *b* y devuelve *verdad* si ambas son ciertas.
- *a* `OR` *b*: disyunción lógica, evalúa *a* y *b* y devuelve *verdad* si alguna de las dos es cierta.
- `NOT` *c*: negación lógica, devuelve el valor lógico contrario a la evaluación de *c*.

Operadores de comparación

Para cada condición simple, antes de agruparlas mediante los operadores lógicos anteriores, podemos emplear operadores como:

- $=$ Igual que, $<>$ Distinto de
- $<$ Menor que, $>$ Mayor que
- $<=$ Menor o igual que ($>=$ Mayor o igual que)
- `x BETWEEN a AND b` Devuelve los registros en los que el valor de campo *a* esté entre *a* y *b*, ambos inclusive.
- *LIKE* para comparar cadenas que cumplan ciertos patrones
- *IN* para chequear si un valor pertenece a una lista

Funciones de agregado

Las funciones de agregado se usan dentro de una cláusula `SELECT` en grupos de registros (*un grupo por cada combinación de valores de* `GROUP BY`*, o un único grupo global si no hay* `GROUP BY`) para devolver un valor para cada posible grupo.

- `AVG` calcula el promedio de valores de un campo determinado
- `COUNT` devuelve el número de registros del grupo
- `SUM` devuelve la suma de valores de un campo determinado para el grupo
- `MAX` devuelve el valor más alto de un campo especificado (`MIN` el más bajo)

Índice

1 Introducción
2 Data Definition Language
3 Data Manipulation Language
4 Select
5 Group by
6 Joins
7 Otros
8 Bibliografía

Consultas de Selección

Las consultas de selección se utilizan para indicar al motor de datos que devuelva información de las bases de datos. Esta información es devuelta en forma de conjunto de registros que se pueden almacenar en una nueva tabla.

- `SELECT Campos FROM Tabla;`
  Donde campos es la lista de campos que se deseen recuperar y tabla es el origen de los mismos.
- Por ejemplo:
  `SELECT Nombre, Telefono FROM Clientes;`

Consultas de Selección

- Se puede especificar el orden en que se desean recuperar los registros de las tablas mediante la cláusula `ORDER BY`

```
SELECT CodigoPostal, Nombre, Telefono
FROM Clientes
ORDER BY Nombre;
```

- Se puede indicar si el orden de los registros es ascendente, `ASC` (valor por defecto) o descendente, `DESC`.

```
SELECT CodigoPostal, Nombre, Telefono
FROM Clientes
ORDER BY CodigoPostal DESC;
```

Consultas de Selección

Otros modificadores aportan diversas acciones para seleccionar campos o para limitar los registros obtenidos:

- `*` Devuelve todos los campos de la tabla.
- `LIMIT n` Limita el resultado devuelto a los primeros *n* registros obtenidos por la tabla. Va al final de toda la consulta `SELECT`.
- `DISTINCT` Omite repeticiones de registros cuyos campos seleccionados coincidan totalmente. Va tras la palabra `SELECT`.

Consultas de Selección

Ejemplos

```sql
SELECT * FROM Empleados;

SELECT Nombre, Apellido
FROM Estudiantes
ORDER BY Nota LIMIT 25;

SELECT DISTINCT Apellido
FROM Empleados;
```

Alias (AS)

- En determinadas circunstancias es necesario asignar un nombre nuevo a alguna columna determinada de un conjunto de registros devuelto por una consulta.
- Usamos la palabra reservada `AS`, que se encarga de asignar el nombre que deseamos a la columna deseada.
- Por ejemplo,

```
SELECT DISTINCT Apellido AS Empleado
FROM Empleados;
```

Estableciendo condciones para la consulta

- Operadores lógicos

```
SELECT *
FROM Empleados
WHERE (Sueldo > 100 AND Sueldo < 500) OR
(Provincia='Madrid' AND Estado='Casado');
```

- Operador **BETWEEN**:

```
SELECT *
FROM Pedidos
WHERE CodPostal BETWEEN 28000 AND 28999;
```

- Combinado con **NOT**:

```
SELECT *
FROM Pedidos
WHERE cp NOT BETWEEN 28000 AND 28999;
```

Operador LIKE

Permite comparar una cadena con un patrón. Su sintaxis es:
`expresión LIKE modelo`
y admite comodines (algunos dependen del SGBD[6]):

- `_` (guión bajo) sirve para cualquier carácter unitario
- `%` comodín para cero o más caracteres
- Una clase **`[...]`** puede tomar cualquier valor entre los corchetes. Por ejemplo, **`[abc]`** indica que en esa posición pueden ir **`a`**, **`b`** o **`c`**. Para un rango de valores, usamos el guión medio: **`[a-z]`** indica cualquier letra y **`[0-9]`** dígito.
- `^` acepta cualquier carácter menos los indicados. Por ejemplo, `[^oa]` acepta cualquier carácter menos o y a.

Como vemos en este enlace, Access emplea símbolos distintos.

[6] En `MySQL` solo se aceptan los 2 primeros, para patrones complejos empleamos `REGEXP`

Operador IN

Este operador devuelve los registros cuyo campo indicado coincide con alguno de los dados en una lista. Su sintaxis es:
`expresión [NOT] IN (valor1, valor2, ...)`

- Por ejemplo:

```
SELECT *
FROM Pedidos
WHERE Provincia IN ('Madrid', 'Cádiz', 'Sevilla');
```

Cuantificador EXISTS

El cuantificador existencial en SQL es `EXISTS`. La expresión sería:

```
WHERE EXISTS (SELECT ... FROM ...)
```

Esta condición se cumple (es verdadera) si, y sólo si, el resultado de evaluar la consulta especificada por

```
SELECT ... FROM ...
```

devuelve algún registro (*i.e.*, no es el conjunto vacío). Esta expresión interna al paréntesis se denomina *subconsulta*.

Ejemplos con WHERE

```sql
SELECT Apellidos, Salario
FROM Empleados
WHERE Salario > 21000;
```

```sql
SELECT Apellidos, Nombre
FROM Empleados
WHERE Apellidos = 'King';
```

```sql
SELECT Apellidos, Nombre
FROM Empleados
WHERE Apellidos Like 'S%';
```

```sql
SELECT Apellidos, Salario
FROM Empleados
WHERE Salario Between 200 and 300;
```

```sql
SELECT Apellidos, Nombre, Ciudad
FROM Empleados
WHERE Ciudad IN ('Sevilla', 'Los Ángeles', 'Barcelona')
   OR Nombre NOT LIKE  '%B%';
```

Índice

1 Introducción
2 Data Definition Language
3 Data Manipulation Language
4 Select
5 Group by
6 Joins
7 Otros
8 Bibliografía

Agrupamiento - GROUP BY

`GROUP BY` *lista_campos*
combina los registros con valores idénticos para *lista_campos* en un grupo, para el que se mostrará un único registro. Si se incluye una función SQL agregada (Sum, Count...) en el SELECT, se obtiene un valor del cálculo para cada registro (grupo). Su sintaxis es:

```
SELECT expresiones
FROM tabla
[WHERE lista_criterios]
GROUP BY lista_campos;
```

Por ejemplo:

```
SELECT Id_Familia, Sum(Stock)
FROM Productos
GROUP BY Id_Familia;
```

Agrupamiento - HAVING

- Una vez `GROUP BY` ha combinado los registros, `HAVING` filtra los registros agrupados que satisfagan las condiciones del `HAVING`.
- Es similar a `WHERE`, determina qué registros se seleccionan. Pero a diferencia de éste (que afecta a los registros originales), `HAVING` filtra los registros resultantes de la agrupación, según los resultados de los campos calculados.
- Es decir, una vez se han agrupado los registros utilizando `GROUP BY` y se han hecho los cálculos agregados, `HAVING` determina cuáles de los nuevos registros se van a mostrar.

```
SELECT Id_Familia, SUM(Stock)
FROM Productos
WHERE NombreProducto Like 'BOS%'
GROUP BY Id_Familia
HAVING SUM(Stock) > 100
```

Agrupamiento - AVG

Calcula, para cada grupo, la media aritmética de los valores del campo especificado para los distintos registros del grupo. Su sintaxis es **`AVG(expr)`**, con **`expr`** siendo el *campo* (o expresión derivada de campos) que contiene los datos numéricos a los que calcular la media.

```
SELECT AVG(Gastos) AS Promedio
FROM Pedidos
WHERE Gastos > 100;
```

Agrupamiento - Count

- Calcula el número de registros devueltos por una consulta, para cada grupo.
  Su sintaxis es: `Count(expr)`, con `expr` el nombre del campo que desea contar.
- Los operandos de `expr` pueden incluir el nombre de un campo (del tipo que sea, incluyendo texto), una constante o una función
- No cuenta los registros que tienen campos `NULL`, a menos que `expr` sea el carácter `*`:

```
SELECT Count(*) AS Total
FROM Pedidos;
```

COUNT

Algunos ejemplos adicionales

Con `COUNT` podemos contar el número de ocurrencias de cierto campo (valores no nulos). Por ejemplo, para contar el número de empleados con la ciudad informada:

```
SELECT COUNT(ciudad)
FROM EMPLEADOS
```

Ahora bien, si queremos saber el número de ciudades (sin repeticiones), incluimos `DISTINCT`:

```
SELECT COUNT(DISTINCT ciudad)
FROM EMPLEADOS
```

Agrupamiento - Max, Min

`Min(expr)` y `Max(expr)` devuelven, respectivamente, el mínimo o el máximo de un conjunto de valores contenidos en un campo de una consulta. Así, **`expr`** es el campo o expresión (incluyendo constantes, llamadas a funciones, etc.) sobre el que realizar el cálculo.

```sql
SELECT Min(gasto) AS gmin
FROM Pedidos
WHERE pais = 'España';
```

```sql
SELECT Max(gasto) AS gmax
FROM Pedidos
WHERE pais = 'España';
```

Agrupamiento - Sum

`Sum(expr)` devuelve la suma de valores para el campo o expresión especificada. Los operandos de `expr` pueden incluir nombres de campos de tablas, constantes, llamadas a funciones y operadores.

```
SELECT Sum(PrecioUnidad * Cantidad) AS Total
FROM DetallePedido;
```

Índice

1 Introducción
2 Data Definition Language
3 Data Manipulation Language
4 Select
5 Group by
6 Joins
7 Otros
8 Bibliografía

Producto cartesiano

El producto cartesiano entre dos relaciones (tablas) se realiza, en su forma más elemental, mediante la separación por comas. Así, sea:

- **T1** con 3 columnas y 5 filas.
- **T2** con 4 columnas y 7 filas.

Podemos devolver el producto cartesiano mediante:

```
SELECT *
FROM T1, T2;
```

y éste resultado tendrá 7 (=3+4) columnas y 35 (=5 * 7) filas.

Joins

En general, denominamos JOIN al producto cartesiano filtrado por ciertas condiciones que restrinjan el resultado a los registros que cumplan ciertas condiciones (usando valores de ciertas columnas de las tablas involucradas). Si hay campos con el mismo nombre, los prefijamos.

```
SELECT * FROM TABLA1, TABLA2
WHERE TABLA1.ID=7 AND TABLA2.ID=9;
```

Podemos usar alias para no tener que escribir el nombre completo de la tabla:

```
SELECT * FROM TABLA1 a, TABLA2 b
WHERE a.ID=7 AND b.ID=9;
```

Joins

Podemos incluir restricciones de igualdad que afecten a columnas de distintas tablas.

```
SELECT * FROM TABLA1, TABLA2
WHERE TABLA1.ID=TABLA2.ID;
```

Para evitar la columna **ID** duplicada, lo más común es usar `INNER JOIN`, con `ON`:

```
SELECT a.NOMBRE, b.DEPTO
FROM TABLA1 a INNER JOIN TABLA2 b ON a.ID=b.ID;
```

o bien `USING`[7]:

```
SELECT a.NOMBRE, b.DEPTO
FROM TABLA1 a INNER JOIN TABLA2 b USING(ID);
```

[7] En según que SGBD y versión, también podremos indicar `NATURAL JOIN`, y omitir `USING`: `SELECT T1.NOMB, T2.DEPT FROM T1 NATURAL JOIN T2`

Joins

Reflexividad

Para hacer JOIN a la propia tabla, lo más común es renombrar dos veces la tabla:

```
SELECT a.NOMBRE, b.NOMBRE
FROM EMPLEADOS a INNER JOIN EMPLEADOS b
ON a.ID = b.ID_SUPERVISOR;
```

Índice

1 Introducción
2 Data Definition Language
3 Data Manipulation Language
4 Select
5 Group by
6 Joins
7 Otros
8 Bibliografía

Subconsultas

Podemos tener una consulta contenida dentro de otra. Tiene carácter temporal y se resuelve antes de la consulta principal.

```
SELECT producto, precio
FROM tabla
WHERE precio = (SELECT MAX(precio) FROM tabla);

SELECT producto, precio
FROM tabla
WHERE precio <> (SELECT MAX(precio) FROM tabla);

SELECT producto, precio
FROM tabla
WHERE producto NOT IN (SELECT producto FROM tabla2);
```

CASE - Primer uso

Formato general:

```
CASE
   WHEN cond1 THEN expr1
  [WHEN cond2 THEN expr2]
  [ELSE expr3]
END
```

Ejemplo:

```
SELECT Nombre, CASE WHEN Turno = 'Mañana' THEN 1
                    ELSE 2 END as Nivel
FROM Alumnos;
```

Devuelve el campo 'Nombre', y una segunda columna 'Nivel' formada a partir de la Columna 'Turno' sustituyendo el valor 'Mañana' por 1 y cualquier otro valor por 2.

ANY

Formato general:

```
Operador ANY Subconsulta
```

`ANY` se usa con un operador de comparación, y el resultado de la misma se evalúa como *Verdadero* si la comparación resulta cierta **para algún valor** devuelto por la subconsulta.

Por ejemplo:

```
SELECT s1
FROM t1
WHERE s1 > ANY (SELECT s1 FROM t2);
```

COALESCE

Devuelve el primer valor conocido (no NULL) de una lista.

```
SELECT Nombre,
       COALESCE(FijoCasa, FijoTrabajo, Móvil) Telefono
FROM Clientes;
```

Devuelve el nombre y un teléfono de cada cliente: si se conoce, el fijo de casa; si no el fijo del trabajo y si no el móvil, en ese orden.

Creación de Tablas/Vistas a partir de consultas

Podemos crear tablas a partir de subconsultas:

```
CREATE TABLE TABLA1 [AS]
  SELECT Titulo, Autor
  FROM Libros;
```

Se guarda así la nueva tabla, incluyendo su esquema y datos.

De forma similar, podemos definir una vista:

```
CREATE VIEW VISTA1 AS
  SELECT Titulo, Autor
FROM Libros;
```

De forma que luego podremos hacer consultar sobre `TABLA1` como si fuera una tabla, pero no se almacena.

Índice

1 Introducción
2 Data Definition Language
3 Data Manipulation Language
4 Select
5 Group by
6 Joins
7 Otros
8 Bibliografía

Bibliografía I

Mercedes Marqués
Apuntes de Bases de Datos.
Universidad Jaume I en Castellón (2011)
ISBN: 978-84-693-0146-3

Carme Martín Escofet
El lenguaje SQL.
UOC (2007)
P06/M2109/02149

Alan Beaulieu
Aprende SQL.
Anaya Multimedia - O'Reilly, Segunda Ed. (2009)
ISBN: 978-84-4152-637-2

Bibliografía II


Luis Grau Fernandez, Ignacio López Rodríguez
Problemas de Bases de Datos.
Sanz Y Torres, S.L. - 3ª Edición (2006)
ISBN: 978-84-960-9469-7


Dolores Cuadra, Elena Castro, Ana Mª Iglesias, Paloma Martínez, Fco. Javier Calle, César de Pablo, Harith Al-Jumaily, Lourdes Moreno, Sonia García Manzano, José Luis Martínez, Jesica Rivero, Isabel Segura
Desarrollo De Bases De Datos: Casos Prácticos Desde El Análisis A La Implementación.
RA-MA Editorial - 2ª Edición (2013)
ISBN: 978-84-996-4124-9

Bibliografía III


Adoracion De Miguel Castaño, Paloma Martínez Fernández, Elena Castro Galán
Diseño De Bases De Datos: Problemas resueltos.
RA-MA Editorial - 1ª Edición (2001)
ISBN: 978-97-015-0687-5


SQL Zoo - `https://sqlzoo.net/`


SQL Tutorial - `https://www.w3schools.com/sql`


SQLite Online - `https://sqliteonline.com/`


SQLite Fiddle - `http://sqlfiddle.com/`