apuntes_si1_parcial2_sql.pdf

CarlosBynd_480854

Sistemas Informaticos I

3º Grado en Ingeniería Informática

Escuela Politécnica Superior
Universidad Autónoma de Madrid

Sistemas Informáticos 1
Bases de Datos Distribuidas

Parte 2.- SQL (Structured Query Language)

Podemos dividir el lenguaje SQL en dos sublenguajes:

- DDL: Lenguaje de definición de datos:
  - Definición de esquemas de relación.
  - Borrado de relaciones.
  - Creación de índices.
  - Modificación de esquemas de relación.
  - Órdenes para la definición de vistas.
  - Órdenes para especificar las restricciones de integridad que deben cumplir los datos almacenados en la base de datos.
  - Órdenes para especificar derechos de acceso para las relaciones y vistas.
- DML: Lenguaje interactivo de manipulación de datos:
  - Incluye un lenguaje de consultas, basado en el álgebra relacional (y en el cálculo de tuplas).
  - Incluye órdenes para insertar, borrar y modificar tuplas de la base de datos.

Apartado 1.- SQL, DDL: Lenguaje de definición de datos.

Primero vamos a hablar de los tipos de datos que existen en SQL:

- char(n): cadena de longitud fija. La longitud es n caracteres.
- varchar(n): cadena de longitud variable. La longitud máxima es n caracteres.
- int/integer: entero.
- smallint: entero corto.
- numeric(p,d): número en formato de coma fija, con precisión de p dígitos, con d dígitos a la derecha de la coma decimal.
- real, double precision: número en coma flotante y número en coma flotante con doble precisión.
- float(n): número en coma flotante con una precisión no menor de n dígitos.

1.1.- Creación y destrucción de Tablas

Para crear una tabla se utiliza la orden: CREATE, cuya sintaxis es la siguiente:

```
CREATE TABLE tabla_ejemplo (
        columna1 char(20),
        columna2 integer);
```

En cuanto al borrado de tablas, se utiliza el comando DROP, con la siguiente sintaxis:

```
DROP TABLE tabla_ejemplo;
```

1.2.- Restricciones

En SQL existen una serie de restricciones que pueden añadirse a una tabla:

- Check: restricción arbitraria.
- Not-Null: el atributo no acepta valores nulos.
- Unique: el atributo no acepta valores repetidos.
- Primary Key: el atributo es clave primaria (no acepta mi valores nulos ni valores repetidos)
- Foreign Key: el atributo es clave extranjera.

A continuación se muestra una serie de ejemplos de uso de cada una de las restricciones mencionadas anteriormente:

Ejemplo CHECK: permite especificar que los valores de una columna deben satisfacer una expresión. Por ejemplo, ser positivos:

```
CREATE TABLE productos (
        producto_no integer,
        nombre text,
        precio numeric(10,2) CHECK (precio>0));

CREATE TABLE productos (
        producto_no integer,
        nombre text,
        precio numeric(10,2) CONSTRAINT
                precio_positivo CHECK (precio>0));
```

Ejemplo NOT NULL: indica que el atributo en cuestión, no puede ser nulo:

```
CREATE TABLE productos (
        producto_no integer NOT NULL,
        nombre text NOT NULL,
        precio numeric(10,2));
```

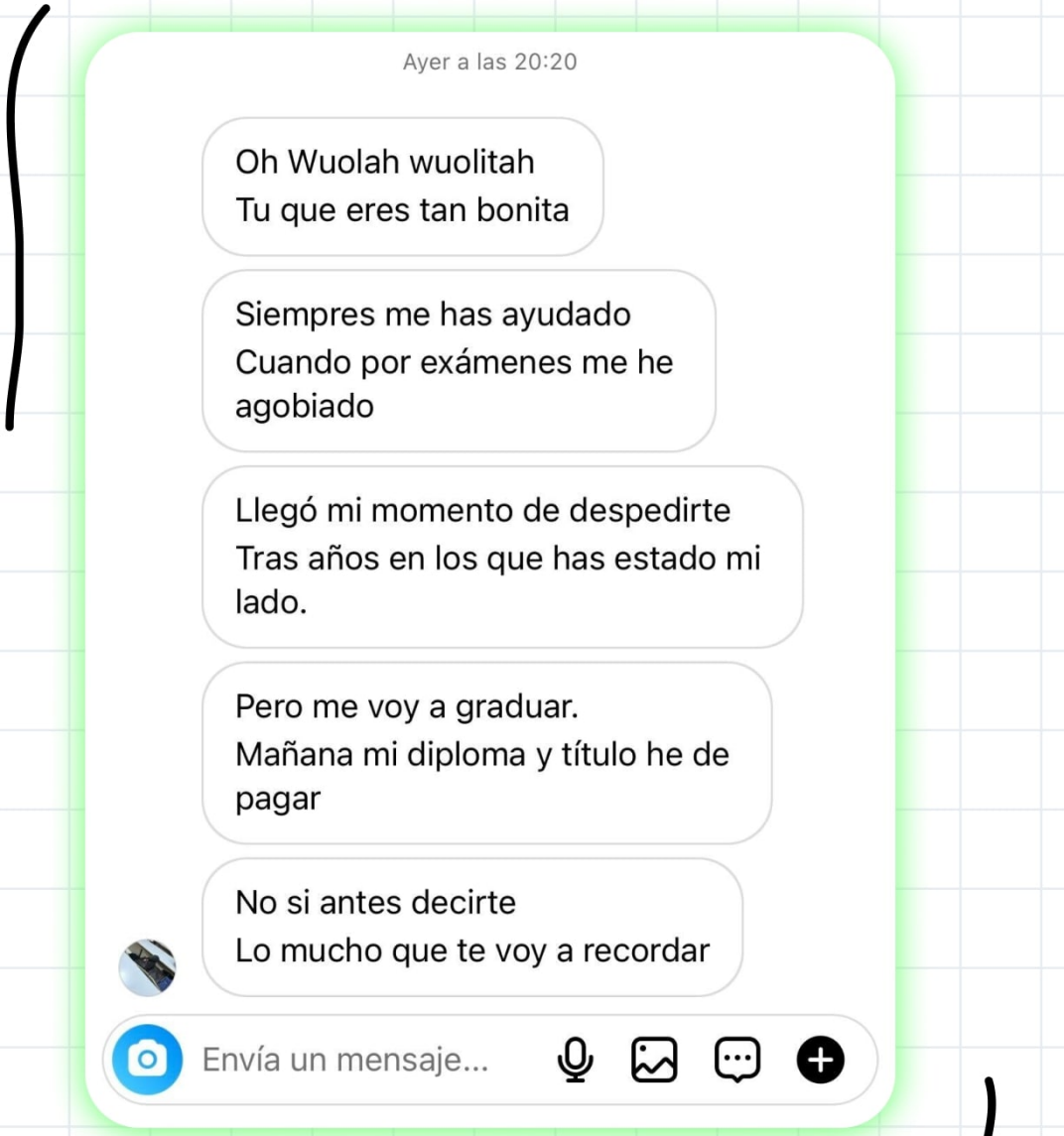
Tambien puede ocurrir que existan varias restricciones referidas al mismo atributo. El orden de ellas no importa:

```
CREATE TABLE productos (
        producto_no integer NOT NULL,
        nombre text NOT NULL,
        precio numeric(10,2) NOT NULL CHECK (precio>0));
```

Ejemplo UNIQUE: asegura que un determinado valor no aparece repetido en una columna.

Que no te escriban poemas de amor cuando terminen la carrera 🚩🚩🚩🚩🚩🚩

(a nosotros por suerte nos pasa)



WUOLAH

```
CREATE TABLE productos (
        producto_no integer UNIQUE,
        nombre text,
        precio numeric(10,2));
```

También puede ponerse como:

```
CREATE TABLE productos (
        producto_no integer,
        nombre text,
        precio numeric(10,2),
        UNIQUE(producto_no,nombre));
```

Ejemplo PRIMARY KEY: una clave primaria indica que el atributo no puede estar repetido y no puede ser nulo.

```
CREATE TABLE productos (
        producto_no integer PRIMARY KEY,
        nombre text,
        precio numeric(10,2));

CREATE TABLE ejemplo (
        a integer,
        b integer,
        c integer,
        PRIMARY KEY (a, c));
```

Ejemplo FOREIGN KEY: la restricción REFERENCES asegura que los valores de una determinada columna debe ser idénticos a los valores que aparecen en otra determinada columna que debe estar en otra tabla, tal y como se muestra a continuación:

```
CREATE TABLE productos (
        producto_no integer PRIMARY KEY,
        nombre text,
        precio numeric(10,2));

CREATE TABLE pedidos (
        pedido_no integer PRIMARY KEY,
        producto_no integer REFERENCES productos(producto_no),
        cantidad integer);
```

De esta manera, no será posible crear pedidos sobre productos que no existan.

1.3.- Modificar tablas

Para eliminar la tabla 'nombre_de_la_tabla' y toda la información relacionada con ella de la base de datos, usamos la orden:

```
DROP TABLE nombre_de_la_tabla;
```

Si queremos añadir atributos a una relación (es decir, añadir una columna a una tabla), utilizamos la orden ALTER TABLE:

```
ALTER TABLE nombre_de_la_tabla ADD COLUMN columna integer;
```

El valor inicial de los atributos es nulo (a no ser que se especifique un valor por defecto). Si queremos borrar una columna en concreto de una tabla, utilizamos:

```
ALTER TABLE nombre_de_la_tabla DROP columna;
```

También podemos querer añadir o modificar una restricción. Aquí se muestra unos cuantos ejemplos:

```
ALTER TABLE productos ADD CHECK (nombre <> '');
ALTER TABLE productos ADD CONSTRAINT some_name UNIQUE (producto_no);
ALTER TABLE productos ADD FOREIGN KEY (producto_group_id) REFERENCES grupo_productos;
```

```
ALTER TABLE productos ALTER COLUMN producto_no SET NOT NULL;
ALTER TABLE productos ALTER COLUMN precio SET DEFAULT 7.77;
ALTER TABLE productos RENAME COLUMN producto_no TO product_number;
```

Es posible que haya herencia entre tablas, es decir, una tabla puede estar basada en otra. La sintaxis es la siguiente:

```
CREATE TABLE ciudades (
    nombre char(30),
    poblacion float,
    altura int);

CREATE TABLE capitales (
    pais char(30)
) INHERITS (ciudades);
```

De esta manera, cada fila de la tabla 'capitales' contiene todos los campos de la tabla 'ciudades'.

1.4.- Modificaciones de la Base de Datos

La orden que ofrece SQL para poder modificar la Base de Datos es INSERT. Su sintaxis es:

```
INSERT INTO R(A1, A2, …, An) VALUES (V1, V2, …, Vn);
```

Ejemplo de inserción:

```
INSERT INTO
        pelicula (id, titulo, agno, puntuacion, votos)
VALUES (1, 'Star Wars', 1977, 8.9, 14182);
```

Los valores a insertar pueden provenir de una consulta.
Se puede omitir la lista de los atributos de la tabla destino si se suministran todos, en el mismo orden en que se definieron, durante lla inserción.

```
INSERT INTO pelicula
VALUES (1, 'Star Wars', 1977, 8.9, 14182);
```

No es imprescindible proveer valores para todos los atributos:

- La tupla creada tendrá el valor por defecto o valor nulo (NULL) para todos aquellos atributos a los que no se les asigne un valor.
- De todas formas, es siempre recomendable usar DEFAULT, NULL cuando no se suministre un valor.

Apartado 2.- SQL, DML: Lenguaje interactivo de manipulación de datos.

Para el desarrollo de los ejemplos de este apartado, se van a utilizar las siguientes tablas con estos atributos:

reparto	**actor**	**pelicula**
actor_id	id	id
pelicula_id	nombre	titulo
ord		agno
		puntuacion
		votos

2.1.- Consultas sencillas en SQL

La sintaxis de una consulta SQL sencilla es la siguiente:

```
SELECT atributos FROM tabla WHERE condición
```

La anterior consulta lista los atributos pertenecientes a la tabla 'tabla' que satisfacen la condición 'condición'.
SQL admite que existan tuplas duplicadas.
Para poder crear condiciones, se pueden utilizar las expresiones algebraicas mayor y menor que '>', '<', la expresión igual '=', la expresión distinto que '<>' y otras más.
También se puede utilizar la instrucción LIKE, que permite el uso de patrones ('wildcards'), donde destacan los comodines '_' (un caracter) y '%' (varios caracteres).
Para encontrar todas las películas (y su puntuación) que empiezan por la cadena 'Star':

```
SELECT titulo, puntuacion
FROM pelicula
WHERE titulo LIKE 'Star%';
```

Para encontrar todas las películas con 's' en su título, deberíamos ejecutar la siguiente consulta:

```
SELECT titulo
FROM pelicula
WHERE titulo LIKE '%''s%';
```

Otra sentencia, que incluye todas las funcionalidades de LIKE, y además, permite utilizar un subconjunto de expresiones regulares es [NOT] SIMILAR TO.

- Una de las dos alternativas: |.
- Repetición de lo anterior cero o más veces: *.
- Repetición de lo anterior una o más veces: +.
- Agrupación para crear un único objeto: ().
- Especificación de una clase: [...].

Un ejemplo de consulta utilizando un SIMILAR es:

```
SELECT titulo, puntuacion
FROM pelicula
WHERE titulo NOT SIMILAR TO
        '%(S|s)tar [A-z]rek%' AND titulo SIMILAR TO '%Star%';
```

2.2.- Productos Cartesianos

Gran parte de la potencia de las bases relacionales se basa en la posibilidad de combinar dos (o más) relaciones.

El producto cartesiano de dos relaciones se consigue enumerando cada relación en la orden FROM.

Por ejemplo, para obtener el reparto de 'Pulp Fiction' (su identificador como película es 2) ejecutamos la siguiente consulta:

```
SELECT nombre
FROM actor, reparto
WHERE pelicula_id=2 AND actor_id=id;
```

2.3.- Producto Natural

El producto natural se realiza mediante la orden NATURAL JOIN.

Es imprescindible que los campos tengan el mismo nombre en las tablas con las que se realiza el producto natural.

Suponiendo que el campo clave de actor es 'actor_id', la anterior consulta podría hacerse como:

```
SELECT nombre
FROM actor NATURAL JOIN reparto
WHERE pelicula_id=2;
```

El JOIN toma dos relaciones y devuele otra relación, que es sobre la que se ejecuta la consulta.
La condición del JOIN define qué tuplas de las dos relaciones se corresponden, y cuáles serán los atributos que estarán en la relación resultante. La condición del JOIN puede ser:

- NATURAL JOIN.
- JOIN ON [...].
- USING (A1, A2, …, An).

El tipo de JOIN define cómo las tuplas de una relación que no tiene correspondencia en la otra relación (según la condición de join) son tratadas. El tipo puede ser:

- INNER JOIN.
- LEFT OUTER JOIN.
- RIGHT OUTER JOIN.
- FULL OUTER JOIN.

2.4.- Combinando Consultas

Para combinar consultas, pueden usarse las órdenes UNION, INTERSECT y EXCEPT. Estos operadores eliminan los duplicados que pueda haber (en caso de no querer eliminar los duplicados, añadimos un ALL detrás del operador).
Obviamente, para poder combinar consultas, las subconsultas deben ser compatibles.
Como ejemplo, vamos a devolver los actores comunes a las películas 'Star Trek IV' y 'Star Trek V':

```
(SELECT nombre FROM pelicula, actor, reparto WHERE titulo LIKE 'Star Trek V:%' AND pelicula_id=pelicula.id AND actor_id=actor.id) INTERSECT (SELECT nombre FROM pelicula, actor, reparto WHERE titulo LIKE 'Star Trek IV:%' AND pelicula_id=pelicula.id AND actor_id=actor.id);
```

Apartado 3.- Subconsultas en SQL.

Hasta ahora, las condiciones que poníamos en WHERE, tan solo involucraban valores escalares, pero ahora vamos a ver que puede que aparezca un SELECT como parte de la condición descrita en WHERE.

- Esta subconsulta puede devolver un valor.
- Puede devolver una relación (una tabla) que será procesada valor por valor usando IN, EXISTS, ALL, ANY, BETWEEN...

Como ejemplo, vamos a ver cómo haríamos antes la consulta del reparto de 'Star Wars' (película con id = 1) y como podemos hacerla ahora:

```
SELECT nombre
FROM actor, reparto
WHERE pelicula_id=1 AND actor_id=actor.id;
```

En lugar de hacer un producto escalar, podemos hacer una subconsulta que nos devuelva los identificadores de los actores (de esta manera, inicialmente, usamos una sola tabla):

```
SELECT nombre
FROM actor
WHERE id =
        (SELECT DISTINCT actor_id
        FROM reparto
        WHERE pelicula_id=1);
```

Una forma más correcta de hacer la consulta anterior sería sustituir el '=' por el operador IN. A continuación se muestran una serie de condiciones que involucran relaciones, así como su significado.

- EXISTS R: TRUE si 'R' es una relación no vacía (un valor o una lista con varios valores).
- s IN R: TRUE si 's' está en 'R'.
- s op ALL R: op = {<,>,<>,=,...} TRUE si 's' es mayor, menor, etc que todos los valores de 'R'.
- s op ANY R: TRUE si 's' es mayor, menor, … que al menos un valor de R.

Otro ejemplo de consulta sería la lista de actores que no actúan en ninguna película:

```
SELECT nombre
FROM actor
WHERE NOT EXISTS
        (SELECT actor_id
        FROM reparto
        WHERE actor_id=actor:id);
```

Un ejemplo de consult, primero normal, y después, anidada, usando el operador IN:

```
SELECT titulo
FROM actor, pelicula, reparto
WHERE nombre='Harrison Ford' AND actor.id=actor_id AND pelicula_id=pelicula.id;

SELECT titulo
FROM pelicula
WHERE id IN
        (SELECT pelicula_id
        FROM reparto
        WHERE (actor_id) IN
                (SELECT id
                FROM actor
                WHERE nombre='Harrison Ford'));
```

Por último, hay que destacar que es posible introducir una subconsulta dentro de FROM, para lo cual es obligatorio darle un alias. En el siguiente ejemplo puede verse la consulta para las películas estrenadas en 1978 por orden de número de actores en el reparto:

```
SELECT titulo, cc
FROM (SELECT pelicula_id, COUNT(actor_id)
        AS cc FROM reparto, pelicula
        WHERE agno=1978 AND pelicula_id=id
        GROUP BY pelicula_id) AS Temp, pelicula
WHERE pelicula_id=id
ORDER BY cc;
```

Apartado 4.- Agrupaciones y funciones de agregación en SQL.

Los operadores de agregación son aquellos operadores que calculan un valor único a partir de una columna de valores. Estos operadores son:

- SUM:
- AVG:
- MIN: devuelve el valor más pequeño de la tupla.
- MAX: devuelve el valor más grande de la tupla.
- COUNT: devuelve el número de tuplas.

En ocasiones, no queremos agrupar toda la columna, para ello, usamos la orden GROUP BY, seguido de una lista con los atributos a agrupar. Si hay un GROUP BY es importante señalar que en el SELECT solo pueden haber dos clases de términos: agregaciones y atributos que aparecen en la orden GROUP BY.
Si la orden SELECT tiene agregados solo aquellos atributos que se mencionan en el GROUP BY pueden aparecer no agregados.
Un ejemplo de consulta utilizando un GROUP BY es la siguiente. Listado de actores ordenado por número de veces que son estrellas (solo para aquellos con más de 10 estrellatos):

```
SELECT nombre, stars
FROM actor, (SELECT actor_id, count(pelicula_id) AS stars
        FROM reparto
        WHERE ord=1
        GROUP BY actor_id) AS ranking
WHERE id=actor_id AND stars>10
ORDER BY stars;
```

Esta es la misma consulta, pero utilizando una vista:

```
CREATE VIEW ranking AS
        SELECT actor_id, count(pelicula_id) AS stars
        FROM reparto
        WHERE ord=1
        GROUP BY actor_id;
```

```
SELECT nombre, stars
FROM actor, ranking
WHERE id=actor_id AND stars>10
ORDER BY stars;
```

La sentencia HAVING nos permite seleccionar grupos en base a alguna propiedad del grupo. En el siguiente ejemplo, solo seleccionaremos aquellos cuyo *sum(y)* sea mayor que 3:

```
SELECT x, SUM(y)
FROM test1
GROUP BY x
HAVING SUM(y) > 3;
```

Otro ejemplo es repetir la consulta anterior, la de los estrellatos, pero utilizando HAVING:

```
SELECT nombre, count(pelicula_id) AS stars
    FROM reparto, actor
    WHERE ord=1 AND reparto.actor_id=actor.id
    GROUP BY nombre
    HAVING COUNT(pelicula_id) > 10;
```