Pràctica Guiada DDL

A. Moll

En esta pràctica s'exercitaran els commandaments bàsics del Data Definition Language (DDL) del SQL, com ara *create table, drop table, createdb*, etc.

Optarem per fer servir el PostgreSQL per ésser un dels millors SGBD's codi obert.

Arquitectura de PostgreSQL

PostgreSQL utiliza el modelo cliente/servidor, con un proceso servidor que est´a continuamente en ejecución, esperando peticiones de procesos clientes. El sistema consta de tres componentes:

- ***Postmaster***. Es el demonio supervisor y debe estar en ejecución si alguien quiere tener acceso a la base de datos. Un demonio puede manejar varias bases de datos. Cuando se va a procesar una petición, *postmaster* inicia un proceso llamado *backend*. Normalmente hay un postmaster en ejecución en cada máquina, aunque puede haber varios en ejecución en una misma máquina. En este último caso, cada postmaster debe usar un puerto distinto y un directorio de datos también diferente para trabajar.

- *Backend*. Es un proceso que se utiliza para ejecutar una sentencia SQL. Si se han de ejecutar varias consultas a la vez, *postmaster* inicia un proceso por cada una de ellas. El número máximo de procesos *backend* lo define el administrador de la base de datos.
- *Frontend*. Los usuarios se conectan al servidor de la base de datos con ayuda de un proceso llamado *frontend*. Cuando un usuario se quiere conectar a una base de datos PostgreSQL, el *frontend* establece la conexión. Entonces *postmaster* pone en marcha un proceso *backend* y, a partir de ese momento, el *frontend* se comunica con el *backend* sin necesidad de *postmaster*. Los procesos *frontend* y *backend* no necesitan ejecutarse en la misma máquina.

psql

Los usuarios pueden acceder a una base de datos PostgreSQL utilizando diversos tipos de procesos *frontend*:

- Ejecutando el programa de terminal interactiva **psql**, que permite introducir, editar y ejecutar sentencias SQL sobre una base de datos.
- Utilizando una herramienta gráfica como **PgAccess** para crear y manipular bases de datos.
- Mediante un programa de aplicación hecho a medida que incorpore sentencias SQL para acceder a la base de datos.

En nuestras prácticas utilizaremos la terminal interactiva **psql** para conectarnos a una base de datos y practicar sobre ella el lenguaje SQL. Cuando se ejecuta este programa, se especifica el nombre de la base de datos sobre la que se va a trabajar. Por ejemplo, para trabajar sobre la base de datos llamada **ejemplo** ejecutaremos la siguiente orden:

```
$ psql ejemplo
```

Si no se especifica ninguna base de datos, la base de datos de trabajo es aquella cuyo nombre coincide con el nombre del usuario. A continuación se muestran algunas de las órdenes de **psql** que más utilizaremos:

`\?`	Muestra ayuda sobre las órdenes de **psql**.
`\h [ sentencia ]`	Muestra ayuda sobre las sentencias de SQL.
`\q`	Sale de psql.
`\d nombre`	Describe la tabla/vista/índice/secuencia con ese nombre.
`\d{t|i|s|v}`	Lista tablas/índices/secuencias/vistas.
`\da`	Lista las funciones de grupo.
`\df`	Lista las funciones.
`\do`	Lista los operadores.
`\dS`	Lista las tablas del diccionario de datos.
`\dT`	Lista los tipos de datos.
`\l`	Lista todas las bases de datos.
`\z`	Lista los permisos de acceso a las tablas.
`\e [ fichero ]`	Edita el buffer con el editor externo (o el fichero especificado).

`\g [ fichero ]`	Ejecuta la sentencia del buffer (y manda los resultados al fichero).
`\p`	Lista el contenido del buffer.
`\w fichero`	Escribe el contenido del buffer en el fichero.
`\i fichero`	Lee y ejecuta las sentencias del fichero.
`\r`	Limpia el buffer.
`\o fichero`	Manda los resultados de todas las consultas al fichero.
`\s [ fichero ]`	Muestra el historial de órdenes (o lo guarda en el fichero).
`\! [ orden ]`	Ejecuta un subshell (o una orden del sistema operativo).

Las órdenes `\g` y `\o` también pueden mandar el resultado como entrada de una orden del sistema operativo mediante `\g |orden` y `\o |orden`.

Todas las órdenes que da el usuario pasan a formar parte de un historial y se pueden recuperar utilizando las teclas con las flechas del cursor.

Empezando la sesión

Abre una conexión con **anubis**:

```
ssh anubis -l alxxxxxx
```

donde alxxxxxx será tu nombre de usuario. Crea un directorio para la asignatura llamado IG18, entra en él y crea, a continuación, la base de datos sobre la que vas a trabajar:

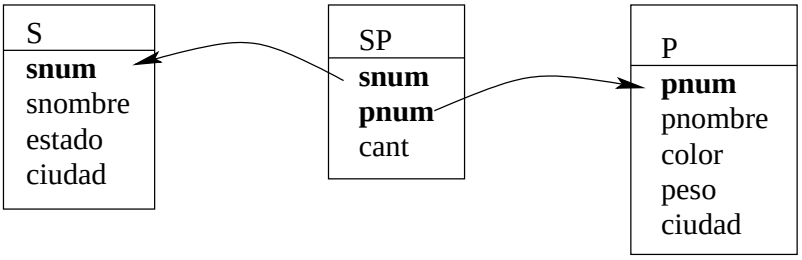
```
createdb alxxxxxx
```

Cada usuario deberá utilizar un nombre diferente para su base de datos, por eso has utilizado tu nombre de usuario. Una vez hecho esto, puedes conectarte a tu base de datos mediante **psql** para empezar a trabajar:

```
psql alxxxxxx
```

Crear tablas

En primer lugar debemos crear las tablas de la base de datos que vamos a utilizar en esta primera sesión de prácticas. Esta base de datos es la ya conocida de proveedores, piezas y envíos, cuyo esquema se muestra a continuación.



La relación S almacena los datos de los **proveedores**: código (snum), nombre (snombre), estado (estado) y ciudad donde se ubica cada proveedor (ciudad). La relación P almacena la información referente a las **piezas**: código (pnum), nombre (pnombre), color (color), peso (peso) y ciudad donde se almacena cada pieza (ciudad). Los atributos S.ciudad y P.ciudad están definidos sobre el mismo dominio (nombres de ciudades). La relación SP almacena los datos sobre los **envíos mensuales** que cada proveedor realiza (snum) de cada pieza que suministra (pnum). Cada uno de estos envíos contiene un número de unidades determinado (cant).

La siguiente sentencia crea la tabla S. Escríbela en un fichero llamado **s.sql** y después ejecútala desde **psql** mediante la orden `\i s.sql`. Observa bien los mensajes que aparecen tras ejecutar la sentencia.

```
CREATE TABLE S (
     snum VARCHAR(2),
     snombre VARCHAR(10),
     estado INTEGER,
     ciudad VARCHAR(10),
     CONSTRAINT cp_s PRIMARY KEY ( snum ), -- clave primaria
     CONSTRAINT ri_s_estado CHECK ( estado>0 ) -- regla de integridad (RI)
);
```

Las sentencias que te permitirán crear las tablas P y SP aparecen a continuación pero están incompletas. Complétalas, escribe cada una en un fichero con extensión **.sql** y después ejecútalas desde **psql**. Define los atributos peso y cant como enteros y establece la longitud de todas las cadenas a diez caracteres, excepto para los códigos de proveedor y de pieza.

```
CREATE TABLE P(
     pnum VARCHAR(2),
     pnombre ......,
     color ......,
     peso ......,
     ciudad ......,
     CONSTRAINT cp_p PRIMARY KEY ......, -- clave primaria
     CONSTRAINT ......-- RI: el peso debe ser mayor que cero
);
```

```
CREATE TABLE SP(
     snum ......,
     pnum ......,
     cant ......,
     CONSTRAINT cp_sp PRIMARY KEY ( snum, pnum ), -- clave primaria
     CONSTRAINT ca_sp_s FOREIGN KEY ( snum ) REFERENCES S
            ON DELETE RESTRICT ON UPDATE CASCADE, -- clave ajena a S
     CONSTRAINT ca_sp_p FOREIGN KEY ( pnum ) REFERENCES P
            ON DELETE CASCADE ON UPDATE CASCADE, -- clave ajena a P
     CONSTRAINT ......-- RI: la cantidad ha de ser mayor que cero
);
```

Ahora utiliza la orden `\d` de **psql** para ver la descripción que se ha guardado sobre las tablas que acabas de crear. Una vez hecho esto, contesta a la siguiente pregunta ¿por qué hay algunas columnas que tienen el modificador not null?

. . . . . .

Insertar datos

Una vez creadas las tablas, vamos a insertar datos en ellas. Las tres sentencias que aparecen a continuación insertan una fila en cada una de las tablas. Nótese que las cadenas de caracteres se encierran entre comillas simples.

```
INSERT INTO S VALUES ('S1', 'Salazar', 20, 'Londres');
INSERT INTO P VALUES ('P1', 'tuerca', 'verde', 12, 'París');
INSERT INTO SP VALUES ('S1', 'P1', 300);
```

En las preguntas que se plantean a continuación debes actuar de la siguiente forma:

- Contesta a la pregunta sin ejecutar la sentencia indicada.
- Ejecuta la sentencia después de haber contestado.
- La ejecución realizada te permitirá saber si has respondido correctamente. Si tu respuesta no es correcta, debes preguntar al profesorado para que te la pueda explicar.

Escoge una respuesta para la siguiente pregunta ¿qué crees que sucederá si se ejecuta la siguiente sentencia? `INSERT INTO SP VALUES ('S1', 'P2', 200);`

1. Se insertará una nueva fila en SP (envío de S1 y de P2).
2. Se insertarán dos nuevas filas: una en SP y otra en P (para P2).
3. Se producirá un error porque . . . . . .

Una vez hayas contestado a la pregunta, ejecuta la sentencia y observa los mensajes que se muestran. De este modo sabrás si has contestado correctamente.

¿Qué crees que sucederá si se ejecuta la siguiente sentencia?

```
INSERT INTO SP VALUES ('S1', 'P1', 400 );
```

. . . . . .

Ejecuta la sentencia y comprueba si has contestado correctamente.

¿Qué crees que sucederá si se ejecuta la siguiente sentencia?

```
INSERT INTO S VALUES ( 'S2', 'Jaimes', 0, 'París');
```

. . . . . .

Ejecuta la sentencia y comprueba si has contestado correctamente.

Ahora debes llenar las tablas con los datos que se muestran a continuación. Escribe las sentencias de inserción para cada tabla en un fichero distinto y luego ejecútalos desde **psql** (orden `\i`). Fíjate que en la tabla P hay dos nulos (ausencias de valor). En la sentencia de inserción deberás utilizar el indicador `NULL` para no insertar valores en las filas correspondientes.

```
 snum |  snombre   | estado |   ciudad
------+------------+--------+------------
 S1   | Salazar    |     20 | Londres
 S2   | Jaimes     |     10 | París
 S3   | Bernal     |     30 | París
 S4   | Corona     |     20 | Londres
 S5   | Aldana     |     30 | Atenas
(5 filas)
```

```
 pnum |  pnombre   |   color    | peso |   ciudad
------+------------+------------+------+------------
 P1   | tuerca     | verde      |   12 | París
 P2   | perno      | rojo       |      | Londres
 P3   | birlo      | azul       |   17 | Roma
 P4   | birlo      | rojo       |   14 | Londres
 P5   | leva       |            |   12 | París
 P6   | engrane    | rojo       |   19 | París
(6 filas)
```

```
 snum | pnum | cant
------+------+------
 S1   | P1   |  300
 S1   | P2   |  200
 S1   | P3   |  400
 S1   | P4   |  200
 S1   | P5   |  100
 S1   | P6   |  100
 S2   | P1   |  300
 S2   | P2   |  400
 S3   | P2   |  200
 S4   | P2   |  200
 S4   | P4   |  300
 S4   | P5   |  400
(12 filas)
```

Consultar datos

La sentencia de consulta de datos es `SELECT`. Esta sentencia la vamos a estudiar durante las sesiones de prácticas del primer semestre, por lo que aquí se hará una breve presentación.

La sentencia `SELECT` tiene varias cláusulas: `SELECT, FROM, WHERE, GROUP BY, HAVING, ORDER BY`. Cada cláusula tiene una función. Para presentar la sentencia veremos sólo las tres primeras en su uso más simple. La cláusula `SELECT`, a pesar de ser la primera que aparece en la sentencia, es la última que se ejecuta. La ejecución empieza por la cláusula `FROM` en la que se indica la tabla de la base de datos sobre la que se va a realizar la consulta. Veamos un ejemplo:

```
SELECT *
FROM S;
```

La sentencia anterior realiza una consulta sobre la tabla S mostrando todas sus filas y, para cada fila, todas sus columnas (mediante el * que aparece en la cláusula `SELECT`). Esta sentencia responde a la siguiente consulta: mostrar un listado con los datos de todos los proveedores.

Para mostrar un subconjunto de las columnas de la tabla, especificaremos éstas en la cláusula `SELECT`. Por ejemplo, si se pide mostrar el número, el nombre y la ciudad de cada proveedor, el resultado se obtendrá mediante la siguiente sentencia:

```
SELECT snum, snombre, ciudad
FROM S;
```

Al realizar una consulta sobre una tabla, si no mostramos en el resultado la clave primaria, pueden aparecer filas repetidas. Para que cada fila aparezca sólo una vez se utiliza el modificador `DISTINCT` tras la cláusula `SELECT`.

```
SELECT DISTINCT ciudad
FROM S;
```

La sentencia anterior responde a la consulta: ¿en qué ciudades hay proveedores? Ejecuta la misma sentencia sin el modificador `DISTINCT` y comprueba que el resultado es diferente.

Cuando en la consulta se quiere realizar una restricción, de modo que sólo se muestren las filas que cumplen una determinada condición, ésta se especifica en la cláusula `WHERE`. Ejecuta los ejemplos que se muestran a continuación y escribe junto a ellos a qué consulta de datos responden.

```
SELECT *
FROM S
WHERE ciudad = 'Londres';

SELECT *
FROM SP
WHERE pnum IN ('P2','P4','P6')
AND cant BETWEEN 200 AND 300;
```

Actualizar datos

La sentencia de actualización de datos `UPDATE` puede afectar a una sola fila o a varias filas a la vez dentro de la misma tabla. Consulta el contenido de la tabla S utilizando la sentencia `SELECT` y ejecuta, a continuación, la siguiente sentencia:

```
UPDATE S SET   estado = 50
         WHERE snum = 'S2';
```

Consulta, de nuevo, el contenido de la tabla S y fíjate en el cambio que se ha producido. La sentencia `UPDATE` que acabas de ejecutar ha actualizado el estado del proveedor S2; afecta a una sola fila porque la condición que deben cumplir las filas a actualizar (`WHERE`) es una condición de igualdad sobre la clave primaria. Como sabes, la clave primaria tiene un valor distinto en cada fila.

Ejecuta la siguiente sentencia:

```
UPDATE S SET   estado = estado + 10
         WHERE ciudad = 'Londres';
```

Consulta el contenido de la tabla S y comprueba los cambios que se han realizado. Fíjate que la sentencia `UPDATE` afectará ahora a varias filas: las correspondientes a los proveedores de Londres. Además, se ha actualizado el valor de una columna en función de su antiguo valor.

Si se omite la cláusula `WHERE`, la actualización se realiza sobre toda las filas de la tabla. Consulta el contenido de la tabla SP y ejecuta después la siguiente sentencia.

```
UPDATE SP SET cant = TRUNC(cant * 1.1);
```

Consultando de nuevo el contenido de la tabla verás los cambios que se han realizado. Ya que la columna cant es de tipo entero, se ha truncado el resultado de aumentar su valor en un 10 %.

También es posible actualizar varias columnas a la vez en una misma fila o en un conjunto de filas, tal y como se muestra en el siguiente ejemplo. Antes de ejecutarlo, contesta a esta pregunta ¿a cuántas filas afectará la sentencia? (fíjate bien en los datos de la tabla) .....

```
UPDATE P SET   color = 'amarillo', peso = peso - 14
         WHERE color = 'rojo';
```

Ejecuta la sentencia y comenta qué ha sucedido y porqué.

.....

En la cláusula `WHERE` se pueden incluir predicados que involucren subconsultas. Una subconsulta es una sentencia `SELECT` anidada en otra sentencia SQL.

```
UPDATE S SET estado = 60
         WHERE snum IN ( SELECT snum
                         FROM   SP
                         WHERE  cant > 300 );
```

¿Qué hace la sentencia anterior? .....

.....

Borrar datos

Los datos se borran mediante la sentencia `DELETE`. Del mismo modo que en la actualización, se puede borrar una sola fila o varias filas a la vez de una misma tabla. Para seleccionar las filas a borrar se incluye una cláusula `WHERE` (si se omite, se borran todas las filas de la tabla).

Consulta el contenido de la tabla SP y ejecuta después la siguiente sentencia:

```
DELETE FROM SP WHERE snum = 'S2';
```

Consulta, de nuevo, la tabla SP y verás que algunas filas han sido eliminadas.

Fíjate ahora en las reglas de borrado que has establecido para las dos claves ajenas ¿qué crees que ocurrirá si ejecutas la siguiente sentencia?

```
DELETE FROM S WHERE snum = 'S1';
```

.....

Tras contestar, ejecuta la sentencia y comprueba si tu respuesta es correcta.

¿Qué crees que ocurrirá si ejecutas esta sentencia?

```
DELETE FROM P WHERE pnum = 'P2';
```

. . . . . .

Tras contestar, ejecuta la sentencia y comprueba si tu respuesta es correcta.

En la cláusula `WHERE` de la sentencia `DELETE` también se pueden incluir predicados que involucren subconsultas.

Eliminar tablas

La sentencia que elimina una tabla de la base de datos es `DROP`. La siguiente sentencia elimina la tabla S:

```
DROP TABLE S;
```

¿Se ejecuta con éxito la sentencia anterior? ¿por qué?

. . . . . .

. . . . . .

Antes de terminar esta primera sesión, debes eliminar las tablas de la base de datos que acabas de crear. Escribe a continuación el orden en que se deben ejecutar las sentencias `DROP` para que no quede ninguna tabla. Con la orden `\d` de `psql` puedes comprobar las tablas que hay en tu base de datos.

. . . . . .

. . . . . .

. . . . . .

Después de salir de `psql` (`\q`), elimina la base de datos que has creado al principio de la sesión:

```
dropdb alxxxxxx
```

y cierra la conexión con **anubis**.