Actividad 2.1. Introducción al SGBD PostgreSQL y al lenguaje SQL

OCW VJ1220 Bases de datos

Objetivos

- Crear una base de datos de PostgreSQL y acceder a ella mediante pgAdmin.
- Crear tablas en SQL (sentencia `CREATE TABLE`) incluyendo la definición de la clave primaria, las claves ajenas, las reglas de comportamiento de las claves ajenas y también reglas de integridad sencillas.
- Insertar, actualizar y borrar datos de una base de datos (sentencias `INSERT`, `UPDATE` y `DELETE`).
- Determinar, a priori, el comportamiento de las sentencias `INSERT`, `UPDATE` y `DELETE` en función de la definición de las tablas de una base de datos: si van a producir un error o no, y en este caso, a cuántas/qué filas van a afectar.
- Eliminar tablas en SQL (sentencia `DROP TABLE`) y de saber qué efectos produce sobre la base de datos esta operación.

Qué hacer

Lee los apartados del 4.1 al 4.4 del **capítulo 4** en donde se da una visión general del lenguaje SQL. Lo que leerás allí lo practicarás a continuación con los ejercicios que se plantean en esta actividad.

PostgreSQL

PostgreSQL es un SGBD relacional que utiliza el modelo cliente/servidor, con un proceso servidor que está 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.

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

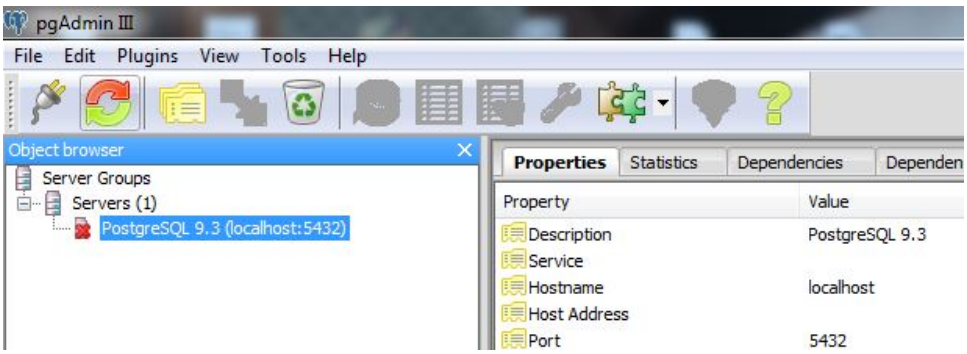
- Ejecutando el programa de terminal interactiva **psql**, que permite introducir, editar y ejecutar sentencias SQL sobre una base de datos.
- Utilizando una herramienta con interfaz gráfica como **PgAdmin** para crear y manipular bases de datos.
- Mediante un programa de aplicación hecho a medida que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

Puesto que en nuestro caso vamos a practicar con el lenguaje SQL de manera interactiva, usaremos PgAdmin.

Para comenzar a trabajar deberás instalar PostgreSQL[1] y pgAdmin[2] en tu ordenador.

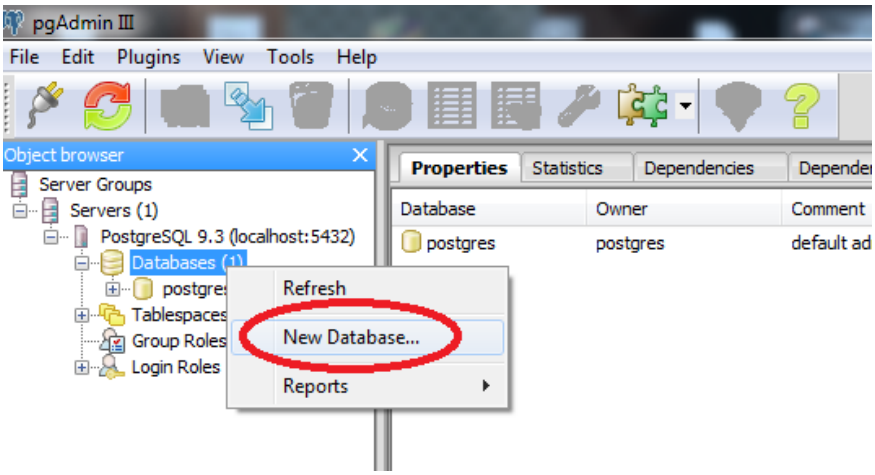
Crear la base de datos

Abre pgAdmin y pincha sobre el servidor que hay en tu ordenador (*localhost*) para conectarte a él.

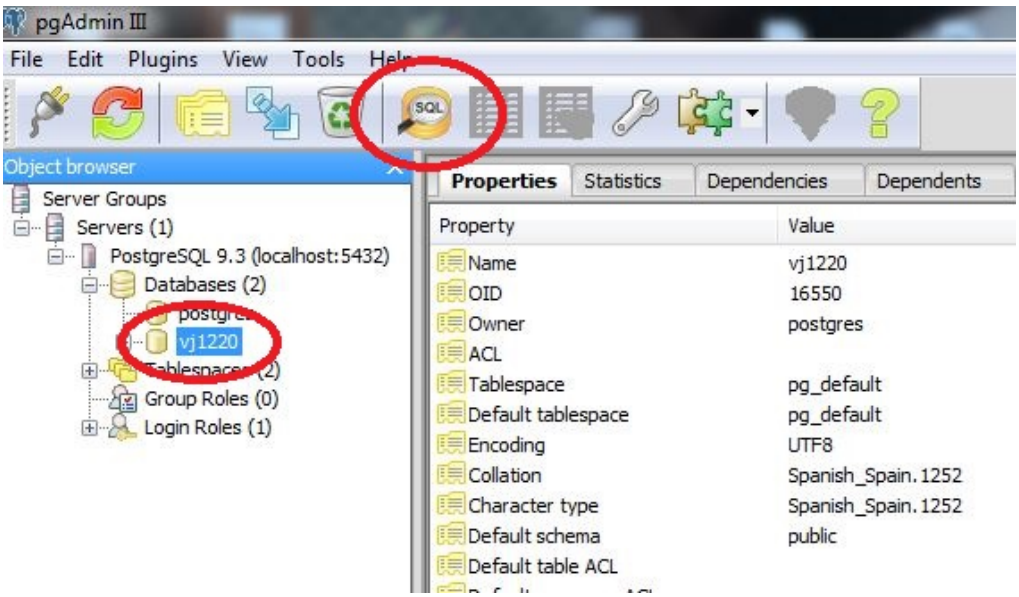


Una vez conectados al servidor local, vamos a crear la base de datos de esta actividad. Pincha sobre *Databases* y pulsa el botón derecho del ratón para abrir el menú contextual: selecciona *New Database* y da un nombre a tu base de datos.

[1] http://www.postgresql.org/download/
[2] http://www.pgadmin.org/download/

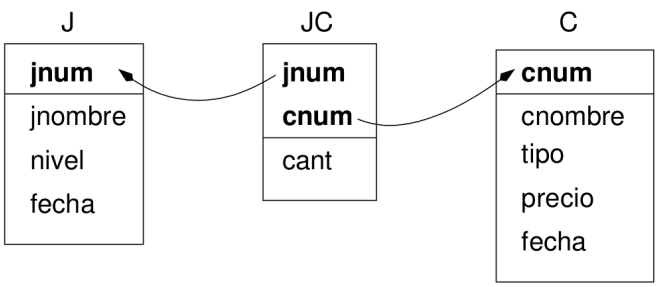


Conéctate a tu base de datos pinchando sobre ella y pulsa después el botón *Execute arbitrary SQL queries*. Se abrirá una ventana en la que podrás ejecutar las sentencias SQL de esta actividad.



Crear tablas

En primer lugar debemos crear las tablas de la base de datos que vamos a utilizar en esta actividad. Esta base de datos es la ya conocida de jugadores, campeones y partidas, cuyo esquema se muestra a continuación.



La tabla `J` almacena los datos de los **jugadores**: identificador (`jnum`), nombre (`jnombre`), nivel (`nivel`) y fecha de ingreso en el juego (`fecha`). La tabla `C` almacena la información referente a los **campeones**: identificador (`cnum`), nombre (`cnombre`), tipo (`tipo`), precio (`precio`) y fecha en la que se introdujo en el juego (`fecha`). La tabla `JC` almacena el número de **partidas** (`cant`) que cada jugador ha realizado (`jnum`) con cada campeón (`cnum`).

Las siguientes sentencias crean las tablas de la base de datos, deberás completarlas antes de ejecutarlas. Las partes a completar se han marcado con líneas de puntos (debes eliminarlas por completo antes de ejecutar las sentencias). Define los atributos `precio` y `cant` como números enteros y establece la longitud de todas las cadenas a 15 caracteres, excepto para los identificadores de jugador y de campeón.

Observa bien los mensajes que aparecen tras ejecutar las sentencias por si hubieras cometido algún error.

```
CREATE TABLE J (
  jnum    VARCHAR(2),
  jnombre VARCHAR(15),
  nivel   INTEGER,
  fecha   DATE,
  CONSTRAINT cp_j PRIMARY KEY ( jnum ), -- clave primaria
  CONSTRAINT ri_j_nivel CHECK ( nivel BETWEEN 1 AND 100 )
        -- regla de integridad (RI): el nivel debe estar entre 1 y 100
);

CREATE TABLE C(
  cnum VARCHAR(2),
  cnombre ..................,
  tipo    ..................,
  precio  ..................,
  fecha   ..................,
  CONSTRAINT cp_c PRIMARY KEY ........................., -- clave primaria
  CONSTRAINT .....................................................................
                  -- RI: el precio debe ser mayor que 150
);

CREATE TABLE JC(
  jnum ..................,
  cnum ..................,
  cant ..................,
  CONSTRAINT cp_jc PRIMARY KEY ( jnum, cnum ), -- clave primaria
  CONSTRAINT ca_jc_j FOREIGN KEY ( jnum ) REFERENCES J
    ON UPDATE CASCADE ON DELETE RESTRICT, -- clave ajena a J
```

```
  CONSTRAINT ca_jc_c FOREIGN KEY ( cnum ) REFERENCES C
     ON UPDATE CASCADE ON DELETE CASCADE, -- clave ajena a C
   CONSTRAINT ..........................................................
                    -- RI: la cantidad ha de ser mayor que 0
);
```

Insertar datos

Una vez creadas las tablas, vamos a insertar datos en ellas. Antes de empezar, vamos a cambiar el formato de las fechas para que coincida con el que vamos a trabajar (europeo y separado por barras inclinadas):

```
SET DATESTYLE TO SQL,EUROPEAN;
```

Ejecuta las tres sentencias que aparecen a continuación (quizá debas reordenarlas). Fíjate que insertan una fila en cada una de las tablas. Verás que en SQL las cadenas de caracteres y las fechas se encierran entre comillas simples.

```
INSERT INTO JC VALUES ('J1', 'C1', 300);
INSERT INTO C VALUES ('C1', 'Akali', 'asesino', 790, '11/05/2010');
INSERT INTO J VALUES ('J1', 'Salazar', 20, '29/06/2010');
```

Escoge una respuesta para la siguiente pregunta ¿qué crees que sucederá si se ejecuta ahora la siguiente sentencia?

```
INSERT INTO JC VALUES ('J1', 'C2', 200);
```

1. Se insertará una nueva fila en `JC` (partidas de `J1` y de `C2`).
2. Se insertarán dos nuevas filas: una en `JC` y otra en `C` (para `C2`).
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 JC VALUES ('J1', 'C1', 400 );
```

Ejecuta la sentencia y comprueba si has contestado correctamente.

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

```
INSERT INTO J VALUES ( 'J2', 'Jaimes', 0, '15/07/2011');
```

Ejecuta la sentencia y comprueba si has contestado correctamente.

Ahora debes llenar las tablas con los datos que insertan las sentencias que encontrarás a continuación. Fíjate que en la tabla `J` hay dos nulos (ausencias de valor). En la sentencia

de inserción utilizamos el indicador `NULL` para no insertar valores en las columnas correspondientes.

```
INSERT INTO J VALUES ('J2', 'Jaimes', 10, '15/07/2011'),
                     ('J3', 'Bernal', 30, '24/09/2010'),
                     ('J4', 'Corona', 20, '25/12/2012'),
                     ('J5', 'Aldana', 30, '24/04/2011');
INSERT INTO C VALUES ('C2', 'Brand',   'asesino', NULL, '24/09/2010'),
                     ('C3', 'Caitlyn', 'mago',    880,  '01/01/2011'),
                     ('C4', 'Diana',   'carry',   880,  '24/09/2010'),
                     ('C5', 'Draven',  NULL,      975,  '11/05/2010'),
                     ('C6', 'Elise',   'mago',    975,  '11/05/2010');
INSERT INTO JC VALUES ('J1', 'C2', 200), ('J1', 'C3', 400),
                      ('J1', 'C4', 200), ('J1', 'C5', 100),
                      ('J1', 'C6', 100), ('J2', 'C1', 300),
                      ('J2', 'C2', 400), ('J3', 'C2', 200),
                      ('J4', 'C2', 200), ('J4', 'C4', 300),
                      ('J4', 'C5', 400);
```

Al terminar las inserciones, los datos que contendrán las tablas serán los siguientes:

Tabla J

```
 jnum | jnombre | nivel |   fecha
------+---------+-------+------------
 J1   | Salazar |    20 | 29/06/2010
 J2   | Jaimes  |    10 | 15/07/2011
 J3   | Bernal  |    30 | 24/09/2010
 J4   | Corona  |    20 | 25/12/2012
 J5   | Aldana  |    30 | 24/04/2011
```

Tabla C

```
 cnum | cnombre |  tipo   | precio |   fecha
------+---------+---------+--------+------------
 C1   | Akali   | asesino |    790 | 11/05/2010
 C2   | Brand   | asesino |        | 24/09/2010
 C3   | Caitlyn | mago    |    880 | 01/01/2011
 C4   | Diana   | carry   |    880 | 24/09/2010
 C5   | Draven  |         |    975 | 11/05/2010
 C6   | Elise   | mago    |    975 | 11/05/2010
```

Tabla JC

```
 jnum | cnum | cant
------+------+------
 J1   | C1   |  300
 J1   | C2   |  200
 J1   | C3   |  400
 J1   | C4   |  200
 J1   | C5   |  100
 J1   | C6   |  100
 J2   | C1   |  300
 J2   | C2   |  400
 J3   | C2   |  200
 J4   | C2   |  200
 J4   | C4   |  300
 J4   | C5   |  400
```

Consultar datos

La sentencia de consulta de datos es `SELECT`. Esta sentencia la vamos a estudiar a fondo durante el curso por lo que en esta sesión se hará una breve presentación.

La sentencia `SELECT` tiene varias cláusulas: `SELECT`, `FROM`, `WHERE`, `GROUP BY`, `HAVING`, `ORDER BY`. Cada cláusula tiene una función distinta. 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 J;
```

La sentencia anterior realiza una consulta sobre la tabla `J` 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 los datos de todos los jugadores*.

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

```
SELECT jnum, jnombre, fecha
FROM J;
```

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

```
SELECT DISTINCT nivel
FROM J;
```

La sentencia anterior responde a la consulta: *¿en qué niveles hay jugadores*? 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 formula una frase que indique la consulta de datos que responden.

```
SELECT *
FROM J
WHERE nivel = 20;
```

```
SELECT *
FROM JC
WHERE cnum IN ('C2','C4','C6')
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 `J` utilizando la sentencia `SELECT` y ejecuta, a continuación, la siguiente sentencia:

```
UPDATE J SET nivel = 50
       WHERE jnum = 'J2';
```

Consulta, de nuevo, el contenido de la tabla `J` y fíjate en el cambio que se ha producido. La sentencia `UPDATE` que acabas de ejecutar ha actualizado el estado del jugador `J2`; 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 J SET nivel = nivel + 10
         WHERE fecha BETWEEN '01/05/2010' AND '30/04/2011';
```

Consulta el contenido de la tabla `J` y comprueba los cambios que se han realizado. Fíjate que la sentencia `UPDATE` afectará ahora a varias filas: las correspondientes a los jugadores que ingresaron en el juego en el periodo especificado. Además, se ha actualizado el valor de una columna en función de su antiguo valor, aumentándolo en diez unidades.

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

```
UPDATE JC 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 ejecutar la sentencia, contesta a esta pregunta ¿a cuántas filas afectará? (para responder fíjate bien en los datos de la tabla)

```
UPDATE C SET tipo = 'support', precio = precio - 750
         WHERE tipo = 'mago';
```

Ejecuta la sentencia y comprueba si tu predicción es correcta..

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

```
UPDATE J SET nivel = 60
       WHERE jnum IN ( SELECT jnum
                       FROM JC
                       WHERE cant > 300 );
```

¿Puedes decir a qué consulta responde la sentencia anterior? Actualiza a 60 el nivel de los jugadores que .....

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 `JC` y ejecuta después la siguiente sentencia:

```
DELETE FROM JC WHERE jnum = 'J2';
```

Consulta, de nuevo, la tabla `JC` 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 J WHERE jnum = 'J1';
```

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

¿Qué crees que ocurrirá si ejecutas esta sentencia?

```
DELETE FROM C WHERE cnum = 'C2';
```

Tras contestar, ejecuta la sentencia y comprueba si tu predicción 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 J;
```

¿Se ejecuta con éxito la sentencia anterior? Si no es así, averigua el porqué.