



BASES DE DATOS



Guía Taller I

Conexión con la Base de Datos - Introducción a PostgreSQL

1. Conexión con la Base de Datos: se debe realizar a partir de algún programa que implemente el protocolo **SSH (Secure Shell)**, a partir del cual es posible acceder a máquinas remotas a través de una red. Para nuestro caso usamos este protocolo debido a que usa técnicas de cifrado que hacen que la información que viaja por el medio de comunicación vaya de manera no legible y ninguna tercera persona pueda descubrir el usuario y contraseña de la conexión, ni lo que se escribe durante toda la sesión.

Existen distintos programas que permiten realizar conexiones a partir de este protocolo, a continuación se presentan dos alternativas:

SecureCRT: es una herramienta eficaz que protege datos, contraseñas, o cuentas de usuarios que son enviados a través del protocolo SSH. También es una utilidad ideal para conectar sistemas remotos a través del protocolo antes mencionado. SecureCRT posee todas las funciones de un cliente tipo Telnet, como por ejemplo, logins automáticos, nombrado de sesiones, impresión, selección de color, etc. El programa se puede descargar desde Internet como un período de prueba gratuito que caduca a los 30 días de uso, solo se deben llenar ciertos datos antes de descargarlo.

[DESCARGAR SecureCRT](#)

Pasos para la conexión con SecureCRT:

Seleccionamos el protocolo a utilizar: para las versiones recientes sería SSH2

Colocamos el Hostname: es el nombre del dominio o el IP al cual se desea conectar. Para nuestro caso es 150.186.36.174

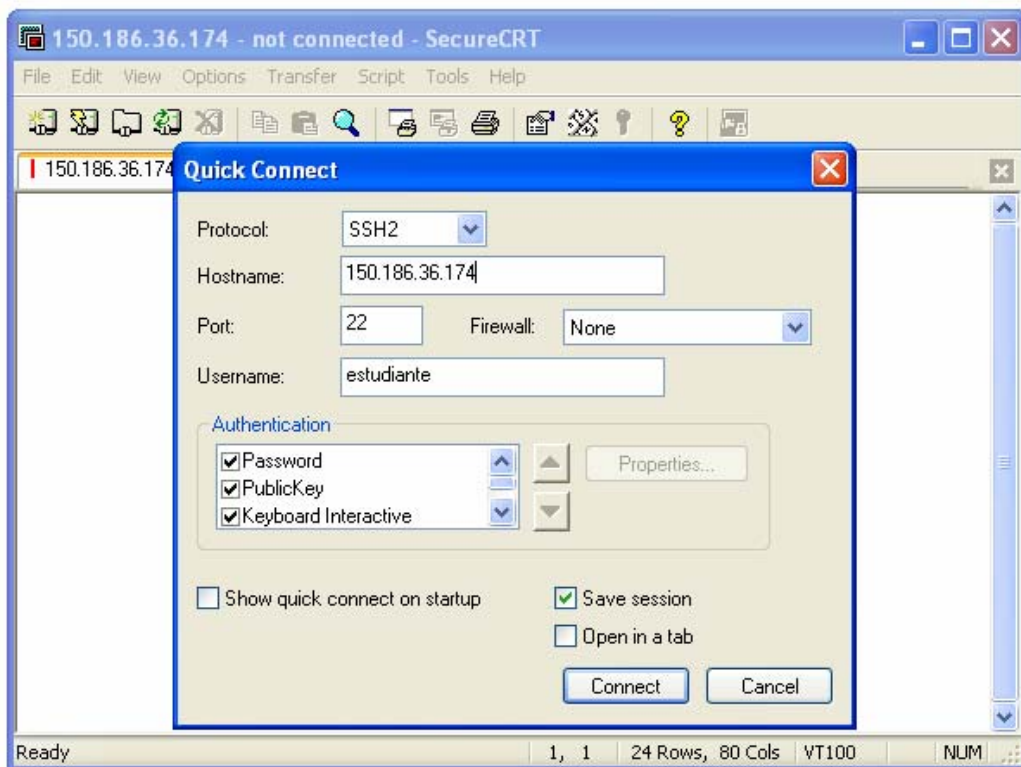
Se denota el puerto (port): se coloca 22 (predeterminado)

Se escribe el Nombre de Usuario (username): colocamos estudiante

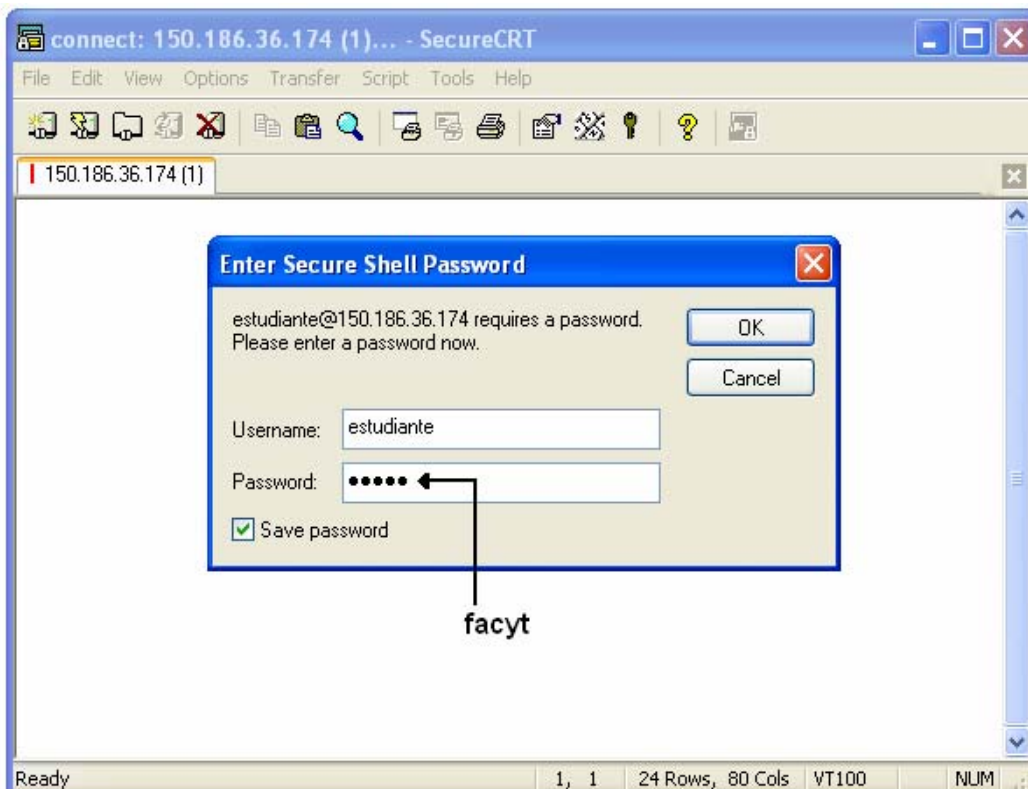
Se elige el tipo de Autenticación (authentication): se debe colocar Password

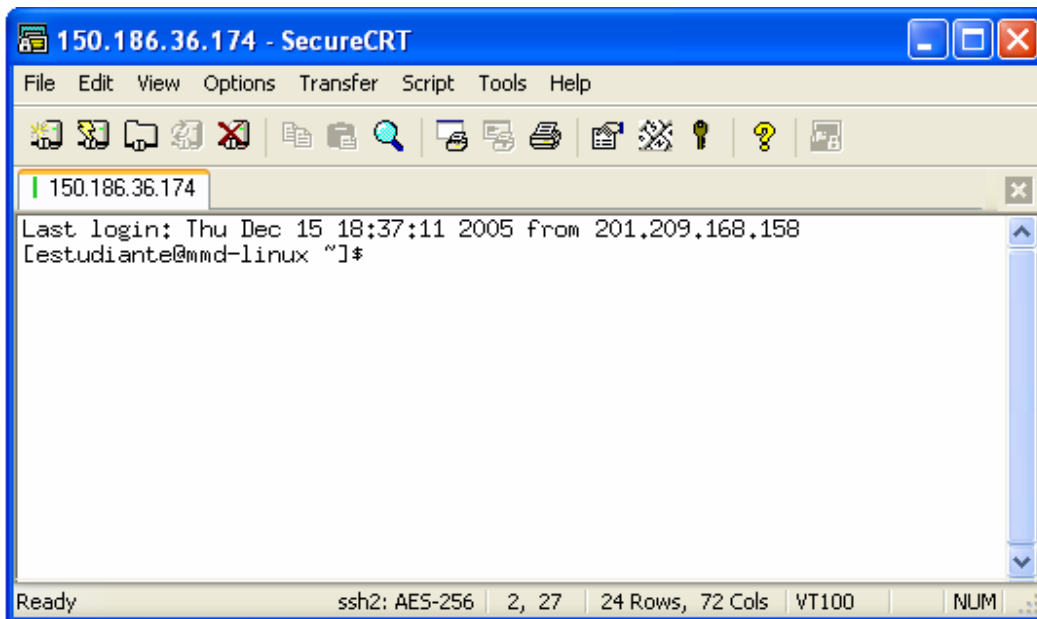
Nota: dependiendo de la versión en la cual se esté trabajando existen opciones adicionales, pero con utilizar las opciones predeterminadas será suficiente para nuestro caso.

Una vez hecho esto se debe ver en la pantalla algo más o menos así:



1. Haga clic en "Connect" y se le presentara una pantalla llamada "Enter Password", en la cual aparecerá **username** (estudiante). En el área del Password escriba **facyt** y si lo desea seleccione la opción de recordar contraseña. La pantalla se debería ver así:





Una vez que aparezca la pantalla anterior ya estará conectado al servidor, pero faltará un paso más para conectarse a la Base de Datos en PostgreSQL, el cual consta de escribir el siguiente código:

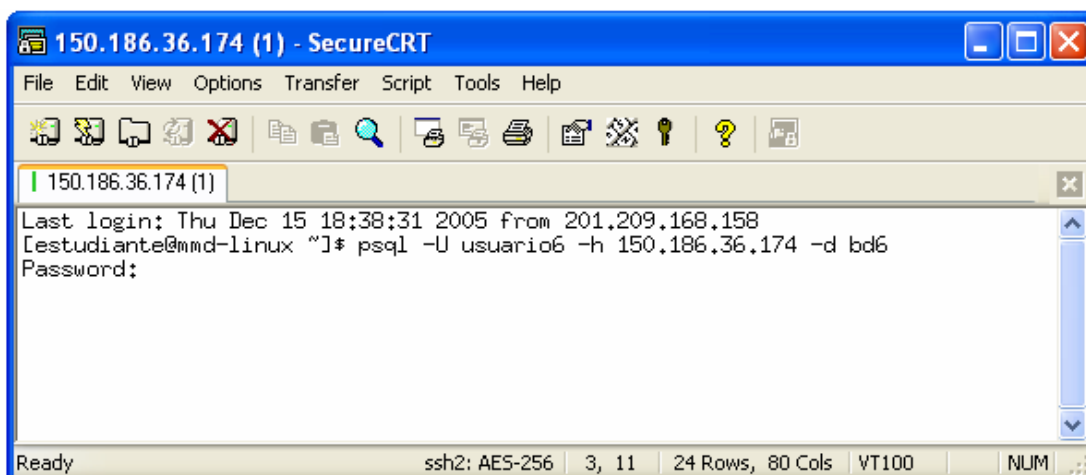
```
psql -U usuarioNro -h 150.186.36.174 -d bdNro
```

Ejemplo: `psql -U usuario15 -h 150.186.36.174 -d bd15`

El **Nro** dependerá del número que le haya sido asignado en el taller del día Martes 13/12/05

Luego presione ENTER y le será solicitado un **password** (ver la siguiente figura), el cual será **usuarioNro**, (una vez más **Nro** dependerá de su número de usuario) y presione **ENTER**

Ejemplo: usuario15



Ya hecho esto estará conectado a su Base de Datos y podrá realizar las operaciones correspondientes.

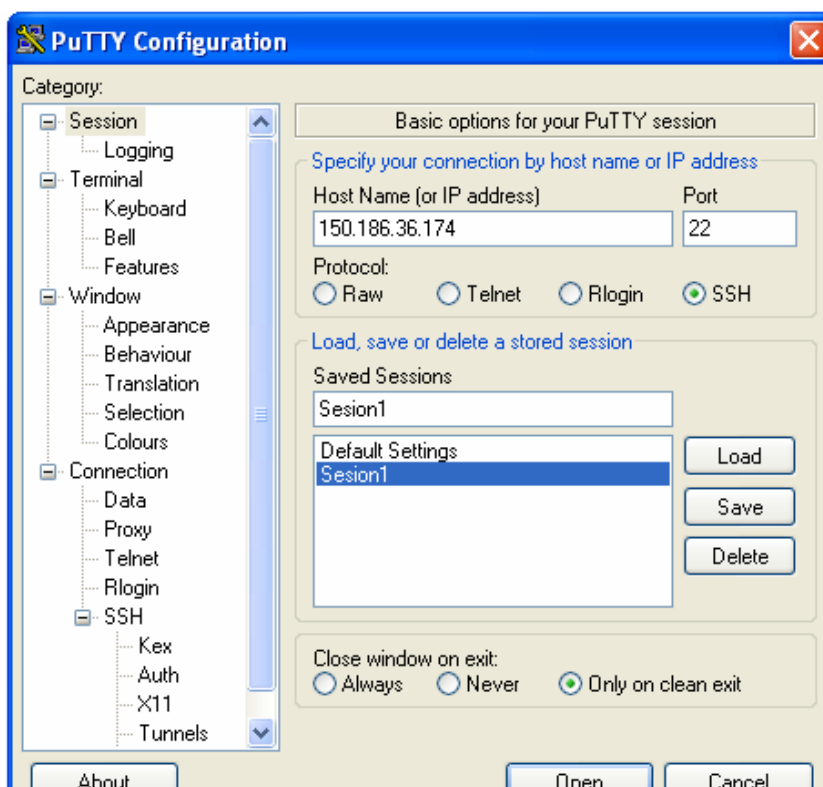
PuTTY: para aquellos que optan por alternativas en el área de software libre existe una herramienta muy efectiva en lo que a conexiones de tipo SSH se refiere, el PuTTY. PuTTY es muy sencillo de utilizar, la mayor o menor complejidad a la hora de usarlo depende principalmente de los conocimientos que usted tenga para realizar tareas ejecutando comandos SSH desde el terminal. Una ventaja importante de este programa es que no se debe instalar en el computador, sólo debe ejecutarlo y seguir los pasos para realizar la conexión. Existen versiones para sistemas UNIX o Windows en todas sus versiones y puedes descargar el programa desde la siguiente página.

[DESCARGAR PuTTY](#)

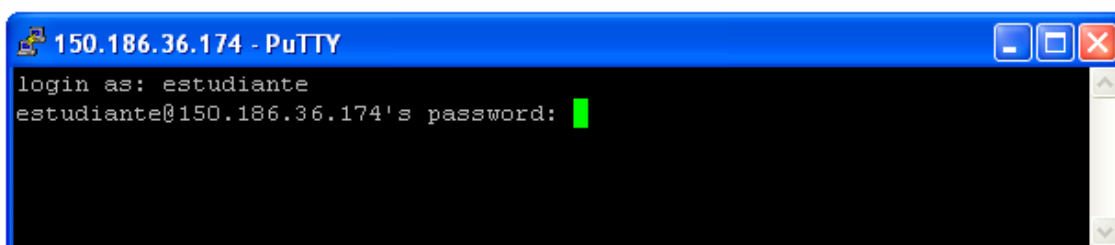
Pasos para la conexión con PuTTY:

1. En el menú de configuración seleccione la categoría **Session**.
2. Introduzca el nombre del dominio o IP en el campo **Host Name**. Para nosotros es **150.183.36.174**
3. Seleccione el protocolo **SSH**.
4. Introduzca un nombre para esta conexión en el campo **Saved Sessions**, puede ser el que usted desee, pues se utilizará para guardar su sesión. En el caso de que desee esto, haga clic en **Save**
5. En la opción **close window on exit** seleccione **“only on clean exit”**.

Una vez hecho esto se debe ver la siguiente pantalla:



6. Ahora aparecerá una consola solicitando el nombre de usuario, colocamos **estudiante** y como Password **facyt**, se verá lo siguiente:



```
150.186.36.174 - PuTTY
login as: estudiante
estudiante@150.186.36.174's password: █
```

Presione la tecla **ENTER**, y aparecerá otras líneas en la pantalla

Ahora sólo falta realizar un paso más para conectarse a la Base de Datos en PostgreSQL, el cual consta de escribir el siguiente código:

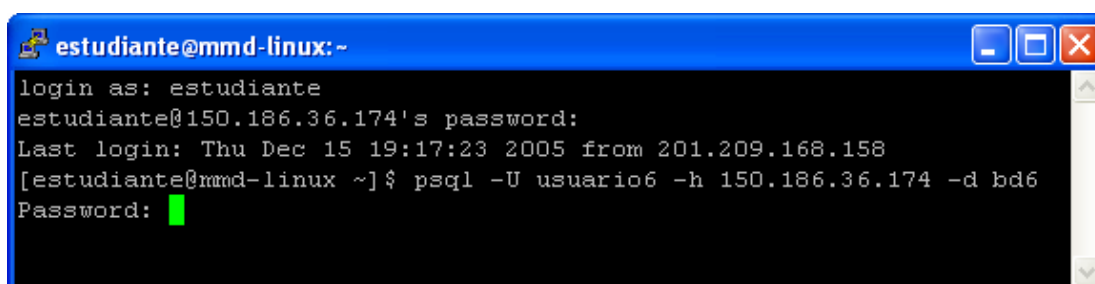
```
psql -U usuarioNro -h 150.186.36.174 -d bdNro
```

Ejemplo: `psql -U usuario15 -h 150.186.36.174 -d bd15`

El **Nro** dependerá del número que le haya sido asignado en el taller del día Martes 13/12/05

Luego presione ENTER y le será solicitado un **password** (ver la siguiente figura), el cual será **usuarioNro**, (una vez más **Nro** dependerá de su número de usuario) y presione **ENTER**

Ejemplo: usuario15



```
estudiante@mmd-linux: ~
login as: estudiante
estudiante@150.186.36.174's password:
Last login: Thu Dec 15 19:17:23 2005 from 201.209.168.158
[estudiante@mmd-linux ~]$ psql -U usuario6 -h 150.186.36.174 -d bd6
Password: █
```

Ya hecho esto estará conectado a su Base de Datos y podrá realizar las operaciones correspondientes.

2. Introducción a PostgreSQL: los sistemas de mantenimiento de Bases de Datos relacionales tradicionales (DBMS) soportan un modelo de datos que consisten en una colección de relaciones con un nombre, que contienen atributos de un tipo específico. En los sistemas comerciales actuales, los tipos posibles incluyen numéricos de punto flotante, enteros, cadenas de caracteres, cantidades monetarias y fechas. Está generalmente reconocido que este modelo será inadecuado para las aplicaciones futuras de procesamiento de datos. Sin embargo, como se ha mencionado, esta simplicidad también hace muy difícil la implementación de ciertas aplicaciones. **Postgres** ofrece una potencia adicional, además de ser una alternativa en software libre, ya que incorpora los siguientes cuatro conceptos adicionales básicos en una vía en la que los usuarios pueden extender fácilmente el sistema:

Clases
Herencia
Tipos
Funciones

Otras características aportan potencia y flexibilidad adicional son:

Restricciones (Constraints)

Disparadores (Triggers)

Reglas (Rules)

Integridad Transaccional

Estas características colocan a **Postgres** en la categoría de las Bases de Datos identificadas como *objeto-relacionales*.

En 1996, los desarrolladores de Postgres, después de haber desarrollado ya varias versiones de este DBMS decidieron cambiar el nombre, y lo llamaron PostGreSQL (exactamente para su versión 6.0) para reflejar la relación entre Postgres y las versiones recientes de SQL. Se crearon nuevas mejoras y modificaciones, que repercutieron en un 20-40% más de eficiencia, así como la incorporación del estándar SQL92, aunque ya para la fecha se ha agregado también el SQL99. Actualmente el grupo de desarrollo de PostgreSQL lanzó la versión 8.0 del mismo.

Sentencias Principales: como se mencionó anteriormente, PostgreSQL está basado en el estándar SQL, por lo que trabaja con ciertas sentencias definidas por el mismo. A continuación se encuentran explicadas de manera sencilla y con ejemplos las sentencias principales utilizadas por PostgreSQL. Siempre se debe indicar el final de una sentencia por medio de un punto y coma (;).

CREATE

El lenguaje SQL incluye un conjunto de comandos para definición de datos. En este caso nos basaremos en el principal, aquel que crea una nueva relación.

SINTAXIS: CREATE TABLE nombre_tabla // Tabla a crear
(atr_1 **tipo_atr_1**,
atr_2 **tipo_atr_2**,
...,
atr_k1 **tipo_atr_k**); // Atributos de la tabla con sus tipos

EJEMPLOS:

CREATE TABLE persona (CI **int4**, nombre **varchar (30)**, edad **int2**);

Principales Tipos de Datos permitidos por SQL:

- Int2** - Entero de 2 bytes con signo
- Int4** - Entero de 4 bytes con signo
- Char (n)** - Cadena de Caracteres de longitud **n** fija
- Varchar (n)** - Cadena de Caracteres de longitud **n** variable
- Bool** - Tipo Booleano
- Date** - Tipo Fecha (año/mes/día, ejemplo 2005,12,15)
- Float** - Número de Punto Flotante

Definición de claves con CREATE TABLE: existen distintas opciones

CREATE TABLE persona (CI **int4 primary key**, nombre **varchar (30)**, edad **int2**);

CREATE TABLE persona (CI **int4**, nombre **varchar (30)**, edad **int2**, **primary key** (CI));

CREATE TABLE ciudad (nombre_c **varchar(30)**, pais **varchar (30)**, **primary key** (nombre_c, pais));

DROP

El lenguaje SQL permite también la eliminación de tablas, de índices, o de vistas, lo cual es posible a través de la sentencia DROP.

SINTAXIS: **DROP TIPO_DATO** // Eliminar el dato de tipo TIPO_DATO
nombre_dato ; // con el nombre nombre_dato

EJEMPLOS:

DROP TABLE persona; // Elimina la tabla persona

SELECT

Es una de las sentencias principales, ya que a partir de ella se recuperan (muestran) los datos almacenados.

SINTAXIS: **SELECT** atr_1, atr_2, ..., atr_k // Atributos a mostrar
FROM table_1, table_2, ..., table_k // Tablas de los atributos utilizados
WHERE condicion_x ; // Condición que deben cumplir los atributos a mostrar

EJEMPLOS:

SELECT *
FROM persona

Resulta...

CI	nombre	edad
15.300.000	Luis	15
16.060.980	Diana	19

Utilizando * en la instrucción SELECT se muestran todos los atributos de la Tabla. Nótese que en SQL ejecutar la sentencia anterior, o ejecutar la siguiente, arroja el mismo resultado, pues como se mencionó anteriormente, el final de una sentencia viene dado por un punto y coma (;).

SELECT * FROM persona ;

Resulta...

CI	nombre	edad
15.300.000	Luis	15
16.060.980	Diana	19

Ahora con cualificaciones, y definición de atributos a mostrar:

```
SELECT CI, nombre
FROM persona
WHERE CI > 15.000.000 ;           Resulta...
```

```
      CI      | nombre
-----+-----
15.300.000 | Luis
16.060.980 | Diana
```

En la cláusula WHERE también se pueden utilizar conectores lógicos como AND, OR, NOT

```
SELECT CI, nombre
FROM persona
WHERE CI > 15.000.000 AND edad < 18;           Resulta...
```

```
      CI      | NOMBRE
-----+-----
15.300.000 | Luis
```

Manipulación de Datos

Además de las sentencias ya antes descritas, SQL permite también la manipulación de datos de las tablas, en este caso, de las distintas tuplas que se encuentren en ella.

INSERT

Esta sentencia inserta una tupla dentro de una tabla

SINTAXIS: **INSERT INTO** nombre_tabla (atr_1, atr_2, ..., atr_k) // Insertar en nombre_tabla
VALUES (valor_atr1, valor_atr2, ..., valor_atrk) ; // Atributos con los valores...

EJEMPLOS:

```
INSERT INTO persona (CI, nombre, edad) VALUES (3291859, 'Pedro', 'Pérez') ;
```

Existen distintas maneras de insertar una nueva tupla en una tabla. Si tomamos en cuenta el ejemplo anterior, podemos darnos cuenta de que se denotan todos los atributos que se van a ingresar, así como los valores para cada uno de ellos. Si se desea crear una

tupla con valores para cada atributo de la tabla, se puede realizar la inserción de la siguiente manera (tomando el ejemplo anterior):

```
INSERT INTO persona VALUES (3291859, 'Pedro', 'Pérez');
```

NOTA: si la inserción se va a realizar de la manera descrita en el ejemplo anterior, se deben colocar los valores para cada atributo en el mismo orden que se encuentran los atributos dentro de la tabla. En ese caso:

```
INSERT INTO persona VALUES (3291859, 'Pedro', 'Pérez'); // Está bien hecho  
INSERT INTO persona VALUES ('Pérez', 'Pedro', 3291859); // Es una inserción errada
```

En el caso en el que no se vayan a denotar valores para todos los atributos de la tupla a insertar, se deben definir explícitamente cuales son los valores a insertar, e igualmente que en el caso anterior, se deben colocar los valores para cada atributo en el mismo orden que se encuentran los atributos a insertar. Entonces:

```
INSERT INTO persona (CI, Apellido) VALUES (3291859, 'Pérez'); // Está bien hecho  
INSERT INTO persona (CI, Apellido) VALUES ('Pérez', 3291859); // Está mal hecho
```

UPDATE

Esta sentencia modifica una o más tuplas dentro de una tabla

```
SINTAXIS: UPDATE nombre_tabla // Actualizar nombre_tabla ,  
          SET atr_1 = valor_atr1, atr_2 = valor_atr2, ..., atr_k = valor_atrk // Cambiar  
          WHERE condicion_x; // Atributos que cumplan con la condicion_x
```

EJEMPLOS:

```
UPDATE persona  
SET nombre = 'Johan', apellido = 'González'  
WHERE CI = 3291859;
```

DELETE

Esta sentencia elimina una o más tuplas dentro de una tabla

```
SINTAXIS: DELETE FROM nombre_tabla // Eliminar de nombre_tabla  
          WHERE condicion_x; // La tupla que cumpla con la condicion_x
```

EJEMPLOS:

```
DELETE FROM persona  
WHERE nombre = 'Johan' ;
```

Existen muchas variaciones y cláusulas adicionales para cada una de las funciones descritas, las cuales facilitan el uso de la Base de Datos y permiten obtener los resultados esperados de manera más efectiva. Se lea al usuario la tarea de reforzar esta información a partir de otras documentaciones.