



Guía teórica de Postgres

Nivel I

UNEWEB



Índice

| | |
|--|----|
| Conceptos básicos de bases de datos. | 3 |
| Base de datos | 3 |
| Tabla | 3 |
| Diferencia entre registro y campo | 4 |
| Modelo Entidad-Relación | 4 |
| Representación gráfica del Modelo Entidad – Relación | 5 |
| SQL Estándar | 6 |
| Introducción a postgresql. | 7 |
| Características y Ventajas | 7 |
| Otros Sistemas de Gestión de Bases de Datos | 9 |
| Instalación de PostgreSql en Linux y Windows. | 10 |
| Acceder a la consola de postgres (psql shell). | 15 |
| Operaciones básicas en las Bases de datos y tablas. | 16 |
| Crear una base de datos | 17 |
| Eliminar bases de datos | 17 |
| Renombrar una base de datos | 18 |
| Conectar con Bases de Datos Creadas | 18 |
| Tipos de datos soportados por postgresql | 19 |
| Crear una Tabla. | 20 |
| Describir la estructura de una tabla | 21 |
| Alteración básica de tablas. | 21 |
| Operaciones elementales en las bases de datos (CRUD). | 22 |
| CRUD-CREATE | 22 |

| | |
|---|----|
| Insertar datos en una tabla | 24 |
| CRUD-READ | 27 |
| CRUD- UPDATE | 29 |
| DELETE-CRUD | 31 |
| TRUNCATE TABLE | 32 |
| Integridad referencial. | 33 |
| Clave Primaria | 33 |
| Clave Foránea o referenciada | 36 |
| Uso de la Acción CASCADE | 38 |
| ¿Cómo identificar las claves foráneas? ¿En qué tabla se define una clave foránea? | 38 |
| Funciones de agrupamiento. | 39 |
| Operadores Relacionales | 40 |
| Operadores lógicos | 41 |

Conceptos básicos de bases de datos.

Para asegurar la mayor retención de los contenidos que en este curso se verán a continuación, es importante tener conocimientos básicos en teoría de bases de datos, he aquí un breve repaso.

Base de Datos:

Una base de datos es un “almacén” que nos permite guardar grandes cantidades de información de forma organizada para que luego podamos encontrarla y utilizarla fácilmente.

Más formalmente, una base de datos es una serie de datos organizados y relacionados entre sí, los cuales son recolectados y explotados por los sistemas de información de una empresa o negocio en particular.

Tabla:

Se refiere al tipo de modelado de datos, donde se guardan los datos recogidos por un programa. Su estructura general se asemeja a la vista general de un programa de Hoja de cálculo.

Las tablas se componen de dos estructuras:

- **Registro:** es cada una de las filas en que se divide la tabla. Cada registro contiene datos de los mismos tipos que los demás registros. Ejemplo: en una tabla de nombres y direcciones, cada fila contendrá un nombre y una dirección.
- **Campo:** es cada una de las columnas que forman la tabla. Contienen datos de tipo diferente a los de otros campos. En el ejemplo anterior, un campo contendrá un tipo de datos único, como una dirección, o un número de teléfono, un nombre, etc.

Diferencias entre registro y campo

Los campos y los registros son dos componentes básicos de una base de datos, que es una colección organizada de información, o datos. El término "campos" se refiere a columnas, o categorías verticales de datos. El término "registros" se refiere a las filas, o agrupaciones horizontales de datos de campo.

Modelo Entidad – Relación.

Los diagramas o modelos entidad-relación (denominado por su siglas, [ERD](#) "Diagram Entity relationship") son una herramienta para el modelado de datos de un sistema de información. Estos modelos expresan entidades relevantes para un sistema de información, sus inter-relaciones y propiedades.


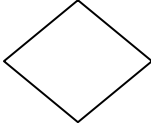
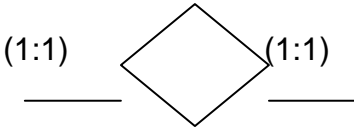
El modelo entidad-relación se definen Entidades y Relaciones, donde:

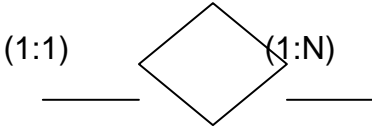
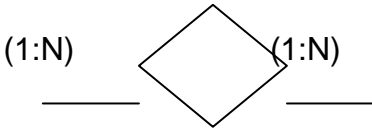
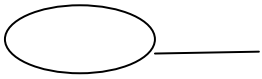
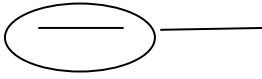
- Una **Entidad** es un objeto del mundo real sobre el que se quiere almacenar información (Ej: una persona). Las entidades están compuestas de *atributos* que son los datos que definen el objeto (para la entidad persona serían ci, nombre, apellidos, dirección,...). Entre los atributos habrá uno o un conjunto de ellos que no se repite; a este atributo o conjunto de atributos se le llama **clave** de la entidad, (para la entidad persona una clave sería CI).
- Un **Atributo** es aquel que define las propiedades de una entidad (Ej: los atributos de una persona son: nombre, apellido, edad, estatura, peso, etc).
- Una **Relación** es una asociación entre entidades. Las relaciones pueden ser de tres tipos:
 - ✓ **Relaciones uno a uno (1:1)**: Las entidades que intervienen en la relación se asocian una a una (Ej: la entidad HOMBRE, la entidad MUJER y entre ellos la

relación MATRIMONIO).

- ✓ **Relaciones uno a muchos (1:N):** Una ocurrencia de una entidad está asociada con muchas (n) de otra (Ej: la entidad EMPRESA, la entidad TRABAJADOR y entre ellos la relación TRABAJAR-EN).
- ✓ **Relaciones muchos a muchos (N:N).**-Cada ocurrencia, en cualquiera de las dos entidades de la relación, puede estar asociada con muchas (N) de la otra y viceversa (Ej: la entidad ALUMNO, la entidad ASIGNATURA y entre ellos la relación MATRÍCULA).

Representación gráfica del Modelo Entidad – Relación

| Símbolo | Significado |
|---|--------------------|
|  | Entidad |
|  | Relación |
|  | Relación uno a uno |

| | |
|--|---------------------------------|
|  | <p>Relación uno a muchos</p> |
|  | <p>Relación muchos a muchos</p> |
|  | <p>Atributo</p> |
|  | <p>Atributo Clave</p> |

SQL Estándar

SQL se ha convertido en el lenguaje de consulta relacional (se basa en el *modelo de datos relacional*) más popular. El nombre “SQL” es una abreviatura de *Structured Query Language* (Lenguaje de consulta estructurado).

Como en el caso de los más modernos lenguajes relacionales, SQL está basado en el cálculo relacional de tuplas. Como resultado, toda consulta formulada utilizando el cálculo relacional de tuplas (o su equivalente, el álgebra relacional) se puede formular también utilizando SQL.

En resumen, SQL nos permite realizar consultas a nuestras bases de datos para mostrar, insertar, actualizar y borrar datos.

Introducción a postgresSQL

El Sistema Gestor de Bases de Datos Relacionales Orientadas a Objetos conocido como PostgreSQL (y brevemente llamado Postgres95) está derivado del paquete Postgres escrito en Berkeley. Con cerca de una década de desarrollo tras él, PostgreSQL es el gestor de bases de datos de código abierto más avanzado hoy en día, ofreciendo control de concurrencia multi-versión, soportando casi toda la sintaxis SQL (incluyendo subconsultas, transacciones, y tipos y funciones definidas por el usuario), contando también con un amplio conjunto de enlaces con lenguajes de programación (incluyendo C, C++, Java, perl, tcl y python).

Características y Ventajas

Postgres ofrece una potencia adicional sustancial a Los sistemas de mantenimiento de Bases de Datos relacionales tradicionales (DBMS) al incorporar 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:

- 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*.

Además de haberse realizado corrección de errores, con PostgreSQL, el énfasis ha pasado a aumentar características y capacidades, aunque el trabajo continúa en todas las áreas. Algunas mejoras son:

- Los tipos internos han sido mejorados, incluyendo nuevos tipos de fecha/hora de rango amplio y soporte para tipos geométricos adicionales.
- Se han añadido funcionalidades en línea con el estándar SQL92, incluyendo claves primarias, identificadores entrecomillados, forzado de tipos cadena literales, conversión de tipos y entrada de enteros binarios y hexadecimales.
- La velocidad del código del motor de datos ha sido incrementada aproximadamente en un 20-40%, y su tiempo de arranque ha bajado el 80% desde que la versión 6.0 fue lanzada.
- Se han implementado importantes características del motor de datos, incluyendo subconsultas, valores por defecto, restricciones a valores en los campos (constraints) y disparadores (triggers).

PostgreSQL 9.x incorpora nuevas características y funciones avanzadas en materia de seguridad, soporte de aplicaciones, seguimiento y control, rendimiento y almacenamiento de datos especiales.

Otros Sistemas de Gestión de Bases de Datos

SGBD libres

- [Firebird](#)
- [SQLite](http://www.sqlite.org) (<http://www.sqlite.org>) Licencia Dominio Público
- [DB2 Express-C](http://www.ibm.com/software/data/db2/express/) (<http://www.ibm.com/software/data/db2/express/>)
- [Apache Derby](http://db.apache.org/derby/) (<http://db.apache.org/derby/>)
- [MariaDB](http://mariadb.org/) (<http://mariadb.org/>)
- [MySQL](http://dev.mysql.com/) (<http://dev.mysql.com/>)

SGBD no libres

- [MySQL](#): Licencia Dual, depende del uso.
- [dBase](#)
- [FileMaker](#)
- [Fox Pro](#)
- [IBM DB2](#): Universal Database (DB2 UDB)
- [Interbase](#)
- [Microsoft Access](#)
- [Microsoft SQL Server](#)
- [NexusDB](#)
- [Open Access](#)
- [Oracle](#)
- [WindowBase](#)

SGBD no libres y gratuitos

- [Microsoft SQL Server Compact Edition Basica](#)
- [Sybase ASE Express Edition para Linux](#) (edición gratuita para [Linux](#))
- [Oracle Express Edition 10](#) (solo corre en un servidor, capacidad limitada)

Instalación de PostgreSQL en Linux y Windows

Los pasos para instalar PostgreSQL en Linux o Windows son los siguientes:

- Descargar la versión del programa de instalación que corresponda a Linux o Windows desde el siguiente link: <http://www.enterprisedb.com/crossover-postgresql>
- Seleccionar la última versión.

Para el caso de **Linux**:

- Una vez descargado el programa abrir un terminal.
- Ubicarse en el directorio donde se descargó el programa (usualmente está en el directorio de descargas).
- Como el programa se grabó sin permisos de ejecución en Linux hay que definir este permiso antes de arrancar el programa, esto se hace de la siguiente manera:

```
chmod +x postgresql-9.x.x-x-linux.run  
sudo ./postgresql-9.x.x-x-linux.run
```

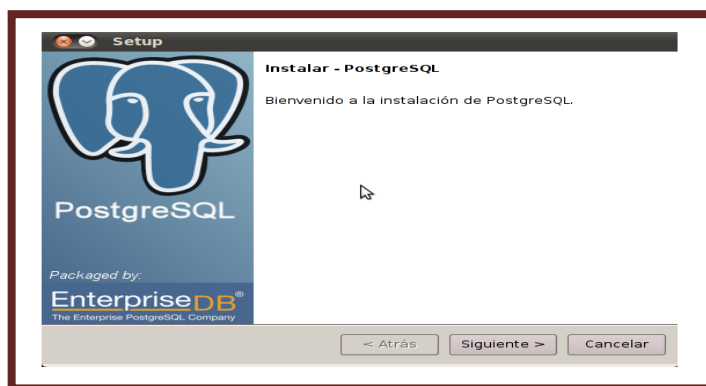
NOTA: la x dependerá de la versión que usted ha descargado, debe colocar exactamente el nombre del archivo.

Para el caso de **Windows**:

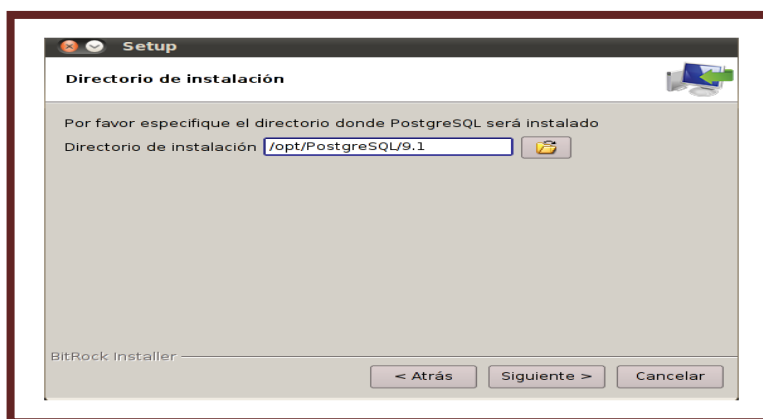
- Una vez descargado el programa hacer doble click sobre el archivo.exe.

Para **Windows** y **Linux**:

- La primera pantalla que se muestra es la bienvenida al instalador de PostgreSQL. A partir de ahora tendremos que pulsar "Siguiente" cada vez que queramos avanzar al siguiente paso:



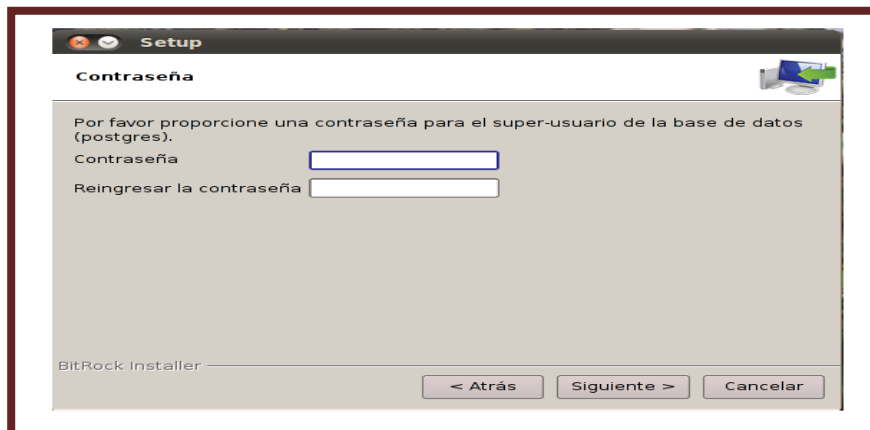
- En el siguiente paso tendremos que definir el directorio donde se van a instalar todos los programas que vienen con esta versión de PostgreSQL. En nuestro caso, utilizaremos el valor por defecto que el programa nos sugiere. /opt/PostgreSQL/9.x en Linux y c:\\Program Files\\PostgreSQL\\9.x en Windows.



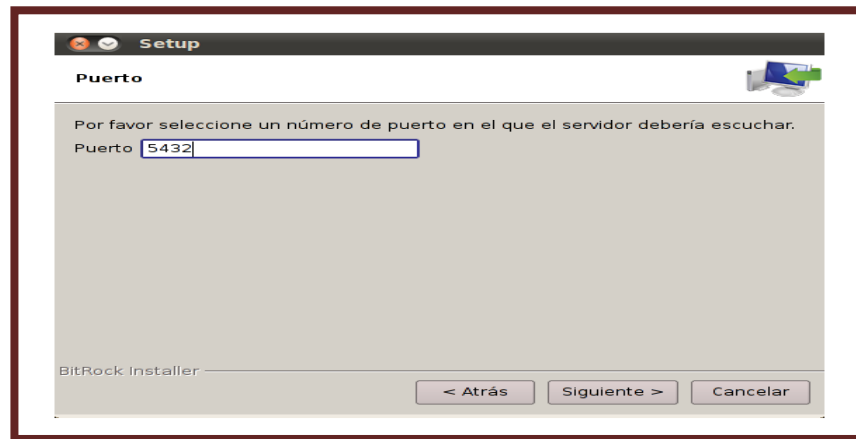
- En este paso tenemos que definir el directorio de datos en donde se crearán las bases de datos. De nuevo, en nuestro caso utilizaremos el valor por defecto que el programa nos sugiere. /opt/PostgreSQL/9.x/data en linux y c:\\Program Files\\PostgreSQL\\9.x\\data en Windows.



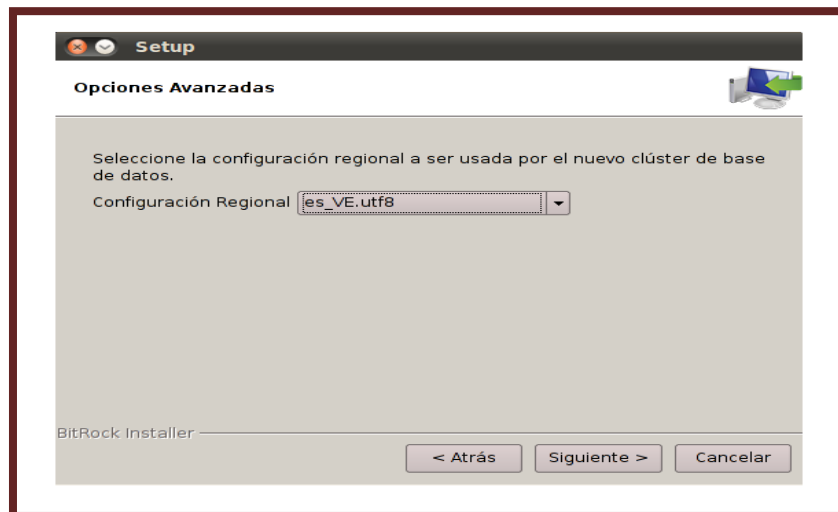
- En este paso hay que definir una clave de acceso para el usuario administrador de nuestra base de datos PostgreSQL.



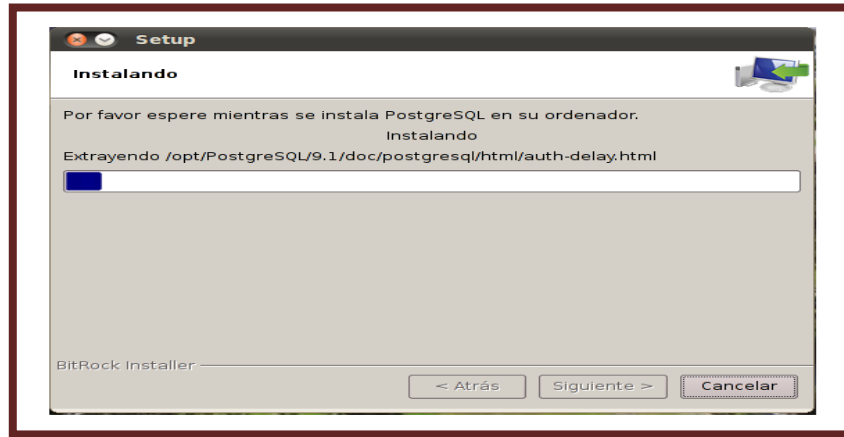
- Ahora hay que especificar el puerto que PostgreSQL utilizará para escuchar por conexiones. En nuestro caso dejamos el valor por defecto, 5432.



- Una vez que hemos terminado con los pasos básicos, el programa entra en la sección de opciones avanzadas. En este paso tenemos que decidir la 'Configuración Regional' que queremos utilizar:



- Pulsamos por última vez "Siguiente" y esperamos a que el programa termine de instalar todo:



- Una vez terminada la instalación, podremos salir del instalador pulsando "Terminar". En este último paso el instalador nos da la opción de arrancar automáticamente un programa llamado "Stack Builder". En nuestro caso no vamos a ver nada relacionado con "Stack Builder" y por eso borramos la elección de arrancarlo automáticamente antes de pulsar "Terminar".



Acceder a la consola de postgres (psql shell).

Caso Windows:

- Ubicarnos en la carpeta donde instalamos el archivo.
- Si hicimos la instalación por defecto estará dentro de la carpeta de nuestro disco duro en archivos de programa (Program Files x86).
- Luego buscamos la carpeta PostgreSQL e ingresamos a ella, seguidamente conseguiremos una carpeta de la versión que instalamos, ingresamos a ella y luego buscamos la carpeta llamada scripts.
- Una vez dentro de la carpeta scripts, le damos doble clic al archivo **runpsql**, y ya estaremos en la consola de comandos de postgres.
- Allí ingresaremos los datos que hemos configurado, en el caso de solo haber puesto una clave al momento de la instalación, no ingresar datos en las secciones especificadas a continuación, solo debemos pulsar la tecla enter:
 - server [localhost],
 - Database [postgres],
 - port [5432],
 - username [postgres].
- Al llegar a la sección de **contraseña para usuario postgres** solo debemos indicar la contraseña que al momento de la instalación definimos.

Caso Linux:

- Ingresamos a la terminal y ejecutamos el siguiente comando:

psql -U nombreUsuario -W -h iphost nombreBaseDeDatos

Parámetros:

- U** es el usuario de la base
 - W** mostrará el prompt de solicitud de password
 - h** IP del servidor de la base de datos en caso nos conectemos remotamente sino bastaría con poner localhost
-
- Si hicimos la instalación por defecto, solo tenemos el usuario postgres por lo tanto escribiríamos lo siguiente: **psql -U postgres -W -h localhost** presionamos enter y colocamos la contraseña que definimos al momento de la instalación.

Operaciones básicas en las Bases de datos y tablas.

Primero que nada veamos la siguiente lista de comandos básicos en postgresql.

\l ---> Te muestra las bases de datos existentes.

\d ----> Te muestra las relaciones (tablas, secuencias, etc.) existentes en la base de datos.

\d [nombre_tabla] ---> Para ver la descripción (nombre de columnas, tipo de datos, etc.) de una tabla.

\c [nombre_bd] ---> Para conectarte a otra base de datos.

SHOW search_path; ---> Para ver la ruta de búsqueda actual.

SET search_path TO [nombre_esquema]; ---> Para actualizar la ruta de búsqueda.

\q -----> Para salir de psql

Crear una base de datos.

Para crear una base de datos debemos escribir lo siguiente:

```
CREATE DATABASE nombreBD;
```

Ejemplo:

```
CREATE DATABASE curso_postgresql;
```

Eliminar Bases de Datos

Podemos eliminar una o muchas bases de datos con el comando DROP DATABASE utilizándolo de esta manera:

```
DROP DATABASE nombreDB;
```

Si deseamos borrar multiples bases de datos podrmos hacer lo siguiente:

```
DROP DATABASE nombreDB1,nombreDB2,.....,nombreDBn;
```

Lo podemos ver en el siguiente ejemplo:

```
DROP DATABASE curso_postgresql;
```

NOTA: Es importante el uso del parámetro IF EXISTS a la hora de ejecutar las consultas, ya que cuando intentas borrar una base de datos que no existe ocurrirá un error.

Para evitar este error podemos agregar este parámetro a la sentencia DROP y así hacer que la sentencia no conlleve a errores si existen o no las bases de datos que queremos borrar.

Ejemplo:

```
DROP DATABASE IF EXISTS nombreDB;
```

En este caso si la tabla existe la borra y si no existe el sistema te advierte que no existe la tabla, pero la consulta no arroja un error.

Renombrar una base de datos

Para renombrar una base de datos necesitamos alterarla de la siguiente manera:

```
ALTER DATABASE nombreBD RENAME TO nombreNuevoDB;
```

Ejemplo:

```
ALTER DATABASE curso_postgresql RENAME TO curso_pg;
```

Conectar con Bases de Datos Creadas.

Para poder usar o crear tablas dentro de una base de datos, primero debemos seleccionarla o conectarnos con ella usando el comando:

```
\c nombreBD;
```

Ejemplo:

```
\c curso_pg;
```

Tipos de datos soportados por postgresql.

| Nombre | Alias | Descripcion |
|-----------------------------|--------------------|---|
| bigint | int8 | entero con signo de ocho bytes |
| bigserial | serial8 | entero autoincremental de ocho bytes |
| bit [(n)] | | cadena de bits de longitud fija |
| bit varying [(n)] | varbit | cadena de bits de longitud variable |
| boolean | bool | Booleano lógico (verdadero/falso) |
| box | | rectángulo en un plano |
| bytea | | datos binarios ("arreglo de bytes") |
| character varying [(n)] | varchar [(n)] | cadena de caracteres de longitud variable |
| character [(n)] | char [(n)] | cadena de caracteres de longitud fija |
| cidr | | dirección de red IPv4 o IPv6 |
| circle | | circulo en un plano |
| date | | fecha de calendario (año, mes, día) |
| double precision | float8 | número de punto flotante de precisión doble (8 bytes) |
| inet | | dirección de equipo de IPv4 o IPv6 |
| integer | int, int4 | entero con signo de cuatro bytes |
| interval [fields] [(p)] | | lapso de tiempo |
| line | | linea infinita en un plano |
| lseg | | segmento de linea en un plano |
| macaddr | | Dirección MAC (Media Access Control) |
| money | | importe monetario |
| numeric [(p, s)] | decimal [(p, s)] | numérico exacto de precisión seleccionable |
| path | | camino geométrico en un plano |
| point | | punto geométrico en un plano |
| polygon | | camino cerrado geométrico en un plano |
| real | float4 | número de punto flotante de precisión |

| | | |
|---|------------|---|
| | | simple (4 bytes) |
| smallint | int2 | entero con signo de dos bytes |
| serial | serial4 | entero autoincremental de cuatro bytes |
| text | | cadena de caracteres de longitud variable |
| time [(p)] [without time zone] | | hora del día (sin zona horaria) |
| time [(p)] with time zone | timetz | hora del día, incluyendo zona horaria |
| timestamp [(p)] [without time zone] | | fecha y hora (sin zona horaria) |
| timestamp [(p)] with time zone | timestampz | fecha y hora, incluyendo zona horaria |
| tsquery | | consulta de búsqueda de texto |
| tsvector | | documento de búsqueda de texto |
| txid_snapshot | | instantánea de ID de transacción a nivel de usuario |
| uuid | | identificador universalmente único |
| xml | | datos XML |

Crear una Tabla.

La sintaxis general para crear una tabla es la siguiente:

```
CREATE TABLE nombre_tabla(
    nombre_campo1 tipo_de_dato,
    ...
    nombre_campoN tipo_de_dato
);
```

Describir la estructura de una tabla:

Con el comando `\d nombreTabla` podemos saber la estructura de la tabla, el nombre de sus campos y tipos de datos asociados, ejemplo:

```
\d nombre_tabla;
```

Alteración básica de tablas.

La sintaxis básica de **ALTER TABLE** para añadir una nueva columna en una tabla existente es la siguiente:

```
ALTER TABLE nombre_tabla ADD nombre_columna tipo_de_dato;
```

La sintaxis básica de **ALTER TABLE** para **DROP COLUMN** en una tabla existente es la siguiente:

```
ALTER TABLE nombre_tabla DROP COLUMN nombre_columna;
```

La sintaxis básica de **ALTER TABLE** para cambiar el **TIPO DE DATOS** de una columna de una tabla es la siguiente:

```
ALTER TABLE nombre_tabla MODIFY COLUMN nombre_columna tipo_de_dato;
```

La sintaxis básica de **ALTER TABLE** para añadir un **NOT NULL** restricción a una columna de una tabla es la siguiente:

```
ALTER TABLE nombre_tabla MODIFY nombre_columna tipo_de_dato NOT NULL;
```

Operaciones elementales en las bases de datos (CRUD).

CRUD-CREATE

CREATE es la sentencia utilizada para la creación de tablas dentro de una base de datos, corresponde a la primera de las acciones elementales de las bases de datos siendo la letra C la que la identifica dentro de la palabra CRUD.

Crear una Tabla

La sintaxis general para crear una tabla es la siguiente:

```
CREATE TABLE nombre_tabla(  
    nombre_campo1 tipo_de_dato,  
    ...  
    nombre_campoN tipo_de_dato  
);
```

Ejemplo:

```
CREATE TABLE cliente (  
    cedula varchar(15),  
    nombre varchar(50),  
    apellido varchar(50),  
    direccion varchar(100)  
);
```

Consideraciones:

- Cada campo con su tipo debe separarse con comas, excepto el último.
- Cuando se crea una tabla debemos indicar su nombre y definir al menos un campo con su tipo de dato.
- Los nombres de tablas pueden utilizar cualquier caracter alfabético o numérico, el primero debe ser un caracter alfabético y no puede contener espacios en blanco.
- Si intentamos crear una tabla con cuyo nombre ya existe mostrará un mensaje indicando que ya hay una tabla con ese nombre en la base de datos.

NOTA:

Cuando tenemos que crear una tabla debemos pensar en sus campos y en el tipo de datos que los representará.

Podemos crear una tabla de prueba para poner en práctica algunos de los tipos de datos más utilizados en la actualidad.

```
CREATE TABLE prueba (  
    campoBool bool,  
    campoChar char(5),  
    campoMoney money,  
    campoDate date,  
    campoTime time,  
    campoTimestamp timestamp,  
    campoReal real,  
    campoText text  
);
```


En el caso de crear una tabla para empleados:

```
CREATE TABLE empleados (  
    cedula INT,  
    nombre VARCHAR(30),  
    apellido VARCHAR(30),  
    cargo VARCHAR(30),  
    tiempo_servicio INT  
);
```

Insertar datos en una tabla

Permite agregar una o más filas a una tabla en una base de datos relacional. La sintaxis básica es la siguiente:

```
INSERT INTO nombretabla(nombre_campo1,..., nombre_campoN)  
VALUES(valor_campo1,...,valor_campoN);
```

Veamos por partes que quiere decir cada una de las partes que conforman la sentencia:

| | Significado |
|--------|---|
| INSERT | Palabra clave que indica que la sentencia de SQL que se quiere insertar datos en la Base de Datos. |
| INTO | Indica la tabla en la que se quiere insertar los datos, señalando cada uno de los campos de la tabla. |
| VALUES | Indica los valores que se van a insertar para cada campo. |

Notas:

- Los datos que corresponden a cadenas de caracteres se colocan entre comillas simples.
- Es importante ingresar los valores en el mismo orden en que se nombran los campos.
- Si el valor de una columna no se inserta se le asignará un valor nulo por defecto.

Un registro es una fila de la tabla que contiene los datos propiamente dichos. Cada registro tiene un dato por cada columna.

Al ingresar los datos de cada registro debe tenerse en cuenta la cantidad y el orden de los campos.

Ahora vamos a agregar un registro a la tabla:

```
INSERT INTO prueba (campoBool, campoChar, campoMoney, campoDate, campoTime,
campoTimestamp, campoReal, campoText) VALUES
(true,'hola',2000,'12.12.12','01:12:23',current_timestamp,23.4,'texto variable');
```

También podemos insertar varios registros en una sola sentencia como vemos a continuación:

```
INSERT INTO empleados (cedula, nombre, apellido, cargo, tiempo_servicio) VALUES
(20345666,'Mario','Perez','Administrador',4),
(12345632,'Maria','Ortiz','Computista',14),
(9345728,'Juana','Cabrera','Contadora',8)
;
```

Usamos “INSERT INTO”. Especificamos los nombres de los campos entre paréntesis y separados por comas y luego los valores para cada campo, también entre paréntesis y separados por comas.

Es importante ingresar los valores en el mismo orden en que se nombran los campos, si ingresamos los datos en otro orden, nos aparece un mensaje de error y los datos se guardan de modo incorrecto o simplemente no llegan a guardarse.

Note que los datos ingresados, como corresponden a campos de cadenas de caracteres se colocan entre comillas simples. Las comillas simples son OBLIGATORIAS, mientras que los datos numéricos podemos escribirlos sin o con comillas.

Con la sentencia SELECT podemos seleccionar los datos de una tabla, esta sentencia se explicara con más detalle en la sección siguiente.

```
SELECT * FROM prueba;
```

```
SELECT * FROM empleados;
```

CRUD-READ

En este tema hablaremos de la sintaxis de la sentencia Select y uso básico de la misma, que nos permite seleccionar registros que cumplan con algunas características que especifiquemos en dicha sentencia. Corresponde a la segunda sigla (la R) de la abreviatura CRUD cuyo significado es READ.

La sentencia SELECT permite consultar los datos almacenados en una tabla de la base de datos. La sintaxis básica es la siguiente:

```
SELECT * FROM nombretabla;
```

Veamos por partes que quiere decir cada una de las partes que conforman la sentencia:

| | Significado |
|--------|--|
| SELECT | Palabra clave que indica que la sentencia de SQL que se quiere ejecutar es de selección. |
| FROM | Indica la tabla (o tablas) desde la que queremos recuperar los datos. |
| * | El asterisco (*) indica que se seleccionan todos los campos de la tabla. |

Notas:

- Se puede especificar el nombre de los campos que se quieren ver separándolo por comas, por ejemplo:

```
select titulo,autor from libros;
```

- Los datos aparecen ordenados según la lista de selección.

Podemos especificar el nombre de los campos de la tabla empleados que queremos ver separándolos por comas:

```
SELECT nombre, apellido FROM empleados;
```

En la sentencia anterior la consulta mostrará sólo los campos "nombre" y "apellido". En la siguiente sentencia, veremos los campos correspondientes al cargo y cedula de todos los empleados de una tabla creada con esas características:

```
SELECT cargo, cedula FROM empleados;
```

Como podemos ver podemos decidir incluso el orden en que se mostrarán los campos solicitados.

Hemos aprendido cómo ver todos los registros de una tabla. Pero podemos aplicar ciertas condiciones para realizar una búsqueda más específica.

Existe una cláusula, "WHERE" que es opcional, con ella podemos especificar condiciones para la consulta "SELECT". Es decir, podemos recuperar algunos registros, sólo los que cumplan con ciertas condiciones indicadas con la cláusula "WHERE". Por ejemplo, queremos ver el usuario cuyo nombre es "Mario", para ello utilizamos "WHERE" y luego de ella, la condición:

```
SELECT nombre, apellido, cargo FROM empleados WHERE nombre='Mario';
```

Para las condiciones se utilizan operadores relacionales. El signo igual (=) es un operador relacional. Para la siguiente selección de registros especificamos una condición que solicita los usuarios cuya cedula es igual a 17675823:

```
SELECT nombre, apellido, cargo FROM empleados WHERE cedula=17675823;
```

Si ningún registro cumple la condición establecida con el "WHERE", no devolverá ningún resultado.

CRUD- UPDATE

En este tema explicaremos el uso de la sentencia Update que nos permite actualizar datos de uno o más registros en tablas, esta sentencia que modifica datos existentes es usada con mucha frecuencia y suele combinarse con otras palabras reservadas para lograr su objetivo. Representa la letra U de la palabra CRUD.

La sintaxis básica es la siguiente:

```
UPDATE nombre_tabla SET nombre_campo:'nuevo_valor';
```

Veamos por partes que quiere decir cada una de las partes que conforman la sentencia:

| | Significado |
|--------|---|
| UPDATE | Palabra clave que indica que la sentencia de SQL que se quiere ejecutar es de actualizar. |
| SET | Indica el campo o campos que se quieren actualizar. |

Para modificar uno o varios datos de uno o varios registros utilizamos "UPDATE" (actualizar). Por ejemplo, en nuestra tabla "empleados", queremos cambiar los valores de todos los años de servicio, por "6":

```
UPDATE empleados SET tiempo_servicio=6;
```

Utilizamos "UPDATE" junto al nombre de la tabla y "set" junto con el campo a modificar y su nuevo valor.

El cambio afectará a todos los registros.

Podemos modificar algunos registros, para ello debemos establecer condiciones de selección con "WHERE".

Por ejemplo, queremos cambiar el valor correspondiente al tiempo de servicio del empleado llamado 'Mario', queremos como nuevo tiempo de servicio "4", necesitamos una condición "WHERE" que afecte solamente a este registro:

```
UPDATE empleados SET tiempo_servicio=4 WHERE nombre='Mario';
```

Si no encuentra registros que cumplan con la condición del "WHERE", ningún registro es afectado.

Las condiciones no son obligatorias, pero si omitimos la cláusula "WHERE", la actualización afectará a todos los registros.

También se puede actualizar varios campos en una sola Instrucción:

```
UPDATE empleados SET nombre='Marcelo', tiempo_servicio=5 WHERE nombre='Mario';
```

Para ello colocamos "UPDATE", el nombre de la tabla, "SET" junto al nombre del campo y el nuevo valor y separado por coma, el otro nombre del campo con su nuevo valor.

DELETE-CRUD

El uso de la sentencia Delete que nos permite borrar filas de uno o más registros en tablas, esta sentencia que borra datos existentes es usada con mucha frecuencia y suele combinarse con otras palabras reservadas para lograr su objetivo. Explicaremos la diferencia en el uso de sentencias similares como DROP y TRNCATE Esta sentencia representa la letra D de la palabra CRUD.

Borra uno o más registros existentes en una tabla.. La sintaxis básica es la siguiente:

```
DELETE FROM nombre_tabla;
```

Veamos por partes que quiere decir cada una de las partes que conforman la sentencia:

| | Significado |
|--------|---|
| DELETE | Palabra clave que indica que la sentencia de SQL que se quiere ejecutar es de borrar. |
| FROM | Indica la tabla (o tablas) desde la que queremos borrar los datos. |

Nota:

- La instrucción anterior elimina todos los registros de la tabla.

Si sólo se quiere eliminar algunas filas con un valor en específico, se utilizaría la cláusula 'WHERE' con la que se establece la condición que debe cumplir para borrar las filas. Por ejemplo: Si se quiere eliminar todas las filas cuyo nombre de usuario es "Marcelo", entonces se escribiría lo siguiente:

```
DELETE FROM usuarios  
WHERE nombre='Marcelo';
```


Notas:

- Si solicitamos el borrado de un registro que no existe, es decir, ningún registro cumple con la condición especificada, ningún registro será eliminado.
- Si no colocamos una condición, se eliminan todos los registros de la tabla nombrada.

TRUNCATE TABLE

Aprendimos que para borrar todos los registros de una tabla se usa "DELETE" sin condición "WHERE".

También podemos eliminar todos los registros de una tabla con "TRUNCATE TABLE". Por ejemplo, queremos vaciar la tabla "proveedores", usamos:

```
TRUNCATE TABLE empleados;
```

La sentencia "TRUNCATE TABLE" vacía la tabla (elimina todos los registros) y vuelve a crear la tabla con la misma estructura.

La diferencia con "DROP TABLE" es que esta sentencia borra la tabla, "TRUNCATE TABLE" la vacía.

La diferencia con "DELETE" es la velocidad, es más rápido "TRUNCATE TABLE" que "DELETE" (se nota cuando la cantidad de registros es muy grande) ya que éste borra los registros uno a uno.

Otra diferencia es la siguiente: cuando la tabla tiene un campo "AUTO_INCREMENT", si borramos todos los registros con "DELETE" y luego ingresamos un registro, al cargarse el valor en el campo autoincrementable, continúa con la secuencia teniendo en cuenta el valor mayor que se había guardado; si usamos "TRUNCATE TABLE" para borrar todos los registros, al ingresar otra vez un registro, la secuencia del campo autoincrementable vuelve a iniciarse en 1.

Por ejemplo, tenemos la tabla "cuadernos" con el campo "codigo" definido "serial", y el valor más alto de ese campo es "5", si borramos todos los registros con "DELETE" y luego ingresamos un registro sin valor de código, se guardará el valor "6"; si en cambio, vaciamos la tabla con "TRUNCATE TABLE", al ingresar un nuevo registro sin valor para el código, iniciará la secuencia en 1 nuevamente.

Integridad referencial

Es un conjunto de reglas que utilizan las Bases de Datos para asegurar que las filas (registros) de las tablas relacionadas son válidos y que no se borren o cambien datos relacionados de forma accidental produciendo errores. Gracias a la integridad referencial se garantiza que un registro siempre se relacione con otras entidades válidas, es decir, que existen en la base de datos.

Clave Primaria

Una clave primaria es una clave que ha sido diseñada para identificar de manera única a los registros de una tabla. La selección de una clave primaria es muy importante en el diseño de una base de datos, ya que es un elemento clave de los datos que facilita la unión de tablas y el concepto total de una base de datos relacional. Las claves primarias deben ser únicas y no nulas. Si tenemos por ejemplo la tabla 'Libros' cuyos atributos son: código, título, autor y precio; vemos que puede establecerse el código como clave primaria, ya que, su valor no se repite.

La sintaxis para definir una clave primaria es la siguiente:

```
CREATE TABLE nombre_tabla(  
    nombre_campo1 tipo_de_dato,  
    ...  
    nombre_campoN tipo_de_dato,  
    primary key (nombre_campo)  
  
);
```

Por ejemplo: Tomando en cuenta la tabla 'Libros' mencionada anteriormente y quisiéramos asegurarnos que cada Libro tendrá un código único y diferente lo definiríamos de la siguiente manera:

```
CREATE TABLE libros(  
    codigo varchar(10),  
    titulo varchar(40),  
    autor varchar(20),  
    precio decimal(4,2),  
    primary key(codigo)  
);
```

Como vemos, lo que se hace para definir una clave primaria es agregar 'primary key' y entre paréntesis el nombre del campo que será la clave. Una tabla sólo puede tener una clave primaria. Cualquier campo (de cualquier tipo) puede ser clave primaria, debe cumplir como requisito, que sus valores no se repitan ni sean nulos.

Por eso se debe tener mucho cuidado a la hora de insertar datos en una clave primaria.

Cuando no llevamos cuenta de la cantidad de registros que insertamos, es necesario que la clave primaria sea auto incremental y entera.

Esta característica la posee de manera implícita el tipo de dato SERIAL el cual es muy usado como tipo de dato para las claves primarias

En este ejemplo vamos a crear una tabla con una clave primaria así como también haremos una inserción en la misma.

```
CREATE TABLE cuentos(  
    codigo serial,  
    titulo varchar(40),  
    autor varchar(20),  
    precio money,  
    primary key(codigo)  
);
```

A la hora de insertar datos en esta tabla no debemos preocuparnos por insertar datos en el campo de código porque corresponde al tipo de dato serial.

```
INSERT INTO cuentos (titulo,autor,precio) VALUES ('La noche de los feos','Mario  
Benedetti',3000);
```

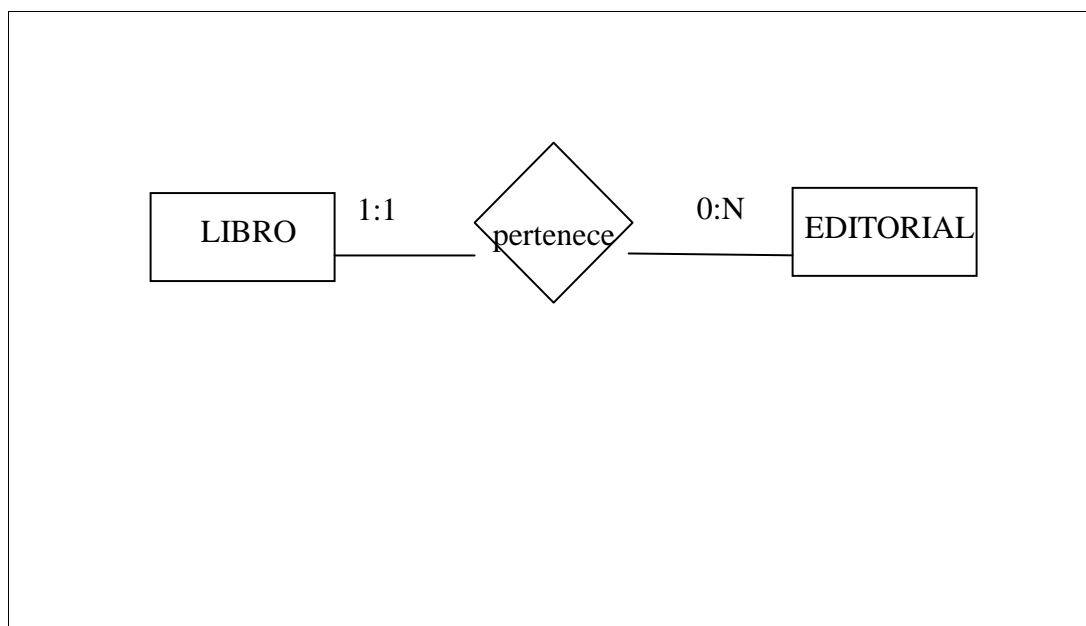
Si seleccionamos los datos de la tabla cuentos nos daremos cuenta que el sistema automáticamente colocó un valor automático al campo código que es una clave primaria sin necesidad de nosotros insertarlo.

```
SELECT * FROM cuentos;
```

Clave Foránea o referenciada

Una clave foránea es simplemente un campo en una tabla que se corresponde con la clave primaria de otra tabla.

Por ejemplo: Si al ejemplo de la tabla 'Libros' le agregamos las editoriales de los libros cuyos campos son: código y nombre; pudieramos decir que la tabla 'Libros' puede tener una clave foránea llamada 'codEditorial' para enlazar los libros con las editoriales. Gráficamente sería así:



La sintaxis para definir una clave foránea es la siguiente:

```
CREATE TABLE nombre_tabla(  
    nombre_campo1 tipo_de_dato,  
    ...  
    nombre_campoN tipo_de_dato,  
    primary key (nombre_campo),
```

```
foreign key(nombre_campo) references tabla (campo)

);
```

Por ejemplo: Tomando en cuenta la tabla 'Libros' con sus atributos y la tabla 'Editorial' con sus atributos, definiríamos en Postgres9.x la relación de la siguiente manera:

| | |
|---|---|
| <pre>CREATE TABLE editorial(idEditorial varchar(10), nombre varchar(40), primary key(idEditorial));</pre> | <pre>CREATE TABLE libros(codigo varchar(10), titulo varchar(40), autor varchar(20), precio numeric(4,2), codEditorial varchar(10), primary key(codigo), foreign key (codEditorial) references editorial (idEditorial));</pre> |
|---|---|

Las claves foráneas y las claves primarias deben ser del mismo tipo para poder enlazarse. En conclusión, una clave foránea es un campo empleado para enlazar datos de 2 tablas.

Uso de la Acción CASCADE

Esta acción borra o actualiza automáticamente todas las referencias activas. Para borrar o actualizar los registros se requiere que la tabla que utilice esta acción cuente con claves foráneas. Por ejemplo:

```
CREATE TABLE libros(  
    codigo serial,  
    titulo varchar(40),  
    autor varchar(20),  
    precio numeric(4,2),  
    codEditorial serial,  
    primary key(codigo),  
    foreign key (codEditorial) references editorial (idEditorial)  
    ON UPDATE CASCADE ON DELETE CASCADE  
);
```

¿Cómo identificar las claves foráneas? ¿En qué tabla se define una clave foránea?

Para cada entidad del esquema se creará una tabla con tantos campos como atributos tenga la entidad. Un atributo será clave foránea cuando se requiera tener información asociada de otra tabla.

Cuando traducimos el modelo ER a tablas en Postgres9.1, las relaciones nos ayudarán a identificar donde debe ir un campo con clave foránea. Las relaciones son las siguientes:

- Relación 1-1 se pueden reflejar incluyendo en una de las dos tablas un campo en el que poder colocar la clave del elemento de la otra tabla con el que se está relacionado. Ese nuevo campo sería una clave foránea.
- Relación **1-N** se representan de forma muy parecida a como se ha explicado para las relaciones 1-1. La diferencia está en que ahora no es indiferente donde se coloque la clave foránea, esta debe estar obligatoriamente en la tabla de 'uno' (1); y además, para este caso si se permitirá que haya valores repetidos en dicho campo.
- Para representar la Relación **N-N** en tablas lo que se hace es crear una nueva tabla solamente para la relación. Esta nueva tabla tendrá dos claves foráneas y su propia clave estará formada por la unión de las claves foráneas.

Funciones de agrupamiento

Son funciones que se utilizan para determinar estadísticas relacionadas con un conjunto de valores. Dentro de las más utilizadas se pueden nombrar:

MAX (Valor máximo)

Devuelve el valor más alto de un campo seleccionado. La sintaxis es la siguiente:

```
SELECT MAX(campo) FROM nombre_tabla;
```

MIN (Valor mínimo)

Devuelve el valor más pequeño de un campo seleccionado. La sintaxis es la siguiente:

```
SELECT MIN(campo) FROM nombre_tabla;
```


SUM (Sumas o Totales)

Devuelve la suma de un conjunto de valores de un campo específico de la tabla. La sintaxis es la siguiente:

```
SELECT SUM(campo) FROM nombre_tabla;
```

COUNT

Devuelve el número de filas que cumplen con una consulta. La sintaxis es la siguiente:

```
SELECT COUNT(campo) FROM nombre_tabla;
```

AVG

Retorna el valor promedio de un campo especificado.

```
SELECT AVG (campo) FROM nombre_tabla;
```

Operadores lógicos

Los operadores lógicos son aquellos que permiten establecer una combinación de condiciones en una consulta a la Base de Datos. Dentro de los utilizados en Postgres9.1 se pueden nombrar:

- AND, significa "y",
- OR, significa "o",
- NOT, significa "no".

Por ejemplo: Si queremos mostrar todos los libros cuyo autor sea igual a "Borges" y cuyo precio no supere los 20 bolívares, necesitamos 2 condiciones:

```
SELECT * from libros WHERE (autor='Borges') AND (precio<=20);
```

Operadores Relacionales

Los operadores relacionales (o de comparación) permiten comparar dos expresiones, que pueden ser valores de campos. Hemos utilizado condiciones de igualdad para seleccionar registros de una tabla, por ejemplo:

```
SELECT * FROM libros WHERE autor = 'Borges';
```

En el ejemplo anterior se utiliza el operador relacional de igualdad. Los operadores relacionales vinculan un campo con un valor para que PostgreSQL9.1 compare cada campo especificado con un valor dado. Los operadores relacionales son los siguientes:

| Operador | Significado |
|----------|---------------|
| = | igual |
| <> | distinto |
| > | mayor |
| < | menor |
| >= | mayor o igual |
| <= | menor o igual |

Se pueden seleccionar también, por ejemplo, los registros cuyo autor sea diferente de "Borges", para ello usamos la condición:

```
SELECT * FROM libros WHERE autor <> 'Borges';
```

Se pueden comparar valores numéricos. Por ejemplo, si se quieren mostrar los títulos y precios de los libros cuyo precio sea mayor a 20 bolívares:

```
SELECT titulo, precio FROM libros WHERE precio > 20;
```