Base de Datos I
Curso No: TWB22B

Volumen 5
Fundamentos de PostgreSQL

Unidad 1
Fundamentos de PostgreSQL

Objetivos del Aprendizaje

- Listar las capacidades de PostgreSQL.
- Describir la licencia BSD.
- Describir copyleft
- Describir la historia de PostgreSQL y su evolución.
- Listar las principales características de PostgreSQL.
- Describir los niveles de jerarquía en PostgreSQL.
- Discutir sobre la arquitectura de PostgreSQL.
- Describir el funcionamiento de PostgreSQL.
- Describir los catálogos del sistema de PostgreSQL
- Explicar como es implementada la seguridad en PostgreSQL.

Objetivos del Aprendizaje

- Describir el sistema de privilegios de PostgreSQL.
- Listar las similitudes y diferencia entre PostgreSQL y MySQL
- Describir el cliente psql

Introducción

- PostgreSQL es el más avanzado sistema administrador de base de datos de código abierto (open source).
- Tiene todas las características de un sistema comercial, soporta funcionalidades avanzadas como: Vistas, Procedimientos Almacenados y Disparadores (triggers).
- Tiene un sofisticado mecanismo de seguridad que impide el acceso no autorizado a la información.
- PostgreSQL es licenciado bajo BSD, esto lo hace libre para todo uso, incluyendo comercial.

Licencia BSD

- BSD son las iniciales de Versión de Software Berkeley (Berkeley Software Distribution), es un proyecto desarrollado por la Universidad de California en Berkeley.
- BSD es comparable a GPL en el sentido de pretender garantizar la libertad de compartir y modificar software libre, aunque existe una diferencia en el sentido que esta licencia permite quitar la libertad al software; es decir, aprovechar un trabajo de BSD y convertirlo en software comercial cerrado, quitando las libertades originales.
- La licencia BSD no ofrece ninguna garantía de funcionamiento, en si, garantías expresas o implícitas están negadas.

Copyleft

- Software libre cuyos términos de distribución no permiten a los re-distribuidores agregar restricciones a cada copia del software, aun modificado debe ser libre.
- Cualquiera que redistribuye con o sin cambios, debe dar la libertad de copiarlo y modificarlo.
- Cuando se habla de software libre, se refiere a libertad del código fuente, no al precio.

Historia de PostgreSQL

- La historia de PostgreSQL comienza como un proyecto de investigación en 1980 en la universidad de Berkeley, el proyecto original fue un manejador de base de datos llamado Ingres.
- Desde 1982 hasta 1985 el proyecto es abandonado.
- En 1985 se retoma el proyecto rebautizándolo postgres (post-ingres).
- El proyecto se dio por finalizado con la versión 4.2, debido al gran auge que estaba teniendo, lo cual causó la imposibilidad de mantenimiento por parte de los desarrolladores.
- En 1994 dos estudiantes Andrew Yu y Jolly Chen reescribieron gran parte del código de postgres, sustituyeron el lenguaje POSTQUEL y añadieron un intérprete de SQL.

Historia de PostgreSQL

- Esta nueva versión se lanzo en 1995 como Postgres95, la cual fue liberada como un proyecto libre (Open Source).
- En 1996, después de estudios realizados se decidió cambiar el nombre, por uno más comercial PosgreSQL, este se obtuvo a partir de la combinación de postgres y SQL.
- PostgreSQL es un sistema administrador de bases de datos basado en el modelo relacional, aunque incorpora algunos conceptos del paradigma Orientado a Objetos, tales como la herencia, soporta el estándar SQL como lenguaje de consulta y está implementado siguiendo la arquitectura cliente-servidor.

Características de PostgreSQL

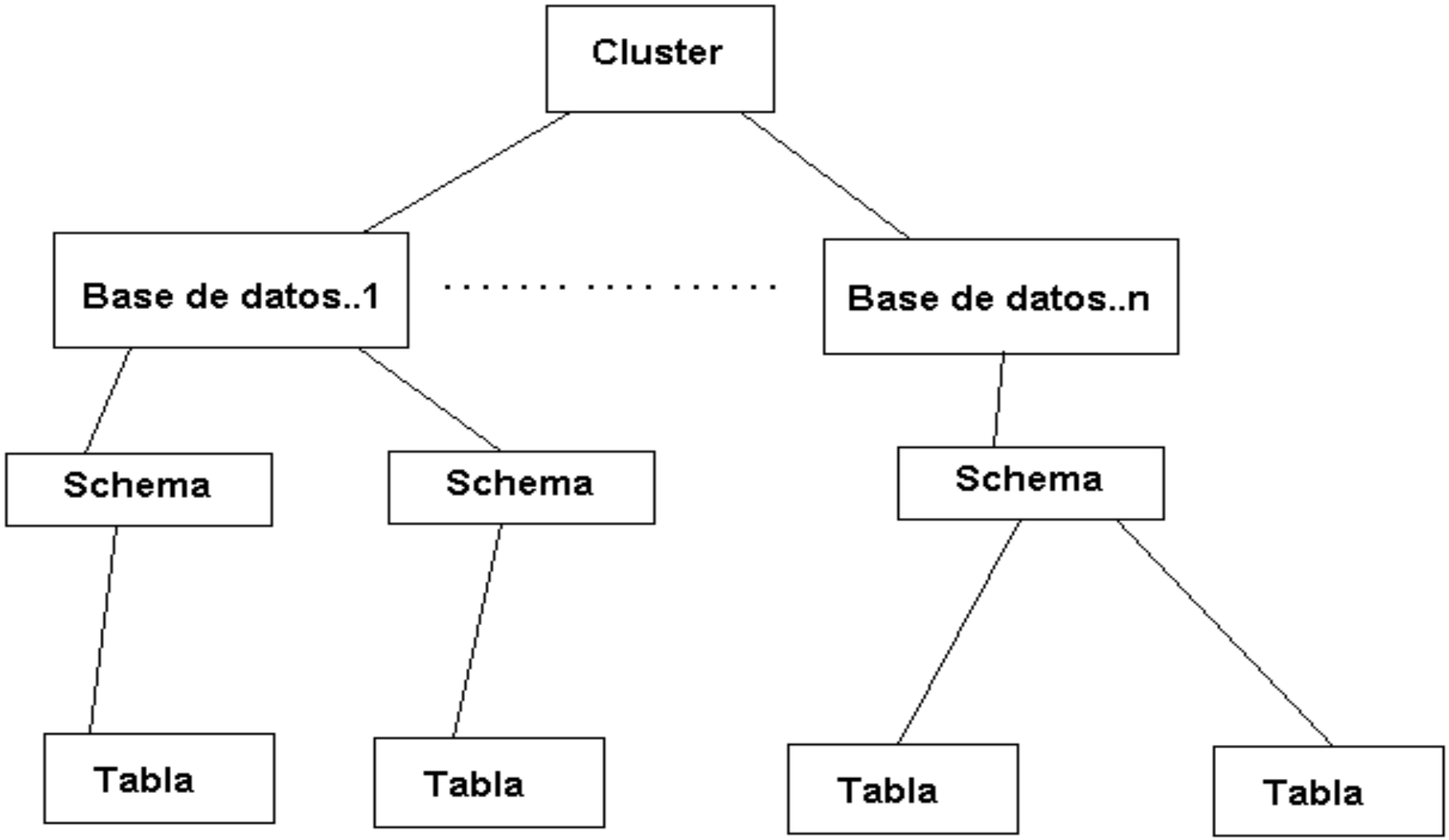
- Sistema administrador de base de datos Objeto-Relacional.
- Extensibilidad
- Soporta el estándar SQL
- Integridad Referencial
- Múltiples API's disponibles
- Lenguajes Procedurales
- MVCC (Multi-Version Concurrency Control)
- Cliente/Servidor
- Write Ahead Logging (WAL)(Escritura a continuación del registro

Características de PostgreSQL

Características Operacionales:

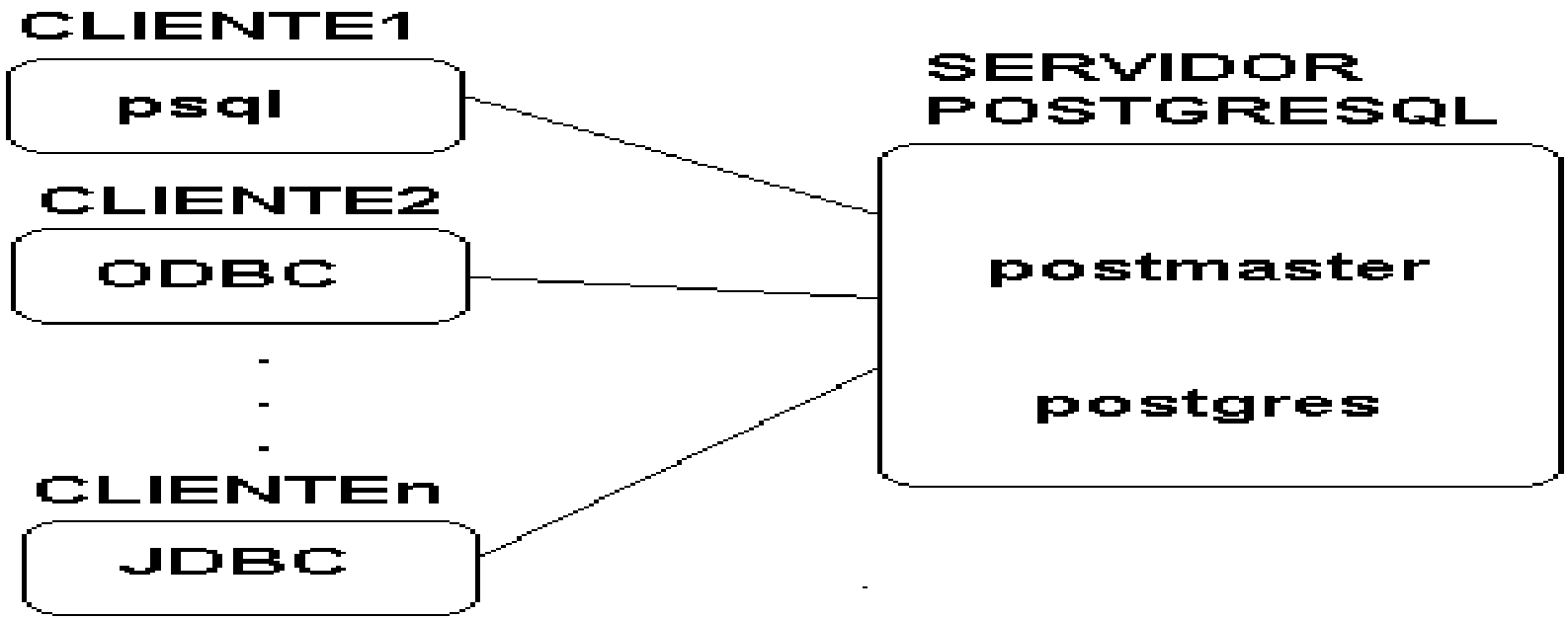
- Transacciones (Cumple completamente con ACID).
- Disparadores (Triggers).
- Restricciones (Constrains), Integridad Referencial.
- Respaldo y Recuperación (Backup & Recovery).
- Uniones externas. (Outer Joins).
- Sintaxis ANSI SQL 89, 92 y 99.
- Orientado a Objetos
- Soporte para Unicode
- Sub-consultas
- Soporte para consultas con UNION y UNION ALL.
- Índices parciales
- Herramientas que permiten generar SQL portable, para compartir con otros sistemas compatibles con SQL

Niveles de jerarquía en PostgreSQL



Arquitectura de PostgreSQL

- PostgreSQL esta implementado sobre una arquitectura cliente/servidor.
- Está dividido en tres grandes procesos: postmaster, postgres (backend) y cliente (frontend)



Funcionamiento de PostgreSQL

- En la arquitectura de funcionamiento de postgreSQL un proceso cliente se conecta exactamente a un proceso servidor
- PostgreSQL implementa un proceso “supervisor” llamado postmaster que corre en segundo plano esperando que surjan peticiones, normalmente hay un postmaster en ejecución en cada máquina, aunque puede haber varios en ejecución en una misma máquina
- Siempre que un pedido de conexión es detectado el proceso “supervisor” crea y asigna un nuevo proceso llamado postgres que es el encargado de manejar y dar respuesta a las peticiones del proceso cliente
- Si existen peticiones de consultas SQL concurrentes un proceso postgres es creado por cada una ellas, el proceso postmaster es el encargado de iniciar estos procesos

¿Porque seleccionar a PostgreSQL?

- PostgreSQL es un sistema administrador de base de datos de código libre “open source”
- Comparable en funcionamiento con sistemas comerciales de la talla de Oracle, Sybase o DB2, es usado en grandes organizaciones, universidades y en aplicaciones de misión critica
- Posee una gran escalabilidad, manejo de transacciones, soporte para disparadores (triggers) y procedimientos almacenados
- El soporte a estándares de SQL esta garantizado con SQL99 y SQL92, entre otros
- Por ser software libre permite que sea instalado en muchos lugares, es decir la instalación seria ilimitada, ya que no se requiere licencias adicionales de instalación.

¿Porque seleccionar a PostgreSQL?

- Modelos de negocios más rentables, ahorros considerables en costos de operación
- No existe la posibilidad de ser auditado para verificar cumplimiento de licencia en ningún momento
- Flexibilidad para hacer investigación y desarrollo sin necesidad de incurrir en costos adicionales de licenciamiento
- Existencia de una importante comunidad de profesionales y entusiastas de PostgreSQL de los que las compañías pueden obtener beneficios y contribuir a nuevos desarrollos
- PostgreSQL ha sido diseñado y creado para tener un mantenimiento y ajuste mucho menor que los productos de los proveedores comerciales, conservando todas las características, estabilidad y rendimiento

Deficiencias del PostgreSQL

- PostgreSQL es un magnífico sistema administrador de bases de datos, capaz de competir con muchos sistemas comerciales, aunque carece de algunas características:
- Como por ejemplo un conjunto de herramientas que permitan una fácil gestión de los usuarios y de las bases de datos que contenga el sistema
- Por otro lado, la velocidad de respuesta que ofrece este sistema con bases de datos relativamente pequeñas puede parecer un poco deficiente
- El sistema administrador de base de datos es un sistema no distribuido
- No existe esfuerzo focalizado de marketing, como en otros sistemas de código abierto “open source”
- El soporte que ofrece la herramienta a nivel empresarial no es muy bien difundido

Catálogo del sistema de PostgreSQL

- En los sistemas administradores de base de datos el catálogo del sistema es utilizado para guardar información referente a la metadata
- El catálogo del sistema puede ser consultado a través de sentencias SQL de la misma forma que una tabla cualquiera.
- Algunas tablas del catálogo del sistema serán listadas abajo:

  - pg_database
  - pg_indexes
  - pg_class
  - pg_view
  - pg_attribute
  - pg_constraint
  - pg_type
  - pg_shadow

Seguridad en PostgreSQL

- Para los administradores de base de datos la seguridad de la información es uno de los puntos más importantes
- Niveles que pueden implementarse para garantizar la seguridad de la información en PostgreSQL:
    A. Las conexiones de los clientes al servidor de la base de datos están permitidas, por defecto, únicamente en conexiones locales
    B. Las conexiones de los clientes se pueden restringir por dirección IP y/o por nombre de usuario mediante el archivo pg_hba.conf situado en el directorio data
    C. A cada usuario de PostgreSQL se le asigna un nombre de usuario y (opcionalmente) una contraseña
    D. Los usuarios pueden ser incluidos en grupos, y el acceso a las tablas puede restringirse en base a esos grupos

Seguridad en PostgreSQL

- La seguridad de un sistema debe ser equilibrada, no son recomendables los extremos:
  - Es menos conveniente muchos mecanismos de seguridad que entorpecen el trabajo de usuarios autorizados
  - Dejar puertas abiertas donde usuarios no autorizados puedan ingresar y causar daños al sistema

- En PostgreSQL deben de tomarse en cuenta tres aspectos para tener niveles altos de seguridad:

  1.- Seguridad de los archivos de datos de PostgreSQL
  2.- Seguridad en los accesos de clientes (ssl)
  3.- Otorgar y denegar acceso a tablas y usuarios.

Sistema de privilegios de acceso en PostgreSQL

- El mecanismo mediante el cuál se aseguran los datos y la integridad es el sistema de privilegios de PostgreSQL
- Es un mecanismo que está basado en la validación de usuarios que intentan conectarse a un servidor de base de datos y en la asignación de privilegios tales como SELECT, INSERT, UPDATE, DELETE, etc
- Cualquier usuario que cree un objeto de base de datos se convierte automáticamente en su propietario
- Usando la sentencia GRANT es posible otorgar privilegios a otros usuarios sobre objetos de base de datos

Comparación entre MySQL y PostgreSQL

- MySQL:
  - **Sistemas Operativos**: Linux, Windows FreeBSD ,MacOSX, Solaris, HPUX, AIX
  - **Soporte a ANSI SQL**: El motor SQL de MySQL esta escrito sobre los fundamentos de SQL92 y da soporte a SQL99. En MySQL se pueden construir sentencias que usan tablas que se encuentran en diferentes base de datos. Existe soporte para left y right outer joins. MySQL 4.1 maneja sub-consultas, se estima que MySQL 5.x soporte Vistas (view)
  - **Rendimiento:** MySQL es muy rápido en la ejecución de sentencias SQL simples y complejas, MySQL es conveniente para ser usado en aplicaciones WEB
  - **Estabilidad:** En las primeras versiones MySQL era bastante inestable, ofrecía pocas funcionalidades, la desconexiones sin motivo estaban presentes a cada momento
  - **Integridad de Datos:** MySQL ofrece soporte a integridad referencial y transacciones a través del motor de almacenamiento InnoDB
  - **Seguridad:** MySQL tiene un excepcional control de acceso, es posible otorgar o revocar cualquier derecho basado en usuarios, tabla o máquina (host)
  - **Soporte para**: Sub-consultas, Replicación de base de datos, Clave foránea, ODBC y JDBC

Comparación entre MySQL y PostgreSQL

- PostgreSQL:
  - **Sistemas Operativos**: Windows, sistemas tipo Unix (Linux, all BSDs, HP-UX,…)
  - **Soporte a ANSI SQL**: PostgreSQL soporta muy bien SQL92/99. PostgreSQL es capaz de manejar sentencias SQL complejas, sub-consultas, vistas, transacciones, left, full y right outer joins, no es posible usar tablas que se encuentren en diferentes base de datos **Rendimiento:** PostgreSQL es lento en la ejecución, el tiempo de respuesta del proceso backend “postgres” es pobre con respecto a otros manejadores ejecutando la misma sentencia
  - **Estabilidad:** Las series de PostgreSQL 6.x eran muy inestables, en la serie 7.x muchos de estas y otras fallas se solucionaron, aumentado la estabilidad, la serie 8 promete ser muy estable, en muchos lugares esta siendo usada en ambientes de producción
  - **Integridad de Datos:** PostgreSQL maneja excelentemente la integridad referencial, soportando la definición de claves primarias, foráneas, entre otras. Maneja transacciones, bloqueos
  - **Seguridad:** PostgreSQL tiene un mecanismo de control de acceso parecido a MySQL, se basa en tres aspectos importantes : 1.- Seguridad de los archivos de datos de PostgreSQL, 2.- Seguridad en los acceso de clientes (ssl) y 3.- Otorgar y denegar acceso a tablas y usuarios específicos
  - **Soporte para**: Sub-consultas, Replicación de base de datos, Clave foránea, Vistas, Full Joins, ODBC y JDBC

Cliente psql

- El cliente psql es una aplicación hecha en C, frecuentemente es llamado monitor interactivo o terminal interactivo, sirve como cliente, monitor y front-end hacia el motor de PostgreSQL
- Permite escribir comandos SQL interactivamente, enviarlos a PostgreSQL y visualizar los resultados de la consulta
- Las sentencias SQL pueden leerse desde un archivo, además provee varios meta-comandos (un meta-comando es una instrucción que es atendida por el monitor más no por el backend o motor de postgreSQL)
- psql es parte de la distribución normal de PostgreSQL
- Sintaxis en psql:
  - psql [–h nombre_del_host] –d base_De_datos –U nombre_de_usuario

Cliente psql

- Ejemplo:
- Psql –h localhost –d template1 –U postgres

```
student1@LIBMP8214:~> psql -h localhost -d template1 -U postgres
Welcome to psql 8.0.2, the PostgreSQL interactive terminal.

Type:  \copyright for distribution terms
       \h for help with SQL commands
       \? for help with psql commands
       \g or terminate with semicolon to execute query
       \q to quit

template1=#
```

- Línea de comandos psql
  - \h Para obtener ayuda de comandos SQL
  - \? Para obtener ayuda de comandos específicos de psql
  - \g Para ejecutar sentencias SQL
  - \q Para salir

Cliente psql

- Editar un archivo de texto con la herramienta vi
- Escribir la sentencia: select * from pg_database;
- Guardar el archivo y colocar como nombre “entrada.txt”
- Ahora levantamos una consola psql:
    - psql –h localhost –d template1 –U postgres
- y ejecutamos el siguiente comando:
    - \i entrada.txt
- La utilidad principal de este metacomando, es permitir la ejecución de sentencias SQL, por ejemplo que hayan sido obtenidas del proceso de respaldo de una base de datos

Resumen

- Usted debe ser capaz de:
- Listar las capacidades de PostgreSQL.
- Describir la licencia BSD.
- Describir copyleft
- Describir la historia de PostgreSQL y su evolución.
- Listar las principales características de PostgreSQL.
- Describir los niveles de jerarquía en PostgreSQL.
- Discutir sobre la arquitectura de PostgreSQL.
- Describir el funcionamiento de PostgreSQL.
- Describir los catálogos del sistema de PostgreSQL
- Listar las similitudes y diferencia entre PostgreSQL y MySQL

Unidad 2
Ejecución de sentencias SQL con PostgreSQL

Objetivos del Aprendizaje

- Describir el lenguaje SQL
- Describir los tipos de datos soportados por PostgreSQL
- Explicar qué son DDL, DML y DCL
- Describir los privilegios de PostgreSQL
- Escribir una sentencia SELECT simple
- Escribir una sentencia SELECT condicional
- Escribir consultas multitablas
- Generar respaldo de base de datos
- Ejecutar sentencias en el cliente psql

Introducción al SQL

- SQL (Lenguaje de consulta estructurado) es un lenguaje muy sencillo
- Basado en el habla inglés, orientado principalmente a base de datos relacionales y, sobre todo, al manejo de consultas
- SQL consiste de sentencias. Una sentencia esta compuesta de comandos y cláusulas terminadas en punto y coma “;” (en PostgreSQL)
- El objetivo principal de SQL es la realización de consultas y cálculos con los datos de una o varias tablas
- En esta unidad, se da una explicación completa del uso de las sentencias SQL sobre el sistemas administrador de base de datos PostgreSQL

El Lenguaje de consultas de PostgreSQL

- El lenguaje de consultas de PostgreSQL es una variación del lenguaje SQL estándar, y son extensiones propias de PostgreSQL.
- Para listar los comandos propios de PostgreSQL debemos usar \?.
- La relación que se hace entre SQL y el modelo de programación orientada a objetos es: una tabla corresponde a una clase, una fila corresponde a una instancia de una clase y las columnas a los atributos.

Ejecución de sentencias SQL con el programa psql

- El cliente psql es un programa de línea de comandos usado para enviar sentencias SQL al servidor, viene incluido en el paquete de distribución de PostgreSQL
- Para conectarse al servidor, usualmente necesitamos de un nombre de usuario (login) y de una contraseña (password), y si el servidor al que nos deseamos conectar está en una máquina diferente de la nuestra, también necesitamos indicar el nombre o la dirección IP de dicho servidor
- Cuando es iniciado por primera vez solo existen dos base de datos definidas: template0 y template1, las cuáles son la base de todas las base de datos que se definen en el servidor

Ejecución de sentencias SQL con el programa psql

- Es posible conocer las bases de datos que están creadas en el sistema administrador de base de datos, esto se hace a través de la sentencia:
- shell> psql –U postgres -l

```
db2admin@LIBMP8214:~> psql -U postgres -l
        List of databases
   Name    |  Owner   | Encoding
-----------+----------+----------
 template0 | postgres | UNICODE
 template1 | postgres | UNICODE
(2 rows)

db2admin@LIBMP8214:~>
```

- Para crear una base de datos desde la línea de comando del sistema operativo usamos:
- createdb –U postgres test1 [Presione ENTER]

Ejecución de sentencias SQL con el programa psql

- Active una ventana de línea de comandos en el sistema operativo, y ejecute el siguiente comando:
- psql –U postgres test1 [Presione la tecla Enter]
- Si la conexión al servidor PostgreSQL se pudo establecer de manera satisfactoria, se recibirá el mensaje de bienvenida y se activa el prompt de PostgreSQL:

```
db2admin@LIBMP8214:~> psql -U postgres test1
Welcome to psql 8.0.2, the PostgreSQL interactive terminal.

Type:  \copyright for distribution terms
       \h for help with SQL commands
       \? for help with psql commands
       \g or terminate with semicolon to execute query
       \q to quit

test1=#
```

Ejecución de sentencias SQL con el programa psql

- Desde la línea de comandos de psql es posible conectarse a otra base de datos con el comando \c
- test1=#\c template1

```
test1=# \c template1
You are now connected to database "template1".
template1=#
```

Tipos de Datos en PostgreSQL

- Tipos de datos numéricos
- Tipos de datos carácter
- Tipos de datos fecha/hora
- Tipo de dato booleano (boolean)
- Tipos de datos geométrico
- Tipos de datos dirección de red
- Arreglos
- Tipos compuestos

Tipos de Datos en PostgreSQL

- **Tipos de datos numéricos**

Nombre	Tamaño	Rango
Enteros		
Smallint	2 bytes	-32768 a +32767
Integer	4 bytes	-2147483648 a +2147483647
Bigint	8 bytes	-9223372036854775808 a 9223372036854775807
Punto Flotante		
decimal(n,p)	variable	n es la cantidad de dígitos del número y p es la cantidad de dígitos decimales
numeric(n,p)	variable	n es la cantidad de dígitos del número y p es la cantidad de dígitos decimales
Real	4 bytes	6 dígitos decimales de precisión
double precision	8 bytes	15 dígitos decimales de precisión
Autoincremento		
Serial	4 bytes	1 a 2147483647
Bigserial	8 bytes	1 a 9223372036854775807

Tipos de Datos en PostgreSQL

- **Tipos de datos carácter**
  - **CHARACTER(n): puede mantener cadenas fijas de n caracteres, puede ser abreviado como CHAR(n), si el valor "n" no se coloca cuando se define una columna CHARACTER el sistema asume longitud 1**
  - **CHARACTER VARYING(n): Mantiene cadenas de longitud variable, puede ser abreviado como VARCHAR(n), si el valor "n" no se coloca cuando se define una columna CHARACTER VARYING es posible almacenar cadenas de cualquier longitud (La máxima longitud de almacenamiento es 1 GB)**
  - **TEXT: Es equivalente a un VARCHAR sin especificar la longitud, este tipo de dato puede almacenar cadenas de cualquier longitud.**

Tipos de Datos en PostgreSQL

- **Tipos de datos Fecha/Hora**
  - **timestamp: Combinación de fecha y hora**
  - **interval: Representa un intervalo de tiempo**
  - **date: Permite almacenar una fecha**
  - **time: Es una representación genérica de valores de tiempo expresado en horas, minutos y segundos. El formato de almacenamiento es de 'hh:mm:ss'**

Tipos de Datos en PostgreSQL

- **Tipo de datos booleano (boolean)**
  - **PostgreSQL provee el tipo estándar boolean (booleano) de SQL, boolean (booleano) puede tener tres estados “true”, “false” y “desconocido” (unknown) el cual es representado por el valor null**
  - **Valores validos para el estado “true” (Verdadero):**

    TRUE 't' 'true' 'y' 'yes' '1'

  - **Valores validos para el estado “False” (Falso):**

    FALSE 'f' 'false' 'n' 'no' '0'

Tipos de Datos en PostgreSQL

- **Tipos de datos Geométricos**
  - **Punto (POINT)**
  - **Segmento (LSEG)**
  - **Caja (BOX)**
  - **Ruta (PATH)**
  - **Polígono (POLYGON)**
  - **Circula (circle)**

- **Estos tipos de datos son útiles en áreas de la inteligencia de negocios, por ejemplo pueden ser utilizados para representar zonas de potenciales compradores, ubicaciones en autopistas, comercios, etc**

Tipos de Datos en PostgreSQL

- **Tipos de datos dirección de red**
- **PostgreSQL da soporte a tipos de datos para almacenar direcciones Ipv4, IPv6 y MAC.**

Nombre	Tamaño	Descripción
cidr	12 o 24 bytes	Redes IPv4 e IPv6
inet	12 o 24 bytes	Estaciones de trabajo y redes IPv4 e IPv6
macaddr	6 bytes	Direcciones MAC

Tipos de Datos en PostgreSQL

- **Arreglos**
  - **PostgreSQL permite definir columnas como arreglos unidimensionales, bidimensionales y multidimensionales**
- **Un ejemplo de un arreglo multidimensional:**

```
CREATE TABLE salario_empleado (
ci              bigint,
salario         decimal(10,2)[3],
planificacion   text[2][2]);
```

- **Inserción de valores en columnas del tipo arreglo multidimensional:**

```
INSERT INTO salario_empleado VALUES(17546214 ,
'{25487.87,254.45,                    4587.45}',
'{{"Entrenamiento",     "Soporte      Técnico"},
{"Vacaciones","Presentación"}}');
```

Tipos de Datos en PostgreSQL

- **Arreglos**
  - **SELECT * FROM salario_empleado;**

```
test1=# SELECT * FROM salario_empleado;
    ci    |         salario          |                          planificacion
----------+--------------------------+-----------------------------------------------------------
 17546214 | {25487.87,254.45,4587.45} | {{Entrenamiento,"Soporte Tecnico"},{Vacaciones,Presentacion}}
(1 row)

test1=#
```

  - **SELECT ci FROM salario_empleado WHERE salario[1] > 3255 AND salario[2]> 25;**

```
test1=# SELECT ci FROM salario_empleado WHERE salario[1] > 3255 AND salario[2]>25;
    ci
----------
 17546214
(1 row)

test1=#
```

Tipos de Datos en PostgreSQL

- **Tipo compuesto**
  - **Un tipo de dato compuesto describe la estructura de una fila o registro en forma de nombres de campos y tipos de datos**
  - **Ejemplo de tipo compuesto:**

```
CREATE TYPE producto AS (identificador    integer,
nombre_producto  text, id_proveedor     integer,
precio           decimal(10,2));
```

  - **Usando tipos compuestos para definir tablas:**

```
CREATE TABLE deposito (
item      producto,
cantidad     integer);
```

Tipos de Datos en PostgreSQL

- Tipo compuesto
  - Insertando valores en la tabla:

```
INSERT INTO deposito VALUES (ROW(1,'computador 1.0 ghz', 42,1009),40);

INSERT INTO deposito VALUES (ROW(2,'computador 1.2 ghz', 42,999),40);
```

  - Usando tipos compuestos para definir tablas:

```
CREATE TABLE deposito (
item      producto,
cantidad     integer );
```

  - Selección de registros desde una tabla:

```
SELECT (item).nombre_producto FROM deposito;

SELECT      (deposito).item.nombre_producto      FROM deposito WHERE (deposito).item.precio> 999;
```

Sentencias DDL

- CREATE DATABASE: Permite crear una base de datos:
  - CREATE DATABASE nombre_base_de_Datos;
- DROP DATABASE: Elimina una base de datos:
  - DROP DATABASE nombre_base_de_datos;
- CREATE SCHEMA: Los esquemas (schema) contienen objetos de base de datos como por ejemplo tablas
- Ejemplo:
  - CREATE SCHEMA myschema;
- DROP SCHEMA: Esta sentencia permite eliminar un esquema (schema) sí está vació
  - DROP SCHEMA myschema;

Sentencias DDL

- CREATE TABLE: Esta sentencia es utilizada para crear una tabla y definir columnas.
- Sintaxis:

```
CREATE TABLE nombre_tabla( LIKE otra_tabla);

 CREATE [ TEMPORARY | TEMP ] TABLE nombre_tabla (

 nombre_columna tipo [ restricción_columna [ ... ]][,...]

 [, restricción_tabla [, ... ]]) ;
```

Sentencias DDL

- CREATE TABLE
- Ejemplos

1.- CREATE TEMPORARY TABLE gastos(id integer NOT NULL PRIMARY KEY, motivo text, fecha date, monto decimal (10,2)); TEMPORARY o TEMP Tanto los datos como la estructura de las tablas temporales son eliminadas de manera automática al terminar la sesión de usuario actual.	2.-CREATE TABLE empleado(id serial, nombre varchar(50), salario numeric(9,2));
3.- CREATE TABLE cliente(id_cliente INTEGER NOT NULL, nombre VARCHAR(20), PRIMARY KEY (id_cliente)); En MySQL la restricción NOT NULL es opcional cuando se especifica la columna como primary key (Esta implícita con primary key).	4 CREATE TABLE copia_cliente(like cliente);

Sentencias DDL

- Con la sentencia \d es posible obtener información sobre la definición de una tabla:
- Sintaxis:

  \d nombre_tabla

- Ejemplo

```
test1=# \d cliente
              Table "public.cliente"
   Column   |          Type          | Modifiers
------------+------------------------+-----------
 id_cliente | integer                | not null
 nombre     | character varying(20) |
Indexes:
    "cliente_pkey" PRIMARY KEY, btree (id_cliente)

test1=#
```

Sentencias DDL

- CREATE INDEX construye un índice sobre una tabla especificada. Los índices son principalmente usados para mejorar el rendimiento de acceso a los datos.
- Sintaxis:

```
CREATE [ UNIQUE ] INDEX nombre_indice ON nombre_tabla(columnas);
```

- Ejemplo:

```
CREATE TABLE emp (id serial,
nombreemp varchar(50) UNIQUE,
  salario numeric(9,2) NOT NULL,
  nrotelefono VARCHAR(20),
  moneda varchar(4) DEFAULT 'BLV');
  CREATE UNIQUE INDEX telindex ON emp(nrotelefono);
```

Sentencias DDL

- Con ALTER TABLE se puede agregar columnas, eliminar columnas, cambiar el tipo, etc.
- Sintaxis:

`ALTER TABLE nombre_de_tabla Acciones [, ... ]`	Sentencia usada para especificar nuevas columnas, eliminar columnas, cambiar la estructura de columnas, agregar restricciones, etc.
`ALTER TABLE nombre_de_tabla RENAME nombrecolumna TO nuevo_nombre_columna`	Sentencia usada para cambiar el nombre de una columna.
`ALTER TABLE nombre_de_tabla RENAME TO nuevo_nombre`	Sentencia usada para cambiar el nombre de una tabla

Sentencias DDL

- Ejemplos de ALTER TABLE
- Esta sentencia permite agregar una columna país del tipo VARCHAR(10) a la tabla emp:

```
ALTER TABLE emp ADD COLUMN pais VARCHAR(10);
```

- Si se desea seleccionar un valor por defecto para la columna, ejecute la siguiente sentencia:

```
ALTER TABLE emp ALTER COLUMN pais SET DEFAULT 'VNZ';
```

- Con ALTER TABLE es posible cambiar el nombre de una tabla, el siguiente ejemplo lo ilustra:

```
ALTER TABLE emp RENAME TO oldemp;
```

Sentencias DDL

- Ejemplos de ALTER TABLE
- Cambiar el propietario de una tabla:

```
ALTER TABLE oldemp OWNER TO ibm_user;
```

- Agregar una restricción de no nulo (NOT NULL) a número telefónico:

```
ALTER TABLE oldemp ALTER nrotelefono SET NOT NULL;
```

1.- ALTER TABLE cliente ADD COLUMN salario INTEGER;	2.- ALTER TABLE cliente DROP COLUMN salario;
3.- ALTER TABLE cliente ADD COLUMN salario DOUBLE PRECISION;	

Sentencias DML

- INSERT: Sentencia usada para insertar datos en una tabla
- Sintaxis:

```
INSERT INTO nombre_tabla [ ( columna [, ...]) ]
{DEFAULT VALUES | VALUES ( { expression
|DEFAULT } [, ...] ) | Consulta_SELECT }
```

- CARGAR DATOS MASIVOS (COPY): Para cargar datos en una tabla desde un archivo externo se usa la sentencia COPY

```
    COPY table FROM { 'filename' | stdin }
[   [USING] DELIMITERS 'delimiter' ]
    COPY table TO { 'filename' | stdout }
    [[USING] DELIMITERS 'delimiter' ]
```

Sentencias DML

- UPDATE: La sentencia UPDATE es usada para actualizar valores de columnas de una tabla
- Sintaxis:

```
UPDATE nombre_tabla
  SET nombrecol1=valor, [nombrecol2=valor, ...]
[WHERE Condición]
```

- DELETE: La sentencia DELETE es usada para eliminar registros de una tabla
- Sintaxis:

```
DELETE FROM table [ WHERE condición ]
```

- PostgreSQL provee una sentencia que procesa la eliminación de todos los datos de una tabla rápidamente, esta es TRUNCATE:
- Sintaxis:

```
TRUNCATE [ TABLE ] name
```

Sentencias DQL

- SELECT: La sentencia SELECT se usa para recuperar filas seleccionadas de una o más tablas Sintaxis:

Sintaxis:

```
SELECT [ ALL | DISTINCT] |
* | nombre_columna [ AS nombre_salida ] [,
...]
[ FROM Fuente [, ...] ]
[ WHERE condición ]
[ GROUP BY nombre_columna [, ...] ]
[ HAVING condición [, ...] ]
[ORDER BY nombre_columna [ ASC | DESC [, ...]
]
[ LIMIT { cantidad | ALL } ]
[ OFFSET inicio ]
```

Sentencias DQL

- DISTINCT: Es usado para restringir la aparición de filas repetidas
- Ejemplo:
  - SELECT DISTINCT nombre FROM cliente;
- WHERE: La mejor utilidad de una base de datos es la posibilidad de hacer consultas en función de ciertas condiciones
- Ejemplo:
  - SELECT * FROM cliente WHERE nombre='Luis';
- ALIAS: El alias se usa como un nombre de columna en expresiones, también es posible definir alias a las tablas de la cláusula FROM.
- Ejemplo:
  - SELECT nombre ||', '|| salario Nombre_Salario FROM cliente ORDER BY Nombre_Salario;

Sentencias DQL

- GROUP BY: Es posible agrupar filas en la salida de una sentencia SELECT usando la cláusula GROUP BY, si la cláusula HAVING esta presente, esta elimina los grupos que no satisfacen una condición dada
- Ejemplo:
  - SELECT count(id_cliente), nombre FROM cliente GROUP BY nombre;
  - SELECT count(id_cliente), nombre FROM cliente GROUP BY nombre HAVING count(id_cliente) >= 2;
- ORDER BY: Es usada para dar ordenamiento a los datos recuperados en una sentencia SELECT, por defecto es ascendente
- Ejemplo:
  - SELECT id_cliente, nombre FROM cliente ORDER BY id_cliente, nombre;

Sentencias DQL

- LIMIT: La cláusula LIMIT permite limitar el número de filas recuperadas por la sentencia SELECT
- Ejemplo:
  - SELECT * FROM cliente LIMIT 3;
- OFFSET: La cláusula OFFSET especifica el número de filas a ser obviadas antes de comenzar las filas retornadas
- Ejemplo:
  - SELECT * FROM cliente OFFSET 3;
- Operador IN: El operador IN se usa para realizar comparaciones con una lista de valores.
- Ejemplo:
  - SELECT * FROM cliente WHERE nombre IN ('Jesus', 'Pedro', 'Nelson');

Sentencias DQL

- Operador BETWEEN: El operador BETWEEN se usa para comprobar si cierto valor está dentro de un rango dado
- Ejemplo:
  - `SELECT id_cliente,nombre FROM cliente WHERE salario >= 1000 AND salario <= 50000;`
- La consulta anterior puede ser escrita usando el operador BETWEEN.
  - SELECT id_cliente,nombre FROM cliente WHERE salario BETWEEN 1000 AND 50000;
- Operador IS: Los operadores IS NULL e IS NOT NULL son utilizados para verificar si una expresión determinada es o no nula
- Ejemplo:
  - SELECT * FROM cliente WHERE nombre IS NOT NULL;

Sentencias DQL

- Operador LIKE: Se usa para hacer comparaciones entre cadenas y patrones. El resultado es verdadero (1) si la cadena se ajusta al patrón, y falso (0) en caso contrario
- '%' Es usado para hacer coincidir cualquier número de caracteres, incluso ninguno.
- '_' Es usado para hacer coincidir con un único carácter.
- Ejemplo:
  - SELECT * FROM cliente WHERE nombre LIKE 'Miguel%';

Consultas Multitablas

- Los datos están distribuidos en diferentes tablas debido a la normalización
- Enlaza tablas de datos por medio de un valor de atributo común en ambas tablas
- Puede ser aplicada a dos o más tablas para poder recuperar registros de múltiples tablas
- Tipos de JOIN:
  - Cartesian JOIN (Producto Cartesiano)
  - INNER JOIN
  - Right Outer
  - Left Outer
  - Full Outer
  - Self

Consultas Multitablas

- PRODUCTO CARTESIANO (CARTESIAN JOIN): Hace corresponder todas las filas de la primera tabla con todas las filas de la segunda tabla y presenta una combinación de todos los registros de ambas tablas.

- Ejemplo
  - SELECT * FROM personal, automovil;

Consultas Multitablas

- INNER JOIN: Las composiciones internas (INNER JOIN) se definen a partir de un producto cartesiano, eliminado las filas que no cumplen la condición de composición
- Ejemplos
  - SELECT * FROM personal, automovil WHERE personal.no_personal=automovil.no_personal;
- Reescribiendo la sentencia anterior con JOIN:

```
SELECT * FROM personal JOIN automovil ON (personal.no_personal=automovil.no_personal);
```

```
SELECT * FROM personal INNER JOIN automovil ON (personal.no_personal=automovil.no_personal);
```

```
SELECT * FROM personal JOIN automovil USING(no_personal);
```

Consultas Multitablas

- RIGHT OUTER JOIN: La sentencia Right Outer JOIN hace corresponder los registros de la tabla del lado derecho con los registros de la tabla del lado izquierdo basándose en la igualdad de valores que se especifica en la condición JOIN
- Ejemplos
  - SELECT * FROM personal RIGHT OUTER JOIN automovil ON (personal.no_personal=automovil.no_personal);
- Puede ser escrito como:
  - SELECT * FROM personal RIGHT OUTER JOIN automovil USING(no_personal);

Consultas Multitablas

- LEFT OUTER JOIN: La sentencia Left Outer JOIN hace corresponder los registros de la tabla del lado izquierdo con los registros de la tabla del lado derecho basándose en la igualdad de valores que es especificada en la condición JOIN
- Ejemplo
  - SELECT * FROM personal LEFT OUTER JOIN automovil ON (personal.no_personal=automovil.no_personal);
- Puede ser escrito como:
  - SELECT * FROM personal LEFT OUTER JOIN automovil USING (no_personal);

Consultas Multitablas

- FULL OUTER JOIN: La sentencia Full Outer Join hace corresponder los registros de la tabla del lado izquierdo con los registros del lado derecho y los registros de la tabla del lado derecho con los registros de la tabla del lado izquierdo basándose en la igualdad de valores que es especificada en la condición JOIN
- Ejemplo
  - SELECT * FROM personal FULL OUTER JOIN automovil ON (personal.no_personal=automovil.no_personal);
- Puede ser escrito como:
  - SELECT * FROM personal FULL OUTER JOIN automovil USING (no_personal);

Consultas Multitablas

- SELF JOIN: Es posible realizar un JOIN en una misma tabla si una tabla tiene dos atributos que comparten el mismo valor

- Ejemplos

  – SELECT S.nombre, M.nombre FROM personal S, personal M WHERE S.no_grnt = M.no_personal;

SubConsultas

- Subconsulta Correlacionada: Cuando la subconsulta hija hace referencia a una o más columnas de la subconsulta padre.
- Ejemplo
  - **SELECT automovil.modelo**

    FROM automóvil WHERE EXISTS (SELECT no_personal

    FROM personal WHERE

    personal.no_personal = automovil.no_personal);
- Subconsulta No Correlacionada: No existe referencia de las columnas entonces se dice que es no-correlacionada, PostgreSQL permite utilizar subconsultas en la cláusula FROM por ejemplo:
- Ejemplo
  - **SELECT * FROM (SELECT * FROM automovil) AS subconsul;**

(Es mandatorio usar alias en la subconsulta interna)

Usuarios y Privilegios

- Cada cluster de base de datos contienen un conjunto de usuarios de base de datos
- Un usuario es propietario de un objeto de base de datos (por ejemplo: una tabla) y puede otorgar privilegios sobre estos objetos a otros usuarios
- Consultando la tabla pg_user se obtienen los usuarios existentes: SELECT usename FROM pg_user;
- El metacomando **\du** ejecutado desde la línea de comandos del psql lista los usuarios de base de datos disponibles
- Grupos: Los grupos son una manera lógica de agrupar usuarios para administración sencilla de privilegios, los privilegios pueden ser otorgados o revocados a todo un grupo. Para crear un grupo usamos el comando
  - CREATE GROUP name;

Usuarios y Privilegios

- Privilegios: Los privilegios son derechos que reciben usuarios sobre objetos de base de datos, estos permiten realizar operaciones como por ejemplo: eliminar datos de tablas, crear tablas, definir restricciones, etc.
- Son otorgados con la sentencia GRANT, la sintaxis es:

GRANT { { SELECT | INSERT | UPDATE | DELETE | RULE | REFERENCES | TRIGGER } [,...] | ALL [PRIVILEGES] }

ON [TABLE] nombre_tabla [, ...] TO { usuario | GROUP nombre_grupo | PUBLIC } [, ...] [WITH GRANT OPTION]

GRANT { { CREATE | TEMPORARY | TEMP } [,...] | ALL [PRIVILEGES] } ON DATABASE basededatos [, ...] TO { usuario | GROUP nombre_grupo | PUBLIC } [, ...] [WITH GRANT OPTION]

Usuarios y Privilegios

- Otorgar el privilegio INSERT sobre la tabla cinta a todos los usuarios:
- Ejemplos
  - GRANT INSERT ON cinta TO PUBLIC;
  - GRANT ALL PRIVILEGES ON monitores TO marcos;
- Usando el metacomando \z desde la línea de comando de psql, es posible obtener información acerca de los privilegios existentes, por ejemplo:
  - Test1=#\z cliente

Usuarios y Privilegios

- SENTENCIA REVOKE: Para revocar privilegios se usa la sentencia REVOKE, la sintaxis es similar a la sentencia GRANT:

REVOKE [GRANT OPTION FOR]

{ { SELECT | INSERT | UPDATE | DELETE | REFERENCES | TRIGGER } [,...] | ALL [PRIVILEGES] } ON [TABLE] nombre_tabla [, ...]

FROM { usuario | GROUP nombre_grupo | PUBLIC } [, ...] [CASCADE]

REVOKE [GRANT OPTION FOR] { { CREATE | TEMPORARY | TEMP [,...] | ALL [PRIVILEGES] } ON DATABASE basededatos [, ...]

FROM { usuario | GROUP nombre_grupo | PUBLIC } [, ...] [CASCADE]

Si **GRANT OPTION FOR** es especificado solo la opción “grant option” para el privilegio es revocada, no el privilegio en si mismo.

Usuarios y Privilegios

- Ejemplos REVOKE:
- Revocar el privilegio INSERT sobre la tabla cinta a todos los usuarios:
  - REVOKE INSERT ON cinta FROM PUBLIC;
- Revocar todos los privilegios disponibles a el usuario marcos sobre la tabla monitores:
  - REVOKE ALL PRIVILEGES ON monitores FROM USER marcos; ENTENCIA REVOKE:

Características avanzadas de PostgreSQL

- Vistas: PostgreSQL soporta la definición de vistas (view), las vistas en si no son tablas físicas, generalmente se usan vistas para restringir el acceso a la información
- Ejemplos
  - CREATE VIEW mi_personal AS SELECT no_personal, nombre_personal, no_division FROM personal;
  - SELECT * FROM mi_personal;

Características avanzadas de PostgreSQL

- Herencia: La herencia es una característica de los manejadores de base de datos orientados a objetos, permite la reutilización de código, es decir definir la estructura de una tabla a partir de una ya creada.
- Veamos el siguiente ejemplo:

```
CREATE TABLE ciudades (
nombre            text,
cantidadpoblacion integer,
codigoarea        varchar(4),
pais              varchar(20),
estado            varchar(20),
capital boolean);
```

Características avanzadas de PostgreSQL

- Si se desea consultar todas las ciudades, se construye la siguiente vista para obtener la información:
  - CREATE VIEW vista_ciudades AS

    SELECT nombre, codigoarea, pais, estado FROM ciudades;
- Si se quiere obtener solo la ciudades que son capitales podríamos construir otra vista para obtener el resultado:
  - CREATE VIEW vista_capitales AS

    SELECT nombre, codigoarea, pais, estado FROM ciudades WHERE capital=TRUE;

Características avanzadas de PostgreSQL

- Es posible hacer uso de herencia en postgreSQL y definir una nueva estructura para las capitales a partir de la definición de ciudades

```
CREATE TABLE ciudades_h (
nombre          text,
cantidadpoblacion integer,
codigoarea      varchar(4),
pais varchar(20),
estado varchar(20));
```

Tabla Padre

```
TABLE capitales_h () INHERITS (ciudades_h);
```

Tabla descendiente

Características avanzadas de PostgreSQL

- Por ejemplo, la siguiente consulta lista las ciudades (incluyendo las capitales) que tienen una población de 5000 personas:
  - SELECT nombre, pais FROM ciudades_h

    WHERE cantidadpoblacion = 5000;
- Si solo se desea listar las ciudades que no son capitales, ejecutamos la siguiente sentencia:
  - SELECT nombre, pais FROM ONLY ciudades_h

    WHERE cantidadpoblacion = 5000;
- Aquí el ONLY antes de ciudades_h indica que la consulta solo se ejecuta sobre la tabla ciudades_h y no sobre los descendientes

Respaldo y Restauración

- pg_dump -- Permite generar respaldo de una base de datos postgreSQL en un archivo de script (sentencias SQL), .
- La sintaxis es la siguiente:
  1. Ejecutar desde la línea de comandos de Linux:
  2. pg_dump [opciones...] [basededatos]
  3. Después presione [ENTER] y coloque su contraseña, en el archivo estarán las sentencias DDL y DML

  – Ejemplo:

  Respaldar una base de datos (solo estructura) (“>” es equivalente a -f):

  ```
  pg_dump –U postgres test > db.out
  ```

  Generar las sentencias INSERT

  ```
  pg_dump –U postgres -d test -f db.out
  ```

Resumen

Ahora que ha completado esta unidad, usted debe ser capaz de:

- Describir el lenguaje SQL
- Describir los tipos de datos soportados por PostgreSQL
- Explicar qué son DDL, DML y DCL
- Describir los privilegios de PostgreSQL
- Escribir una sentencia SELECT simple
- Escribir una sentencia SELECT condicional
- Escribir consultas multitablas
- Generar respaldo de base de datos
- Ejecutar sentencias en el cliente psql