Introducción

Las Bases de Datos ofrecen numerosas ventajas frente a los sistemas tradicionales, por lo que cada vez más son requeridos personales capacitados para el área Gestión de Base de Datos en las Empresas.
En esta materia, se realizará una introducción a los conceptos básicos del área de base de datos, los diferentes tipos de bases de datos existentes en la actualidad y los sistemas manejadores de bases de datos. Además de enfocar aspectos relacionados con modelaje, planificación, diseño y administración de los mismos.

¿Qué es una Base de Datos?
Una Base de Datos puede ser definida como un conjunto de datos interrelacionados, representando informaciones sobre un dominio especifico. Por datos podemos comprender como “hechos conocidos” que pueden ser almacenados y que poseen un significado implícito.
Ejemplo: Lista telefónica

¿Qué es un Sistema de Base de Datos?

Una Base de datos puede ser creada y mantenida por un Sistema Gerenciador de Base de Datos (SGBD) que consiste en un conjunto de aplicaciones desarrolladas especialmente para esta tarea. El conjunto formado por una Base de datos más las aplicaciones que manipulan al mismo es llamado de "Sistema de Base de Datos".

Por tanto, podemos decir que un sistema de Base de Datos consiste en una colección de datos interrelacionados y una colección de programas para proveer acceso a esos datos. El objetivo principal de un sistema de bases de datos es proveer un ambiente que sea adecuado y eficiente para el uso en la recuperación y almacenamiento de informaciones.

Principales Módulos de un Sistema de Base de Datos

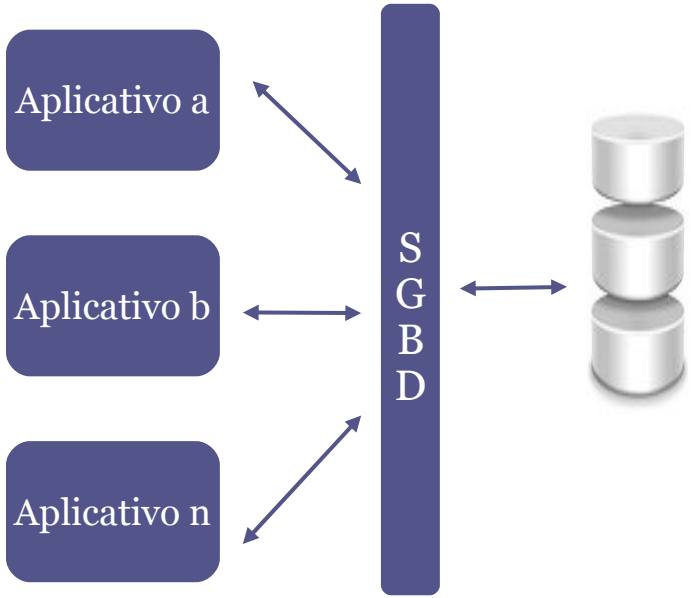
- Inclusión
- Alteración
- Eliminación
- Consulta

Sistema Gerenciador de Base de Datos

Un SGBD es una colección de programas que permiten al usuario definir, construir y manipular Bases de Dados para las más diversas finalidades

En el gráfico se ilustra la función del SGBD:

El usuario emite una solicitud de acceso
El SGBD intercepta la solicitud y analiza.
El SGBD inspecciona los esquemas externos (o sub-esquemas) relacionados a aquel usuario, los mapeamientos, y la definición de la estructura de almacenamiento.
El SGBD realiza las operaciones solicitadas en la base de datos almacenada.



Características generales de un SGBD:

- *Garantía de Integridad.*
- *Garantía de Seguridad*
- *Copias de seguridad ("backup") y recuperación.*
- *Control de concurrencia.*
- *Optimización de comandos DML.*
- *Diccionario de Dados.*
- *Desempeño*

SGBD más utilizados

Ventajas en el uso de Base de Datos

Eliminación o Reducción de Redundancias: La redundancia innecesaria de datos lleva al almacenamiento excesivo de informaciones, ocupando espacio que podría estar siendo utilizado por otras informaciones.
Acceso compartido a datos: Un SGBD permite que múltiples usuarios accedan a la base de datos al mismo tiempo manteniendo el control de concurrencia para asegurar que los resultados de actualizaciones sean correctas.
Restricciones de Seguridad: Un SGBD debe proveer un subsistema de autorización y seguridad, el cual es utilizado por el DBA para crear "cuentas" y especificar las restricciones de las mismas; el control de restricciones se aplica tanto al acceso a los datos como al uso de los softwares inherentes al SGBD.
Independencia de Datos : Es la capacidad de modificar una definición de esquema en un nivel, sin afectar la definición de esquema en un nivel más alto.

Independencia Física de datos: Capacidad de modificar el esquema físico sin necesitar reescribir los programas de aplicación.
Independencia Lógica de datos: Capacidad de modificar el esquema conceptual sin a necesidad de reescribir los programas de aplicación.

Esquema Físico: Descripción de la estructura de almacenamiento físico de la base de datos, indicando detalladamente los datos almacenados y los caminos de acceso a la base de datos.
Esquema Conceptual: Descripción global de la base de datos que no contiene detalles del modo como los datos están físicamente almacenados.

Administradores y usuarios de Base de Datos

Administrador de Datos
Desarrolla y administra centralizadamente estrategias, procedimientos, prácticas y planes capaces de disponibilizar los dados corporativos necesarios, cuando sean necesarios, con integridad, privacidad, documentación y en forma compartida.
Participa del relevamiento de datos y reglas de negocio de la empresa.
Elabora y/o acompaña la confección de modelos. Participa de pesquisa de softwares de apoyo, relacionados a área de AD, así como SGBD.

Administrador de Base de Dados
Define la creación del esquema original de la base de datos, a partir de los modelos lógicos.
Define la estructura de almacenamiento del método de acceso, modificación de la organización física y del esquema.
Concede autorización para acceso a datos. Especificación de restricciones e integridades.

Usuarios de Base de Datos
Programadores de Aplicaciones
Usuarios Simples
Usuarios Especializados

Estructuras de Datos de un SGBD

Archivo de Datos
Almacenan los datos propiamente dichos.

Diccionario de Datos
Almacenan informaciones sobre la estructura de la base de datos.

Índices
Proporcionan acceso rápido a los items de datos con valores específicos.

Modelos de Datos

Los modelos de datos pueden ser básicamente de dos tipos:

Alto nivel o modelo de datos conceptual: Provee una visión más próxima del modo como los usuarios visualizan los datos realmente;

Bajo nivel o modelo de datos físico: Que provee una visión más detallada del modo como los datos están realmente almacenados en la computadora.

Esquemas e Instancias

En cualquier modelo de datos utilizado, es importante distinguir la "descripción" de la base de datos por si propio. Esa descripción es llamada de "esquema de una base de datos" y es especificada durante el proyecto. Generalmente, pocos cambios ocurren en el esquema de la base de dados.
Los datos almacenados en una base de dados en un determinado instante de tiempo forman un conjunto llamado de "instancia de la base de datos". La instancia altera toda vez que una alteración en la base de datos es hecha.

El SGBD es responsable por garantizar que toda instancia de la base de datos satisfaga al esquema de la base de datos, respetando su estructura y sus restricciones.
El esquema de una base de datos también puede ser llamado de "intención" de una base de datos y la instancia de "extensión" de una base de datos.

Lenguajes de los sistemas de Gestión de BD

Lenguaje de Definición de Datos - (Data Definition Language - DDL)

Lenguaje que define las aplicaciones, archivos y campos que irán componer la base de datos (comandos de creación y actualización de la estructura de los campos de los archivos).”
Para la definición de los esquemas conceptual e interno puede utilizarse un lenguaje llamado DDL (Data Definition Language – Lenguaje de Definición de Datos). El SGBD posee un compilador DDL que permite la ejecución de las declaraciones para identificar las descripciones de los esquemas y para almacenarlas en el catálogo del SGBD. El DDL es utilizado en SGBDs donde la separación entre los niveles interno y conceptual no es muy clara.

¿Dónde quedan almacenadas estas definiciones?
Diccionario de Datos (Data Dictionary - DD): archivo que contiene metadatos; esto es, “datos acerca de datos”. Este archivo es consultado antes de que los datos reales sean leídos o modificados en el sistema de base de datos.

Ejemplos
CREATE TABLE
CREATE VIEW

Ejemplo 1
CREATE TABLE cliente ;
(ClienteID Integer PRIMARY KEY, Nombre character(20), Ciudad character(20))
Ejemplo 2
CREATE VIEW mivista AS SELECT * FROM cliente WHERE ciudad="ASUNCION”

Lenguaje de Manipulación de Datos - (Data Manipulation Language - DML)

Lenguaje que define las aplicaciones, archivos y campos que irán componer la base de datos (comandos de creación y actualización de la estructura de los campos de los archivos).

Una vez que el esquema este compilado y la base de datos este poblada, se usa un lenguaje para hacer la manipulación de los datos, la DML (Data Manipulation Language – Lenguaje de Manipulación de datos).

¿En que consiste la manipulación de datos?

- La recuperación de la información almacenada en la base de datos.
- La inserción de nuevas informaciones en la base de datos.
- La remoción de informaciones en la base de datos

Comandos DML. Ejemplos

INSERT
UPDATE
DELETE
SELECT

Ejemplo 1
USE cliente
INSERT INTO cliente (clienteID, Nombre, Ciudad) VALUES (1200, "Ramon", "Asunción")

Ejemplo 2
UPDATE cliente SET ciudad = “San Lorenzo”

Ejemplo 3
DELETE FROM cliente WHERE ClienteID = "2300“

Ejemplo 4
SELECT * FROM cliente WHERE ClienteID = "2300"

Lenguaje de Control de Datos - (Data Control Language - DCL)

Un Lenguaje de Control de Datos es un lenguaje proporcionado por el sistema de gestión de base de datos, que incluye una serie de comandos SQL que permiten al administrador controlar el acceso a los datos contenidos en la base de datos. Algunos ejemplos de comandos incluídos en el DCL son los siguientes:

GRANT : Permite dar permisos a uno o varios usuarios o roles para realizar tareas determinadas.
REVOKE : Permite eliminar permisos que previamente se han concedido con GRANT.

Las tareas sobre las que se pueden conceder o denegar permisos son las siguientes:

- CONNECT
- SELECT
- INSERT
- UPDATE
- DELETE
- USAGE

Reglas de Integridad Relacional

Regla de integridad de las entidades

Ningún componente de la clave primaria de una relación base puede aceptar nulos. Es la primera de todas las reglas generales de integridad del modelo relacional. Con nulos queremos decir información faltante por alguna razón, por ejemplo, si la propiedad no es aplicable o si el valor se desconoce, etc.
Entonces, entendemos por nulo, sencillamente un valor o representación que por convención no representa valor real alguno del atributo aplicable.
En caso de las claves primarias compuestas, la regla de integridad de entidades dice que cada valor individual de la clave primaria debe ser no nulo en su totalidad.

Regla de integridad referencial

La base de datos no debe contener valores de clave ajena sin concordancia. Con esto expresamos la segunda regla general de integridad del modelo relacional: la de integridad referencial. Con el término "valores de claves ajenas sin concordancia" queremos decir aquí un valor no nulo de clave ajena para el cual no existe un valor concordante de la clave primaria en la relación objetivo pertinente.
Sin duda, es obvia la justificación de esta regla: así como los valores de la clave primaria representan identificadores de entidades, así los valores de la clave ajena representan referencias a entidades. La regla de integridad referencial dice tan solo que si B hace referencia a A, entonces A debe existir.

Claves Primarias y claves ajenas

Claves Primarias

En términos informales, la clave primaria de una relación es sólo un identificador único para esa relación; esta clave puede ser simple o compuesta.
Además, es posible, aunque poco usual, tener una relación con más de un identificador único. En un caso así decimos que la relación tiene varias claves candidatas; entonces escogeríamos una de esas claves candidatas como clave primaria, y a las demás llamaríamos claves alternativas.
El término "clave candidata" definimos de la siguiente manera: El atributo K de la relación R es una clave candidata de R si y solo si satisface las siguientes propiedades:
1. Unicidad: En cualquier momento dado, no existen dos tuplas en R con el mismo valor de K.
2. Minimalidad: Si K es compuesto, no será posible eliminar ningún componente de K sin destruir la propiedad de unicidad.

Así pues, ¿Por qué son importantes las claves primarias?
Una respuesta obvia a esta pregunta es que el manejo de claves primarias es un requisito para el manejo de claves ajenas, como veremos en el siguiente punto. Una respuesta más fundamental es que las claves primarias constituyen el mecanismo de direccionamiento a nivel de tuplas básico en un sistema relacional. Es decir, el único modo, garantizado por el sistema, de localizar alguna tupla específica es por el valor de su clave primaria.

Claves Ajenas

Una clave ajena es un atributo (quizá compuesto) de una relación R2 cuyos valores deben concordar con los de la clave primaria de alguna relación R1.
Un valor de clave ajena representa una referencia a la tupla donde se encuentra el valor correspondiente de la clave primaria.

Ejemplo de aplicación de Claves Primarias y Claves Ajenas

Relaciones

Equivalencias Informales de la Terminología

Debe entenderse que las equivalencias mostradas son solo aproximadas, porque los términos formales del modelo relacional tienen definiciones precisas, pero los equivalentes informales solo poseen definiciones aproximadas.

Propiedades de las Relaciones

1. No existen tuplas repetidas
2. Las tuplas no están ordenadas (de arriba hacia abajo)
3. Los atributos no están ordenados (de izquierda a derecha)
4. Todos los valores de los atributos son atómicos (indivisibles)

Propiedades de las Relaciones

1. **No existen tuplas repetidas**: Esta propiedad es consecuencia del hecho de que el cuerpo de la relación es un conjunto matemático (es decir, un conjunto de tuplas), y en matemáticas los conjuntos por definición no incluyen elementos repetidos. Por cierto, esta primera propiedad nos sirve de inmediato para ilustrar la diferencia entre una relación y una tabla, porque una tabla podría contener filas repetidas, al faltar una disciplina que evite tal situación, pero una relación no puede contener tuplas repetidas (pues si una "relación" contiene tuplas duplicadas, no es una relación por definición).

2. **Las tuplas no están ordenadas** (de arriba hacia abajo): No existe el concepto de direccionamiento por posición y tampoco el de adyacencia. Por tanto, no puede hablarse de la "quinta tupla" o "la tupla 97", tampoco existe la "siguiente tupla".

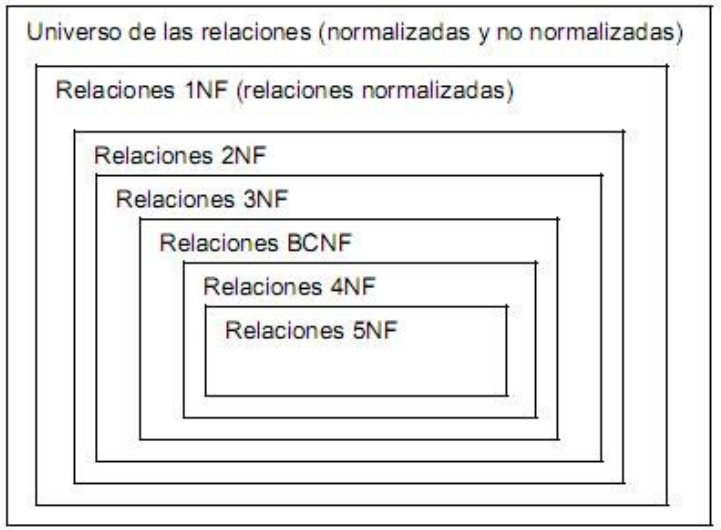
3. **Los atributos no están ordenados** (de izquierda a derecha): Esta propiedad se desprende del hecho de que la cabecera de una relación se define también como conjunto (es decir, un conjunto de atributos o, dicho en forma más precisa, de pares atributo-dominio). Por ejemplo, los atributos de la relación S podrían haberse presentado en el orden SNombre, Ciudad, Situac, S#, de todos modos sería la misma relación.

4. **Todos los valores de los atributos son atómicos**: Podemos expresar esta propiedad de manera más sencilla de este modo: en cada posición de fila y columna dentro de la tabla, siempre existe un solo valor, nunca una lista de valores. O en otra forma equivalente; las relaciones no contiene grupos repetitivos. Si una relación satisface esta condición, se dice que esta normalizada (una forma equivalente de de expresar esto es decir que la relación esta en la primera forma normal)

Formas Normales

Consiste en definir el formato lógico adecuado para las estructuras de datos identificados en el proyecto lógico del sistema, con el objetivo de minimizar el espacio utilizado por los datos y garantizar la integridad y confiabilidad de las informaciones.
La normalización es hecha, a través del análisis de los datos que componen las estructuras utilizando el concepto llamado "Formas Normales (FN)". Las FN son conjuntos de restricciones los cuales los datos deben satisfacer.

Se puede decir que la estructura está en la primera forma normal (1FN), si los datos que la componen satisfacen las
restricciones definidas para esta etapa. La normalización completa de los datos es hecha, siguiendo las restricciones de las cuatro formas normales existentes, siendo que el paso de una FN a otra es hecha teniendo como base el resultado obtenido en la etapa anterior, o sea, en la FN anterior.



Para realizar la normalización de los datos, es primordial que sea definido un campo clave para la estructura, campo este que permitirá identificar los demás campos de la estructura.

Como sugiere la siguiente figura, todas las relaciones normalizadas están en la 1NF; algunas relaciones 1NF están también en la 2NF, y algunas relaciones 2NF están también en la 3NF.

Primera Forma Normal (1FN)

Consiste en retirar de la estructura los elementos repetitivos, o sea, aquellos datos que pueden componer una estructura de vector. Podemos afirmar que una estructura está normalizada en la 1FN, si no posee elementos repetitivos.

Ejemplo:

Estructura original:

Ventas
Número
Fecha de emisión
Cod. del Cliente
Nombre del Cliente
Dirección
Mercaderías Vendidas
dónde para cada mercadería tenemos:
Código de la mercadería,
Descripción,
Cantidad,
Precio de venta
Total de la venta de esta mercadería
Total General de la Venta

Analizando la estructura expuesta, observamos que existen varias mercaderías en una única Venta, siendo por tanto elementos repetitivos que deberán ser retirados.

Estructura en la primera forma normal (1FN):

Ventas
Número
Fecha de emisión
Cód. del Cliente,
Nombre del Cliente,
Dirección
Total General de la Venta

ItemVenta
Numero
Código de la mercadería
Descripción
Cantidad
Precio de venta
Total de la venta de esta mercadería

Como resultado de esta etapa ocurre una descomposición de los datos en dos estructuras:
Primera estructura (Archivo Ventas): Datos que componen la estructura original, excluyendo los elementos repetitivos.
Segunda estructura (Archivo ItemVenta): Datos que componen los elementos repetitivos de la estructura original, tenido como clave el campo clave de la estructura original (Número) y el campo clave de la estructura de repetición (Código de la Mercaderías)

Segunda Forma Normal (2FN)

Consiste en retirar de las estructuras que poseen claves compuestas (campo clave siendo formado por más de un campo), los elementos que son funcionalmente dependiente de parte de la clave. Podemos afirmar que una estructura está en la 2FN, si ella está en la 1FN y no posee campos que son funcionalmente dependientes de parte de clave.

Estructura en la primera forma normal (1FN):
Ventas
Número, Fecha de emisión, Cód. del Cliente, Nombre del Cliente, Dirección y total General de la Venta
ItemVenta
Numero, Código de la mercadería, Descripción, Cantidad, Precio de venta y total de la venta de esta mercadería

Estructura en la segunda forma normal (2FN):
Ventas
Número, Fecha de emisión, Cód. del Cliente, Nombre del Cliente, Dirección y total General de la Venta
ItemVenta
Numero, Código de la mercadería, Cantidad, Precio de venta y total de la venta de esta mercadería
Mercaderías
Código de la Mercadería, Descripción, Precio de venta

Como resultado de esta etapa, hubo un desdoblamiento del archivo ItemVenta (el archivo Ventas, no fue alterado, por no poseer clave compuesta) en dos estructuras a saber:

Primera estructura (Archivo ItemVenta): Contiene elementos originales, siendo excluidos los datos que son dependientes apenas del campo Código de la Mercaderías.
Segundo estructura (Archivo Mercaderías): Contiene los elementos que son identificados apenas por el Código de la Mercadería, o sea, independientemente de la Venta, la descripción y el precio de venta serán constantes.

Tercera Forma Normal (3FN)

Consiste en retirar de las estructuras los campos que son funcionalmente dependientes de otros campos que no son claves. Podemos afirmar que una estructura está en la 3FN, si ella está en la 2FN y no posee campos dependientes de otros campos no claves.

Estructura en la segunda forma normal (2FN):
Ventas
Número, Fecha de emisión, Cód. del Cliente, Nombre del Cliente, Dirección y total General de la Venta
ItemVenta
Numero, Código de la mercadería, Cantidad, Precio de venta y total de la venta de esta mercadería
Mercaderías
Código de la Mercadería, Descripción, Precio de venta

Estructura en la tercera forma normal (3FN):
Ventas
Número, Fecha de emisión, Cód. del Cliente y total General de la Venta
ItemVenta
Numero, Código de la mercadería, Cantidad, Precio de venta y total de la venta de esta mercadería
Mercaderías
Código de la Mercadería, Descripción, Precio de venta
Clientes
Código del Cliente, Nombre del cliente, Dirección

Como resultado de esta etapa, hubo un desdoblamiento del archivo Ventas, por ser el único que poseía campos que no eran dependientes de la clave principal (Número), una vez independizado de Ventas, el Nombre y la Dirección del cliente son inalterados. Este procedimiento permite evitar inconsistencia en los datos de los archivos y ahorrar espacio por eliminar el almacenamiento frecuente y repetidas veces de estos datos. A cada VENTA al Cliente determinado, habrá el almacenamiento de estos datos y podrá ocurrir incoherencia entre ellos.

Las estructuras alteradas fueron por los motivos, a saber:

Primera estructura (Archivo Ventas):
Contiene los elementos originales, siendo excluidos los datos que son dependientes apenas del campo Código del Cliente (informaciones referentes al cliente).

Segunda estructura (Archivo Clientes):
Contiene los elementos que son identificados apenas por el Código del Cliente, o sea, independiente de la Venta, el Nombre y la dirección de los clientes serán constantes.

Luego de la normalización, las estructuras de los datos están proyectadas para eliminar las inconsistencias y redundancias de los datos, eliminando de esta forma cualquier problema de actualización del sistema. La versión final de los datos podrá sufrir alguna alteración, para atender las necesidades específicas del sistema, a criterio del analista de desarrollo durante el proyecto físico del sistema.

Transacciones

Conceptos, arquitectura

Las transacciones en bases de datos cliente/servidor son de extrema importancia y su uso representa una de las mayores evoluciones cuando se migra de una base de datos desktop a un verdadero SGBD (Sistema Gerenciador de Base de Datos). Ahora veremos algunos conceptos de transacciones, detalles de la arquitectura versioning, y las diferencias entre varios tipos de aislamientos transaccionales, además del relacionamiento entre ellos.

Arquitectura de versioning

Es importante tener una noción básica de lo que es versioning, el sistema gerenciador de concurrencia utilizado en el servidor, también llamado de Multi-Generational Architecture (MGA). En esta arquitectura, cuando se altera alguna información en la base de datos, versiones temporales de los registros alterados son creados y permanecen activas durante el tiempo que fuere necesario, permitiendo que el registro quede accesible – con su contenido original – a otras transacciones, que estarán teniendo siempre una vista consistente de los datos.

Las versiones temporarias de los registros, obviamente, ocupan espacio en la base de datos. Cuando los registros temporales no fueren más necesarios, el proceso de recolección de basuras (garbage collection) se encarga de marcarlos como basura para futuro reaprovechamiento del espacio ocupado. Por esa razón, una base de datos nunca disminuye de tamaño (a no ser cuando un backup/restore es ejecutado).

Esa es una medida eficiente, pues la utilización de nuevo espacio en disco es mucho más onerosa que el reaprovechamiento de un espacio ya utilizado.

Como versioning crea versiones temporales de los registros con los que se esta trabajando, en una situación de falla o error, el SGBD apenas marca el plan de la transacción como RollBack haciendo con que todos los registros asociados a ella sean descartados, dejando la base de datos consistente en cuestión de seguridad.

Transacciones en la Práctica

Para los que no estén acostumbrados con el concepto de transacciones, una forma simple de entenderlo es la siguiente: imagine que una transacción define un bloque de operaciones realizadas en una o más tablas en la base de datos, siendo que si alguna de esas operaciones fallan, entonces todas serán deshechas. La confirmación (commit) o la cancelación (rollback) de la transacción determina su fin.

En un ejemplo práctico, imagine un sistema de facturación con una operación de emisión de comprobante de venta que automáticamente descuenta los productos del stock y envia los movimientos en el sistema de cuentas a cobrar. En este proceso, se puede hacer con que las operaciones realizadas en las tablas involucradas pertenezcan en una única transacción – en el caso de que ocurra un error – pueden ser revertidas todas las alteraciones generadas por la facturación en curso con un simple rollbak, haciendo con que la base de datos permanezca en un estado consistente.

Comandos para manipulación de transacciones

Básicamente son tres los comandos principales responsables por la manipulación de transacciones:

Begin: Como su nombre sugiere, ese comando es responsable por el inicio de una transacción.

Commit: Indica al SGBD que todas las alteraciones, inserciones y remociones realizadas en la base de datos por la transacción deben ser efectivamente grabadas.

RollBack: Dice al servidor de base de datos que todas las operaciones realizadas por la transacción deben ser desechas.

Una transacción debe ser siempre encerrada por un commit o rollback. Es muy importante que las transacciones queden abiertas por el menor tiempo posible, a fin de evitar problemas como perdida de desempeño, entre otros

El algoritmo clásico para operaciones con datos dentro de una transacción es:

```
// Inicio de la transacción
Begin;
        Manipulación de datos a través de componentes (query, etc.)
        relacionados con la transacción, realizando inserts, updates, deletes, etc)
        // Confirmacion de la transacción
        Commit;
        except
        // En caso de error
        Rollbak;
end;
```

Control de Usuarios

Privilegios

Cuando un objeto de base de datos es creado, es atribuido un dueño al mismo. El dueño es el usuario que ejecutó el comando de creación. Para cambiar el dueño de una tabla, índice, secuencia o vista debe ser utilizado el comando ALTER TABLE. Por padrón, solamente el dueño (o un superusuario) puede hacer cualquier cosa con el objeto. Para permitir el uso por otros usuarios, deben ser concedidos privilegios.

Existen varios privilegios distintos: SELECT, INSERT, UPDATE, SELETE, RULE, REFERENCES, TRIGGER, CREATE, TEMPORARY, EXECUTE, USAGE y ALL PRIVILEGES.

Comandos GRANT y REVOKE

El derecho de modificar o remover un objeto es siempre privilegio apenas del dueño.
Es utilizado el comando GRANT para conceder privilegios. Por tanto, si joel fuere un usuario existente, y tb_cuentas fuere una tabla existente, el privilegio de actualizar la tabla puede ser concedido mediante el comando
***GRANT UPDATE ON tb_cuentas TO joel*;**

Este comando debe ser ejecutado por el dueño de la tabla. Para conceder privilegios a un grupo debe ser utilizado el comando:
GRANT SELECT ON tb_cuentas TO GROUP grp_finanzas;

El nombre especial de usuario PUBLIC puede ser utilizado para conceder el privilegio para todos los usuarios del sistema. Escribiendo ALL en el lugar del privilegio especifica la concesión de todos los privilegios.

Para revocar un privilegio debe ser utilizado el comando REVOKE
REVOKE ALL ON tb_cuentas FROM PUBLIC;

Los privilegios especiales del dueño de la tabla (DROP(remover), GRANT(conceder), REVOKE (revocar), etc) son siempre implícitos al hecho de ser dueño, no pudiendo ser concedidos o revocados. Pero el dueño de la tabla puede decidir revocar sus propios privilegios comunes como, por ejemplo, haciendo que una tabla sea de solo lectura para el mismo y para los otros.

Control de concurrencia multiversión (MVCC o MCC)

Es un método comúnmente utilizado por SGBD, para ofrecer un acceso concurrente a la base de datos.
MVCC proporciona a cada usuario conectado a la base de datos con una "instantánea“ para trabajar. Ningún cambio realizado podrá ser visto por otros usuarios de la base de datos hasta que la transacción reciba un **commit**.

Esto garantiza que una operación no tenga que esperar, ya que la base de datos mantiene varias versiones del mismo objeto.

Si una transacción (T i) quiere escribir en un objeto, y si hay otra operación (T k), debe preceder a la primera transacción realizada para que la operación de escritura tenga éxito.

Cada objeto tendría también una lectura de hora, y si la operación T intenta escribir en el objeto P, y la fecha de la transacción es anterior a la lectura del objeto, la operación T se aborta y reinicia. De lo contrario, T crea una nueva versión de P y establece la lectura / escritura de tiempo de operación.
.
La evidente desventaja de este sistema es el costo de almacenamiento de varias versiones de los objetos en la base de datos. Por otro lado, el hecho de que el objeto nunca se halle bloqueado, puede ser importante para el volumen de trabajo que afectan sobre todo a la lectura de valores de la base de datos.

MVCC es particularmente experto en la implementación verdadera del **snapshot isolation** (aislamiento instantáneo), algo que otros métodos de control de concurrencia con frecuencia tienen incompleto o con alto costo en cuanto a desempeño .

Historia de PostgreSQL

La herramienta conocida actualmente como PostgreSQL tuvo su origen en un proyecto llamado Postgres, en la Universidad Berkeley, en California (EUA), en 1986. Un equipo orientado por el Prof. Michael Stonebraker fue designado para crear un modelo y las reglas de un nuevo sistema de almacenamiento de datos, con el apoyo de diversos órganos.

La primera versión de demostración estuvo lista en 1987. En 1989, la primera versión estable fue lanzada, con sucesivos lanzamientos anuales de nuevas versiones con diversas correcciones de bugs. A mediados del 1991, su código fue adquirido por la empresa Illustra Information Technologies, la cual se fisionó con Informix, hoy pertenece a la IBM; y es utilizando como gerenciador de base de datos en un importante proyecto científico.

Vale la pena comentar que la empresa Informix era de propiedad del Prof. Michael Stonebraker , orientador del proyecto Postgres, y que la misma fue comprada por US$. 1 billón por la IBM en 2001.

El primer gran cambio en el proyecto POSTGRES ocurrió en 1994. Debido a la creciente popularidad que la herramienta estaba adquiriendo, el proyecto fue encerrado dando origen al programa Postgres95, que trajo una gran ventaja en su primera versión: la incorporación del lenguaje SQL, por los desarrolladores Andrew Yu y Jolly Chen, sustituyendo al lenguaje PostQUEL anteriormente utilizada. En la época, también el programa fue totalmente compatibilizado con el padrón ANSI C, volviéndose portable para mas de una plataforma, entre otras diversas mejoras, que lo convirtieron en un ícono entre las bases de datos.

En el año 1996, nuevas mejorías surgieron, y el nombre Portgres95 ya estaba desactualizado. Nuevamente la denominación de la herramienta fue cambiada, esta vez por el nombre por la cual lo conocemos hoy: PostgreSQL. Actualmente se encuentra en su version 8.4 estable, contando con todas las principales características que un SGBD puede disponibilizar.

Licencia de Uso

A diferencia de la mayoría de los softwares libres existentes en el mercado, el PostgreSQL no utiliza la licencia GNU para regularizar su utilización, pero si la licencia BSD (Berkeley Software Distribution).

Originada juntamente con el sistema operativo FreeBSD, la licencia BSD obtuvo reconocimiento y varios otros softwares actualmente también la utilizan gracias a su libertad frente a otras licencias de software.

Como puede notarse, tanto el PostgreSQl como la licencia BSD tuvieron origen en el mismo local; la Universidad de Berkeley, en California. Este es el primer factor que hace que el PostgreSQL utilice esta licencia, pues los intereses iniciales de la herramienta y de la licencia tenían puntos comunes.

La licencia BDS posee innumerables ventajas sobre la licencia GNU, o mejor, innumerables restricciones menos de las impuestas por la otra licencia. Esto lo convierte en un código mucho mas accesible para diversos tipos de utilizaciones, incluyendo la libre utilización de la herramienta, incluso para fines comerciales.

Utilizaciones Recomendadas

El PostgreSQL se encuentra en una versión perfectamente estable y confiable, con los principales recursos existentes en bases de datos pagadas disponibles en el mercado. Sus capacidades pueden suplir las necesidades de pequeñas, medianas y grandes aplicaciones.

Capacidad de Almacenamiento

No tiene limites de tamaño para sus bases de datos, siendo la única limitación para tal criterio el hardware disponible por la computadora en la que el PostgreSQL este almacenando sus informaciones. Su limitación se da a nivel de tablas, con un limite máximo de 32 TB por tabla. Además es posible tener registros con hasta 1.6 TB, campos con hasta 1 GB, tablas con hasta 1600 campos e índices ilimitados para aceleración en resultado de búsqueda .

Breve resumen de la capacidad de almacenamiento	
Tamaño máximo de una base de datos	Ilimitado
Tamaño máximo de una tabla	32 TB
Tamaño máximo de un registro	1.6 TB
Tamano máximo de una campo	1 GB
Cantidad máxima de campos por tabla	De 250 a 1600, dependiendo de los tipos de datos utilizados
Tamano máximo de índices por tabla	Ilimitado

Compatibilidades

Ahora veremos algunas informaciones generales sobre la compatibilidad del PostgreSQL, en lo que respecta a sistemas operativos, lenguajes de programación , plataformas de desarrollo y versión de SQL utilizada.

Para los programadores y desarrolladores existen bibliotecas y drivers de conexión para las principales plataformas y lenguajes utilizados, pudiendo citarse las siguientes: C/C++, Java/JSP, PHP, ASP, .NET, Perl, Python, Ruby, Tcl y driver ODBC, entre otros.

Sobre ambientes de instalación, el PostgreSQL es una herramienta extremamente portable, disponibilizando instalaciones para diversos sistemas operativos, como por ejemplo:

- *Linux*
- *Unix*
- *Max OS X Server*
- *Windows*

Tratándose de compatibilidad con lenguaje SQL, el PostgreSQL ya cuenta con varios recursos implementados de la versión ANSI SQL 2003, siendo la primera base de datos implementar algunas de las más recientes definiciones en la historia de SQL como Gin y DTrace.

Principales Características

SGBD relacional con soporte a ACID (transacciones)

Posee soporte a operaciones ACID (Atomicidad, Consistencia, Isolamiento y Durabilidad). Cada una de estas propiedades garantiza una parte de la calidad de los servicios disponibilizados por las bases de datos. En lo que respecta a la integridad referencial, el PostgreSQL posee y valida este tipo de operaciones en sus relacionamientos almacenados en tablas.

Replicación

Así como las demás bases de datos, PostgreSQL ofrece recursos necesarios para realizar la replicación entre servidores. La ventaja sobre las otras bases de datos es que su licencia abre este recurso para uso gratuito incluso para aplicaciones comerciales, a diferencia de otras licencias de software libre, utilizadas por otras bases de datos.

Clúster (alta disponibilidad)

Visionando expandir su capacidad para mas de un servidor (hardware) recurrente de las limitaciones de procesamiento, es posible configurar el PostgreSQL para actúe como un clúster de informaciones.

La utilización de clúster envuelve el uso de dos o mas computadoras, interligadas y sincronizadas entre si, para que ambas puedan atender las demandas venidas de usuarios de la aplicación o base de datos en cuestión, en la teoría, duplicando la capacidad de utilización.

Multithreads

El PostgreSQL gerencia varias conexiones con base de datos de una única vez, utilizando el recurso miltithreads ofrecido por los sistemas operativos. De esta forma, mas de una persona puede acceder a la misma información sin ocasionar atrasos o filas de accesos. Algunas operaciones fuerzan el uso de filas de acceso a datos, principalmente aquellas en que mas de un usuario este intentando realizar un acceso de grabación en los mismos datos. El PostgreSQL administra estas situaciones para que los datos no sean corrompidos.

Seguridad SSL y Criptografía

El soporte nativo a SSL ya está embutido en PostgreSQL, posibilitando crear conexiones seguras a partir de estos canales, tanto para transmisión de informaciones de login como para aquellas consideradas sigilosas. Además el PostgreSQL ofrece extensibilidad para utilización de algoritmos de criptografía como SHA1 y MD5 (ya nativo en sus últimas versiones)

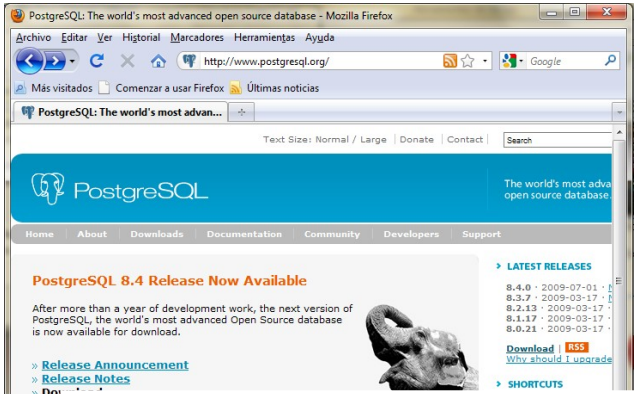
SQL

Basado en los padrones establecidos por ANSI SQL, el PostgreSQL adopta este criterio en la implementación de sus funciones.

Incorporable en aplicaciones gratuitamente

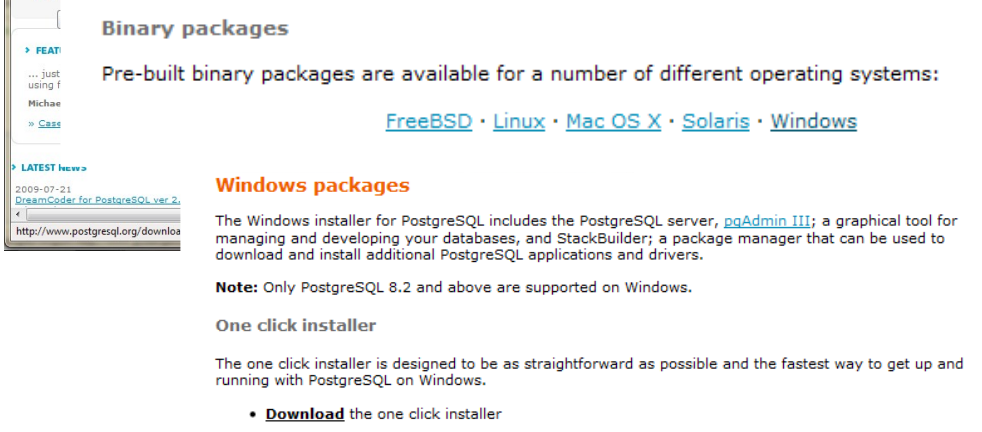
Por utilizar la licencia de uso BSD, el PostgreSQL puede libremente ser incorporada por aplicaciones personales y/o comerciales, sin ningún costo.

Obteniendo una copia de PostgreSQL



El primer paso necesario para instalar el PostgreSQL, es la obtención de una copia del instalador, accediendo a la dirección www.postgresql.org.

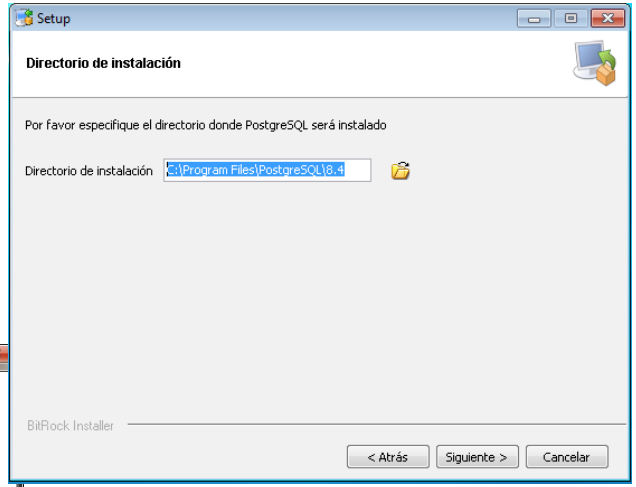
En la página inicial, seleccione el botón **Download**. Este link llevará a una conexión para el FTP, donde deberá seleccionar el sistema operativo para el cual desea obtener el instalador.



Instalando PostgreSQL en Windows

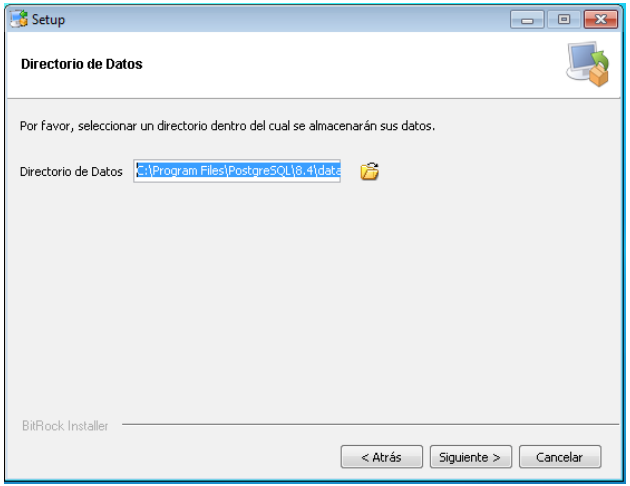
Paso 1:

Después de realizar el download del archivo de instalación, deberá ejecutar el instalador. Visualizará la pantalla de bienvenida, seleccione la opción **Siguente**.



Paso 2:

Deberá especificar el directorio en el cual se instalará el PostgreSQL

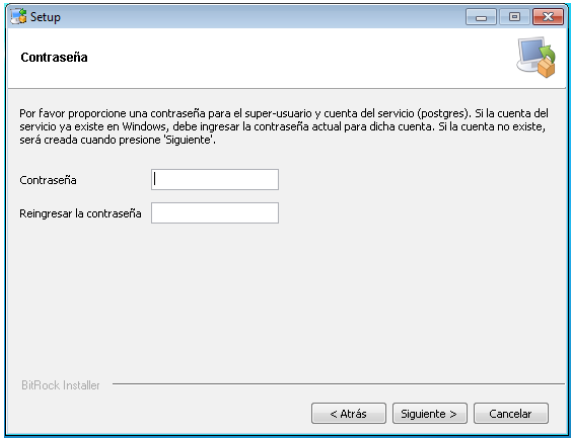


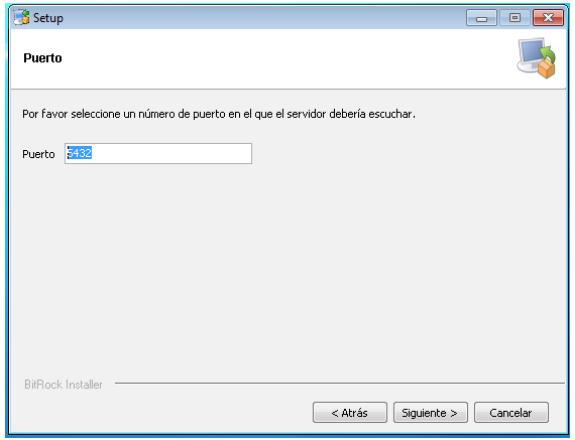
Paso 3:

Deberá seleccionar el directorio dentro del cual se almacenarán los datos.

Paso 4:

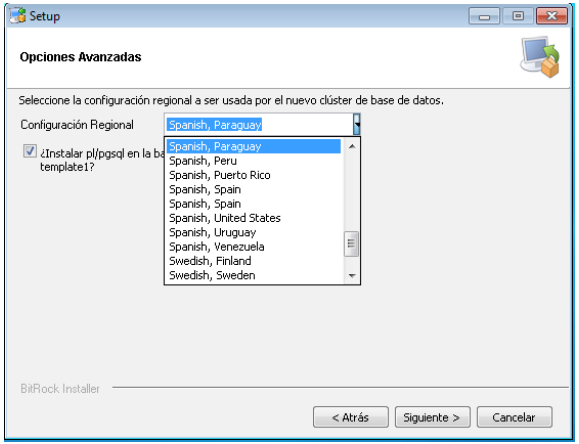
En esta etapa se crea el súper-usuario y cuenta se servicio **postgres**.. La seña asignada a la cuenta, será establecida para el usuario del sistema operativo, como también para el usuario de la base de datos.





Paso 5:

Ahora deberá seleccionar el numero de puerto en el que el servidor escuchará. El puerto predeterminado del PostgreSQL es el 5432. Pero podrá ser asignado otro número, siempre que no se encuentre en uso.

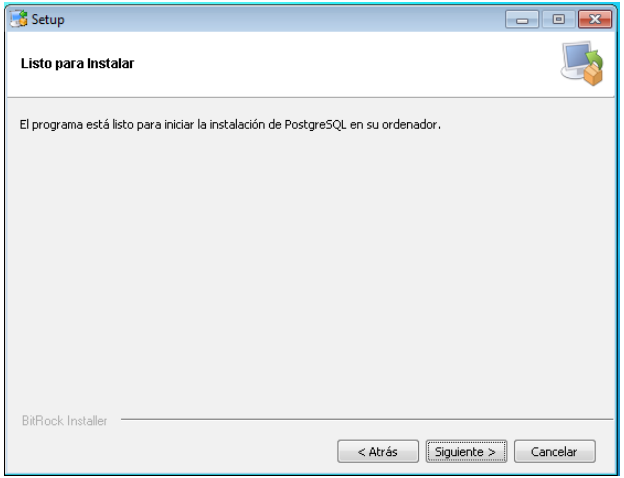


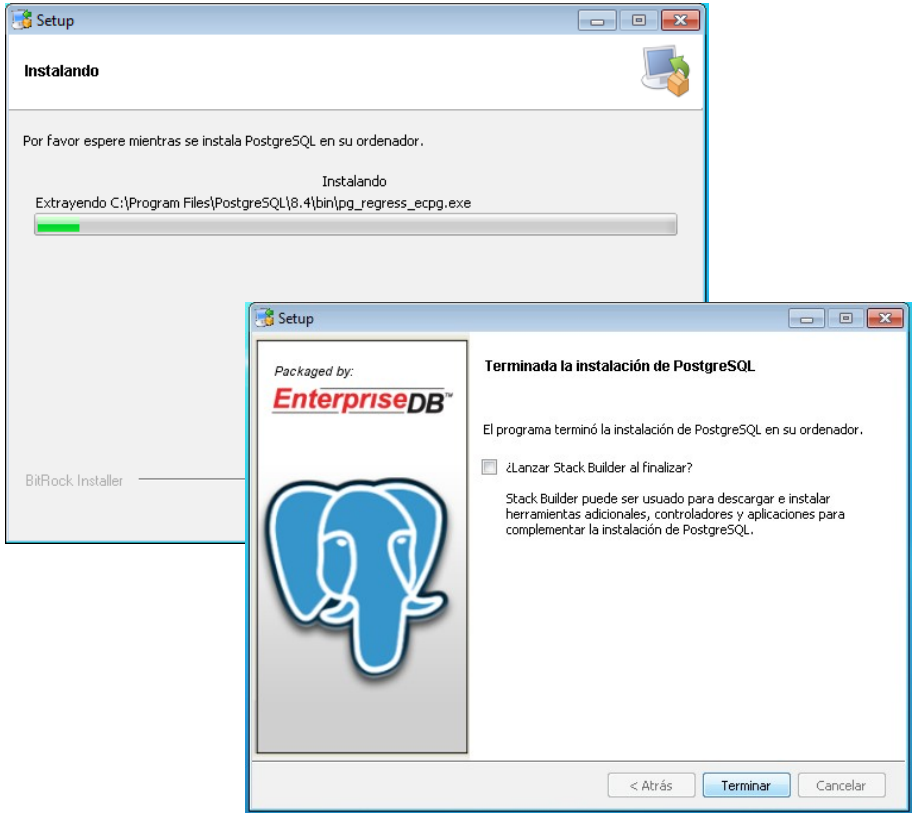
Paso 6:

Seleccione la configuración regional (Spanish, Paraguay), y marque la opción para instalar el pl/pgsql, que es el lenguaje procedural nativo del PostgreSQL.

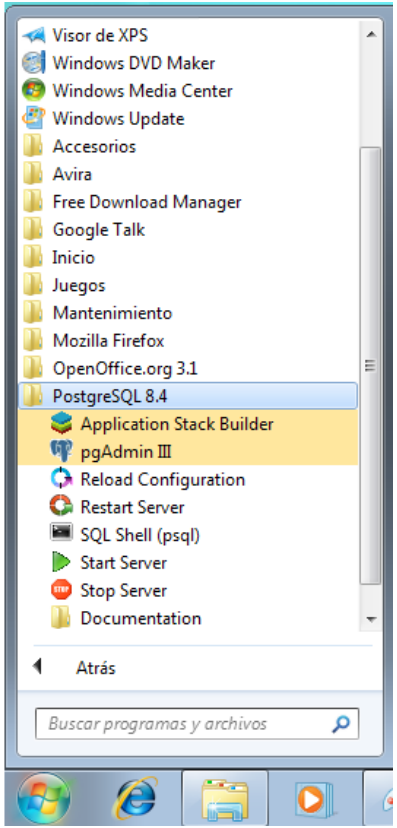
Paso 7:

Con todos los parámetros previos ingresados, se inicia la instalación en el sistema operativo. Seleccione la opción Siguiente para proseguir.





Configuraciones iniciales



Una vez finalizada la instalación, podrá visualizar una serie de herramientas, siendo las principales, las que a continuación se detallan:

pgAdminIII: Herramienta gráfica de administración de la base de datos.

psql: Consola interactiva de administración de la base de datos.

Reload Configuración: Releer la configuración de la base de datos, después de algún cambio introducido, sin parar el servicio activo.

Restart Server: Parar y volver a levantar el servicio de la base de datos.

Start Server: Levantar el servicio.

Stop Server: Parar el servicio.

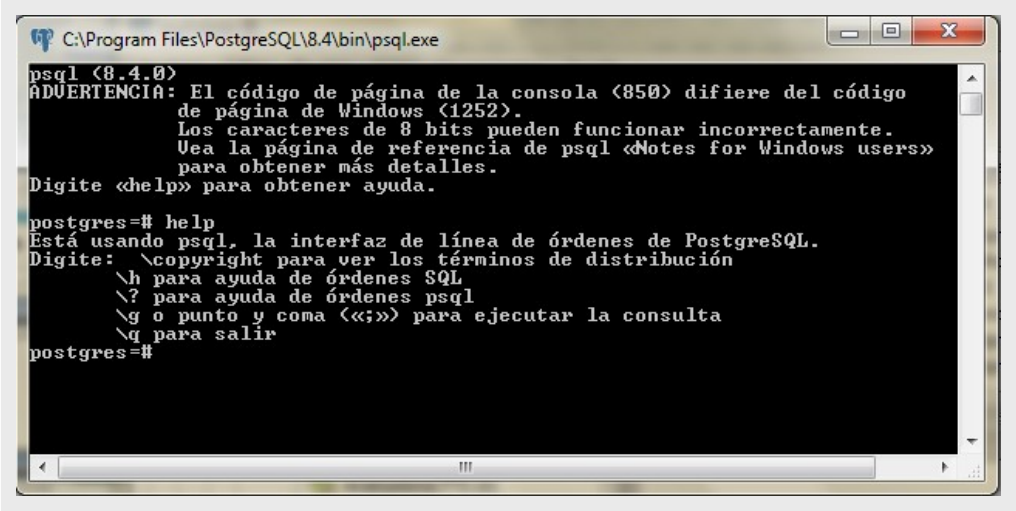
Por defecto, el servicio es levantado en forma automática, al iniciar el sistema operativo. Esta configuración puede ser alterada accediendo a la administración de servicios.

Herramientas de Administración

Existen varias herramientas gráficas de alta calidad para administrar la bases de datos.

PSQL

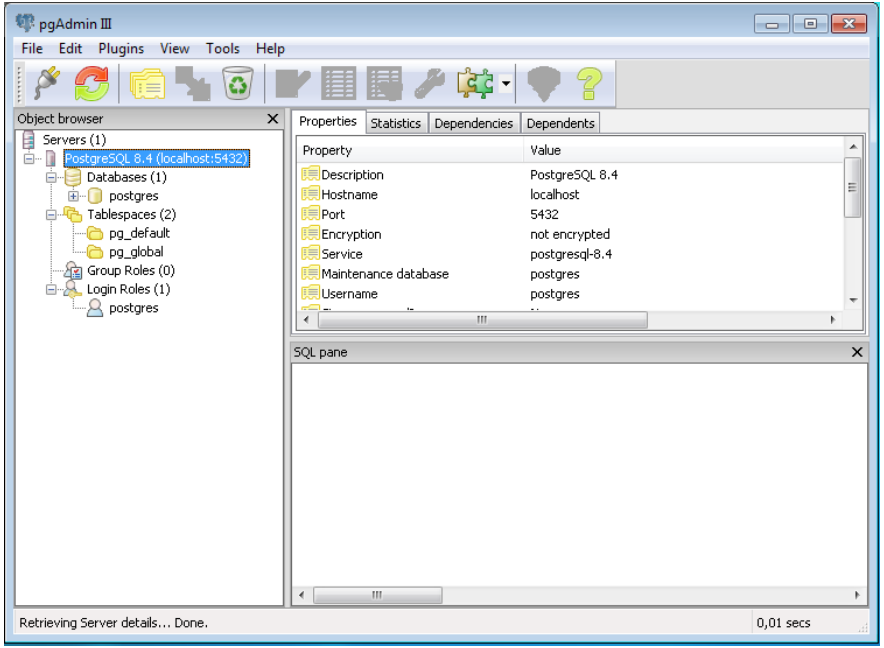
Es la herramienta canónica para la ejecución de sentencias SQL a través del shell del SO. Es una herramienta de tipo frontend que permite describir sentencias SQL, ejecutarlas y visualizar sus resultados. El método de ingreso puede ser mediante la inserción directa del código en la consola, o la ejecución de sentencias dentro de un archivo de texto. Provee de diversos meta-comandos para la ejecución de las sentencias, así como diversas opciones tipo shell propias de la herramienta



PHPPgAdmin

Es un poderosa herramienta de administración basada en un interfaz Web para bases de datos PostgreSQL. Además de la funcionalidad básica, dispone de soporte para procedimientos almacenados, triggers y vistas. Las versiones de punta van mano a mano con el desarrollo del servidor PostgreSQL. Esta versión es una de la mas famosa de los administradores GUI para PostgreSQL.

pgAdminIII



pgAdmin es una interfaz de diseño y gestión de base de datos PostgreSQL, diseñada para funcionar en la mayoría de sistemas operativos.

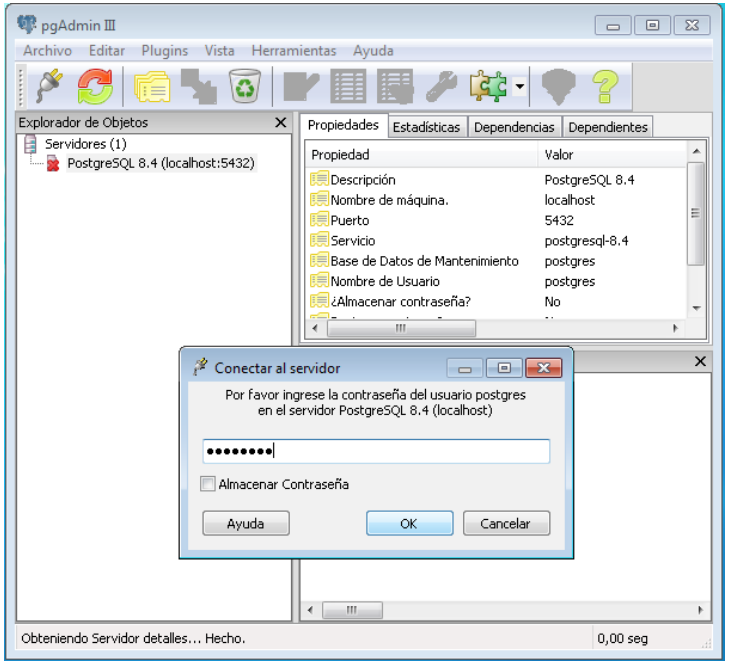
El software está escrito en C++ y utiliza la excelente plataforma wxWidgets.

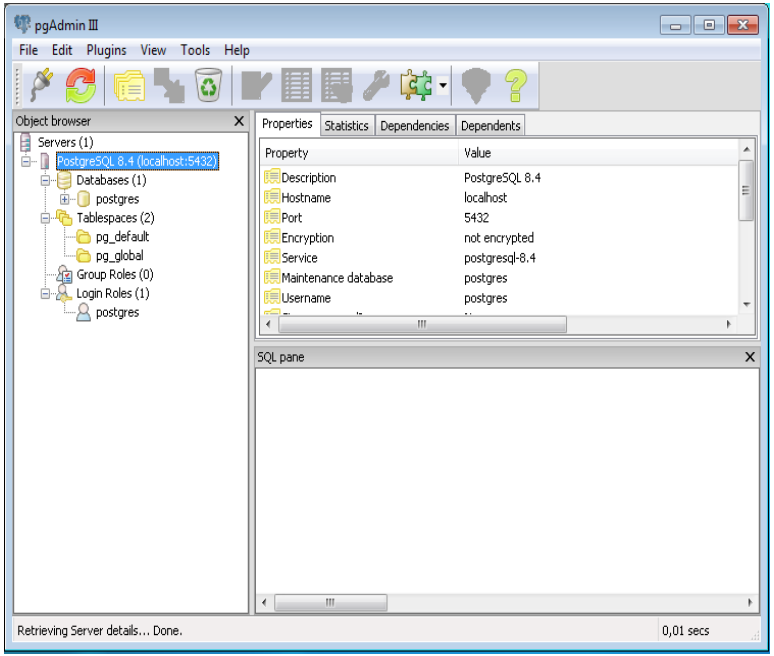
La aplicación es utilizada por decenas de miles de usuarios de todo el mundo.

Esta es la herramienta de administración que utilizaremos para el desarrollo de contenido de la materia.

Accediendo a la Herramienta de Administración

Al acceder a la herramienta de administración pgAdminIII, visualizamos un servidor, que se encuentra instalado en **localhost:5432** (o el puerto escogido durante la instalación). Para acceder se deberá digitar la contraseña del usuario solicitado.



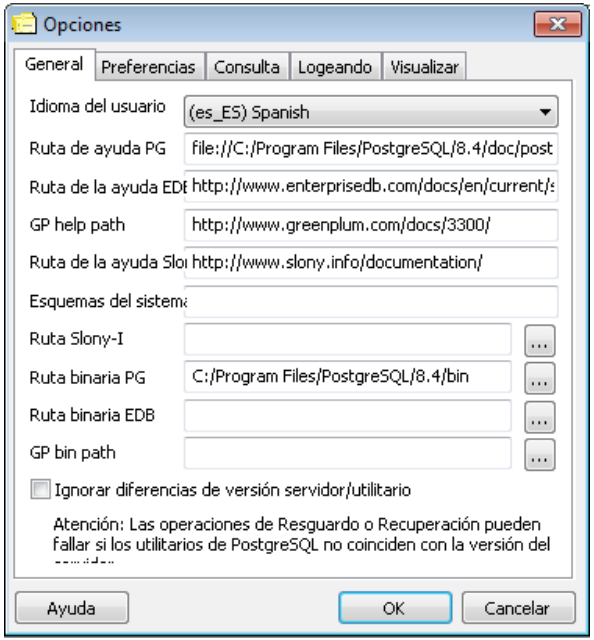


Una vez comprobada la validez de la seña, podrá acceder a la base de datos.

Podrá manipular los objetos (tablas, vistas, funciones, trigger, roles, etc), según los permisos que tenga.

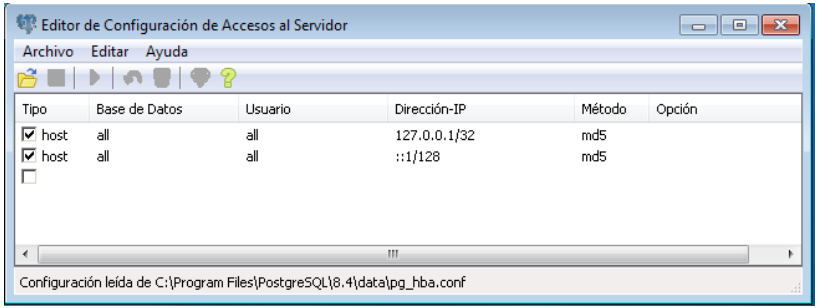
pgAdminIII. Configuración de Idioma

Para modificar el idioma de la herramienta de administración pgAdminIII, deberá acceder al menú Archivo, Opciones; en el ítem Idioma de Usuario y seleccionar la configuración deseada.



Archivos de Configuración

Editando el archivo **pg_hba.conf** podrá configurar los permisos de acceso al servidor, por dirección IP, por Base de Datos y/o por usuarios.



Otras configuraciones del motor se encuentran en los archivos **postgresql.conf** y **pgpass.conf**.

Introducción al Lenguaje SQL

La sigla SQL abrevia la expresión *Structure Query Language* (Lenguaje Estructurado de Consulta), nombre dado al lenguaje responsable por la interacción con los datos almacenados en la memoria de las bases de datos relacionales.

Las herramientas de manipulación gráfica de datos (pgAdmin, por ejemplo), utiliza el lenguaje SQL para interactuar con la base de datos; las terminales de texto también hacen uso del recurso.

Los recursos disponibilizados por este lenguaje son los siguientes:

Consultas: Posibilidad de capturar de la base de datos, informaciones almacenadas.

Actualizaciones: Hace posible la actualización de la información almacenada en la base de datos, a partir de sistemas que posean conexión con éste.

Filtros y actualizaciones: Permite que los datos retornados en una consulta puedan ser ordenados conforme a algún criterio, además de disponibilizar comandos que posibilitan formatear los resultados.

Lenguaje SQL. Recursos

Lenguaje de Definición de Datos – DDL *(Data Definition Language)*

Es un lenguaje proporcionado por el sistema de gestión de base de datos que permite a los usuarios de la misma, llevar a cabo las tareas de definición de las estructuras que almacenarán los datos así como de los procedimientos o funciones que permitan consultarlos.

Lenguaje de Manipulación de Datos – DML *(Data Manipulation Language)*

Es un lenguaje proporcionado por el sistema de gestión de base de datos que permite a los usuarios llevar a cabo las tareas de consulta o manipulación de los datos, organizados por el modelo de datos adecuado.

Lenguaje de Control de Datos – DCL (*Data Control Language*)

Es un lenguaje proporcionado por el sistema de gestión de base de datos que incluye una serie de comandos SQL que permiten al administrador controlar el acceso a los datos contenidos en la base de datos.

Clasificación	Sentencias
DDL	CREATE ALTER DROP
DML	INSERT UPDATE DELETE SELECT
DCL	GRANT REVOKE

Lenguajes de Definición de Datos - DDL

CREATE

Este comando crea un objeto dentro de la base de datos. Puede ser una tabla, vista, índice, trigger, función, procedimiento o cualquier otro objeto que el motor de la base de datos soporte.

SINTAXIS: **CREATE DATABASE: Crea una base de datos**

CREATE DATABASE *name*
[[WITH] [OWNER [=] *dbowner*]
[TEMPLATE [=] *template*]
[ENCODING [=] *encoding*]
[LC_COLLATE [=] *lc_collate*]
[LC_CTYPE [=] *lc_ctype*]
[TABLESPACE [=] *tablespace*]
[CONNECTION LIMIT [=] *connlimit*]]

SINTAXIS: **CREATE TABLE: Crea un nueva tabla**

CREATE TABLE *nombre_tabla* (
nombre_columna_1 *tipo_dato_1* [DEFAULT *default_expr*] [NOT NULL] ,
nombre_columna_n *tipo_dato_n* [DEFAULT *default_expr*] [NOT NULL] ,
[CONSTRAINT *nombre_clave_primaria*] PRIMARY KEY (*nombre_columna* [USING INDEX TABLESPACE *tablespace*] ,
[CONSTRAINT *nombre_clave_unica*] UNIQUE (*nombre_columna* [,....]) [USING INDEX TABLESPACE *tablespace*],
[CONSTRAINT *nombre_clave_foránea*] FOREIGN KEY (*nombre_columna*]
[CONSTRAINT *nombre_chk* CHECK *(condicion1,* [,...] *)*] ***;***
REFERENCES *nombre_tabla_referencia (nombre_campo_referencia)*] [MATCH FULL | MATCH PARTIAL |
MATCH SIMPLE] [ON DELETE *action*] [ON UPDATE *action*] }
[DEFERRABLE | NOT DEFERRABLE] [INITIALLY DEFERRED | INITIALLY IMMEDIATE]

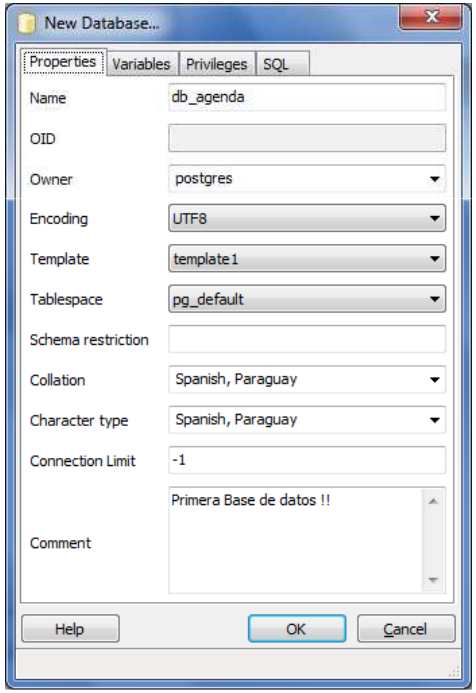
SINTAXIS: **CREATE VIEW: Crea una nueva vista**

CREATE OR REPLACE VIEW *nombre_vista* AS
SELECT *nombre_columna_1* [AS alias_columna_1] [, ...]
FROM *nombre_tabla1* [ALIAS] [, ...]
WHERE *condicion1* [, ...]
[UNION
SELECT *nombre_columna_1* [AS alias_columna_1] [, ...]
FROM *nombre_tabla1* [ALIAS] [, ...]
WHERE *condicion1* [, ...]
]
ORDER BY nombre_campo1 [ASC | DESC] [, ...] ;

Práctica de Laboratorio Nro.1

Crear una base de datos denominada **db_agenda,** cuyo propietario deverá ser el usuario postgres, con codificación UTF8, sin limite de conexión y que esté basada en la plantilla template1. Agregar un comentario con el texto siguiente “Primera Base de Datos”

```
CREATE DATABASE db_agenda
  WITH OWNER = postgres
       ENCODING = 'UTF8'
       LC_COLLATE = 'Spanish, Paraguay'
       LC_CTYPE = 'Spanish, Paraguay'
       CONNECTION LIMIT = -1;
COMMENT ON DATABASE db_agenda IS 'Primera Base de Datos';
```



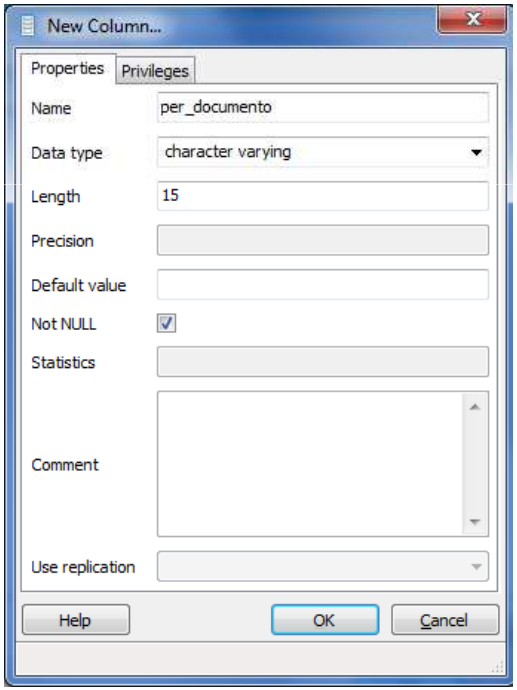
Práctica de Laboratorio Nro.2

Crear una tabla denominada **tb_persona,** que deberá estar contenida dentro de la base de datos **db_agenda**, y que posea los campos siguientes:

per_id	serial
per_documento	character varying (15)
per_nombre	character varying (30)
per_apellido	character varying (30)
per_telefono	character varying (20)
per_activo	boolean

✓Los campos per_documento y per_nombre no deberán permitir valores nulos.
✓El campo per_activo tendrá como valor predeterminado true.
✓Crear una clave primaria con el campo per_id, y una clave unica con el campo per_documento.

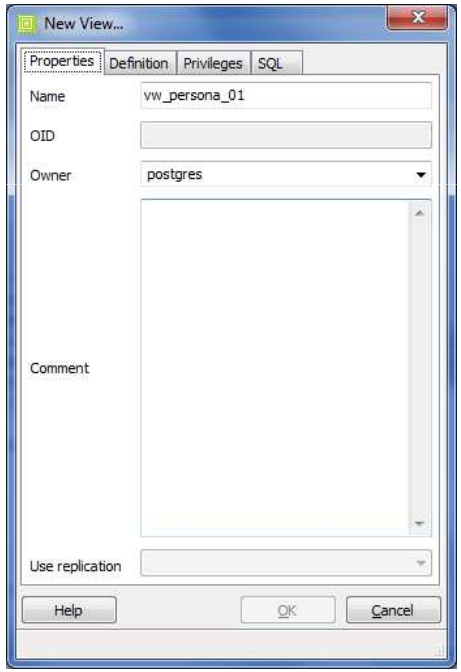
```
CREATE TABLE tb_persona
(
per_id serial,
per_documento character varying(15)  NOT NULL,
per_nombre character varying(30) NOT NULL,
per_apellido character varying(30),
per_telefono character varying(15),
per_activo boolean DEFAULT true,
CONSTRAINT per_id_pkey PRIMARY KEY (per_id),
CONSTRAINT per_documento_ukey UNIQUE (per_documento)
);
```

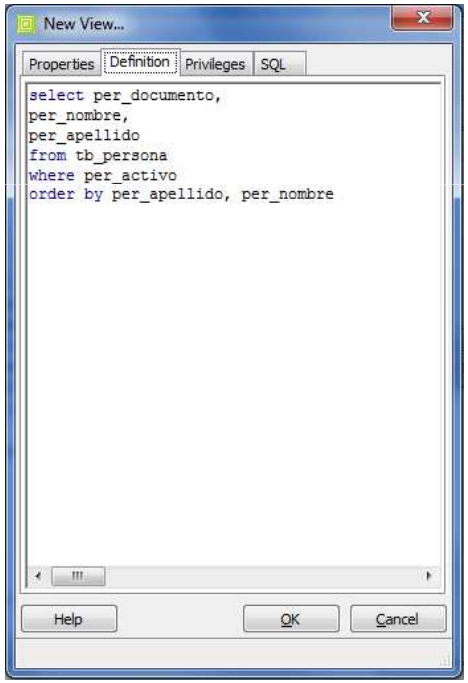


Práctica de Laboratorio Nro.3

Crear una vista de la tabla **tb_persona**, denominada **vw_persona_01**.
La vista deberá contener los campos per_documento, per_apellido y per_nombre.
Deberá esta ordernada por per_apellido y per_nombre y solo deberá contener personas con situación Activa (per_activo = true).

```
CREATE OR REPLACE VIEW vw_persona_01 AS
 SELECT  per_documento, per_nombre,  per_apellido
   FROM tb_persona
   WHERE per_activo
   ORDER BY per_apellido, per_nombre;
```





Lenguajes de Definición de Datos - DDL

ALTER

Este comando permite modificar la estructura de un objeto- Se pueden agregar / quitar campos a una tabla, modificar el tipo de un campo, agregar / quitar índices a una tabla, etc.

ALTER DATABASE, cambia los atributos de una base de datos

SINTAXIS: ALTER DATABASE *name*
[[WITH] *option* [...]]

Las opciones pueden ser:
CONNECTION LIMIT *connlimit*
ALTER DATABASE *name* RENAME TO *newname*
ALTER DATABASE *name* OWNER TO *new_owner*
ALTER DATABASE *name* SET TABLESPACE *new_tablespace*
ALTER DATABASE *name* SET *configuration_parameter* { TO | = } { *value* | DEFAULT }
ALTER DATABASE *name* SET *configuration_parameter* FROM CURRENT
ALTER DATABASE *name* RESET *configuration_parameter*
ALTER DATABASE *name* RESET ALL

ALTER TABLE, cambia los atributos de una tabla

SINTAXIS: ALTER TABLE *name action* [, ...]

Donde las acciones pueden ser:

RENAME [COLUMN] *column* TO *new_column*
RENAME TO *new_name*
SET SCHEMA *new_schema*
DROP CONSTRAINT *name_constraint*;
DROP COLUMN *name_column*;
ALTER *name_column* TYPE *type*;
ADD COLUMN per_nuevo integer;

Práctica de Laboratorio Nro.4

Acción ADD COLUMN

Agregue la columna per_sexo a la tabla tb_persona, con las características siguientes:

✓No deberá aceptar valores nulos
✓El valor predetermiando del campo deberá ser **M**

```
ALTER TABLE tb_persona
ADD COLUMN per_sexo character varying(1)
NOT NULL
DEFAULT 'M';
```

Práctica de Laboratorio Nro.5

Acción ADD CONSTRAINT

Agregue una restricción a la tabla tb_persona, en el campo per_sexo.
El mismo deberá aceptar solo los valores 'M' o 'F'; que indicaran el sexo Masculino o Femenino.

```
ALTER TABLE tb_persona
ADD CONSTRAINT
per_sexo_chk CHECK (per_sexo = 'M' or per_sexo = 'F');
```

Lenguajes de Definición de Datos - DDL

DROP

Este comando elimina un objeto de la base de datos. Puede ser una tabla, vista, índice, trigger, función, procedimiento o cualquier otro objeto que el motor de la base de datos soporte. Se puede combinar con la sentencia ALTER.

El comando DROP es utilizado para eliminar tablas, vistas, funciones, y la propia base de datos.

SINTAXIS: **DROP TABLE** *nombre_tabla*;
DROP VIEW *nombre_vista*;
DROP DATABASE *nombre_database;*
ALTER TABLE *nombre_tabla* **DROP COLUMN** *nombre_columna*;

Práctica de Laboratorio Nro.6

Elimine la vista vw_persona_01 y vuelva a crearla, incluyendo el campo per_id, en la primera posición de la misma.

```
Drop view vw_persona_01;

CREATE OR REPLACE VIEW vw_persona_01 AS
 SELECT per_id,  per_nombre,  per_apellido
   FROM tb_persona
   WHERE per_activo
   ORDER BY per_apellido, per_documento;
```

Práctica de Laboratorio Nro.7

Elimine el campo **per_telefono** de la tabla tb_persona

```
ALTER TABLE tb_persona
DROP COLUMN per_telefono
```

Lenguajes de Manipulación de Datos - DML

Es un lenguaje proporcionado por el sistema de gestión de base de datos que permite a los usuarios llevar a cabo las tareas de consulta o manipulación de los datos, organizados por el modelo de datos adecuado.

Incluye las sentencias siguientes:

Sentencia	Sintaxis
INSERT Inserta una columna en la tabla	INSERT INTO *table* [(*column* [, ...])] { DEFAULT VALUES | VALUES ({ *expression* | DEFAULT } [, ...]) [, ...] | *query* } [RETURNING * | *output_expression* [[AS] *output_name*] [, ...]]
UPDATE Actualiza datos en la tabla	UPDATE [ONLY] *table* [[AS] *alias*] SET { *column* = { *expression* | DEFAULT } | (*column* [, ...]) = ({ *expression* | DEFAULT } [, ...]) } [, ...] [FROM *fromlist*] [WHERE *condition* | WHERE CURRENT OF *cursor_name*] [RETURNING * | *output_expression* [[AS] *output_name*] [, ...]]
DELETE Elimina columnas de la tabla	DELETE FROM [ONLY] *table* [[AS] *alias*] [USING *usinglist*] [WHERE *condition* | WHERE CURRENT OF *cursor_name*] [RETURNING * | *output_expression* [[AS] *output_name*] [, ...]]
SELECT Recupera columnas desde una tabla o vista	SELECT [ALL | DISTINCT [ON (*expression* [, ...])]] * | *expression* [[AS] *output_name*] [, ...] [FROM *from_item* [, ...]] [WHERE *condition*] [GROUP BY *expression* [, ...]] [HAVING *condition* [, ...]] [{ UNION | INTERSECT | EXCEPT } [ALL] *select*] [ORDER BY *expression* [ASC | DESC | USING *operator*] [NULLS { FIRST | LAST }] [, ...]] [LIMIT { *count* | ALL }]

Práctica de Laboratorio Nro.8

Inserte los siguientes datos en la tabla tb_persona:

Documento	Nombres	Apellidos	Sexo	Activo
2343235	Pedro	Cantero	M	true
3456765	Estela	Sanabria	F	true
4567544	Jose	Leiva	M	false
1230005	Mariza	Vera	F	true

```
INSERT INTO tb_persona(per_id, per_documento, per_nombre, per_apellido, per_sexo, per_activo)
   VALUES (default, '2343235', 'Pedro', 'Cantero', 'M', true);

INSERT INTO tb_persona(per_id, per_documento, per_nombre, per_apellido, per_sexo, per_activo)
   VALUES (default, '3456765', 'Estela', 'Sanabria', 'F', true);

INSERT INTO tb_persona(per_id, per_documento, per_nombre, per_apellido, per_sexo, per_activo)
   VALUES (default, '4567544', 'Jose', 'Leiva', M', false);

INSERT INTO tb_persona(per_id, per_documento, per_nombre, per_apellido, per_sexo, per_activo)
   VALUES (default, '1230005', 'Mariza', 'Vera', 'F, true);
```

Práctica de Laboratorio Nro.9

Selecciona de la tabla tb_persona el registro que cumpla la siguiente condición: per_documento = '4567544'

```
SELECT   *
FROM     tb_persona
WHERE per_documento = '4567544';
```

Práctica de Laboratorio Nro.10

Toma el valor de per_id, retornado de la consulta anterior. Luego, actualiza el campo per_nombre con el dato "Jose Ramón", cuando el per_id sea igual al valor del per_id recuperado.

```
UPDATE tb_persona
  SET per_nombre= 'Jose Ramon'
 WHERE per_id  = 3;
```

```
UPDATE tb_persona
  SET per_nombre= 'Jose Ramon'
 WHERE per_id  = (select per_id   from    tb_persona   where   per_documento = '4567544');
```

Práctica de Laboratorio Nro.11

Elimina los registros de la tabla tb_persona, cuyo campo per_activo sea igual a false.

```
DELETE FROM tb_persona
 WHERE per_activo = false;
```

```
DELETE FROM tb_persona
 WHERE   !per_activo;
```

Lenguajes de Control de Datos - DCL

GRANT — Define privilegios de acceso

El comando GRANT tiene dos variantes básicas: una que otorga privilegios sobre una base de datos objeto (cuadro, columna, vista, secuencia, la base de datos, servidor extranjero, la función, lenguaje procedural, esquema o tablas), y que otorga un pertenencia a un rol.

REVOKE — Remueve privilegios de acceso asignados

El comando REVOKE retira los privilegios concedidos anteriormente a partir de una o más funciones. La palabra clave PUBLIC se refiere al grupo implícito definido para todos los roles.

GRANT. Sintaxis

GRANT { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES | TRIGGER } [,...] | ALL [PRIVILEGES] }
ON [TABLE] *tablename* [, ...] TO { [GROUP] *rolename* | PUBLIC } [, ...] [WITH GRANT OPTION]

GRANT { { SELECT | INSERT | UPDATE | REFERENCES } (*column* [, ...]) [,...] | ALL [PRIVILEGES] (*column* [, ...]) }
ON [TABLE] *tablename* [, ...] TO { [GROUP] *rolename* | PUBLIC } [, ...] [WITH GRANT OPTION]

GRANT { { USAGE | SELECT | UPDATE } [,...] | ALL [PRIVILEGES] }
ON SEQUENCE *sequencename* [, ...] TO { [GROUP] *rolename* | PUBLIC } [, ...] [WITH GRANT OPTION]

GRANT { { CREATE | CONNECT | TEMPORARY | TEMP } [,...] | ALL [PRIVILEGES] }
ON DATABASE *dbname* [, ...] TO { [GROUP] *rolename* | PUBLIC } [, ...] [WITH GRANT OPTION]

GRANT { USAGE | ALL [PRIVILEGES] }
ON FOREIGN DATA WRAPPER *fdwname* [, ...] TO { [GROUP] *rolename* | PUBLIC } [, ...] [WITH GRANT OPTION]

GRANT { USAGE | ALL [PRIVILEGES] }
ON FOREIGN SERVER *servername* [, ...] TO { [GROUP] *rolename* | PUBLIC } [, ...] [WITH GRANT OPTION]

GRANT { EXECUTE | ALL [PRIVILEGES] }
ON FUNCTION *funcname* ([[*argmode*] [*argname*] *argtype* [, ...]]) [, ...] TO { [GROUP] *rolename* | PUBLIC } [, ...] [WITH GRANT OPTION]

GRANT { USAGE | ALL [PRIVILEGES] }
ON LANGUAGE *langname* [, ...] TO { [GROUP] *rolename* | PUBLIC } [, ...] [WITH GRANT OPTION]

GRANT { { CREATE | USAGE } [,...] | ALL [PRIVILEGES] }
ON SCHEMA *schemaname* [, ...] TO { [GROUP] *rolename* | PUBLIC } [, ...] [WITH GRANT OPTION]

GRANT { CREATE | ALL [PRIVILEGES] } ON TABLESPACE *tablespacename* [, ...] TO { [GROUP] *rolename* | PUBLIC } [, ...] [WITH GRANT OPTION]

GRANT *role* [, ...] TO *rolename* [, ...] [WITH ADMIN OPTION]

REVOKE. Sintaxis

REVOKE [GRANT OPTION FOR] { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES | TRIGGER } [,...] | ALL [PRIVILEGES] }
ON [TABLE] *tablename* [, ...] FROM { [GROUP] *rolename* | PUBLIC } [, ...] [CASCADE | RESTRICT]

REVOKE [GRANT OPTION FOR] { { SELECT | INSERT | UPDATE | REFERENCES } (*column* [, ...]) [,...] | ALL [PRIVILEGES] (*column* [, ...]) }
ON [TABLE] *tablename* [, ...] FROM { [GROUP] *rolename* | PUBLIC } [, ...] [CASCADE | RESTRICT]

REVOKE [GRANT OPTION FOR] { { USAGE | SELECT | UPDATE } [,...] | ALL [PRIVILEGES] }
ON SEQUENCE *sequencename* [, ...] FROM { [GROUP] *rolename* | PUBLIC } [, ...] [CASCADE | RESTRICT]

REVOKE [GRANT OPTION FOR] { { CREATE | CONNECT | TEMPORARY | TEMP } [,...] | ALL [PRIVILEGES] }
ON DATABASE *dbname* [, ...] FROM { [GROUP] *rolename* | PUBLIC } [, ...] [CASCADE | RESTRICT]

REVOKE [GRANT OPTION FOR] { USAGE | ALL [PRIVILEGES] }
ON FOREIGN DATA WRAPPER *fdwname* [, ...] FROM { [GROUP] *rolename* | PUBLIC } [, ...] [CASCADE | RESTRICT]

REVOKE [GRANT OPTION FOR] { USAGE | ALL [PRIVILEGES] }
ON FOREIGN SERVER *servername* [, ...] FROM { [GROUP] *rolename* | PUBLIC } [, ...] [CASCADE | RESTRICT]

REVOKE [GRANT OPTION FOR] { EXECUTE | ALL [PRIVILEGES] }
ON FUNCTION *funcname* ([[*argmode*] [*argname*] *argtype* [, ...]]) [, ...] FROM { [GROUP] *rolename* | PUBLIC } [, ...] [CASCADE | RESTRICT]

REVOKE [GRANT OPTION FOR] { USAGE | ALL [PRIVILEGES] }
ON LANGUAGE *langname* [, ...] FROM { [GROUP] *rolename* | PUBLIC } [, ...] [CASCADE | RESTRICT]

REVOKE [GRANT OPTION FOR] { { CREATE | USAGE } [,...] | ALL [PRIVILEGES] }
ON SCHEMA *schemaname* [, ...] FROM { [GROUP] *rolename* | PUBLIC } [, ...] [CASCADE | RESTRICT]

REVOKE [GRANT OPTION FOR] { CREATE | ALL [PRIVILEGES] }
ON TABLESPACE *tablespacename* [, ...] FROM { [GROUP] *rolename* | PUBLIC } [, ...] [CASCADE | RESTRICT]

REVOKE [ADMIN OPTION FOR] *role* [, ...] FROM *rolename* [, ...] [CASCADE | RESTRICT]

Práctica de Laboratorio Nro.12

Crear un rol de grupo denominado **rl_abm,** con tiempo de validez indefinido.

```
CREATE ROLE rl_abm
  VALID UNTIL 'infinity';
```

Crear un rol de login (*nombre_usuario*), que sea miembro del rol **rl_abm** y que solo pueda recibir herencia

```
CREATE ROLE nombre_usuario LOGIN
  VALID UNTIL 'infinity';

GRANT rl_abm TO nombre_usuario;
```

Conceder permiso de SELECT, UPDATE, INSERT y DELETE, sobre la tabla tb_persona al rol de grupo rl_abm.

```
GRANT SELECT, UPDATE, INSERT, DELETE
ON TABLE tb_persona
TO GROUP rl_abm;

GRANT ALL ON TABLE tb_persona_per_id_seq TO GROUP rl_abm;
```

Gerenciando una Base de Datos

Gerenciar una base de datos es una tarea, que dependiendo de la complejidad de las bases en cuestión y de las herramientas utilizadas, puede ser fácil o extremadamente compleja.

Existen diversas herramientas que realizan este enlace entre el usuario administrador y el PostgreSQL. Pero es importante conocer lo que cada comando de estas herramientas realiza en cada acción. Por ese motivo, veremos el psql**,** que es el cliente mas comúnmente utilizado para esta función.

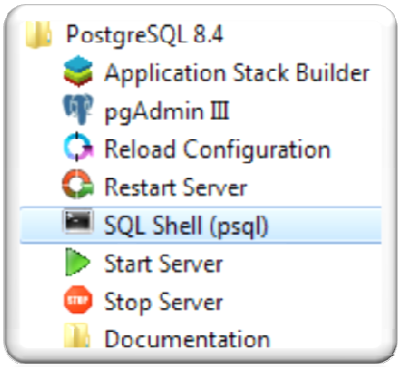
El **psql** es una consola interactiva que opera en modo texto, que podría considerarse como una desventaja con relación a otras opciones en lo que respecta a la practicidad. Pero es el modo más completo, pues permite que todo y cualquier comando SQL sea completamente manipulado y alterable por el usuario.

Las herramientas de administración gráficas utilizan una conexión similar al del psql para ejecutar sus acciones e interactuar con el usuario; ganando con la practicidad pero perdiendo un poco de libertad con la falta de acceso total al comando SQL que será ejecutado.

La decisión de que herramienta utilizar queda a criterio de la persona que administra el servidor. Muchas veces, mas de una herramienta es necesaria, pues en gran parte de las actividades es recomendada la herramienta gráfica; en cuanto que en otros momentos, como exportación de datos y otros específicos de seguridad, el psql puede ser una alternativa mas interesante.

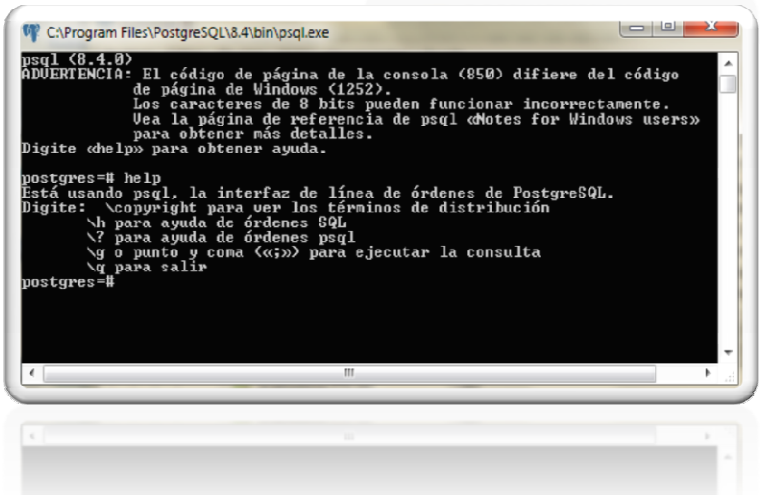
Acciones de Administración de BD

La mayoría de las acciones presentadas a continuación, pueden también ser ejecutadas utilizando la herramienta gráfica.



Como acceder a la consola

El psql es un programa-cliente que acompaña al PostgreSQL en su instalación; por tanto podrá acceder al mismo mediante la opción SQL Shell(psql)



Para obtener ayuda en la terminal psql, es posible utilizar los comandos:

\h	Lista de opciones de ayuda para la sintaxis SQL
\?	Lista de opciones de ayuda para los comandos psql

Gerenciando tablespaces

Los tablespaces son definiciones de locales de almacenamiento lógico de las informaciones del servidor. En otras palabras, se tratan de directorios existentes en el sistema operativo; posibilitando almacenar informaciones del servidor en locales distintos, lo que puede ocurrir por los mas diversos motivos: políticas de backup, utilización de más de un disco rígido, organización, etc.

Creando un tablespace

Para crear un tablespace utilice la sintaxis siguiente:

CREATE TABLESPACE *nombre* LOCATION *'localización'*;

En la sintaxis presentada, en necesario informar el *nombre* del tablespace que esta siendo creado e informar la *localización* o dirección lógica (directorio) donde los objetos que utilizaran este tablespace deben ser almacenados. Además, es necesario que el directorio en cuestión esté vacio para que el comando tenga éxito en su ejecución, y que el usuario tenga privilegios de utilización en el sistema operativo.

Visualizando tablespaces

Para visualizar los tablespaces disponibles y registrados en su servidor, utilice el comando siguiente: **\db**

Los tablespaces que empiecen con **pg_** son internos del PostgreSQL y no deben ser excluídos.

Alterando un tablespace

Son dos, las informaciones que pueden ser modificadas en un tablespace: nombre y propietario

Por cuestiones internas del PostgreSQL, no es posible alterar la dirección lógica; y si esta fuere la necesidad, el proceso debe ser de exclusión y re inclusión del mismo, con su respectiva política de backup y restore, en el caso de que exista información almacenada.

Sintaxis para alterar el nombre del tablespace: **ALTER TABLESPACE *nombre* RENAME *nuevo_nombre*;**

Sintaxis para alterar el nombre del propietario: **ALTER TABLESPACE *nombre* OWNER TO *nuevo_propietario*;**

Eliminando un tablespace

Para eliminar un tablespace utilice la sintaxis: **DROP TABLESPACE *nombre*;**

No es posible excluir un tablespace que no este vacio. Si contiene datos, será necesario, vaciar el directorio antes de ejecutar este comando.

Gerenciando una Base de Datos

El objeto más básico en un SGBD es la base de datos, pues cualquier otro objeto que venga a ser creado posteriormente, como tablas, vistas, procedimientos; todos estarán de alguna forma directamente relacionados con datos almacenados en algún local. Siendo "local", alguna base de datos.

La base de datos debe ser creada en forma optimizada para economizar recursos de hardware y para atender de forma rápida y eficiente, debe ser seguro para que las informaciones sean utilizadas solamente por personas autorizadas, debe ser consistente y no debe fallar.

Conocer todas esas necesidades es gerenciarlas, y es lo que marca la diferencia entre un desarrollador y un DBA. Ahora veremos los comandos básicos relacionados con el gerenciamiento, luego abordaremos las cuestiones relativas a seguridad y optimización.

Creando una base de datos

Para crear una base de datos se utiliza el comando **CREATE DATABASE** *nombre_bd*

Necesariamente se deberá poseer tal derecho de acceso (role CREATEDB) en e PostgreSQL. Otros argumentos opcionales pueden ser usados, al crear la base de datos :

Argumento	**Descripción**
OWNER *usuario*	Es posible infomar cual es el usuario responsible por la base de datos, el cual tendrá amplios poderes sobre la misma, pudiendo gerenciar sus accesos, como todas las estructuras. Generalmente el DBA inluye el nombre del solicitante del área de TI, en la base de datos, como responsable.
TEMPLATE *nombre*	En el caso de que se utilicen bases de datos que sigan un modelo padrón, es posible crear la base siguiendo un modelo de estructura especificado mediante esta variable.
ENCODING *valor*	Este argumento es responsable por indicar que conjunto de caracteres utilizará la base de datos. Es fundamental informar adecuadamente, para no perder informaciones como también poder visualizar en forma correcta valores como caracteres especiales y acentuaciones.
TABLESPACE *nombre*	Define a cual tablespace pertenece rá la base de datos a crear.
CONNECTION LIMIT *valor*	Por medio de este argumento, es posible limitar cuantas conexiones simultáneas podrán estar interactuando con la base de datos . Por padrón es utilizando el valor -1 que representa infinitas conexiones.

Visualizando bases de datos

Para visualizar las bases de datos registradas en su PostgreSQL, utilice el siguiente comando dentro del psql: `\l;`

Este comando le retornará una lista con todas las bases de datos definidas, además de informaciones relacionadas con la codificación utilizada y el propietario de cada una de ellas.

Activando una base de datos

Para ejecutar cualquier comando en una base de datos, primero es necesario conectarse . Para activar o conectarse a una base de datos utilice el comando:

`\c nombre_bd;`

Alterando una base de datos

Para alterar alguna de las propiedades de una base de datos creada, es necesario utilizar el comando ALTER DATABASE. Este comando permite realizar el mantenimiento de informaciones disponibilizadas en la creación, tales como propietario, numero máximo de conexiones, entre otros.

Sintaxis para alterar el nombre de la base de datos:

```
ALTER DATABASE nombre RENAME TO nuevo_nombre;
```

Sintaxis para alterar el propietario de la base de datos:

```
ALTER DATABASE nombre OWNER TO nuevo_propietario;
```

Sintaxis para alterar el limite de conexiones simultaneas

```
ALTER DATABASE nombre CONNECTION LIMIT nuevo_valor;
```

Eliminando una base de datos

Para excluir una base de datos, es necesario ser propietario del mismo, o tener derechos de acceso como administrador del PostgreSQL.

La sintaxis para eliminar la base de datos es la siguiente:

```
DROP DATABASE nombre;
```

Gerenciando Esquemas

Los esquemas separadores lógicos para organización de objetos creados en una base de datos. Una base de datos puede tener decenas de objetos como tablas, vistas e índices, entre diversas otras estructuras. Por medio de los esquemas, es posible organizarlos, atribuyendo cada objeto a un esquema, de acuerdo con su utilización.

Computacionalmente, el recurso de esquemas no ofrece ventajas en el sentido de optimización de recursos, pero si en lo que respecta a la organización y manipulación futura (gerenciamiento de accesos) por esquemas; posibilitando otorgar acceso por esquema a usuarios.

Creando esquemas

Para crear un esquema, utilice la sintaxis siguiente:

CREATE SCHEMA *nombre* [AUTHORIZATION *propietario*];

Visualizando un esquema

Para visualizar los esquemas existentes en un la base de datos activa utilice el comando

\dn

Para visualizar las estructuras pertenecientes a un esquema, utilice el comando

\dtvs *nombre_esquema.

Alterando un esquema

Las dos propiedades presentadas en la definición de un esquema pueden ser alteradas por medio de las sintaxis propias para cada caso:

Alteración del nombre del esquema:

```
ALTER SCHEMA nombre RENAME TO nuevo_nombre;
```

Alteración del nombre del propietario:

```
ALTER SCHEMA nombre OWNER TO nuevo_propietario;
```

Eliminando un esquema

Para excluir un esquema, utilice la sintaxis:

```
DROP SCHEMA nombre [CASCADE];
```

Al excluir un esquema, todos los objetos contenidos en el mismo también serán eliminados, en el caso de que se utilice la opción CASCADE. Caso contrario, un mensaje será mostrado informando que el esquema posee objetos (si fuere el caso) y la operación será abortada.

Gerenciando Tablas

Este tópico trata sobre los comandos básicos para creación y manipulación de estructuras de tablas en PostgreSQL. El gerenciamiento estructural de tablas es uno de los conjuntos de comandos mas completos y personalizables, teniendo una gran cantidad de parámetros que pueden ser analizados.

Creando tablas

Para crear una tabla utilizando el psql, utilice la sintaxis siguiente

```
CREATE TABLE
[nombre_esquema.]nombre_tabla
(
Columnas
) [opciones avanzadas]
```

Visualizando una tabla

Para visualizar la estructura de una tabla usando el psql, utilice el siguiente comando

```
\d nombre_tabla;
```

Alterando una tabla

Las principales acciones que contempla la alteración de tablas se describen a continuación:

Renombrar una tabla

```
ALTER TABLE nombre_tabla RENAME TO nuevo_nombre;
```

Renombrar una columna

```
ALTER TABLE nombre_tabla RENAME nombre_columna TO nuevo_nombre_columna;
```

Adicionar una columna

```
ALTER TABLE nombre_tabla ADD COLUMN nombre_columna tipo_dato
                                    [opciones adicionales];
```

Eliminar una columna

```
ALTER TABLE nombre_tabla DROP nombre_columna [CASCADE];
```

Importante: Al utilizar CASCADE, todos los objetos dependientes de esa tabla y sus registros serán excluidos, no pudiendo ser recuperados posteriormente.

Alterar el tipo de datos de una columna

```
ALTER TABLE nombre_tabla ALTER nombre_columna TYPE nuevo_tipo_dato expresion_sql;
```

La alteración del tipo de dato podrá ser acompañada de una expresión SQL, para indicar los procedimientos de conversión de datos.

Eliminando una tabla

Para excluir una tabla, utilice el comando `DROP TABLE *nombre_tabla* [CASCADE];`

Con esto, la estructura de la tabla será eliminada, como también todos sus registros. Dependiendo de los relacionamientos que posea, el procedimiento no podrá ser realizado. Para forzar la exclusión de todos los objetos relacionados, utilice el argumento CASCADE.

Vaciando una tabla

Para excluir todos los registros de una tabla, sin excluir su estructura, utilice el comando siguiente:

```
TRUNCATE TABLE nombre_tabla;
```

Gerenciando Indices

El índice es un recurso íntimamente ligado con la optimización de un servidor de base de datos, pues prepara algunas estructuras de organización de registros para hacer que las búsquedas y consultas sean más rápidas, precisas y eficientes.

Un índice es la selección de una o más columnas para almacenar sus datos ordenados en la base, de forma transparente para los usuarios.

La creación de índices debe ser analizada con cuidado, y si es posible, con comparación de medidas de tiempo entre consultas sin los índices y consultas con índices, para medir ganancias del servidor, así como su viabilidad.

Creando un índice

Sintaxis: CREATE INDEX *nombre* ON *tabla* [USING *algoritmo*] (*columna);*

Parámetro	Descripción
nombre	Define el nombre del índice
tabla	Representa la tabla que contiene la columna a ser utilizada como base de datos del índice en cuestión.
USING *algoritmo*	Parámetro Opcional. El PostgreSQL disponibiliza una serie de algoritmos para realizar la búsqueda de datos . En el caso de que quiera indicar un algoritmo específico , utilice esta opción. Algoritmos disponibles: HASH, GIST y GIN.
columna	Indica que columna de la tabla tendrá su versión también en índice.

Visualizando los índices de una tabla

Después de conectar a la base de datos, utilice el comando siguiente para visualizar los índices existentes `\di`

Renombrando un índice

Para renombrar un índice utilice el comando siguiente: `ALTER INDEX nombre RENAME TO nuevo_nombre;`

Eliminando un índice

Para eliminar un índice utilice el comando siguiente: `DROP INDEX nombre [tipo_eliminacion];`

Es posible informar el tipo de exclusión, con las opciones siguientes:

CASCADE: Excluye también los objetos dependientes.

RESTRICT : Aborta la operación en el caso de que haya objetos dependientes del índice en cuestión.

Gerenciando Vistas

Las vistas son recursos muy útiles cuando trabajamos con determinados datos contenidos en tablas separadas, o cuyos datos deben ser filtrados por algún criterio.

En bases de datos normalizadas, generalmente un registro completo de una entidad se encuentra distribuido en dos o mas tablas. Una vista puede ser útil, si es utilizada como un comando SELECT que une todas las tablas utilizando sus claves y retorna una única tabla como resultad final, o trayendo de la tabla únicamente registros que cumplan con algún criterio preestablecido.

Importante:

Excluir una vista no excluye los datos de la tabla a las cuales pertenecen los datos. Solamente la estructura de la vista será descartada.

Gerenciando Dominios

Los Dominios también son conocidos como máscaras de datos por su funcionalidad: verificar si el valor de un campo (columna) está acorde con el padrón preestablecido por el usuario.

Para utilizar un dominio, después de crearlo, se debe crear la tabla que contendrá la columna que será validada utilizando el nombre del dominio que fue creado. En el caso de que la tabla ya exista, podrá alterarse el tipo de dato de la columna en cuestión.

Creando un Dominio

Para crear un dominio, utilice la sintaxis siguiente:

```
CREATE DOMAIN nombre  tipo_dato [DEFAULT valor] [CONSTRAINT nombre_regla CHECK (formula)]
```

Ejemplo

```
CREATE DOMAIN codigo
  AS numeric
  DEFAULT 4
  NOT NULL
  CHECK (value >= 1000 and value <  2000);
```

Alterando un Dominio

Las alteraciones de dominios comprenden la inclusión de nuevas reglas de validación o la exclusión de una regla existente.

Adicionar una regla:

ALTER DOMAIN *nombre* ADD CONSTRAINT *nombre_regla* CHECK *(formula);*

Excluir una regla:

ALTER DOMAIN *nombre* DROP CONSTRAINT *nombre_regla;*

Eliminando un Dominio

Para eliminar un dominio, utilice la sintaxis siguiente:

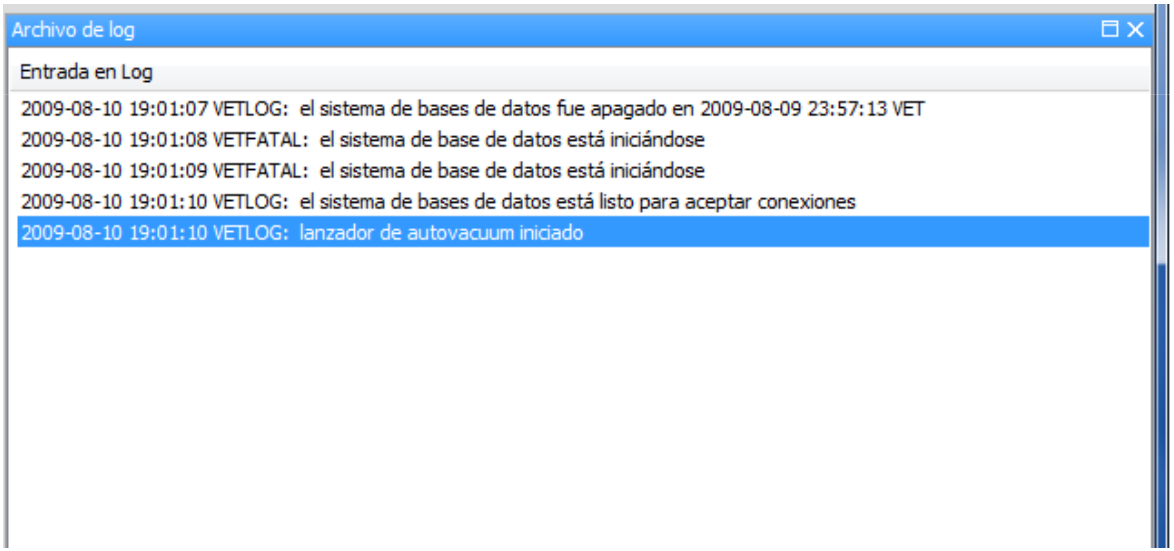
DROP DOMAIN *nombre* [CASCADE]*;*

Importante: Al utilizar CASCADE, todos los objetos dependientes serán excluidos también.

Log

El PostgreSQL puede, y por padrón, realiza el gerenciamiento de sus acciones dentro de uno o mas archivos de log en el sistema. Este tipo de procedimiento es vital para posibilitar futuros análisis de la base de datos, como también del servidor; en la resolución de problemas u otras situaciones que pudiesen ocurrir.

La localización de estos archivos puede ser obtenida mediante el archivo de configuración del PostgreSQL (postgresql.conf)



1- Crear una base de datos denominado db_mis_iniciales, ejemplo ***"db_grgl"***, que sea del dba postgres, que use como plantilla template0 y codificado en utf8.

2- Crear un grupo de usuario denominado gp_mis_iniciales, ejemplo "***gp_grgl_abm***", que pueda heredar permiso de los roles padres, que pueda crear roles y que no caduque

3- Crear un usuario denominado rl_mis_iniciales_01, ejemplo "***rl_grgl_01***" que herede privilegios de gp_grgl_abm y que no caduca

4- Crear la siguiente tabla de acuerdo a la estructura:

Consideraciones a cumplir en la definición de los mismos
- Los campos cli_id, fac_id, cli_documento y fac_numero deben aceptar solo valores únicos
- Los campos cli_nombre, cli_apellido y cli_direccion no deben aceptar valores nulos
- El campo fac_total debe aceptar valores mayores a 1000 y cli_sexo solo M y F
- El campo fac_cliente solo debe aceptar código existente en tb_clientes extendible para borrado y actualizaciones

tb_clientes

Campo	Tipo	Tamaño
cli_id	autoincremental	
cli_documento	carácter	12
cli_nombre	carácter	30
cli_apellido	carácter	30
cli_direccion	carácter	40
cli_telefono	carácter	20
cli_celular	carácter	20
cli_fax	carácter	20
cli_sexo	carácter	1
cli_cumple	fecha	8

tb_facturaciones

fac_id	autoincremental	
fac_numero	numerico	8
fac_fecha	fecha	
fac_cliente	entero	
fac_total	numerico	12,2

5- Cargar los siguientes datos en la tabla

tb_clientes
1, 345758, Pedro, Cantero, Av. Julio Rosas, 061-500, 0983-600600, 061-500000, M, 12-12-1973
2, 754125, Juan, Saldivar, Km 12 Acaray, 061-600, 0983-700700, 061-700000, M, 15-11-1944
3, 685420, Antonia, Benitez, Villa Apua, 061-900, 0983-800800, 061-900000, F, 28-02-1965

tb_facturaciones
1-1200, 17/10/2007, 2, 520.000
2-1201, 17//10/2007, 1 ,350.000
3-1202, 16/10/2007, 2, 480.000
4-1203, 14/11/2007, 2, 200.000
5-1204, 10/11/2007, 3, 500.000

6- Cambiar el privilegio del usuario ***rl_grgl_01,*** de solo consulta y borrado, a la tabla ***tb_clientes***

7- También que el usuario **rl_grgl_01** no puede crear ningún objeto dentro del esquema **public,** es buena practica retirar los permisos nativos de **public**

8- Eliminar el privilegio borrado del usuario ***rl_grgl_01,*** a la tabla ***tb_clientes***

9- Definir una vista de ***tb_clientes*** con los campos documento, nombre, apellido y teléfono denominada ***vw_clientes_00*** *ordenado por nombre*

10- Realizar una consulta de clientes ordenado por nombre de todos los campos

Guardar el scritp en practica01.sql

Preparar un proyecto de base de datos . Que cumplan con los requisitos a seguir

- Crear un usuario denominado **dbacurso_grupo_xx**, que herede permiso de los roles padres, que sea superusuario, pueda crear objetos de base de datos, crear roles y que pueda modificar el catalogo y que no caduque.-

- Crear una base de datos denominada **db_curso_grupo_xx**, que sea del dba **dbacurso_grupo_xx**, que use como plantilla **template0,** y codificado en **utf8**.-

- Crear un grupo de usuario denominado **gp_conecta_grupo_xx**, que solo pueda heredar permiso de los roles padres y que no caduque. Y que por defecto sea asignado a todos los usuarios-

- Crear un grupo de usuario denominado **gp_alta_grupo_xx**, que solo pueda heredar permiso de los roles padres y que no caduque.-

- Crear un grupo de usuario denominado **gp_baja_grupo_xx**, que solo pueda heredar permiso de los roles padres y que no caduque.-

- Crear un grupo de usuario denominado **gp_modifica_grupo_xx**, que solo pueda heredar permiso de los roles padres y que no caduque.-

- Crear un grupo de usuario denominada **gp_lista_grupo_xx** que solo pueda heredar permiso de los roles padres y que no caduque.-

- Crear un grupo de usuario denominada **gp_movimiento_grupo_xx** que solo pueda heredar permiso de los roles padres y que no caduque.-

- Crear los usuarios para cada grupo y asignarles los privilegios respectivos según el grupo **rl_user01_alta_grupo_xx**, **rl_user01_baja_grupo_xx**, **rl_user01_lista_grupo_xx, etc.**-

Estructura de las Tablas

tb_personas

Campo	Tipo	Tamaño		Restricciones
per_id	autoincremental	8		PKEY
per_documento	carácter	12	NO NULO	UK
per_nombre	carácter	20	NO NULO	
per_apellido	carácter	20	NO NULO	
per_direccion	carácter	40	NO NULO	
per_telefono	carácter	20	NO NULO	
per_email	carácter	50		
per_sexo	carácter	1	NO NULO	CHK(M,F)
per_fecha_nacimiento	fecha	8		
per_activo	Logico	1	DEFAULT true	

tb_clientes

Campo	Tipo	Tamaño		Restricciones
cli_id	autoincremental	8		PKEY
cli_idpersona	Entero	8		UK-FK
cli_activo	Logico	1	DEFAULT true	

tb_oficiales

Campo	Tipo	Tamaño		Restricciones
ofi_id	autoincremental	8		PKEY
ofi_idpersona	Entero	4		UK-FK
ofi_activo	Logico	1	DEFAULT true	

tb_tipos_movimientos

Campo	Tipo	Tamaño		Restricciones
tmv_id	autoincremental	8		PKEY
tmv_codigo	character	4		UK
tmv_descripcion	character	40		UK
tmv_debito_credito	character	1		CHK (D,C)
tmv_estado	boolean	1	DEFAULT true	

tb_caja_cuentas

Campo	Tipo	Tamaño		Restricciones
ccu_id	autoincremental	8		PKEY
ccu_descripcion	character	40	NO NULO	
ccu_estado	boolean	1	DEFAULT true	

tb_ahorros

Campo	Tipo	Tamaño		Restricciones
aho_id	autoincremental	8		PKEY
aho_numero	character	4	NO NULO	
aho_fecha_apertura	date	8	NO NULO	
aho_idcliente	Entero	4		FK
aho_idoficial	Entero	4		FK
aho_saldo	numeric	18,2	DEFAULT 0	

tb_ahorros_movimientos

Campo	Tipo	Tamaño		Restricciones
ahm_id	autoincremental	8		PKEY
ahm_idahorro	Entero	8		FK
ahm_idtipomovimiento	Entero	8		FK
ahm_fecha	date	8	NO NULO	
ahm_hora	time	8	DEFAULT current_time	
ahm_comprobante	character	6	NO NULO	UK
ahm_monto	numeric	18,2	NO NULO	DEFAULT 0
ahm_idcuentacaja	Entero	8		FK

1. Controlar que la edad de la persona sea mayor o igual a 18 años.
2. Que no exista dos personas con la misma numeración de documento.
3. Que no exista persona con nombre, apellido y dirección en blanco.
4. Que el genero de la persona sea identificada por [M] y [F] únicamente.
5. Que como minino el nombre y apellido de la persona tenga dos caracteres.
6. Que el grupo de usuario de **maestros, movimientos y reportes** tenga acceso a los datos para insertar, actualizar, borrar y consultar según corresponda.
7. Que no exista ahorrista sin cliente.
8. Que no exista movimientos anterior a fecha de apertura y posterior a fecha actual.
9. Que no se pueda sobregirar las cuentas.
10. Que el tipo de movimientos sea identificada por [D], [C] únicamente.
11. Un listado de ranking de saldo del cliente y de oficial. (Vista)
12. Un listada de extracto de cuenta de ahorro por rango fecha. (Vista)
13. Un listado de cliente que están de cumpleaños según la fecha del día. (Vista)
14. Preparar un script de datos para probar.

– Temario

1. Crear una base de datos denominada db_2practica con propietario postgres, codificado con UTF8, con límite de conexión a 30 usuarios.

2. Crear un grupo de usuario denominado gp_ventas_iniciales, ejemplo "***gp_ventas_pc***", que pueda heredar permiso de los roles padres y que caduque el 31/12/2012 24:00

3. Crear un usuario denominado rl_nombre_apellido, que herede privilegios de gp_ventas_iniciales y que caduca el 31/12/2012 23:59:59

4. Crear la siguiente tabla de acuerdo a la estructura:

tb_mercaderias

mer_id	serial		PK
mer_codigo	varchar(6)	NO NULO	UK
mer_descripcion	varchar(50)	NO NULO	
mer_precio_costo	numeric(12,2)	NO NULO	DEFAULT 0
mer_precio_venta	numeric(12,2)	NO NULO	DEFAULT 0
mer_existencia	numeric(7,2)	NO NULO	DEFAULT 0

tb_clientes

cli_id	serial		PK
cli_descripcion	varchar(50)	NO NULO	UK
cli_estado	boolean	NO NULO	DEFAULT true

tb_ventas

ven_id	serial		PK
ven_numero	numeric(10)	NO NULO	UK
ven_fecha	date	NO NULO	
ven_idcliente	integer		FK

tb_ventas_items

vei_id	serial		PK
vei_idventa	integer		FK
vei_idmercaderia	integer		FK
vei_cantidad	numeric(7,2)	NO NULO	
vei_precio_venta	numeric(12,2)	NO NULO	

Restricciones para las tablas:

El precio de venta de la mercadería debe ser mayor que el precio de costo
La cantidad vendida debe ser mayor a 0
No puede existir registros huérfanos, según los FK

5. Cargar 2 registros como mínimo en las tablas una ves establecida las condiciones.

6. Crear una vista denominada vw_ventas_items con los siguientes campos: vei_id, vei_idventa, vei_idmercaderia, vei_cantidad, vei_precio_venta, mer_descripcion y vei_total_items (Calculado precio venta x cantidad).

7. Crear una vista denominada vw_ventas con los siguientes campos: ven_id, ven_numero, ven_fecha, ven_idcliente, ven_total_ventas (calculado precio venta x cantidad desde tb_ventas_items).

8. Cambiar el privilegio del usuario rl_nombre_apellido, para consultar, borrar e insertar para las tablas tb_ventas y tb_ventas_items, vía grupo de usuario.