Bases de datos relacionales (Parte I)

Introducción a las bases de datos

Competencias

- Reconocer el rol de las bases de datos relacionales.
- Reconocer las características de una RDBMS.
- Aprender qué es el lenguaje estructurado de consultas (SQL).

Introducción

Hoy en día, estamos en constante generación de datos, en cada compra que hacemos, búsqueda en internet, cada vez que tomamos el metro, estamos generando datos. Si estos datos los agrupamos y ordenamos, podemos obtener información importante de ellos y posteriormente realizar toma de decisiones.

Una herramienta que tenemos para almacenar y gestionar estos datos, son las bases de datos y su importancia se basa en que a través de ellas podemos obtener información.

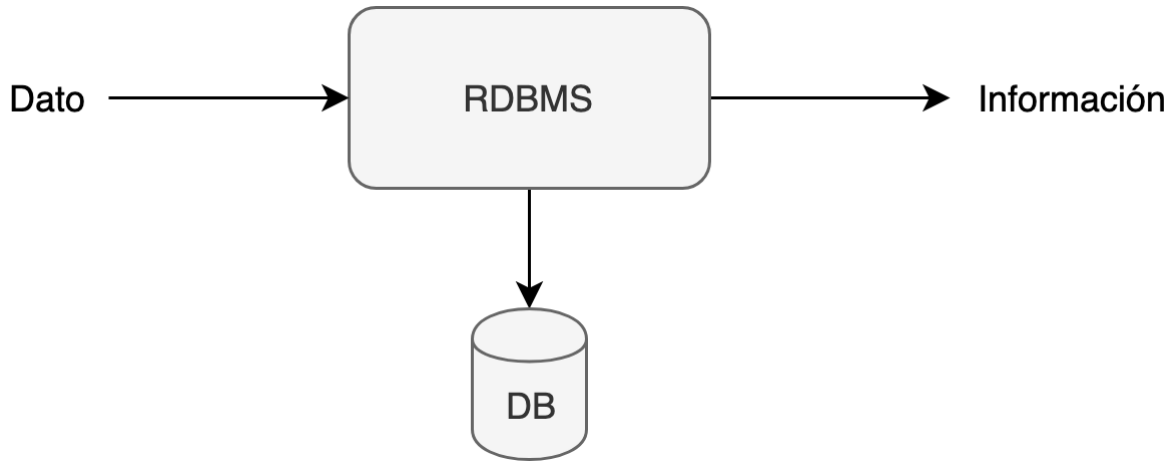

Imagen 1. Bases de datos.
Fuente: Desafío Latam.

Como podemos ver en la imagen 1 tiene un dato, que luego de pasar por un RDBMS (Sistema de gestión de base de datos relacionales) y se obtiene cierta información.
Este sinfín de datos ha tenido como aliado los sistemas que permiten organizar la información o gestores de bases de datos, de tal manera que podamos consultar y analizar nuestros datos de una forma rápida y efectiva. SQL es uno de los lenguajes que nos permite interactuar con los gestores de bases de datos y realizar consultas.

En este capítulo aprenderemos sobre lo que son las bases de datos y cómo interactuar con ellas.

Bases de datos y RDBMS

Las bases de datos se definen como un conjunto de información relacionada que se encuentra ordenada o estructurada. En el ámbito de la informática las bases de datos están categorizadas en bases de datos relacionales y no relacionales, cuando trabajamos bases de datos relacionales dentro de un sistema de gestión hablamos de una RDBMS que por sus siglas en inglés Relational Database Management System significa Sistema de Gestión de Bases de datos Relacionales, nos permiten definir y manipular una o más bases de datos.

Algunas de sus características son:

- El lenguaje de datos para realizar consultas a la base de datos es SQL
- Almacena datos en tablas
- Persiste los datos en forma de filas y columnas
- Recupera datos de una o más tablas
- Proporciona la claves primarias para identificar de forma única las filas y las claves foráneas para la relación entre tablas
- Crea índices para una recuperación de datos más rápida

¿Qué es SQL?

Structured Query Language (Lenguaje estructurado de consultas) es un lenguaje de programación creado para la definición y la manipulación de bases de datos relacionales. El beneficio de este lenguaje es que facilita la administración de datos almacenados.

SQL es un estándar mantenido por ANSI (American International Standard Institute) y por ISO (International Organization for Standardization), por lo que en general todas las bases de datos SQL comparten la misma base, aunque existen variaciones entre los distintos motores.

El lenguaje SQL está compuesto por cláusulas, operadores y funciones de agregado. Estos elementos se combinan en las instrucciones para manipular las bases de datos. En la imagen 2 puedes observar diferentes conceptos usados en una instrucción SQL.

Imagen 2. Estructura del Lenguaje.
Fuente: (Maman, 2011)

Un caso cotidiano

Imaginen un directorio telefónico con los datos de todas las personas de una ciudad, pero se requiere obtener el número de sólo una de ellas.

¿Qué es lo que hay que hacer para encontrarlo?

Buscar en orden alfabético hasta encontrar el nombre de la persona, lo que puede tomar bastante tiempo. Sin embargo, SQL entrega la facilidad de obtener rápidamente el número deseado.

¿Qué pasaría si se quisiera obtener los números de todas las mujeres de la ciudad? ¿O de toda la gente que tenga edad entre los 20 y 30 años? ¿O incluso ambas condiciones juntas?

SQL da la posibilidad de hacer este tipo de consultas, reduciendo el tiempo de ejecución en comparación a un humano realizando la misma tarea.

Tipos de bases de datos

Como mencionamos anteriormente, el objetivo de las bases de datos es ordenar y clasificar datos para transformarlos en información, pero de acuerdo a la naturaleza del negocio que estemos abordando y cómo se estructuran los datos que vayamos a almacenar, existen distintos tipos de bases de datos más afines a cada escenario.

Podemos clasificarlas de acuerdo a los siguientes aspectos:

- **Según la variabilidad de los datos:** Es decir, cómo se estructuran los datos y su grado de modificación en el tiempo. Existen dos tipos: estáticas y dinámicas.

- **Según el contenido:** Se refiere al tipo de datos que estamos almacenando. Por ejemplo: Bibliográficas, directorios, científicas, etc.

- **Según el modelo:** Tiene relación a cómo se estructura la forma en la que se guardan sus datos (descripciones), sus métodos de almacenamiento y recuperación. Algunas de las clasificaciones más conocidas son: jerárquicas, de red, transaccionales, relacionales, multidimensionales, documentales, etc.

PostgreSQL es uno de los proveedores de bases de datos relacionales más conocidos y soporta un modelo de bases relacionales, que consiste básicamente en una colección de tablas que contienen filas y columnas, donde cada fila representa un registro y cada columna representa un atributo del registro contenido en la tabla. Más adelante abordaremos este modelo en profundidad.

Instalación y configuración de PostgreSQL

Competencias

- Reconocer las ventajas de utilizar PostgreSQL.
- Instalar y configurar PostgreSQL en los distintos sistemas operativos.

Introducción

En este capítulo se definirá ¿Qué es PostgreSQL? y se analizarán sus ventajas y desventajas para una mejor toma de decisiones al momento de buscar soluciones a un problema determinado que incluya persistencia de datos. A su vez, aprenderás como se realiza la instalación de gestor de bases de datos en los diferentes sistemas operativos, para tener un entorno de desarrollo listo con el que puedas comenzar a trabajar con las bases de datos relacionales.

¿Qué es PostgreSQL?

De acuerdo a lo descrito por Vannahme, Juba y Volkov (2015), PostgreSQL es un sistema de gestión de base de datos relacionales de código abierto. Hace hincapié en la extensibilidad, creatividad y compatibilidad. Compite con los principales proveedores de bases de datos relacionales, como Oracle, MySQL, SQL Server y otros. Es utilizado por diferentes sectores, incluidos agencias gubernamentales y los sectores público y privado. Es un DBMS (Database Management System) multiplataforma, y se ejecuta en la mayoría de los sistemas operativos modernos, incluidos Windows, MAC y Linux.

Por otra parte, los autores Zea, Molina y Redrován (2017), ofrecen su perspectiva sobre PostgreSQL y lo definen como: "Un sistema de gestión de base de datos objeto-relacional, distribuido bajo licencia BSD y con código fuente disponible libremente. Es uno de los sistemas de gestión de base de datos más potente del mercado".

Ambos autores coinciden en que PostgreSQL es un sistema de base de datos relacional de alta disponibilidad, es decir, es estable, consistente y tolerante a fallos. Gracias a estas características, resulta un sistema robusto muy utilizado en la actualidad a nivel empresarial.

Ventajas y desventajas de PostgreSQL

Como se mencionó anteriormente, esta es una herramienta utilizada para el manejo de las bases de datos y a diferencia de otros sistemas, permite implementar funciones de distintos lenguajes como C, C++, Java, entre otros. Al ser código abierto, puede ser modificado a gusto del usuario y acomodarse su uso a cualquier persona.

A continuación presentaremos algunas de las principales ventajas y desventajas que debemos considerar al utilizar PostgreSQL:

Ventajas de PostgreSQL	Desventajas de PostgreSQL
Licencia gratuita	Alto consumo de recursos
Disponible para distintos sistemas operativos, como Windows, Linux y Unix, 32 y 64 bits.	Algunos comando o sentencias pueden ser poco intuitivas
Bajo mantenimiento	No tiene soporte en línea. Tiene foros oficiales y una gran comunidad que responde a las dudas.
Estabilidad, no se presentan caídas de bases de datos	
Alto rendimiento	
Gran capacidad de almacenamiento	
Gran escalabilidad, se ajusta al número de CPU y memoria disponible de forma óptima, soportando gran cantidad de peticiones simultáneas	

Tabla 1.Ventajas y desventajas de PostgreSQL.
Fuente: Desafío Latam.

Instalación de PostgreSQL

Un aspecto a considerar es que la instalación de PostgreSQL dependerá del sistema operativo que se esté utilizando:

Para efectos prácticos, se añade la explicación de los procesos de instalación para Windows, Linux y MacOS.

Windows

1. Ir a la página de PostgreSQL y seleccionar "Download the installer" como se puede observar en la próxima imagen.

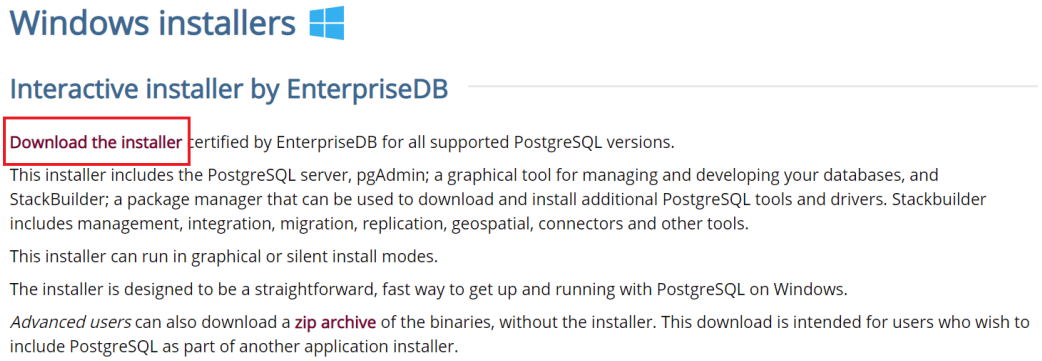

Imagen 3. Ingresar a la página oficial de PostgreSQL.
Fuente: Desafío Latam.

2. Seremos redirigidos a la página de descargas, se debe seleccionar la arquitectura del Sistema Operativo (32 o 64 bits), escoger la versión 10.xx y esperar a que termine la descarga, como muestra la imagen 4.

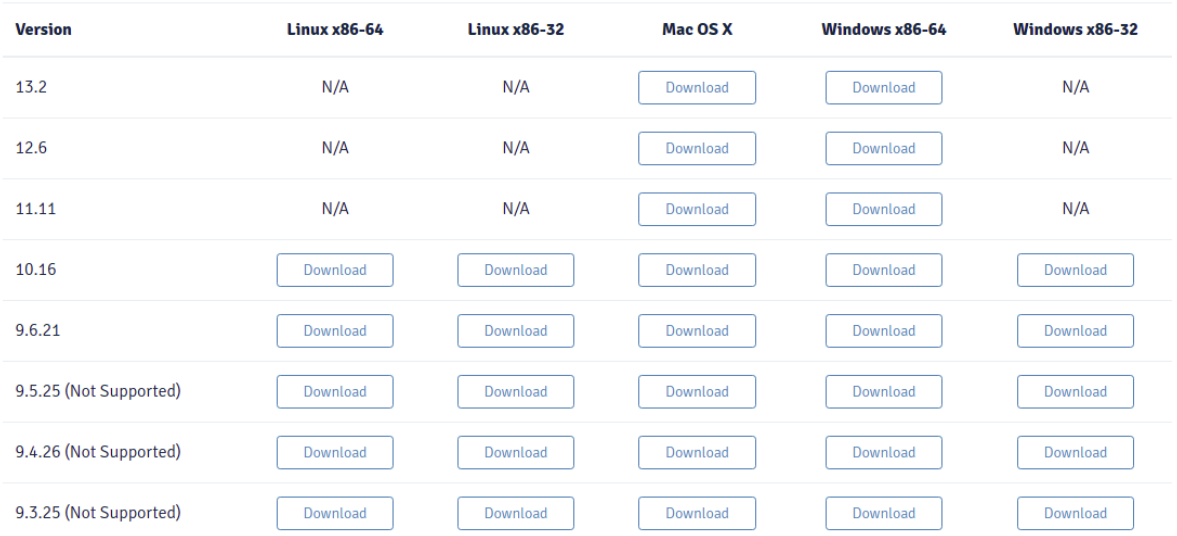

Imagen 4. Descargar el instalador.
Fuente: Desafío Latam.

3. Ejecutar el instalador y seguir las instrucciones, se debe ver algo parecido a la siguiente imagen:

Imagen 5. Ejecutar el instalador.
Fuente: Desafío Latam.

4. Una vez que termine de procesar la instalación, nos mostrará una pantalla de éxito como en la imagen 6, donde sólo se debe presionar finalizar.

Imagen 6. Finalizar la instalación.
Fuente: Desafío Latam.

5. Esto instalará en nuestro computador el software pgAdmin. Lo ejecutaremos para probar que todo esté instalado correctamente, al usar el buscador deberías ver algo como la siguiente imagen.

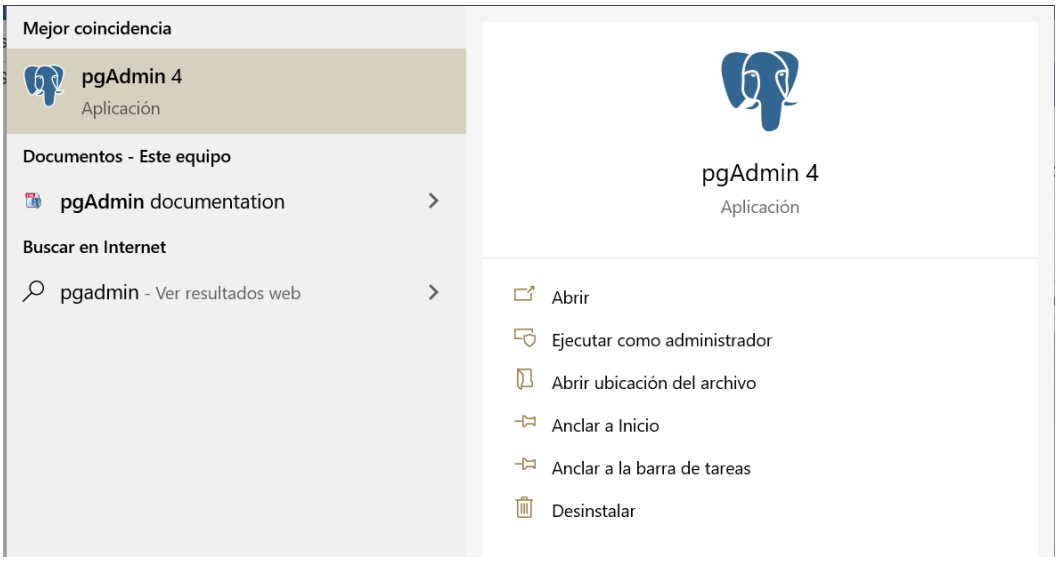

Imagen 7. PgAdmin.
Fuente: Desafío Latam.

6. Se iniciará el administrador en el navegador por defecto y nos pedirá la contraseña que hayamos definido para el administrador como en la imagen 8:

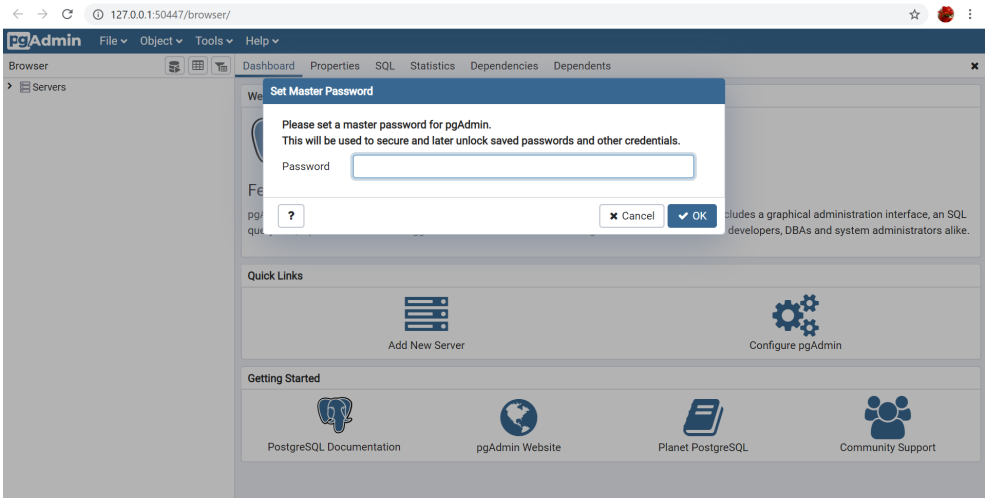

Imagen 8. pgAdmin browser.
Fuente: Desafío Latam.

7. Para verificar que todo se ha instalado correctamente, iremos al menú izquierdo en servers, lo que desplegará la información del servidor corriendo, te dejo la imagen 9 como referencia.

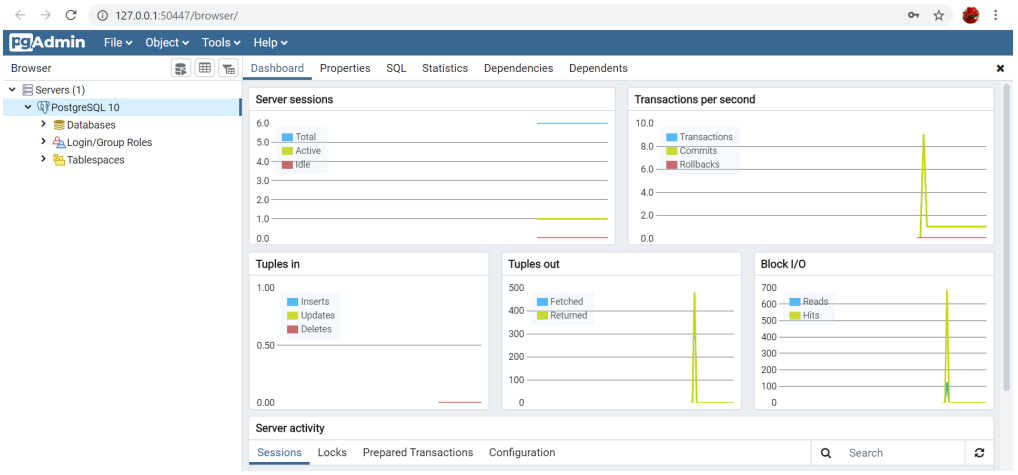

Imagen 9. Servers.
Fuente: Desafío Latam.

Linux

Lo primero que se debe realizar es agregar el repositorio con el siguiente comando:

```
wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc |
sudo apt-key add -
sudo sh -c 'echo "deb http://apt.postgresql.org/pub/repos/apt/
$(lsb_release -sc)-pgdg main" > /etc/apt/sources.list.d/PostgreSQL.list'
```

Luego se actualiza el repositorio, para luego instalar postgresql-10:

```
sudo apt update
sudo apt-get install postgresql-10
```

Para iniciar el motor, se debe usar lo siguiente:

```
sudo systemctl start postgresql.service
```

Para iniciar a la base de datos hay que cambiar de usuario y ejecutar el siguiente comando:

```
sudo su -l postgres
```

Finalmente:

```
psql
```

Configuración de PostgreSQL

Para poder utilizar la consola de PostgreSQL, se debe abrir el SQL Shell. Al abrirlo, preguntará por Server, Database, Port y Username, estos tendrán un valor por defecto entre corchetes, bastará con apretar Enter para continuar.

Luego, se debe ingresar la contraseña de PostgreSQL, esta será la misma que indicó en el instalador, luego de esto la consola quedará lista para trabajar y usar comandos SQL.

MacOS

Para instalar PostgreSQL en MacOS, la instalación de Postgres.app debe realizarse desde el siguiente link. Para instalarlo, una vez que finalice la descarga hay que ejecutarlo y moverlo a la carpeta de aplicaciones.

Luego en la consola, se debe escribir el siguiente comando opcional para luego reiniciar el equipo.

```
sudo mkdir -p /etc/paths.d && echo
/Applications/Postgres.app/Contents/Versions/latest/bin | sudo tee
/etc/paths.d/postgresapp
```

Ahora ya podemos utilizar PostgreSQL, en la imagen 10 se detalla en el menú de aplicaciones el icono de PostgreSQL:

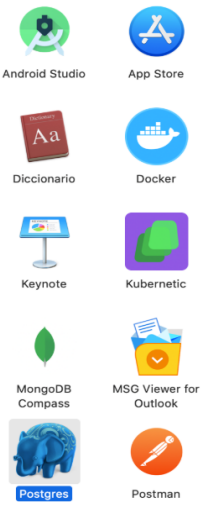

Imagen 10. Menú de aplicaciones.
Fuente: Desafío Latam.

Para ejecutar este gestor de base de datos simplemente debemos presionar doble click sobre el ícono de PostgreSQL. Una vez realizada esta acción, podremos ver en la imagen 11 el servicio corriendo y las bases de datos disponibles:

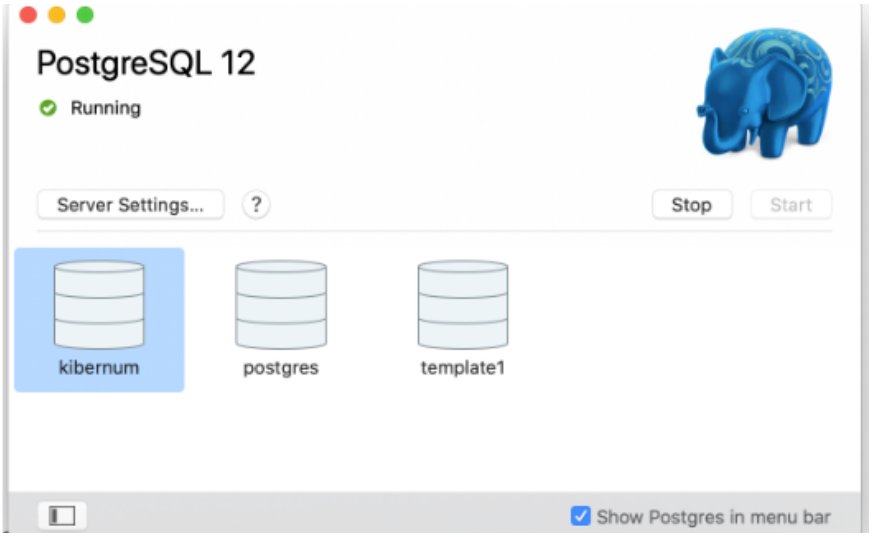

Imagen 11. Servicio PostgreSQL.
Fuente: Desafío Latam.

En caso contrario, se puede entrar al terminal y ejecutar el comando:

```
psql
```

También podemos realizar la instalación mediante el gestor de paquetes Homebrew, si no lo tienes instalado, ingresa al sitio oficial y sigue las instrucciones que allí aparecen.

Una vez instalado Homebrew, ejecutaremos el siguiente comando en la consola

```
brew install postgresql
```

Después de que Homebrew termine la instalación de PostgreSQL ejecutaremos el siguiente comando para iniciar postgre:

```
brew services start postgresql
```

Administrar usuarios y bases de datos

Competencias

- Crear usuarios y asignar permisos para administrar una base de datos.
- Crear bases de datos para almacenar información de forma organizada.

Introducción

Si piensas en el funcionamiento de una biblioteca, para que una persona acceda a un libro debe contar con algún tipo de credencial de socio. Para poder reservar ese libro y tenerlo en préstamo, no sólo requerirá de la credencial, sino también ser del grupo de usuarios que pueden pedir libros y además no contar con ninguna deuda.

Ese método de control es útil para resguardar la seguridad de los libros de la biblioteca y tener un control de quién accede a qué servicios.

De la misma manera, una de las primeras acciones que debemos realizar sobre nuestra base de datos es definir los permisos, es decir, qué usuario puede acceder a ella y qué acciones puede realizar.

A continuación veremos cómo administrar usuarios dentro de una base de datos, viendo qué permisos pueden tener al momento de enfrentarse a una base de datos.

Algunas características de PostgreSQL

En primer lugar, debemos dejar claro algunos puntos con respecto a la sintaxis de PostgreSQL:

- Las instrucciones deben ser terminadas con el símbolo " ; " (punto y coma).
- Hay que tener en cuenta que existen palabras reservadas para variables, ya que estos suelen ser comandos y pasa a ser fácil de confundir.

Algunos comandos son:

ALIAS	AND	AS
CREATE	CREATEDB	CREATEUSER
DATABASE	FROM	INNER
JOIN	LARGE	PASSWORD
WHERE	INSERT	UPDATE

Tabla 2. Palabras reservadas.
Fuente: Desafío Latam.

Existen varios más aparte de estos, los que puedes encontrar en la documentación de PostgreSQL.

- No es sensible a las letras mayúsculas/minúsculas, por lo que se pueden escribir los comandos de cualquiera de estas formas y PostgreSQL lo reconocerá de igual manera.

Administración de usuarios en PostgreSQL

A la hora de trabajar, más de una persona ocupa la información de una misma base de datos. PostgreSQL da la posibilidad de crear usuarios mediante la definición del rol de administrador de base de datos, quien tenga este rol será el encargado de asignar roles, los que van a definir el rango de acciones posibles para usuarios ordinarios; así como la creación/modificación/eliminación de tablas, revocación de roles/usuarios por nombrar algunas de las funcionalidades y solo los usuarios registrados van a poder acceder a la base de datos.

Crear usuarios

Primero, hay que entrar mediante la terminal a la base de datos con el comando:

```
psql
```

Para poder crear un usuario, se debe aplicar el siguiente comando:

```
CREATE USER nombre_usuario;
```

Para ver los usuarios existentes en la base de datos, se usa el comando:

```
\du
```

El cual, lista todos los usuarios existentes en la base de datos, con sus respectivos permisos.

Para poner algún límite a un usuario, se debe hacer lo siguiente:

```
CREATE USER nombre_usuario WITH comando_opcional;
```

Donde algunos de los comandos posibles son:

- `PASSWORD`: Asigna una contraseña al usuario creado.
- `ENCRYPTED PASSWORD`: Le asigna una contraseña encriptada al usuario creado.
- `UNENCRYPTED PASSWORD`: Le asigna una contraseña no encriptada al usuario creado.
- `VALID UNTIL`: La cuenta expirará en la fecha indicada.
- `CREATEDB`: Permite al usuario crear bases de datos.
- `NOCREATEDB`: No permite al usuario crear bases de datos.
- `SUPERUSER`: Puede crear otros usuarios (volveremos a ver esto más adelante).
- `NOSUPERUSER`: No puede crear otros usuarios.

Estos comando pueden ser usados simultáneamente como por ejemplo:

```
CREATE USER Bastian WITH PASSWORD 'contraseña_secreta' VALID UNTIL
'2019-12-31';
```

En este caso, el usuario Bastian tendrá de contraseña 'contraseña_secreta', su cuenta expirará el 31 de Diciembre del 2019 y no tiene permiso para crear bases de datos (en caso de que no se especifique, se considera NOCREATEDB por defecto)

Eliminar usuarios

Para eliminar usuarios sin tener que esperar la fecha de expiración, se puede utilizar el comando DROP USER:

```
DROP USER nombre_usuario;
```

Habrán casos en que se necesite saber cuales son todos los usuarios que tienen acceso a la base de datos, para ello, se hace lo siguiente:

```
SELECT nombre_usuario FROM pg_user;
```

A veces es necesario crear otro usuario que tenga todos los permisos, al igual que el administrador de la base de datos, lo que entonces se busca es crear un superusuario o superuser y para esto se debe aplicar lo siguiente:

```
CREATE USER nombre_usuario WITH SUPERUSER;
```

Permisos para los usuarios

Una vez creados los usuarios, se pueden cambiar los permisos con los siguientes comandos:

- Para dar acceso a todos los privilegios de una base de datos:

```
GRANT ALL PRIVILEGES ON DATABASE database_name TO nombre_usuario;
```

- Para dar permiso de creación de una base de datos se usa:

```
ALTER USER nombre_usuario CREATEDB;
```

- Para el caso de querer transformar al usuario en superuser:

```
ALTER USER myuser WITH SUPERUSER;
```

- Remover el superusuario:

```
ALTER USER username WITH NOSUPERUSER;
```

Cambiar de usuario

Ahora que se han creado los usuarios, es normal preguntarse cómo se puede acceder a la base de datos con alguno de ellos, entonces ahora corresponde aprender cómo cambiar de usuario.

Primeramente, hay que salir de PostgreSQL con el comando `\q` e ingresar usando lo siguiente:

```
psql -U nombre_usuario
```

Pero al hacer esto, la consola arrojará un error indicando que la base de datos no existe:

```
psql: FATAL:  database "nombre_usuario" does not exist
```

Ya que toma por defecto al usuario que estamos ingresando nombre_usuario como el nombre de la base de datos a la que nos queremos conectar.

Para solucionar esto, escribiremos:

```
psql -U nombre_usuario -d nombre_base_de_datos
```

Ahora, para revisar que hemos ingresado de manera correcta, usaremos:

```
SELECT * FROM user;
```

y verán que aparece el nombre de usuario con el que han ingresado a la base de datos:

```
user
--------
nombre_usuario
(1 row)
```

Crear una base de datos

Dentro de un motor de base de datos PostgreSQL, podemos tener varias bases de datos. Para crear una base de datos, usaremos el siguiente comando, el que nos permitirá alojar todas las tablas que se generen posteriormente:

```
CREATE DATABASE nombre_base_de_datos;
```

Pero podemos ver que al crearla no estamos en dicha base de datos, ya que a la izquierda donde escribiremos los comandos nos aparece el nombre de la base de datos a la que estamos conectados.

Para cambiarnos de base de datos, usaremos el comando `\c` de la siguiente forma:

```
\c nombre_base_de_datos
```

Eliminar una base de datos

Tal y como vimos, podemos crear bases de datos pero también eliminarlas, hay que ser consciente y tener cuidado, ya que si los datos de la base de datos no están respaldados, se perderá toda la información.

Para eliminar la base de datos utilizamos el comando:

```
DROP DATABASE nombre_base_de_datos;
```

Operaciones comunes a nivel de consola

En la consola de PostgreSQL existen una serie de comandos que nos van a permitir conectarnos a bases de datos, listar tablas y también listar usuarios. A continuación se presenta una tabla con sus comando más utilizados

Comando	Acción
\c nombre_base	Conectarse a una base de datos específica
\l	Listar todas las bases de datos existentes
\du	Listar todos los usuarios en el motor
\d	Listar todas las relaciones (o tablas) existentes en una base de datos específica
\dt	Lista todas las tablas de una base de datos
\q	Salir de la consola de PostgreSQL
\h	Mostrar la lista de comandos

Tabla 3. Comandos básicos de PostgreSQL.
Fuente: Desafío Latam.

Elementos de una base de datos

Competencias

- Interpretar el concepto de tabla en una base de datos.
- Reconocer cómo se relacionan las tablas mediante claves primarias y foráneas.
- Clasificar los tipos de datos que se pueden utilizar en una base de datos.

Introducción

En este capítulo aprenderás la diferencia entre una base de datos relacional y una base de datos no relacional, para que conozcas el contraste que hay entre estos dos mundos alternos y refuerces tu capacidad de análisis en la toma de decisiones, al crear un software que necesite persistencia de datos. Continuando con el ejemplo del capítulo anterior, empezarás a crear las relaciones entre las tablas por medio de claves primarias y claves foráneas, las cuales sirven para definir las dependencias de las entidades.

Finalmente, aprenderás los diferentes tipos de datos que se almacenan en columnas dentro de las tablas, logrando restringir el ingreso de datos incoherentes a una entidad de contexto definido dentro de las bases de datos.

Bases de datos relacionales versus bases de datos no relacionales

Existen dos tipos de bases de datos: las Relacionales y las No Relacionales.

- Las bases de datos relacionales son aquellas compuestas por varias tablas donde se almacena la información, y posteriormente se relacionan entre sí.

- Las bases de datos No Relacionales son aquellas que siguen esquemas más flexibles de organización, donde no necesariamente todas las entradas tienen la misma estructura.

Un aspecto relevante a tomar en cuenta, es que para implementar bases de datos relacionales o no relaciones, es necesario tener conocimiento previo sobre qué es lo que vamos a almacenar, teniendo en cuenta los siguientes puntos clave:

Relacional	No Relacional
Cuando el volumen de datos no crece o lo hace gradualmente.	Cuando el volumen de datos crece muy rápidamente.
Cuando las necesidades de proceso se pueden asumir en un solo servidor o en N servidores definidos previamente.	Cuando las necesidades del proceso no se pueden prever.
Cuando no existen peaks de uso o son esporádicos.	Cuando existen peaks de uso en múltiples ocasiones.
Cuando requerimos mantener la integridad referencial.	Cuando la información no requiere mantener relación entre los registros.
Estructura de datos mayormente estática.	Estructura de datos variable.

Tabla 4. Relacional v/s No Relacional.
Fuente: Desafío Latam.

Como podemos observar, existen modelos más aptos dependiendo de la naturaleza del negocio que estamos abordando. En esta unidad sólo veremos cómo operar con bases de datos de tipo relacional.

Ejercicio guiado: Directorio telefónico

Crear un registro telefónico donde alojaremos el nombre, apellido, número telefónico, dirección y edad de una serie de individuos. Resulta que el registro en sí será la tabla Directorio_telefonico, donde tendremos todos los campos mencionados anteriormente.

Para empezar a crear nuestra base de datos utilizaremos tablas, donde alojaremos esta información.

Tablas

Una base de datos se compone de múltiples tablas. Cada una de éstas presentarán dos dimensiones:

- Filas, que representan a los registros en la tabla.
- Columnas, que van a representar los atributos ingresados en cada registro, definiendo el tipo de dato a ingresar.

Claves primarias y foráneas

De manera adicional a las filas y columnas, las tablas también cuentan con claves primarias y foráneas. Estas buscan generar identificadores para cada registro de estas tablas mediante algún valor específico de una columna o atributo.

Clave primaria (primary key)

Cuando hacemos referencia a una columna dentro de su tabla de origen, hablaremos de una clave primaria. Esta clave siempre será de carácter único.

Clave foránea (foreign key)

Cuando hacemos referencia a una columna identificadora en otra tabla a la cual hacemos referencia, hablamos de una clave foránea.

Veamos esto con el ejemplo

Supongamos que en base a nuestra tabla `Directorio_telefonico`, deseamos incorporar información de otra tabla llamada `Agenda` que tiene las columnas `nick` y `numero_telefonico`. Ante esta situación, vamos a identificar que la columna `numero_telefonico` en la tabla `Directorio_telefonico` será la clave primaria y la columna `numero_telefonico` en la tabla Agenda corresponderá a la clave foránea. Este comportamiento es posible dado que el número registrado será congruente en ambas tablas, en la siguiente imagen te muestro una referencia de esto.
Nota: Se recomienda siempre crear los nombres de los campos (columnas) sin tildes ni caracteres que contengan símbolos como "ñ".

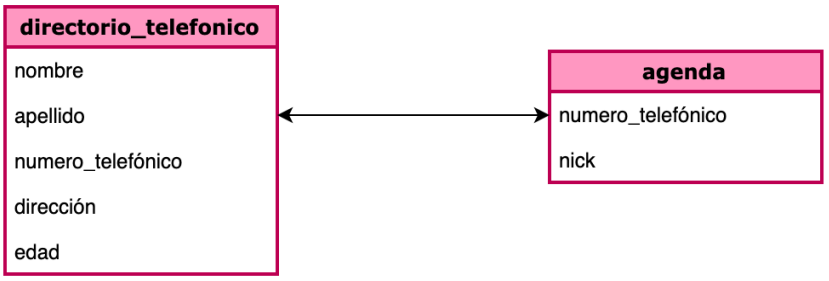

Imagen 12. Modelo relacional.
Fuente: Desafío Latam.

Tipos de datos

Una de las principales características del trabajo con bases de datos, es la necesidad de declarar los distintos tipos de datos existentes en cada campo a completar. Estos aplican restricciones sobre lo que pueden ingresar a los registros. No podemos ingresar caracteres cuando piden un número, ni sobrepasar el límite de caracteres posibles.

Los tipos de datos más comunes son:

- `INT`: Números enteros de 4 bytes que pueden tomar valor desde -2147483648 hasta +2147483647.
- `SMALLINT`: Números enteros de 2 bytes que pueden tomar valor desde -32768 hasta +32767.
- `BIGINT`: Números enteros de 8 bytes que pueden tomar valor desde -9223372036854775808 hasta 9223372036854775807.

- `FLOAT`: Números decimales de 32 bit de precisión.
- `DOUBLE`: Números decimales de 64 bit de precisión con hasta 15 dígitos decimales.
- `CHAR`: Cadena de hasta 255 caracteres de longitud fija.
- `VARCHAR`: Cadena de hasta 65.535 caracteres de longitud variable. A diferencia de CHAR, si no se ocupa toda la memoria, esta queda libre. CHAR ocupará toda la memoria solicitada.
  Para CHAR Y VARCHAR se puede definir un límite máximo de caracteres como CHAR(N) Y VARCHAR(N), si se trata de almacenar un valor que sobrepase el límite, PostgreSQL entregará un error.
- `DATE`: Almacena fecha en formato aaaa-mm-dd. <año>-<mes>-<dia>.
- `TIME`: Almacena el tiempo en horario desde 00:00:00 hasta 24:00:00.
- `TIMESTAMP`: Almacena fecha y hora juntos: yyyy-mm-dd hh:mm:ss.
- `BOOLEAN`: Tiene 3 valores posibles: Verdadero, Falso o NULL.
  Estos pueden representarse para el caso:
    - Verdadero: 1, yes, t o true.
    - Falso: 0, no, false o f.

Instrucciones de creación, inserción, actualización y eliminación de datos

Competencias

- Construir bases de datos para el almacenamiento persistente de información.
- Crear tablas con sus atributos y tipos de datos correspondientes para el direccionamiento de datos.
- Crear claves primarias y foráneas para el enlace de referencias entre tablas.
- Importar fichero con extensión .sql

Introducción

Hasta el momento hemos instalado, configurado y aprendido ciertos comandos de PostgreSQL, y ya es momento que empecemos a crear nuestra primera base de datos con las primeras tablas.

El proceso de vida de una tabla en una base de datos parte con el proceso de Crear, Insertar, Actualizar y Eliminar, que responde a las operaciones elementales que podemos realizar en una tabla. No obstante, las bases de datos no están compuestas de una única tabla sino de varias que se interconectan y referencian para evitar redundancia de datos, almacenamientos en cascada de información y normalizaciones, en este capítulo practicaremos con un ejemplo sobre los directorios de teléfonos y cómo se relacionan con las agendas.

Creación de tablas

La primera tarea que debemos realizar es la creación de la tabla que permitirá almacenar la información en filas y columnas.

Para crear una tabla dentro de nuestro motor, debemos utilizar el comando `CREATE TABLE` acompañado del nombre de la tabla y declarar los atributos con su respectivo tipo de dato, ingresandolos entre paréntesis. La forma canónica de creación es la siguiente:

```
CREATE TABLE nombre_tabla(
    columna1 tipo_de_dato1,
    columna2 tipo_de_dato2,
    columna3 tipo_de_dato3,
    PRIMARY KEY (columnaN)
);
```

Luego de definir los nombres de las columnas y sus tipos de datos separados con "," definimos también con las palabras reservadas "PRIMARY KEY" cuál de estas columnas tomará el rol de clave primaria.

Volviendo a nuestro ejemplo, con el siguiente código vamos a crear la tabla Directorio_telefonico con los campos `nombre`, `apellido`, `numero_telefonico`, `direccion` y `edad`.

```
CREATE TABLE Directorio_telefonico(nombre VARCHAR(25), apellido
VARCHAR(25), numero_telefonico VARCHAR(8), direccion VARCHAR(255), edad
INT, PRIMARY KEY (numero_telefonico) );
```

Comentarios

Las líneas precedidas por `--` representan comentarios específicos sobre cada línea. El siguiente código es una réplica de la creación de la tabla "Directorio_telefonico" realizada en el punto anterior pero con comentarios agregados.

```
    -- Creamos una tabla con el nombre directorio_telefonico
    CREATE TABLE Directorio_telefonico(
    -- Definimos el campo nombre con el tipo de dato cadena con un largo
de 25 caracteres.
    nombre VARCHAR(25),
    -- Definimos el campo apellido con el tipo de dato cadena con un
largo de 25 caracteres.
    apellido VARCHAR(25),
    -- Definimos el campo numero_telefonico con el tipo de dato cadena
con un largo de 8 caracteres.
    numero_telefonico VARCHAR(8),
    -- Definimos el campo dirección con el tipo de dato cadena con un
largo de 255 caracteres.
    direccion VARCHAR(255),
    -- Definimos el campo edad con el tipo de dato entero
    edad INT,
    -- Definimos que el campo numero_telefonico representará la clave
primaria de la tabla.
    PRIMARY KEY (numero_telefonico)
    );
```

Para poder ver la estructura que tiene nuestra tabla, podemos usar el comando:

```
SELECT * FROM nombre_tabla;
```

Donde mostrará la siguiente estructura:

nombre	apellido	numero_telefonico	direccion	edad

Tabla 5. Columnas de la tabla nombre_tabla.
Fuente: Desafío Latam.

Creando una tabla con clave foráneas

Posterior a la creación de la primera tabla, definir los componentes de la segunda tabla `Agenda` con los campo `numero_telefonico y nick`, asignando una clave foránea proveniente de la tabla `Directorio_telefonico`.

Profundicemos sobre la última instrucción. La forma canónica de implementar una clave foránea responde a los siguientes componentes:

```
    -- Creamos una tabla con el nombre agenda
   CREATE TABLE Agenda(
    -- Definimos el campo nick con el tipo de dato cadena con un largo
de 25 caracteres
    nick VARCHAR(25),
    -- Definimos el campo numero_telefonico con el tipo de dato cadena
con un largo de 8 caracteres.
    numero_telefonico VARCHAR(8),
    -- Vinculamos una clave foránea entre nuestra columna
numero_telefonico y su símil en la tabla directorio telefónico
    FOREIGN KEY (numero_telefonico) REFERENCES
Directorio_telefonico(numero_telefonico)
    );
```

De manera similar, nuestra tabla Agenda se ve de la siguiente manera, sin registros de momento.

nick	numero_telefonico

Tabla 6. Columnas de la tabla Agenda.
Fuente: Desafío Latam.

Inserción de datos en una tabla

Para que una base de datos sea útil, debe contener datos, para insertar datos a una tabla existe el comando `INSERT,` que indica la tabla a la que se agregaron datos, los tipos de datos y los valores que queremos ingresar en el registro.

Esto es un proceso ordenado y debemos especificar a qué fila pertenecen los datos que estamos ingresando. Es necesario usar un orden claro, ya que si no la tabla pierde su

estructura, haciendo imposible obtener la información buscada y la base de datos perdería su real utilidad.

La forma canónica de la instrucción INSERT es la siguiente:

```
INSERT INTO nombre_tabla (columna1, columna2, columna3) VALUES (valor1, valor2, valor3);
```

Por ejemplo, ingresamos un registro a las **tablas Directorio_telefonico y Agenda:**

```
    -- Definimos qué tabla vamos a insertar datos
    INSERT INTO Directorio_telefonico
    -- Explicitamos cuáles son las columnas a insertar
    (nombre, apellido, numero_telefonico, direccion, edad)
    -- Con la instrucción VALUES logramos asociada cada columna con un valor específico
    VALUES ('Juan', 'Perez', '12345678' , 'Villa Pajaritos', 21);
```

Ingresemos otros registros más:

```
    INSERT INTO Directorio_telefonico
    (nombre, apellido, numero_telefonico, direccion, edad) VALUES
    ('Fabian', 'Salas', '32846352', 'Playa Ancha',    21);

    INSERT INTO Directorio_telefonico
    (nombre, apellido, numero_telefonico, direccion, edad) VALUES
    ('John', 'Rodriguez',     '23764362', 'Constitución',   21);

    INSERT INTO Directorio_telefonico
    (nombre, apellido, numero_telefonico, direccion, edad) VALUES
    ('Braulio', 'Fuentes', '23781363', 'Rancagua',    19);
```

Hasta el momento tenemos 4 registros, para poder verlos se utiliza el siguiente comando, el que se explicará más adelante:

```
SELECT * FROM nombre_tabla;
```

nombre	apellido	numero_telefonico	direccion	edad
Juan	Perez	12345678	Villa Pajaritos	21
Fabian	Salas	32846352	Playa Ancha	21
John	Rodriguez	23764362	Constitución	21
Braulio	Fuentes	23781363	Rancagua	19

Tabla 7. Registros de la tabla Directorio_telefonico.
Fuente: Desafío Latam.

¿Qué ocurre si tratamos de ingresar el siguiente registro?:

```
INSERT INTO Directorio_telefonico (nombre, apellido, numero_telefonico,
direccion, edad) VALUES ('Marta', 'Fuentes', '23781363', 'Santiago',
24);
```

Aparece el siguiente error:

```
ERROR:  duplicate key value violates unique constraint
"directorio_telefonico_pkey"
DETAIL:  Key (numero_telefonico)=(23781363) already exists.
```

Donde especifica que se está intentado agregar un valor duplicado, violando la restricción de que una clave primaria debe ser de carácter único.

Ahora, ingresemos un par de registros en en la **tabla Agenda** de la siguiente manera:

```
-- Realicemos el mismo procedimiento en la tabla Agenda
INSERT INTO Agenda (nick, numero_telefonico) VALUES ('Juanito',
'12345678');
```

Recordemos que hicimos una relación entre la tabla Directorio_telefonico y Agenda, indicando que la clave foránea de la tabla Agenda es el numero_telefonico de la tabla Directorio_telefonico. Entonces, ¿Qué ocurrirá si tratamos de ingresar otro nick con el mismo número?

```
INSERT INTO Agenda (nick, numero_telefonico) VALUES ('Juancho',
'12345678');
```

No hay problema, ya que el número telefónico existe en la tabla Directorio_telefonico. Ahora intentemos agregar un número que no existe en la tabla de Directorio_telefonico:

```
INSERT INTO Agenda (nick, numero_telefonico) VALUES ('Fabi',
'32846351');
```

y nos aparece el siguiente error:

```
ERROR:  insert or update on table "agenda" violates foreign key
constraint "agenda_numero_telefonico_fkey"
DETAIL:  Key (numero_telefonico)=(32846351) is not present in table
"directorio_telefonico".
```

Especificando que se viola la restricción de la llave foránea, ya que no existe en la tabla de `Directorio_telefonico`.

Ingresemos un par de datos para tener los siguientes registros en las tablas:

nombre	apellido	numero_telefonico	direccion	edad
Juan	Perez	12345678	Villa Pajaritos	21
Fabian	Salas	32846352	Playa Ancha	21
John	Rodriguez	23764362	Constitucion	21
Braulio	Fuentes	23781363	Rancagua	19
Pedro	Arriagada	38472940	Valdivia	23
Matias	Valenzuela	38473623	Nogales	22
Cristobal	Missana	43423244	Con Con	20
Farid	Zalaquett	32876523	La Florida	20
Daniel	Hebel	43683283	San Bernardo	20
Javiera	Arce	94367238	Quilpue	20

Tabla 8. Inserción de registros.
Fuente: Desafío Latam.

Agregamos registros en la tabla de `Agenda`, quedando los siguientes valores.

nick	numero_telefonico
Juanito	12345678
Juancho	12345678
Peter	38472940
Mati	38473623
Cris	43423244
Javi	94367238
Farid	32876523
Dani	43683283

Tabla 9. Registros de la tabla Agenda.
Fuente: Desafío Latam.

Ejercicio propuesto (1)

La empresa Ovalle Electronics SPA necesita una base de datos para almacenar sus productos y el historial de ventas del día a día. Crea una base de datos llamada OvalleElectronicsSPA con las siguientes tablas:

nombre	ID	fecha_creacion	proveedor	categoría
TV RV-25	2468	2020-08-16	Dende SPA	televisores

Tabla 10. Tabla Productos.
Fuente: Desafío Latam.

fecha	ID_Producto	cliente	metodo_pago	referencia
2021-02-01	2468	Bruce Lee	efectivo	34414
2020-11-15	2468	Chuck Norris	débito	43224

Tabla 11. Tabla Ventas.
Fuente: Desafío Latam.

Actualización de registros

A veces nos vemos en la necesidad de actualizar los registros ya existentes, sin embargo, es riesgoso borrarlos e ingresarlos nuevamente (más adelante se profundizará en esto), por lo que si queremos actualizar los datos de un registro, debemos usar el comando `UPDATE` de la siguiente forma:

```
UPDATE nombre_tabla SET columna1=valor_nuevo WHERE condicion;
```

Juan se cambió de casa a Villa Los Leones, por lo que tenemos que actualizar la tabla `Directorio_telefonico`:

```
UPDATE Directorio_telefonico
SET direccion = 'Villa Los Leones'
WHERE nombre = 'Juan';
```

Quedando la tabla de la siguiente forma:

nombre	apellido	numero_telefonico	direccion	edad
Fabian	Salas	32846352	Playa Ancha	21
John	Rodriguez	23764362	Constitucion	21
Braulio	Fuentes	23781363	Rancagua	19
Pedro	Arriagada	38472940	Valdivia	23
Matias	Valenzuela	38473623	Nogales	22
Cristobal	Missana	43423244	Con Con	20
Farid	Zalaquett	32876523	La Florida	20
Daniel	Hebel	43683283	San Bernardo	20
Javiera	Arce	94367238	Quilpue	20
Juan	Perez	12345678	Villa Los Leones	21

Tabla 12. Actualización de registros.
Fuente: Desafío Latam.

Y vemos que la dirección ha sido actualizada. Considera que de no especificar el “where” se actualizarán los valores de esa columna en todos los registros.

Ejercicio propuesto (2)

La empresa Ovalle Electronics SPA registró en la venta con referencia 43224 un método de pago como "débito" y en una auditoría realizada recientemente se comprobó que esa venta fue cancelada por el cliente con una tarjeta de "crédito", por lo que se le pide al programador que se encarga de la base de datos hacer esa corrección. Usa el ejercicio propuesto 1 para este ejercicio.

Eliminación de registros

Cuando estamos completamente seguros de que queremos eliminar un registro de una tabla o incluso la tabla completa, y que de verdad estamos seguros de que la condición que usamos es la correcta, podemos utilizar el comando DELETE. Este lo que hace es eliminar uno, varios o todos los registros de la tabla según la condición dada. La sintaxis para eliminar toda la tabla es:

```
    DELETE FROM tabla;
```

Para poder seleccionar qué registros queremos borrar debemos hacerlo de la siguiente forma:

```
    DELETE FROM tabla WHERE condicion;
```

En nuestro ejemplo eliminaremos a John de la tabla de `Directorio_telefonico` de la siguiente forma:

```
    DELETE FROM Directorio_telefonico WHERE nombre='John';
```

Si quisiéramos borrar todos los datos de una tabla usaremos:

```
    DELETE FROM Agenda;
```

nick	numero_telefonico

Tabla 13. Eliminación de registros con DELETE.
Fuente: Desafío Latam.

Agreguemos un registro a la tabla **Agenda**:

```
    INSERT INTO Agenda
    VALUES('Juanito','12345678');
```

Ahora eliminar a Juan de la tabla de `Directorio_telefonico`

```
    DELETE FROM Directorio_telefonico WHERE nombre='Juan';
```

Obtenemos un error recordándonos que no se puede violar la restricción de clave foránea del número telefónico:

```
    ERROR:  update or delete on table "directorio_telefonico" violates
foreign key constraint "agenda_numero_telefonico_fkey" on table "agenda"
    DETAIL:  Key (numero_telefonico)=(12345678) is still referenced from
table "agenda".
```

Ejercicio propuesto (3)

La empresa Ovalle Electronics SPA decidió dejar de vender el televisor modelo RV-25, por lo que le pide al programador encargado de la base de datos que lo elimine del registro de los productos.

Lo pernicioso del método DELETE

Si bien el comando DELETE existe para ser usado, debe usarse con mucha precaución, es más, sólo el administrador de la base de datos debería ser capaz de eliminar datos de ella.

Este comando es susceptible a errores, ya que si no implementamos correctamente la condición en el comando WHERE, podemos eliminar datos que no teníamos pensado borrar y no hay posibilidad de recuperarlos.

PostgreSQL les da a todos permisos por defecto, sin embargo, utilizar el comando DELETE no es parte de ellos, el administrador de la base de datos es el encargado de otorgar ese poder.

Añadiendo o eliminado columnas

Utilizando la definición anterior de nuestra tabla Agenda, se busca agregar ahora una nota. Ante la eventualidad que deseamos añadir/remover una columna específica de una tabla, podemos implementar la siguiente sintáxis:

```
    ALTER TABLE nombre_tabla
    ADD nueva_columna tipo_de_dato;
```

Para la aplicación de este tipo de instrucción en nuestro ejemplo, procedemos a ejecutar el siguiente código en nuestra consola de PostgreSQL para la inclusión del campo “nota” en nuestra tabla Agenda.

```
    -- Declaramos que alteraremos la tabla Agenda
    ALTER TABLE Agenda
    -- Definimos el campo nick con el tipo de dato cadena con un largo
de 25 caracteres.
    ADD nota VARCHAR(100);
```

La estructura de la tabla Agenda quedaría entonces de la siguiente manera

nick	numero_telefonico	nota

Tabla 14. Tabla Agenda con columna “nota” agregada.
Fuente: Desafío Latam.

Si deseamos eliminar alguna columna en específico, podemos implementar la instrucción DROP de manera análoga a como lo hicimos con ADD.

```
    ALTER TABLE nombre_tabla
    DROP nueva_columna;
```

Ejercicio propuesto (4)

La empresa Ovalle Electronics SPA decidió desestimar el almacenamiento de la fecha de creación de los productos por lo que pide que esa propiedad sea eliminada y además solicita la creación de una nueva propiedad llamada “color” para futuros filtros de búsqueda.

Restricciones

Puede que existan casos en los que nuestras columnas necesiten reglas que cumplir, como que no hayan valores repetidos o que no existan campos vacíos. Es por eso que aparece el concepto de restricciones. En el siguiente listado se muestran las principales restricciones con su respectiva definición:

- `NOT NULL`: La columna no puede tener valores NULL.
- `UNIQUE`: Todos los valores de la columna deben ser diferentes unos a otros.
- `SERIAL`: Restringe a que el valor numérico sea autoincremental.
- `PRIMARY KEY`: Aplica la clave primaria.
- `FOREIGN KEY`: Sirve para enlazar 2 tablas, normalmente referente a una clave primaria.
- `CHECK`: Todos los valores de una columna deben satisfacer una condición en específico.
- `DEFAULT`: Le da un valor por defecto a aquellos registros que no tengan un valor asignado.
- `INDEX`: Sirve para crear y recuperar datos de forma rápida.

Estos se aplican de la siguiente forma:

```
-- Creamos una tabla
CREATE TABLE nombre_tabla(
-- Declaramos una serie de restricciones a cada campo de dato creado
columna1 tipo_de_dato1 restriccion,
columna2 tipo_de_dato2 restriccion,
columna3 tipo_de_dato3 restriccion
);
```

PRIMARY KEY y FOREIGN KEY se manejan de forma distinta, los explicaremos en la siguiente sección. Usaremos de ejemplo las mismas tablas que creamos antes, asignándoles restricciones:

Volveremos a crear las tablas `Directorio_telefónico` y `Agenda`:

```
CREATE TABLE Directorio_telefonico(
nombre VARCHAR(25) NOT NULL,
apellido VARCHAR(25),
numero_telefonico VARCHAR(8) UNIQUE,
direccion VARCHAR(255),
edad INT );
```

```
CREATE TABLE Agenda(
 nick VARCHAR(25) NOT NULL,
 numero_telefonico VARCHAR(8) UNIQUE
 );
```

En este caso, hicimos que `numero_telefonico` sea un valor único que no pueda repetirse, que nombre y nick no puedan tener valores NULL.

Ejercicio propuesto (5)

A la empresa Ovalle Electronics SPA le ha ido muy bien vendiendo televisores y ha empezado a contratar más personal por lo que consideró necesario tener en la base de datos un registro de sus empleados, almacenando su nombre, fecha de ingreso a la empresa, género y rut. Crea una tabla en la misma base de datos de los ejercicios propuestos, especificando que el rut es un valor único y que el nombre no puede ser un dato tipo null.

Restricciones a nivel de PRIMARY KEY y FOREIGN KEY

Como mencionamos antes, las claves primarias sirven para identificar un registro único en una tabla, y la clave foránea sirve para referenciar esa columna desde otra tabla. Se tiene la siguiente imagen:

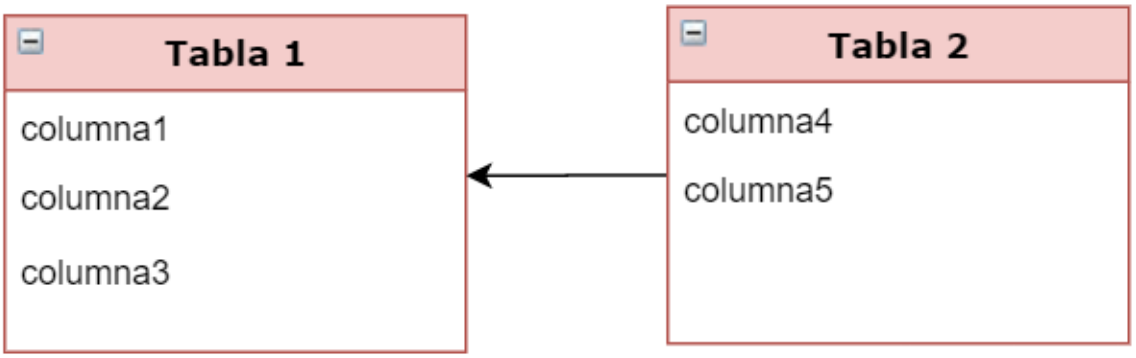

Imagen 13. Primary and Foreign Key.
Fuente: Desafío Latam.

Suponiendo que la columna1 de la tabla 1 fuese su clave primaria y la columna4 de la tabla 2 la foránea, la forma de aplicarlas es la siguiente:

```
CREATE TABLE tabla1(
columna1 tipo_de_dato1 UNIQUE,
columna2 tipo_de_dato2,
columna3 tipo_de_dato3,
PRIMARY KEY (columna1)
);

CREATE TABLE tabla2(
columna4 tipo_de_dato4,
columna5 tipo_de_dato5,
FOREIGN KEY (columna4) REFERENCES tabla1(columna1)
);
```

Nota que se ha agregado la restricción “UNIQUE” en la columna 1 para justamente restringir que los valores de ese atributo en todos los registros deben ser únicos.

Volvamos al ejemplo de `Directorio_telefonico` y `Agenda`, ambos usan los mismos números telefónicos para referirse a las mismas personas. La diferencia aquí, es que obtenemos los números desde `Directorio_telefonico` para llevarlos a `Agenda`.

Es por esto, que `numero_telefonico` será clave primaria (PRIMARY KEY) en `Directorio_telefonico` con la restricción "UNIQUE" para evitar repeticiones y al mismo tiempo existirá este atributo como la clave foránea (FOREIGN KEY) en `Agenda`.

Esto se puede ver reflejado así:

```
CREATE TABLE Directorio_telefonico(
nombre VARCHAR(25),
apellido VARCHAR(25),
numero_telefonico VARCHAR(8) UNIQUE,
direccion VARCHAR(255),
edad INT,
PRIMARY KEY (numero_telefonico)
);

CREATE TABLE Agenda(
nick VARCHAR(25),
numero_telefonico VARCHAR(8),
nota VARCHAR(100),
FOREIGN KEY (numero_telefonico) REFERENCES
Directorio_telefonico(numero_telefonico)
);
```

Ejercicio propuesto (6)

La empresa Ovalle Electronics SPA se dio cuenta que se están generando registros con datos que ya existen en otra tabla, y para evitar esta redundancia innecesaria le pide a su programador en base de datos que cree la relación entre las tablas Productos y Ventas. Vuelve a crear estas tablas asignando la clave primaria y foránea.

Eliminación de una tabla

Hemos creado y modificado tablas, añadido registros y definimos las restricciones de nuestro modelo de datos, pero ¿Qué pasa si queremos eliminar la tabla por completo?

Es riesgoso y se hará en muy pocas ocasiones, ya que una vez eliminada la tabla no tenemos forma de recuperar los datos que almacenaba, ni su estructura.

Insertemos un registro en las tablas creadas recientemente, para ver cómo se comportan:

```
    INSERT INTO Directorio_telefonico VALUES ('Juan','Perez',1234,'Av.
Falsa 123',20);
    INSERT INTO Agenda VALUES ('Juanito',1234,'Juanito dice hola');
```

Imaginemos que queremos eliminar la tabla Agenda, lo haremos de la siguiente manera:

```
    DROP TABLE Agenda;
```

La respuesta de PostgreSQL es:

```
    DROP TABLE;
```

Al consultar sobre la tabla Agenda, nos muestra lo siguiente:

```
    select * from Agenda;

    ERROR:  no existe la relación «agenda»
    LÍNEA 1: select * from agenda;
```

La tabla ha sido eliminada completamente.

Ejercicio propuesto (7)

La empresa Ovalle Electronics SPA ha bajado drásticamente sus ventas, no obstante ha notado en sus empleados un nivel técnico excelente y ha tomado la decisión de dejar de vender productos y cambiar su modelo de negocios para ofrecer servicio técnico, por lo que ya no necesita seguir usando el registro de productos, así que le pide a su programador que elimine la tabla de productos de la base de datos.

Truncado de una tabla

Si lo que realmente necesitamos es eliminar sólo los registros, pero no la tabla en sí, usaremos el comando `TRUNCATE`, de esta manera, limpiaremos todo lo que hayamos insertado en ella.

Debes utilizar este comando sólo si es realmente necesario, ya que al igual que el comando `DROP` no es posible recuperar la información una vez eliminada.

Creemos nuevamente la tabla Agenda y veamos lo que ocurre:

```
CREATE TABLE Agenda(
nick VARCHAR(25),
numero_telefonico VARCHAR(8),
nota VARCHAR(100),
FOREIGN KEY (numero_telefonico) REFERENCES
Directorio_telefonico(numero_telefonico)
);

INSERT INTO Agenda VALUES ('Juanito',1234,'Juanito dice hola');
```

Ejecutemos el comando Truncate sobre la tabla, de la siguiente manera:

```
TRUNCATE TABLE Agenda;
```

La respuesta de PostgreSQL es:

```
TRUNCATE TABLE;
```

Al consultar sobre la tabla Agenda, nos muestra lo siguiente:

```
select * from Agenda;
```

nick	numero_telefonico	nota

(0 filas)

Tabla 15. Truncado de una tabla.
Fuente: Desafío Latam.

Ejercicio propuesto (8)

Luego de un éxito tremendo ofreciendo servicio técnico la empresa Ovalle Electronics SPA ha logrado abrir una sucursal en otro estado del país, por lo que generó una copia de la base de datos para ser usada en esta nueva sede, pero ahora solicita al programador que vacíe la tabla de empleados para volver a llenarla pero con los nuevos trabajadores.

Cargar consultas desde un fichero

A veces tenemos una gran cantidad de sentencias que queremos realizar, y estar tipeando una a una puede ser algo engorroso, sumado a la pérdida de tiempo, y a que se está muy propenso a cometer pequeños errores.

Es por eso que podemos crear ficheros con extensión `.sql` que nos permitirá escribir todos nuestros comandos SQL para poder cargarlos.

La sintaxis para realizar esta operación es:

```
\i ubicación\nombre_fichero.sql
```

Solución de los ejercicios propuestos

1. La empresa Ovalle Electronics SPA necesita una base de datos para almacenar sus productos y el historial de ventas del día a día. Crea una base de datos llamada OvalleElectronicsSPA con las siguientes tablas:

nombre	ID	fecha_creacion	proveedor	categoría
TV RV-25	2468	2020-08-16	Dende SPA	televisores

Tabla 16. Tabla Productos en ejercicio propuesto.
Fuente: Desafío Latam.

fecha	ID_Producto	cliente	metodo_pago
2021-02-01	2468	Bruce Lee	efectivo

Tabla 17. Tabla Ventas en ejercicio propuesto.
Fuente: Desafío Latam.

```
CREATE DATABASE OvalleElectronicsSPA;
```

```
CREATE TABLE Productos(
    nombre VARCHAR(25),
    ID INT,
    fecha_creacion DATE,
    proveedor VARCHAR(25),
    categoria VARCHAR(25)
    );
```

```
INSERT INTO Productos
      (nombre, ID, fecha_creacion, proveedor, categoria)
    VALUES ('TV RV-25', '2468', '2020-08-16', 'Dende SPA',
'televisores');
```

```
CREATE TABLE Ventas(
    fecha DATE,
    ID_Producto INT,
    cliente VARCHAR(25),
    metodo_pago VARCHAR(25),
    referencia INT
    )
```

```
INSERT INTO Ventas
      (fecha, ID_Producto, cliente, metodo_pago, referencia)
    VALUES ('2020-08-16', '2468', 'Bruce Lee' , 'efectivo','34414');
```

```
INSERT INTO Ventas
      (fecha, ID_Producto, cliente, metodo_pago, referencia)
    VALUES ('2020-10-15', '2468', 'Chuck Norris' , 'debito', '43224');
```

2. La empresa Ovalle Electronics SPA registró en la venta con referencia 43224 un método de pago como "débito" y en una auditoría realizada recientemente se comprobó que esa venta fue cancelada por el cliente con una tarjeta de "crédito", por lo que se le pide al programador que se encarga de la base de datos hacer esa corrección. Usa el ejercicio propuesto 1 para este ejercicio.

```
UPDATE Ventas
SET metodo_pago='credito'
WHERE referencia='43224';
```

3. La empresa Ovalle Electronics SPA decidió dejar de vender el televisor modelo RV-25, por lo que le pide al programador encargado de la base de datos que lo elimine del registro de los productos.

```
    DELETE FROM Productos WHERE ID='2468';
```

4. La empresa Ovalle Electronics SPA decidió desestimar el almacenamiento de la fecha de creación de los productos por lo que pide que esa propiedad sea eliminada y además solicita la creación de una nueva propiedad llamada “color” para futuros filtros de búsqueda.

```
ALTER TABLE Productos
DROP fecha_creacion;
```

```
ALTER TABLE Productos
ADD color VARCHAR(25);
```

5. A la empresa Ovalle Electronics SPA le ha ido muy bien vendiendo televisores y ha empezado a contratar más personal por lo que consideró necesario tener en la base de datos un registro de sus empleados, almacenando su nombre, fecha de ingreso a la empresa, género y rut. Crea una tabla en la misma base de datos de los ejercicios propuestos, especificando que el rut es un valor único y que el nombre no puede ser un dato tipo null.

```
CREATE TABLE Empleados(
nombre VARCHAR(25) NOT NULL,
fecha_ingreso DATE,
genero VARCHAR(25),
rut VARCHAR(8) UNIQUE
);
```

6. La empresa Ovalle Electronics SPA se dio cuenta que se están generando registros con datos que ya existen en otra tabla, y para evitar esta redundancia innecesaria le pide a su programador en base de datos que cree la relación entre las tablas Productos y Ventas. Vuelve a crear estas tablas asignando la clave primaria y foránea.

```
CREATE TABLE Productos(
    nombre VARCHAR(25),
    ID INT PRIMARY KEY,
    fecha_creacion DATE,
    proveedor VARCHAR(25),
    categoria VARCHAR(25)
    )
```

```
CREATE TABLE Ventas(
    fecha DATE,
    FOREIGN KEY (ID_Producto) REFERENCES Productos(ID),
    cliente VARCHAR(25),
    metodo_pago VARCHAR(25),
    referencia INT
    );
```

7. La empresa Ovalle Electronics SPA ha bajado drásticamente sus ventas, no obstante ha notado en sus empleados un nivel técnico excelente y ha tomado la decisión de dejar de vender productos y cambiar su modelo de negocios para ofrecer servicio técnico, por lo que ya no necesita seguir usando el registro de productos, así que le pide a su programador que elimine la tabla de productos de la base de datos.

```
    DROP TABLE Productos;
```

8. Luego de un éxito tremendo ofreciendo servicio técnico la empresa Ovalle Electronics SPA ha logrado abrir una sucursal en otro estado del país, por lo que generó una copia de la base de datos para ser usada en esta nueva sede, pero ahora solicita al programador que vacíe la tabla de empleados para volver a llenarla pero con los nuevos trabajadores.

```
    TRUNCATE TABLE Empleados;
```