AWAKELAB

Emprendimiento digital con tecnologías Web

Módulo 7: Introducción a base de datos para emprendimiento digital

---

Aprendizaje Esperado

---

1. Construir consultas a una base de datos utilizando el lenguaje estructurado de consultas SQL y a partir de un modelo de datos para la obtención de información que satisface los requerimientos planteados.

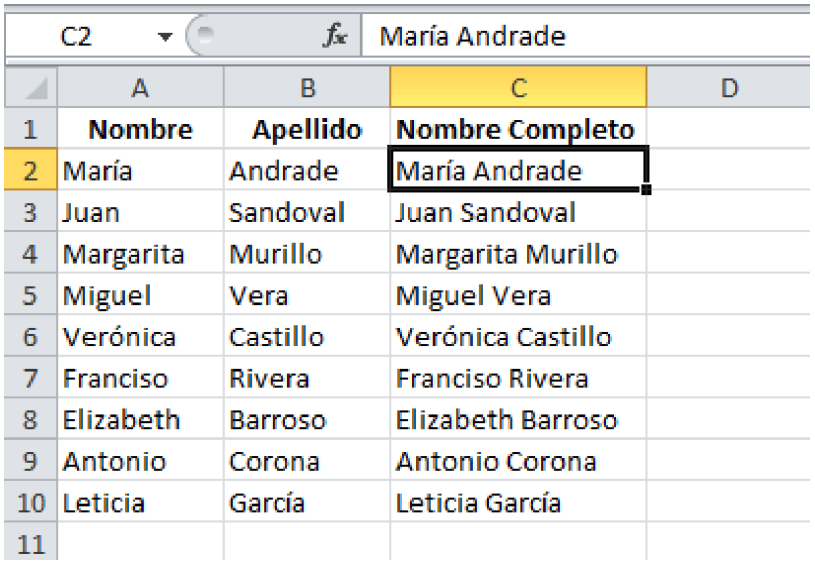
Bases de datos relacionales

1. El rol de las bases de datos relacionales

Las aplicaciones de hojas de cálculo, como Microsoft Excel, se utilizan ampliamente como una forma de almacenar e inspeccionar datos. Es fácil ordenar los datos de diferentes maneras y ver las características y patrones en estos con solo mirarlos.

Desafortunadamente, las personas a menudo confunden una herramienta que es buena para inspeccionar y manipular datos con una herramienta adecuada para almacenar y compartir datos complejos y quizás críticos para el negocio. Las dos necesidades suelen ser muy diferentes.

La mayoría de las personas están familiarizadas con una o más hojas de cálculo y se sentirán cómodas con los datos organizados en un conjunto de filas y columnas. LibreOffice es otra buena alternativa y hay bastantes más.



C2 | María Andrade

	A	B	C	D
1	**Nombre**	**Apellido**	**Nombre Completo**	
2	María	Andrade	María Andrade	
3	Juan	Sandoval	Juan Sandoval	
4	Margarita	Murillo	Margarita Murillo	
5	Miguel	Vera	Miguel Vera	
6	Verónica	Castillo	Verónica Castillo	
7	Franciso	Rivera	Franciso Rivera	
8	Elizabeth	Barroso	Elizabeth Barroso	
9	Antonio	Corona	Antonio Corona	
10	Leticia	García	Leticia García	
11				

Esta sencilla hoja de cálculo, indicada en la imagen anterior, incorpora varias características que serán útiles de recordar cuando comencemos a diseñar bases de datos. Por ejemplo, el nombre y los apellidos se mantienen en columnas separadas, lo que facilita la clasificación de los datos por apellido si es necesario. Entonces, ¿qué hay de malo en almacenar información del cliente en una hoja de cálculo? Las hojas de cálculo están bien, siempre y cuando:

- No tenga demasiados clientes
- No tenga muchos detalles complejos para cada cliente
- No sea necesario almacenar ninguna otra información repetida, como los distintos pedidos que ha realizado cada cliente.
- No quiera que varias personas puedan actualizar la información simultáneamente
- Asegurarse de que se realice una copia de seguridad de la hoja de cálculo con regularidad si contiene datos importantes

Las hojas de cálculo son una idea fantástica y son excelentes herramientas para muchos tipos de problemas. Sin embargo, así como no intentaría (o al menos no debería) intentar clavar un clavo con un destornillador, a veces las hojas de cálculo no son la herramienta adecuada para el trabajo.

Imagínese cómo sería si una gran empresa, con decenas de miles de clientes, mantuviera la copia maestra de su lista de clientes en una simple hoja de cálculo. En una gran empresa, es probable que varias personas

necesiten actualizar la lista. Aunque el bloqueo de archivos puede garantizar que solo una persona actualice la lista a la vez, a medida que aumenta el número de personas que intentan actualizar la lista, pasarán más y más tiempo esperando su turno para editar la lista. Lo que nos gustaría es permitir que muchas personas lean, actualicen, agreguen y eliminen filas simultáneamente, y que la computadora se asegure de que no haya conflictos. Claramente, el bloqueo simple de archivos no será adecuado para manejar este problema de manera eficiente.

Otro problema con las hojas de cálculo son sus estrictas dos dimensiones. Supongamos que también quisiéramos almacenar detalles de cada pedido que realizó un cliente. Podríamos comenzar a poner la información de los pedidos junto a cada cliente, pero a medida que aumentara la cantidad de pedidos por cliente, la hoja de cálculo se volvería cada vez más compleja.

Un sistema de administración de bases de datos (DBMS) suele ser un conjunto de bibliotecas, aplicaciones y utilidades que alivian al desarrollador de aplicaciones de la carga de preocuparse por los detalles de almacenamiento y administración de datos. También proporciona facilidades para buscar y actualizar registros. Los DBMS vienen en varios tipos desarrollados a lo largo de los años para resolver tipos particulares de problemas de almacenamiento de datos.

Cuando se mira superficialmente, una base de datos relacional (RDB), como PostgreSQL, MySQL u Oracle, tiene muchas similitudes con una hoja de cálculo. Sin embargo, cuando se conoce la estructura subyacente de una base de datos, puede ver que es mucho más flexible, principalmente debido a su capacidad para relacionar tablas de formas complejas. Puede almacenar de manera eficiente datos mucho más complejos que una hoja de cálculo, y también tiene muchas otras características que lo convierten en una mejor opción como almacén de datos. Por ejemplo, una base de datos puede administrar varios usuarios simultáneamente.

Pensemos en almacenar nuestra lista de clientes simple de una sola hoja en una base de datos, para ver qué beneficios podría tener.

Las bases de datos se componen de tablas, o en terminología más formal, relaciones. Nos ceñiremos al uso del término tablas. Una tabla contiene filas de datos y cada fila de datos consta de varias columnas o atributos.

1.1. Características de un RDBMS

Varias reglas importantes definen un sistema de gestión de bases de datos relacionales (RDBMS). Todas las filas deben seguir el mismo patrón, ya que todas tienen el mismo número y tipo de componentes. A continuación, se muestra un ejemplo de un conjunto de filas:

```
{"Chile", "CLP", 1000,56}
{"Bélgica", "EUR", 1,34}
```

Cada una de estas filas tiene tres atributos: un nombre de país (cadena), una moneda (cadena) y un tipo de cambio (un número de punto flotante). En una base de datos relacional, todos los registros que se agregan a este conjunto, o tabla, deben seguir el mismo formulario, por lo que no se permiten los siguientes:

```
{"Germany", "EUR"}
```

Esto tiene muy pocos atributos.

```
{"Switzerland", "CHF", "French", "German", "Italian", "Romansch"}
```

Esto tiene demasiados atributos.

```
{1936.27, "EUR", "Italy"}
```

Esto tiene tipos de atributos incorrectos (orden incorrecto).

Además, en cualquier tabla de filas, no debe haber duplicados. Esto significa que en cualquier tabla en una base de datos relacional correctamente diseñada, no puede haber filas o registros idénticos. Esto podría parecer una restricción bastante draconiana. Por ejemplo, en un sistema que registra los pedidos realizados por los clientes, parecería no permitir que el mismo cliente pida el mismo producto dos veces. Más adelante, veremos que hay una manera fácil de evitar este tipo de problemas/requisitos agregando un atributo.

Cada atributo de un registro debe ser "atómico"; es decir, debe ser un solo dato, no otro registro o una lista de otros atributos. Además, el tipo de atributos correspondientes en cada registro de la tabla (elementos de la columna) debe ser el mismo. Técnicamente, esto significa que deben extraerse del mismo conjunto de valores o dominio. En términos prácticos, significa que todos serán una cadena, un número entero, un valor de punto flotante o algún otro tipo admitido por el sistema de base de datos.

El atributo (o atributos) que se utilizan para distinguir un registro en particular en una tabla de todos los otros registros se llama clave primaria o clave principal (primary key). En una base de datos relacional, cada relación o tabla debe tener una clave principal para cada registro para que sea único, diferente de todos los demás en esa tabla.

1.3.- Conociendo las herramientas para consultar una base de datos

Los programas que permiten administrar bases de datos, organizan y almacenan los datos de manera tal que las tablas están indexadas y se pueden contestar preguntas. Estas herramientas están disponibles en un número de configuraciones diferentes y escalables para ser usadas por un individuo o una corporación global. Algunas son fáciles de usar, con bases de datos a manera de interfaz visual, mientras que otras, al final del espectro empresarial, requieren entrenamiento y herramientas especializadas para ser útiles.

A continuación se indicarán las más conocidas y utilizadas en la actualidad.

- **Microsoft Access:** Es un sistema de base de datos personal de Microsoft. Se trata de un producto de software orientado hacia lo visual, lo que hace que quienes no sean programadores puedan crear bases de datos útiles con facilidad. Si bien la estructura de base de datos de Access puede ampliarse para satisfacer las necesidades empresariales, su uso más común es para pequeñas bases de datos individuales o en programas multiusuario de uso limitado. Access integra el lenguaje Visual Basic para aplicaciones, por lo que es un entorno de desarrollo completo.
- **MySQL:** es una base de datos basada en servidor que permite a varios usuarios acceder a múltiples bases de datos. El software funciona en múltiples plataformas, incluyendo la mayoría de las

variedades de UNIX y Windows. Ofrece usabilidad de primer plano limitada y está diseñado como un servidor de base de datos back-end. MySQL se diferencia de otros productos de base de datos por sus costos, la versión no empresarial se distribuye de forma gratuita.

- **SQL Server:** es un servidor de base de datos a nivel empresarial escalable. Este producto se diferencia de la base de datos personal al no proporcionar las herramientas para el usuario que proporciona un producto de base de datos individual. El motor de base de datos se centra en responder rápidamente a las solicitudes del cliente en el formulario de consultas SQL. Estas consultas se pueden generar directamente en SQL Server, o por medio de una interfaz de usuario independiente desarrollada en una variedad de lenguajes de programación. SQL Server está diseñado para manejar bases de datos con millones de registros.
- **Oracle:** es otra base de datos escalable a nivel empresarial. Soporta bases de datos corporativas distribuidas, que permiten al usuario acceder a los datos de forma local o desde bases de datos remotas en una transacción transparente. Las bases de datos distribuidas ayudan a superar las limitaciones físicas de un entorno informático físico.

Instalando la base de datos y sus herramientas utilitarias

Cada motor o software que permite administrar bases de datos tiene su procedimiento particular, dependiendo también del sistema operativo en el que se haga.

En este caso se indicará cómo instalar MySQL en su versión 8.0 en un equipo local, habilitando también una herramienta que permitirá consultar los datos.

1. Acceder al sitio de descarga de complementos de MySQL.
    a. https://www.mysql.com/downloads/ [2]
2. En la parte central, hacer clic en el enlace “MySQL Community (GPL) Downloads »”.
3. En la página que se desplegará, seleccionar la opción “MySQL Installer for Windows”.

4. Aparecerán dos opciones de descarga; se debe elegir la segunda de ellas, la que pesa poco más de 400 MB. Esto se hace presionando el botón “Download”.
5. Con lo anterior se desplegará un botón de login y otro de registro. En realidad no es necesario presionar ninguno de ellos, así que puedes hacer clic en el enlace “No thanks, just start my download”.
6. Se descargará un archivo con extensión *.msi, qué es el instalable para Windows; se debe hacer doble clic sobre él.
7. En la sección que permite elegir un tipo de instalación, se debe seleccionar la primera opción de las cuatro, esto es, “Developer Default”. Esta opción no solo instalará el motor de MySQL en su última versión, sino también MySQL Workbench y otros complementos que te permitirán crear, consultar y administrar bases de datos en un entorno local, e incluso si lo deseas de manera remota. Una vez seleccionada la opción presiona “Next”.
8. En las secciones siguientes indicará datos para la customización del entorno. Se recomienda dejar estos datos tal y como están, especialmente el puerto sobre el cual operará este aplicativo en modo local.
9. Otro aspecto que se solicitará en la instalación es la clave de administrador del servidor local, denominado comúnmente “root”. No debes olvidar esta clave.
10. Una vez que se hayan instalado todas las herramientas necesarias, aparecerá un botón “Finish”, el cual al ser presionado terminará el proceso.

Creando una conexión a la base de datos

Gracias a la instalación del paso anterior, se instalará MySQL como motor de base de datos, y Workbench como herramienta de gestión. Además, es necesario considerar que junto con la instalación de estos aplicativos, se cargarán datos de prueba que permitirán hacer pruebas de concepto.

Para acceder a estos datos se debe seguir las siguientes acciones:

- Ingresa al programa MySQL Workbench (En Windows: Inicio MySQL MySQL Workbench).
- En primera instancia, solo existirá una conexión activa, que corresponde a la instancia creada en el ambiente local. Para ingresar en ella, se debe presionar el botón con nombre “Local

instance MySQL 80". Por cierto, puedes agregar más conexiones haciendo clic en el botón "+" ubicado antes de este recuadro.

- En caso que no se haya ingresado antes a la instancia, te solicitará el usuario y la clave respectiva. Recuerda que no es obligación conectarte siempre con el usuario "root", también lo puedes hacer con un usuario especialmente creado con menos permisos que el usuario administrador.
- A fin de probar consultas sobre una base de datos, existen las bases "sakila" y "world". Para ingresar a alguna de ella, se debe seleccionar la pestaña Schemas ubicada en el menú de la izquierda, junto con el menú "Administration". Ya dentro de esta pestaña, se debe presionar la flecha a un costado izquierdo y con ello desplegar su contenido.
- En cada base de datos, al momento de desplegar sus componentes, se despliegan las opciones:
    - **Tablas:** es un conjunto de datos que contienen los mismos registros. Todos los registros (llamados comúnmente filas) de una tabla tienen los mismos datos que los representan; estos últimos se conocen como campos.
    - **Vistas:** es una definición lógica de una tabla, formada por la unión de uno o más campos de diversas tablas. Las vistas pueden ser lógicas o bien materializadas.
    - **Procedimientos almacenados:** son un conjunto de instrucciones que realiza un conjunto de tareas específica entre una o más tablas, y que retorna un resultado.
    - **Funciones:** una función es una rutina o conjunto de acciones que aplican sobre un dato específico, transformándolo o bien aplicando operaciones de cálculo y/o transformación sobre el mismo.

1.3 Configurando el motor de base de datos

Cada instancia de Motor de base de datos debe configurarse satisfacer los requisitos de rendimiento y disponibilidad definidos para las bases de datos hospedadas por la instancia. El Motor de base de datos incluye opciones de configuración que controlan comportamientos como el uso de recursos y la disponibilidad de características como la los comportamientos de control como el uso de los recursos y la disponibilidad de características como auditoría o recursividad de desencadenador.

Configuración de instancia
Cuando una base de datos se implementa en producción a menudo hay un contrato de nivel de servicio (SLA) que define áreas como los niveles de rendimiento requeridos a partir de la base de datos y el nivel de disponibilidad necesario de la base de datos. Los términos del SLA controlan normalmente los requisitos de configuración para la instancia.

Una instancia se configura normalmente inmediatamente después de que ha instalado. La configuración inicial se determina normalmente por los requisitos del SLA de los tipos de bases de datos planeadas para implementarse en la instancia. Una vez implementadas las bases de datos, los administradores de bases de datos supervisan el rendimiento de la instancia y ajustan la configuración según sea necesaria si las métricas de rendimiento muestran que la instancia no cumple los requisitos del SLA.

Tareas de configuración:

Descripción de la tarea	**Tema**
Describe las distintas opciones de configuración de la instancia y cómo se pueden ver o cambiar estas opciones.	Opciones de configuración de servidor (SQL Server)
Describe cómo ver y configurar las ubicaciones predeterminadas de los archivos de datos y registro nuevos en la instancia.	Ver o cambiar las ubicaciones predeterminadas de los archivos de datos y registro (SQL Server Management Studio)
Describe cómo configurar SQL Server para que use soft-NUMA.	Soft-NUMA (SQL Server)
Describe cómo asignar un puerto TCP/IP a la afinidad del nodo de acceso a memoria no uniforme (NUMA).	Asignación de puertos TCP/IP a nodos NUMA (SQL Server)

Describe cómo habilitar la directiva de Bloquear páginas en memoria de Windows. Esta directiva determina qué cuentas pueden usar un proceso que mantiene los datos en memoria física, impidiendo que el sistema lleve los datos a páginas de memoria virtual (disco).	Habilitar la opción Bloquear páginas en la memoria (Windows)

1.4 Acesso a base de datos a través de terminal

Es muy normal que utilicemos MySQL a través de páginas PHP y para administrar la base de datos utilicemos un programa como PhpMyAdmin, pero a veces no nos queda otro remedio que acceder a la base de datos a través de la línea de comandos, por ejemplo cuando estamos en un servidor remoto al que accedemos por terminal, o cuando no tenemos otra herramienta de interfaz gráfica instalada.
MySQL tiene un programa cliente, que se llama con el mismo nombre de la base de datos (mysql), que sirve para gestionar la base datos por línea de comandos. Ese programa está disponible en cualquier instalación de MySQL y lo tendremos que usar para conectarnos por línea de comandos.

Localizar el cliente MySQL en Windows
En un ordenador Windows ese programa se encuentra en un directorio como:
- C:\Archivos de programa\MySQL\MySQL Server 4.1\bin
- C:\xampp\mysql\
El directorio puede variar, por ejemplo, puede estar localizado en la raíz del disco C:, o en cualquier otro lugar donde podamos haber instalado MySQL. También depende de qué programa has usado para instalar MySQL y qué versión tienes. Una búsqueda en Google seguro que te ayudará a encontrar la carpeta correcta para tu caso.
Para acceder a la consola de MySQL en Windows tendremos que estar situados dentro de ese directorio, o bien colocar esa carpeta en la configuración de PATH.

Localizar el cliente MySQL en Linux
En Linux, por supuesto, también se puede acceder a MySQL por línea de comandos. Posiblemente desde cualquier directorio podamos acceder a la consola de MySQL, sin necesidad de situarse en el directorio donde esté instalado, ya que una vez instalado el motor de base de datos, nos proporciona el comando "mysql", estemos en la carpeta que estemos dentro de nuestro terminal.

Localizar el cliente MySQL en Mac
En Mac dependerá de cómo hemos instalado MySQL. El comando no está siempre disponible en nuestro terminal aunque tengamos instalado el motor de base de datos.

Aquí de nuevo una búsqueda en Google nos podrá decir cómo acceder al comando "mysql" si no está disponible en nuestro programa de terminal. Pero una configuración muy habitual es que hayamos instalado Mamp, en cuyo caso la respuesta la encuentras en la FAQ: Usar el comando mysql en Mac con instalación de Mamp Server.

1.5 Crear una base de datos desde el shell de postgreSQL

Disponemos de dos métodos para crear una base de datos en PostgreSQL. Antes de poder crear una base de datos, debemos de tener instalado PostgreSQL y crear una instancia donde se almacenan los datos.

Métodos
En PostgreSQL existen dos formas para crear una base de datos, una es utilizando un comando de Postgres, por lo cual debemos de conectarnos a una base de datos que exista. Y el otro método es utilizar un ejecutable de la instalación de PostgreSQL.

Comandos
Previamente tenemos que conectarnos a una base de datos que ya exista, por ejemplo «postgres» y desde esta conexión utilizamos el siguiente comando:

```
CREATE DATABASE nuevadb;
```

Esta nueva base de datos será creada con los mismos objetos que contenga la plantilla «template1», que al ser una plantilla predeterminada, se encuentra vacía.

Por lo que es posible crear una base de datos nueva a partir de una plantilla que disponga de una serie de objetos como tablas. Para ello:

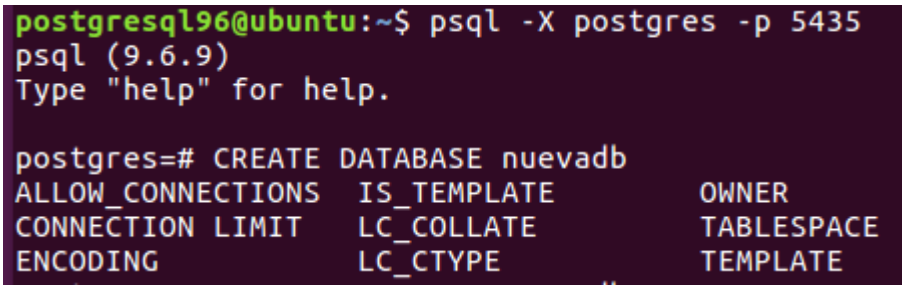
```
CREATE DATABASE nuevadb WITH TEMPLATE plantilladb;
```

Otra de las opciones de las que disponemos es indicarle el propietario de esta nueva base de datos. Simplemente debemos de indicar lo siguiente:

```
CREATE DATABASE nuevadb WITH OWNER usuario;
```

Estas dos opciones y otras más como el tablespace, el juego de caracteres o el limite de conexiones, se puede indicar en una única sentencia separando por espacios cada una de ella:

```
CREATE DATABASE nuevadb TEMPLATE plantilladb OWNER usuario TABLESPACE mydirectorio;
```



Ejecutable para la creación de la BBDD
En el directorio de los binarios de PostgreSQL, disponemos de un ejecutable con el que podemos crear una base de datos, sin tener que conectar a una existente previamente.

De la siguiente forma se crea una base de datos utilizando el ejecutable de PostgreSQL:

```
createdb nuevadb
```

También es posible crear una base de datos a partir de una plantilla utilizando este ejecutable, para ello, indicamos la opción «-T»:

```
createdb nuevadb -T plantilladb
```

Para indicarle el propietario a esta nueva base de datos, utilizamos la opción «-O» de este ejecutable:

```
createdb nuevadb -O usuario
```

Al igual que el comando, este ejecutable te permite indicar las propiedades de la nueva base de datos en una única línea:

```
createdb nuevadb -T plantilladb -O usuario -D mydirectorio
```

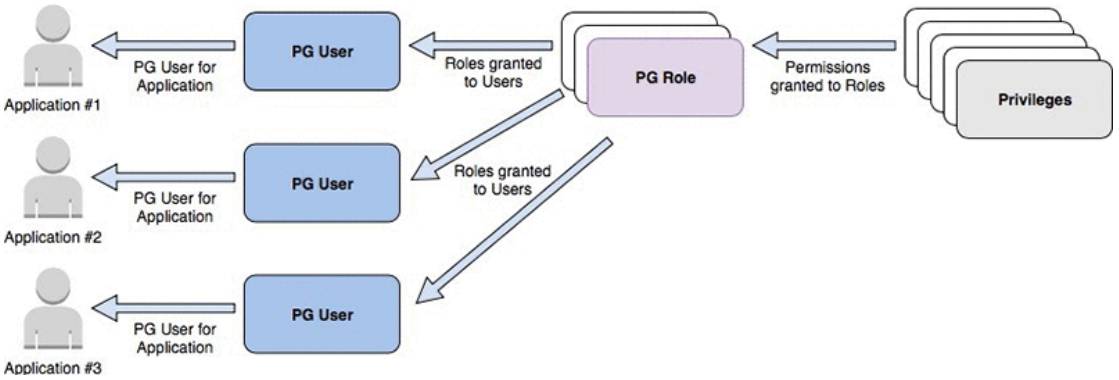
1.6 Crear usuarios desde el shell de postgreSQL

Con PostgreSQL, puede crear usuarios y roles con permisos de acceso granulares. Al nuevo usuario o rol se les debe conceder selectivamente los permisos necesarios para cada objeto de base de datos. Esto da mucho poder al usuario final, pero al mismo tiempo, dificulta potencialmente el proceso de creación de usuarios y roles con los permisos correctos.

PostgreSQL le permite conceder permisos directamente a los usuarios de la base de datos. Sin embargo, como práctica recomendada, se recomienda crear varios roles con conjuntos específicos de permisos basados en los requisitos de aplicación y acceso. Paso seguido, asigne el rol apropiado a cada usuario. Los roles deben utilizarse para aplicar un modelo de privilegios mínimos para acceder a objetos de base de datos. El usuario maestro que se crea durante la creación de instancias de Amazon RDS y Aurora PostgreSQL solo debe utilizarse para tareas de administración de bases de datos, como la creación de otros usuarios, roles y bases de datos. El usuario maestro nunca debe ser utilizado por la aplicación.

El método recomendado para configurar un control de acceso detallado en PostgreSQL es el siguiente:

- Utilice el usuario maestro para crear roles por aplicación o caso de uso, como readonly (solo lectura) y readwrite (escritura).
- Agregue permisos para permitir que estos roles tengan acceso a varios objetos de base de datos. Por ejemplo, el rol de readonly solo puede ejecutar consultas SELECT.
- Conceda a los roles los menos permisos posibles necesarios para la funcionalidad.
- Cree nuevos usuarios para cada aplicación o funcionalidad distinta, como app_user (usuario de aplicación) y reporting_user (usuario de reportes).
- Asigne los roles aplicables a estos usuarios para otorgarles rápidamente los mismos permisos que el rol. Por ejemplo, conceda el rol readwrite aapp_user y conceda el rol de readonly a reporting_user.
- En cualquier momento, puede quitar el rol del usuario para revocar los permisos.



El siguiente diagrama presenta un resumen de estas recomendaciones:

En las siguientes secciones se describen estos pasos en detalle. Puede conectarse al extremo de RDS de la base de datos de PostgreSQL mediante un cliente como psql y ejecutar las sentencias SQL.

1.7 Listar base de datos desde el shell de postgreSQL

Vamos a listar bases de datos (lo que equivale a ejecutar `show databases` en MySQL) y tablas (`show tables` en MySQL) en Postgres, ejecutando consultas desde línea de comandos con la herramienta `psql`. La primera limitación que se encuentra un usuario de MySQL al interactuar por primera vez con un gestor de bases de datos Postgres.

A pesar de que existen herramientas gráficas, como pgAdmin, es necesario conocer cómo se realizan las tareas desde línea de comandos para estar preparado para cuando las papas queman.

La herramienta `psql` provee una terminal interactiva contra servidores de bases de datos PostgreSQL. Es posible utilizarla de forma local (ejecutar `psql` directamente en la consola del servidor de bases de datos), o contra un servidor remoto especificando el host y puerto mediante los parámetros `-h` y `-p`:

```
root@dbserver42:~# psql -U postgres
psql (9.3.3)
Type "help" for help.

postgres=#
```

Para listar las bases de datos de un servidor Postgres (luego de conectarse al mismo) ejecutar el comando `\l`:

```
postgres=# \l
                                  List of databases
   Name    |  Owner   | Encoding |   Collate   |    Ctype    |      Access privileges
-----------+----------+----------+-------------+-------------+-----------------------------
 prueba    | postgres | UTF8     | es_AR.UTF-8 | es_AR.UTF-8 | postgres=CTc/postgres      +
           |          |          |             |             | rol_prueba=Tc/postgres     +
           |          |          |             |             | rol_readonly=Tc/postgres
 postgres  | postgres | UTF8     | es_AR.UTF-8 | es_AR.UTF-8 |
 template0 | postgres | UTF8     | es_AR.UTF-8 | es_AR.UTF-8 | =c/postgres                +
           |          |          |             |             | postgres=CTc/postgres
 template1 | postgres | UTF8     | es_AR.UTF-8 | es_AR.UTF-8 | postgres=CTc/postgres      +
           |          |          |             |             | =c/postgres
(4 rows)

postgres=#
```

Luego, para cerrar la terminal `psql`, ejecutar el comando `\q`:

```
postgres=# \q
root@dbserver42:~#
```

También es posible listar las bases de datos ejecutando directamente el comando `psql -l` en la terminal bash:

```
root@dbserver42:~# psql -U postgres -l
                                  List of databases
   Name    |  Owner   | Encoding |   Collate   |    Ctype    |      Access privileges
-----------+----------+----------+-------------+-------------+-----------------------------
 prueba    | postgres | UTF8     | es_AR.UTF-8 | es_AR.UTF-8 | postgres=CTc/postgres      +
           |          |          |             |             | rol_prueba=Tc/postgres     +
           |          |          |             |             | rol_readonly=Tc/postgres
 postgres  | postgres | UTF8     | es_AR.UTF-8 | es_AR.UTF-8 |
 template0 | postgres | UTF8     | es_AR.UTF-8 | es_AR.UTF-8 | =c/postgres                +
           |          |          |             |             | postgres=CTc/postgres
 template1 | postgres | UTF8     | es_AR.UTF-8 | es_AR.UTF-8 | postgres=CTc/postgres      +
           |          |          |             |             | =c/postgres
(4 rows)
```

En el listado se observan 4 bases de datos, de las cuales 3 son bases de datos de sistema (no se deben eliminar). Cada vez que se crea una base de datos con el comando `CREATE DATABASE`, en realidad se está copiando una base de datos existente (por defecto el modelo estándar de bases de datos "template1"). Entonces, la base de datos "template1" es una plantilla desde la cual se construyen nuevas bases de datos. Si se agregan objetos a esta base de datos, serán copiados en bases de datos subsecuentes cada vez que se utilice el comando `CREATE DATABASE`. El segundo modelo estándar de bases de datos "template0" contiene los mismos datos que posee inicialmente "template1", es decir los objetos estándar predefinidos por la versión de PostgreSQL. **Nunca deben realizarse modificaciones sobre "template0".** La tercera base de datos de sistema, "postgres",

funciona como base de datos por defecto para gestionar conexiones al servidor de bases de datos por parte de usuarios y aplicaciones. Se crea cuando se inicia un cluster de bases de datos y es una simple copia de "template1". Puede eliminarse (*drop*) y recrearse si es necesario. Para mayor información ver la documentación oficial de PostgreSQL: 21.3. Template Databases.

Para listar tablas de una base de datos Postgres existe el comando `\dt`. Conectarse a la base de datos especificando su nombre con el parámetro `-d`, y luego ejecutar el comando `\dt`. Por ejemplo el listado de tablas de una base de datos de un sistema Moodle:

```
root@dbserver42:~# psql -U postgres -d moodle
psql (9.3.3)
Type "help" for help.

moodle=# \dt
                    List of relations
 Schema |               Name                | Type  |  Owner
--------+-----------------------------------+-------+----------
 public | mdl_assign                        | table | postgres
 public | mdl_assign_grades                 | table | postgres
 public | mdl_assign_plugin_config          | table | postgres
 public | mdl_assign_submission             | table | postgres
 public | mdl_assign_user_flags             | table | postgres
 public | mdl_assign_user_mapping           | table | postgres
 public | mdl_assignfeedback_comments       | table | postgres
 public | mdl_assignfeedback_file           | table | postgres
 public | mdl_assignment                    | table | postgres
 public | mdl_assignment_submissions        | table | postgres
 public | mdl_assignsubmission_file         | table | postgres
 public | mdl_assignsubmission_onlinetext   | table | postgres
 public | mdl_backup_controllers            | table | postgres
 public | mdl_backup_courses                | table | postgres
 public | mdl_backup_files_template         | table | postgres
 public | mdl_backup_ids_template           | table | postgres
 public | mdl_backup_logs                   | table | postgres
 public | mdl_badge                         | table | postgres
 public | mdl_badge_backpack                | table | postgres
 public | mdl_badge_criteria                | table | postgres
 public | mdl_badge_criteria_met            | table | postgres
--Más--
```

1.8 Ingresar a una base de datos desde el shell de postgreSQL

Para conectarte a PostgreSQL desde la línea de comandos, sigue estos pasos.

1.- Si usas un servidor local, abre la consola. Si se trata de un servidor remoto, conéctate mediante SSH al servidor.
2.- Escribe el siguiente comando, cambiando BASE_DE_DATOS por el nombre de la base de datos y NOMBRE_USUARIO por tu nombre de usuario.

2.- Consultando información de una tabla

2.1.- El lenguaje estructurado de consultas SQL

SQL es un lenguaje de programación ampliamente utilizado que te permite definir y consultar bases de datos. Ya seas un analista de marketing, un periodista o un investigador que mapea neuronas en el cerebro de una mosca de la fruta, se beneficiará del uso de SQL para administrar objetos de bases de datos, así como para crear, modificar, explorar y resumir datos.

Dado que SQL es un lenguaje maduro que existe desde hace décadas, está profundamente arraigado en muchos sistemas modernos. Un par de investigadores de IBM describieron por primera vez la sintaxis de SQL (entonces llamado SEQUEL) en un artículo de 1974, basándose en el trabajo teórico del científico informático británico Edgar F. Codd. En 1979, una precursora de la empresa de bases de datos Oracle (entonces llamada Relational Software) se convirtió en la primera en utilizar el lenguaje en un producto comercial. Hoy en día, sigue siendo uno de los lenguajes informáticos más utilizados en el mundo y es poco probable que eso cambie pronto.

SQL viene en varias variantes, que generalmente están vinculadas a sistemas de bases de datos específicos. El Instituto Nacional Estadounidense de Estándares (ANSI) y la Organización Internacional de Normalización (ISO), que establecen estándares para productos y tecnologías, proporcionan estándares para el idioma y lo revisan. La buena noticia es que las variantes no se alejan mucho del estándar, por lo que una vez que aprenda las convenciones SQL para una base de datos, podrá transferir ese conocimiento a otros sistemas.

Entonces, ¿por qué debería usar SQL? Después de todo, SQL no suele ser la primera herramienta que las personas eligen cuando están aprendiendo a analizar datos. De hecho, muchas personas comienzan con hojas de cálculo de Microsoft Excel y su variedad de funciones analíticas. Después de trabajar con Excel, es posible que se pasen a Access, el sistema de base de datos integrado en Microsoft Office, que tiene una interfaz gráfica de consulta que facilita el trabajo, lo que hace que las habilidades de SQL sean opcionales.

Pero, como ya explicamos anteriormente en esta clase, Excel y Access tienen sus límites. Excel actualmente permite un máximo de 1.048.576 filas por hoja de trabajo, y Access limita el tamaño de la base de datos a dos gigabytes y limita las columnas a 255 por tabla. No es infrecuente que los conjuntos de datos superen esos límites, especialmente cuando se trabaja con datos descargados de sistemas gubernamentales. El último obstáculo que desea descubrir al enfrentarse a una fecha límite es que su sistema de base de datos no tiene la capacidad para realizar el trabajo.

El uso de un sólido sistema de base de datos SQL le permite trabajar con terabytes de datos, múltiples tablas relacionadas y miles de columnas. Le brinda un control programático mejorado sobre la estructura de sus datos, lo que genera eficiencia, velocidad y, lo más importante, precisión.

SQL también es un excelente complemento de los lenguajes de programación utilizados en ciencias de datos, como R y Python. Si usa cualquiera de esos lenguajes, por ejemplo, puede conectarse a bases de datos SQL y, en algunos casos, incluso incorporar la sintaxis SQL directamente en el lenguaje. Para las personas sin experiencia en lenguajes de programación, SQL a menudo sirve como una introducción

fácil de entender a los conceptos relacionados con las estructuras de datos y la lógica de programación.

Además, conocer SQL puede ayudarle más allá del análisis de datos. Si profundiza en la creación de aplicaciones en línea, encontrará que las bases de datos brindan el poder de backend para muchos frameworks web comunes, mapas interactivos y sistemas de administración de contenido. Cuando necesite excavar debajo de la superficie de estas aplicaciones, la capacidad de SQL para manipular datos y bases de datos será muy útil.

2.2. Recuperando información de una tabla

Antes de consultar datos desde una base, es la acción de crearla. Para esto sigue estos pasos:

- Ingresa a MySQL Workbench
- Haz clic en la conexión de usuario root, e ingresa la clave determinada anteriormente.
- Una vez dentro de la sesión, busca la pestaña “Schemas” en el menú del lado izquierdo.
- Presiona el botón secundario del mouse, y selecciona la opción “Create Schema”.
- En la ventana que aparecerá, ingresa el nombre “escuela” en el campo name, y selecciona un conjunto de caracteres (“charset”); para el idioma español te recomendamos el tipo “UTF-8”, el que contiene tildes y caracteres propios del idioma.
- En la misma ventana anterior, en la elección del subtipo (“collation”), debes seleccionar un tipo de juego de caracteres que se adapte a tus necesidades. Podrías escoger, por ejemplo, el tipo UTF8_general_ci.
- Realizado lo anterior, presiona el botón “Apply”, con lo que se creará el esquema. Este concepto de esquema se encuentra y es equivalente en la mayoría de los SGBDR (MySQL, Oracle, SQL Server, Postgres...). Por supuesto, el concepto de base de datos también, pero puede presentar diferencias significativas según el SGBD. En lo que respecta a MySQL, el término esquema es completamente intercambiable con el término base de datos y, a partir de ahora, se utilizará el término esquema para evitar la ambigüedad con el

software MySQL bautizado a menudo como la "Base de datos MySQL".

El siguiente paso recomendado es crear un usuario que tenga acceso completo solo al esquema recién creado. Con esto se asegura mantener la integridad de los datos, y evitar pérdida de información sensible. Para esto se deben seguir estos pasos:

- Dentro de MySQL Workbench como root, debes seleccionar la pestaña "Administration".
- Haz clic en la opción "Users and privileges".
- En la ventana que se despliega a la derecha, presiona el botón "Add Account".
- Por cada sección ingresa los siguientes datos:
    - Login: Ingresa nombre de usuario y clave (se pide dos veces)
    - Accounts limits: puedes limitar la acción del usuario. Por lo pronto no haremos cambios acá.
    - Administrative roles: puedes configurar roles de administración al usuario. Dado que queremos limitar los permisos del usuario, no alteramos esta sección.
    - Schema privileges: presiona el botón "Add Entry …", y selecciona la base "escuela" creada anteriormente, y presiona "Ok".
    - Se agregará un nuevo registro al listado; presiona el registro, y en la parte de abajo selecciona todos los permisos de la parte izquierda y central (permisos de objeto y de DDL). Una vez realizado presiona el botón "Apply".
- Con esto el usuario estará creado en sistema, y podrás realizar acciones con este usuario sobre la base recién creada.

Para los ejemplos continuaremos utilizando la base de datos creada anteriormente; recuerda que debes ingresar desde ahora con el usuario creado recientemente, y seleccionar el esquema "escuela" como esquema por defecto (botón derecho sobre el esquema, y selecciona la opción "Set as Default Schema"). La sintaxis para crear una tabla en nuestra base de datos es la siguiente:

```
CREATE TABLE profesores (
  id int primary key auto_increment,
  nombre varchar(25),
```

```
    apellido varchar(50),
    escuela varchar(50),
    fecha_de_contratacion date,
    sueldo int
);
```

Ahora bien, para cada columna (id, nombre, apellido) utilizamos un tipo de datos específico. Dado que ya tenemos la tabla profesores, vamos a llenarla con algunos datos; para tal propósito utilizaremos la instrucción INSERT de SQL:

```
INSERT INTO profesores (
nombre, apellido, escuela, fecha_de_contratacion, sueldo)
VALUES
('Juanita', 'Perez', 'Gabriela Mistral', '2011-10-30', 234000),
('Bruce', 'Lee', 'Republica Popular China', '1993-05-22', 780945),
('Juan Alberto', 'Valdivieso', 'Sagrada Concepcion', '2005-08-01', 3400000),
('Pablo', 'Rojas', 'E-34', '2011-10-30', 300000),
('Nicolas', 'Echenique', 'Bendito Corazón de María', '2005-08-30', 8900000),
('Jericho', 'Jorquera', 'A-18 Abrazo de Maipu', '2010-10-22', 67500);
```

También, podemos insertar filas individuales o una a una:

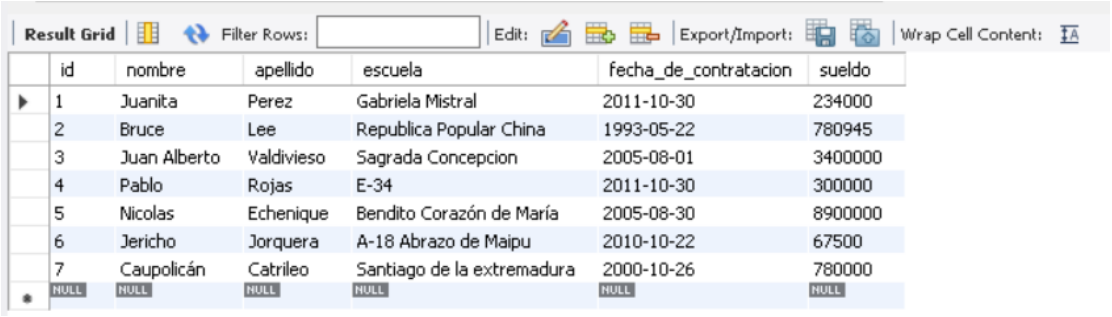
```
INSERT INTO profesores (
nombre, apellido, escuela, fecha_de_contratacion, sueldo)
VALUES
('Caupolicán','Catrileo','Santiago de la extremadura',
'2000-10-26', 780000);
```

Observa que ciertos valores que estamos insertando están entre comillas simples, pero otros no. Este es un requisito estándar de SQL. El texto y las fechas requieren comillas; los números, incluidos los números enteros y decimales, no requieren comillas.
Note que no se utilizó cifra alguna para la columna id, que es la primera columna de la tabla. Esto sucede porque cuando creó la tabla, su secuencia de comandos especificó que esa columna fuera del tipo autoincremental. Entonces, cuando se inserta cada fila, automáticamente se llena la columna id con un número entero que se incrementa automáticamente como veremos a continuación cuando recuperemos (o

leamos) la información de nuestra tabla profesores utilizando la sentencia SELECT de SQL. Una instrucción SELECT puede ser simple, recuperando todo en una sola tabla, o puede ser lo suficientemente compleja como para vincular docenas de tablas mientras maneja múltiples cálculos y filtra por criterios exactos.

Aquí hay una declaración SELECT que recupera cada fila y columna de la tabla llamada profesores:

```
SELECT * FROM profesores;
```

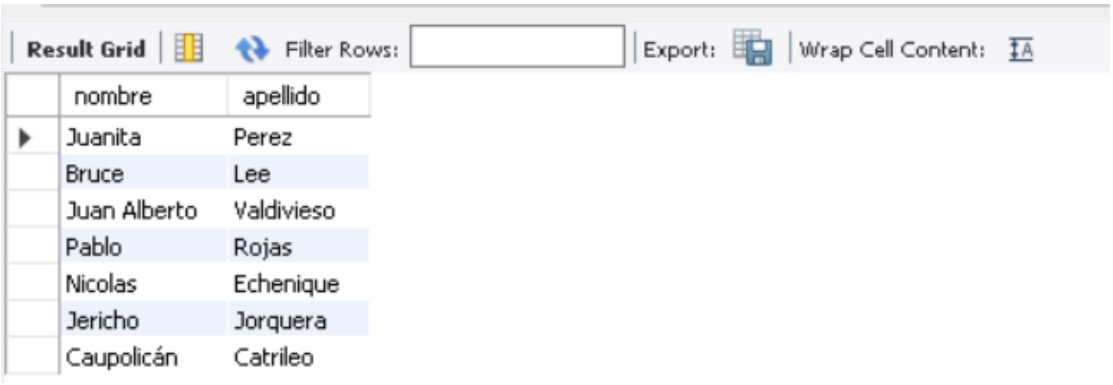


id	nombre	apellido	escuela	fecha_de_contratacion	sueldo
1	Juanita	Perez	Gabriela Mistral	2011-10-30	234000
2	Bruce	Lee	Republica Popular China	1993-05-22	780945
3	Juan Alberto	Valdivieso	Sagrada Concepcion	2005-08-01	3400000
4	Pablo	Rojas	E-34	2011-10-30	300000
5	Nicolas	Echenique	Bendito Corazón de María	2005-08-30	8900000
6	Jericho	Jorquera	A-18 Abrazo de Maipu	2010-10-22	67500
7	Caupolicán	Catrileo	Santiago de la extremadura	2000-10-26	780000
NULL	NULL	NULL	NULL	NULL	NULL

Esta única línea de código muestra la forma más básica de una consulta SQL. El asterisco que sigue a la palabra clave SELECT es un comodín. Un comodín es como un sustituto de un valor: no representa nada en particular y, en cambio, representa todo lo que ese valor podría ser. Aquí, es la abreviatura de "seleccionar todas las columnas". Si hubiera dado un nombre de columna en lugar del comodín, este comando seleccionaría los valores en esa columna. La palabra clave FROM indica que desea que la consulta devuelva datos de una tabla en particular. El punto y coma después del nombre de la tabla le dice a PostgreSQL que es el final de la declaración de consulta.

También podemos consultar un subconjunto de columnas:

```
SELECT nombre, apellido FROM profesores;
```



nombre	apellido
Juanita	Perez
Bruce	Lee
Juan Alberto	Valdivieso
Pablo	Rojas
Nicolas	Echenique
Jericho	Jorquera
Caupolicán	Catrileo

2.3.- Consultas utilizando la llave primaria

Una llave primaria o clave principal (primary key) es una columna o colección de columnas cuyos valores identifican de forma única cada fila de una tabla. Una columna de clave primaria válida impone ciertas restricciones:

- La columna o colección de columnas debe tener un valor único para cada fila.
- La columna o colección de columnas no puede tener valores faltantes.

En el caso que hemos visto hasta ahora con la tabla profesores, el campo id es una especie de llave primaria por defecto. Fíjate que cumple con las restricciones, pero podemos definir nuestras propias llaves primarias.

La llave primaria se puede definir al momento de creación de la tabla. En el primer campo de la tabla se acompaña con el término “primary key”.

```
CREATE TABLE profesores (
   id int,
   nombre varchar(25),
   apellido varchar(50),
   escuela varchar(50),
   fecha_de_contratacion date,
   sueldo int,
   constraint profesores_pk primary key (nombre)
```

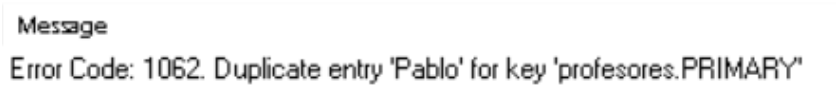
```
);
```

También la podemos agregar una vez que la tabla ya ha sido creada; para este caso definiremos el campo “nombre” como llave primaria:

```
alter table profesores add primary key (nombre);
```

Ahora bien, dado que nombre es nuestra llave primaria debe cumplir con las restricciones anteriormente descritas, es decir no podemos, por ejemplo, en este caso duplicar un nombre:

```
INSERT INTO profesores (nombre, apellido, escuela, fecha_de_contratacion, sueldo) VALUES ('Pablo', 'Lizarraga', 'Sagrado Grial de Montegrande', '2004-08-21', 6800000);
```



Naturalmente nuestro ejemplo es solo ilustrativo, los nombres de personas se repiten frecuentemente en una base de datos; no obstante, no hay dos personas que tengan el mismo número de cédula de identidad, tal condición es ideal para crear una llave primaria si su diseño y tabla lo requieren.

2.4.- Consultas utilizando condiciones de selección

Las consultas que requieren criterios de selección son naturalmente una práctica absolutamente necesaria y frecuente. Veremos algunas típicas, pero existen muchos criterios de selección a la hora de ejecutar estas consultas.

Anteriormente vimos la sintaxis básica para la instrucción SELECT

```
SELECT * FROM profesores;
```

Podemos elegir solo campos que nos interesan descartando los que no nos sirven para una consulta en particular. La regla general es

```
SELECT una_columna, otra_columna, super_columna FROM tabla;
```

Ejemplo 1:

```
SELECT apellido, escuela, sueldo FROM profesores;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

	apellido	escuela	sueldo
▶	Lee	Republica Popular China	780945
	Catrileo	Santiago de la extremadura	780000
	Jorquera	A-18 Abrazo de Maipu	67500
	Valdivieso	Sagrada Concepcion	3400000
	Perez	Gabriela Mistral	234000
	Echenique	Bendito Corazón de María	8900000
	Rojas	E-34	300000

Con esa sintaxis, la consulta recupera todas las filas de solo esas tres columnas. El orden lo podemos elegir de acuerdo a nuestras necesidades, por ejemplo:

```
SELECT sueldo, apellido, escuela FROM profesores;
```

Aunque estos ejemplos son básicos, ilustran una buena estrategia para comenzar a consultar un conjunto de datos. Generalmente, es

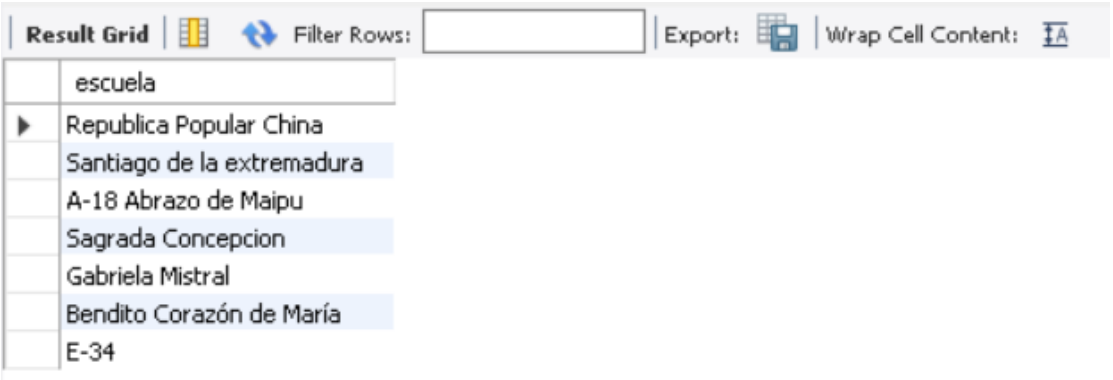
aconsejable comenzar su análisis verificando si sus datos están presentes y en el formato que espera.

Para este ejemplo primero insertemos una nueva fila con un apellido y una escuela repetida:

```
INSERT INTO profesores (nombre, apellido, escuela, fecha_de_contratacion, sueldo) VALUES ('Wong', 'Lee', 'Santiago de la extremadura', '2000-10-26', 780000);
```

Supongamos que necesitamos saber solo las escuelas que existen en nuestros registros, esto lo podemos lograr con **DISTINCT**:

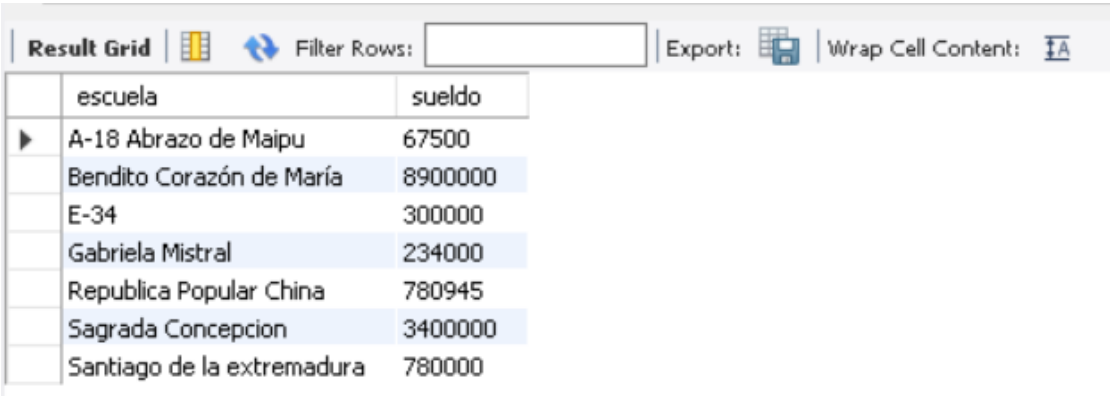
```
SELECT DISTINCT escuela FROM profesores;
```



Pese a que los profesores Catrileo y Wong Lee trabajan en la misma escuela, solo necesitamos las escuelas y obviamente dejar de lado repeticiones innecesarias.

Podríamos ordenar la información para que nos resultará más legible, por ejemplo, alfabéticamente usando **ORDER BY** (y ascendente **ASC** o decreciente **DESC**):

```
SELECT DISTINCT escuela, sueldo FROM profesores ORDER by escuela ASC;
```

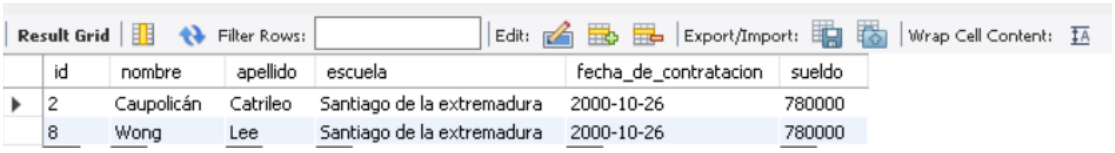


Result Grid | Filter Rows: | Export: | Wrap Cell Content:

escuela	sueldo
A-18 Abrazo de Maipu	67500
Bendito Corazón de María	8900000
E-34	300000
Gabriela Mistral	234000
Republica Popular China	780945
Sagrada Concepcion	3400000
Santiago de la extremadura	780000

Una construcción común en las consultas, son aquellas que usan la palabra clave WHERE (donde). Por ejemplo, si solo queremos saber los profesores que trabajan en cierta escuela podríamos ejecutar:

```
SELECT * FROM profesores WHERE escuela = 'Santiago de la extremadura';
```



Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

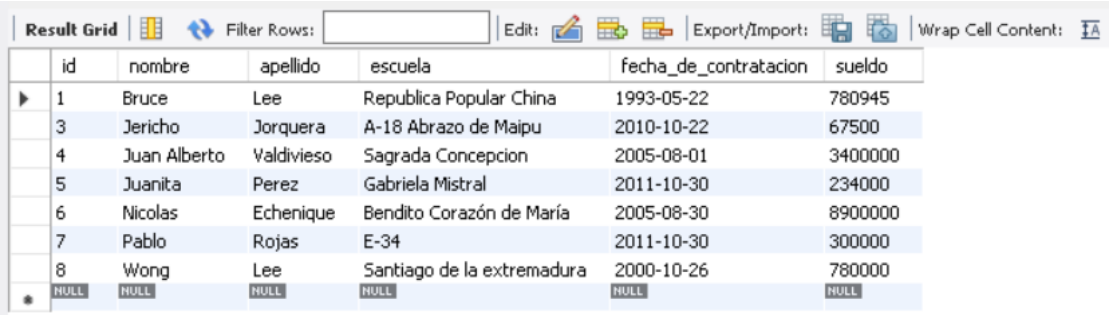
id	nombre	apellido	escuela	fecha_de_contratacion	sueldo
2	Caupolicán	Catrileo	Santiago de la extremadura	2000-10-26	780000
8	Wong	Lee	Santiago de la extremadura	2000-10-26	780000

Existen muchos Operadores que se pueden usar en conjunto con WHERE

- = Igual a - ej. **WHERE escuela = 'E-34'**
- <> o != Distinto de - ej. **WHERE escuela <> 'E-34'**
- > Mayor a - ej. **WHERE sueldo > 100000**
- < Menor que - ej. **WHERE sueldo < 1000000**
- >= Mayor o igual que - ej. **WHERE sueldo >= 100000**
- <= Menor o igual que - ej. **WHERE sueldo <= 1000000**
- BETWEEN Dentro de un rango - ej. **WHERE sueldo BETWEEN 100000 AND 600000**
- IN Coincidir con un grupo de valores - ej. **WHERE apellido IN ('Catrileo', 'Perez')**
- LIKE Coincidir con un patrón (distingue mayúsculas y minúsculas) - ej. **WHERE apellido LIKE 'Catri%'**
- NOT Niega una condición - ej. **WHERE apellido NOT LIKE 'catri%'**

Para ilustrar vamos a ejecutar una de estas instrucciones, el resto se pueden ratificar de forma similar.

```
SELECT * FROM profesores WHERE apellido NOT LIKE 'catri%';
```



Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	nombre	apellido	escuela	fecha_de_contratacion	sueldo
1	Bruce	Lee	Republica Popular China	1993-05-22	780945
3	Jericho	Jorquera	A-18 Abrazo de Maipu	2010-10-22	67500
4	Juan Alberto	Valdivieso	Sagrada Concepcion	2005-08-01	3400000
5	Juanita	Perez	Gabriela Mistral	2011-10-30	234000
6	Nicolas	Echenique	Bendito Corazón de María	2005-08-30	8900000
7	Pablo	Rojas	E-34	2011-10-30	300000
8	Wong	Lee	Santiago de la extremadura	2000-10-26	780000
NULL	NULL	NULL	NULL	NULL	NULL

2.5.- Utilización de funciones en las consultas

Al igual que con el número de operadores, el número de funciones integradas de MySQL es enorme. También vamos a explorar solo algunas de las más comunes, la documentación en línea tiene todo lo que puedas necesitar para tareas más específicas.

- **current_date:** Devuelve la fecha de hoy.
- **current_time:** Devuelve la hora actual (no se devuelve información de fecha).
- **current_timestamp:** Devuelve una marca de tiempo (fecha y hora) de la hora actual.
- **extract(tipo from campo):** Devuelve una parte de un campo especificado en el atributo “tipo”.
- **now():** Devuelve un campo especificado en el texto de una marca de tiempo determinada.

Las funciones antes descritas se pueden probar fácilmente usando SELECT

```
SELECT current_time;
```

Las funciones de cadena (string) se pueden usar para manipular valores de cadena de varias formas. Para estas funciones, cualquier entrada de cadena, incluidas las cadenas de texto, varchar y char, se considerará una entrada válida para la función.

- **lower(string):** Devuelve la cadena en minúsculas.
- **position(substring in string):** Devuelve la posición entera de una subcadena dentro de la cadena.
- **substring(string,from,[for]):** Extrae una subcadena de la cadena a partir de los dígitos especificados.

- **replace(string,from,to):** Reemplaza texto por texto en una cadena determinada.
- **upper(string):** Devuelve la cadena en mayúsculas.

Ejemplo 2:

```
SELECT replace('Monito', 'ito', 'oto');
```

2.6.- Consultas de selección con funciones de agrupación

SQL proporciona una serie de funciones que le permiten realizar recuentos, promedios y otras operaciones agregadas. Algunas de las funciones agregadas más comunes son:

- **avg(expression):** El promedio de todos los valores de entrada. Los valores de entrada deben ser uno de los tipos enteros.
- **count(*):** El número de valores de entrada.
- **count(expression):** El número de valores de entrada no nulos.
- **max(expression):** El valor máximo de expresión para todos los valores de entrada.
- **min(expression):** Los valores mínimos de expresión para todos los valores de entrada.
- **sum(expression):** La suma de la expresión para todos los valores de entrada no nulos.

Ejemplo 3:

Utilizando nuestra base datos de ejemplo de apartados anteriores, busquemos la cantidad de profesores que fueron contratados a partir del año 2010.

```

```

```
select
   count(DISTINCT escuela)
from profesores
WHERE fecha_de_contratacion > '2010-01-01';
```

Ejemplo 4:

O sumemos la cantidad de dinero mensual gastada en todos los profesores registrados

```
SELECT sum(sueldo) AS total FROM profesores;
```

2.7.- Consultando información relacionada en varias tablas

Continuando con nuestro camino para dominar los fundamentos de SQL, en secciones anteriores vimos lo básico de crear consultas con la instrucción SELECT. A continuación vamos a complicar un poco la cosa aprendiendo a realizar consultas en varias tablas de la base de datos al mismo tiempo.

Es habitual que queramos acceder a datos que se encuentran en más de una tabla y mostrar información mezclada de todas ellas como resultado de una consulta. Para ello tendremos que hacer combinaciones de columnas de tablas diferentes. En SQL es posible hacer esto especificando más de una tabla en la cláusula FROM de la instrucción SELECT.

Tenemos varias formas de obtener esta información. Una de ellas consiste en crear combinaciones que permiten mostrar columnas de diferentes tablas como si fuese una sola tabla, haciendo coincidir los valores de las columnas relacionadas.

Este último punto es muy importante, ya que si seleccionamos varias tablas y no hacemos coincidir los valores de las columnas relacionadas, obtendremos una gran duplicidad de filas, realizándose el producto cartesiano entre las filas de las diferentes tablas seleccionadas.

Pero antes de entrar de lleno en este tema, analizaremos los modelos de datos, y su función en la creación de consultas entre varias tablas.

2.8.- Qué es un modelo de datos y cómo leerlo

Modelos de Datos

Durante las décadas de 1960 y 1970, los desarrolladores crearon bases de datos que resolvieron el problema de los grupos repetidos de varias formas diferentes. Estos métodos dan como resultado lo que se denominan modelos para sistemas de bases de datos. La investigación realizada en IBM proporcionó gran parte de la base para estos modelos, que todavía se utilizan en la actualidad.

Un factor principal en los primeros diseños de sistemas de bases de datos fue la eficiencia. Una de las formas comunes de hacer que los sistemas sean más eficientes era imponer una longitud fija para los registros de la base de datos, o al menos tener un número fijo de elementos por registro (columnas por fila). Esto esencialmente evita el problema del grupo que se repite. Si es un programador en casi cualquier lenguaje de procedimiento, verá fácilmente que, en este caso, puede leer cada registro de una base de datos en una estructura C simple. La vida real rara vez es tan complaciente, por lo que debemos encontrar formas de lidiar con datos estructurados de manera inconveniente. Los diseñadores de sistemas de bases de datos hicieron esto mediante la introducción de diferentes tipos de bases de datos.

- Modelo de base de datos jerárquica.
- Modelo de base de datos de red.
- Modelo de base de datos relacional.

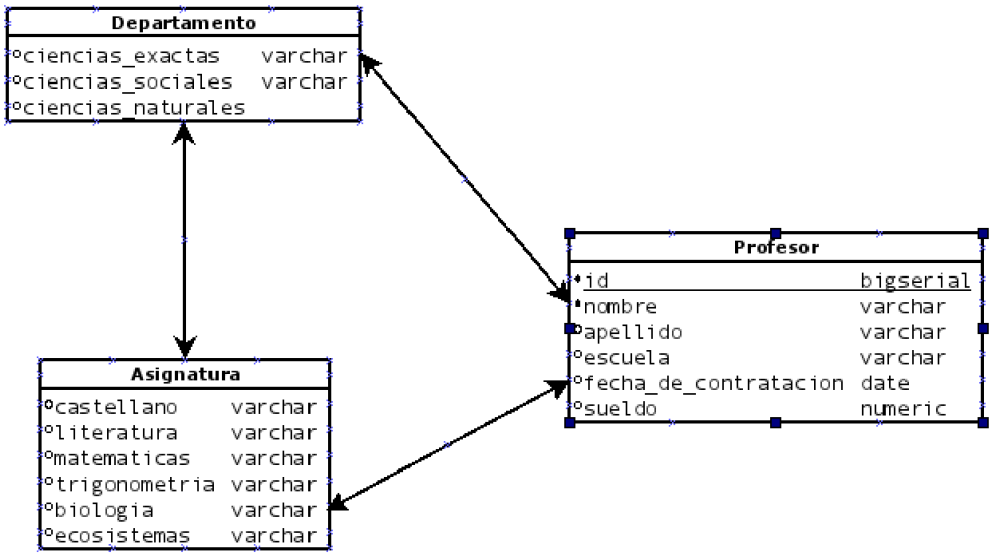
Los primeros dos escapan al alcance de este curso y requerirían probablemente cursos aparte especializados. Nuestro foco será, como hasta acá, el modelo relacional.

No es ningún misterio que una base de datos sea algo que almacena datos. Sin embargo, los modernos sistemas de administración de bases de datos relacionales (RDBMS) de hoy, como MySQL, PostgreSQL, SQL Server, Oracle, DB2 y otros, han ampliado esta función básica al agregar la capacidad de almacenar y administrar datos relacionales. Este es un concepto que merece cierta atención.

Entonces, ¿qué significan los datos relacionales? Es fácil ver que cada dato escrito en una base de datos del mundo real está relacionado de alguna manera con información ya existente. Los productos están relacionados con categorías y departamentos; los pedidos están relacionados con productos y clientes, etc. Una base de datos relacional mantiene su información almacenada en tablas de datos, pero también es consciente de las relaciones entre las tablas.

Estas tablas relacionadas forman la base de datos relacional, que se convierte en un objeto con un significado propio, en lugar de ser simplemente un grupo de tablas de datos no relacionadas. Se dice que los datos se convierten en información solo cuando les damos significado, y establecer relaciones con otros datos es un medio ideal para hacerlo.

La figura muestra una representación simple de tres tablas de datos.



Cuando se dice que dos tablas están relacionadas, esto significa más específicamente que los registros de esas tablas están relacionados. Entonces, si la tabla de productos está relacionada con la tabla de asignatura, esto se traduce en que cada registro de profesores está relacionado de alguna manera con uno de los registros de la tabla de asignatura.

Los diagramas como este se utilizan para decidir qué se debe almacenar en la base de datos. Una vez que sepa qué almacenar, el siguiente paso es decidir cómo se relacionan los datos enumerados, lo que conduce a la estructura física de la base de datos. El diagrama de la figura es solo un boceto que sirve para ir estructurando un modelo (que por supuesto puede sufrir modificaciones).

Entonces, ahora que conoce los datos que desea almacenar, pensemos en cómo se relacionan las tres partes entre sí. Además de saber que los registros de dos tablas están relacionados de alguna manera, también necesita saber el tipo de relación entre ellos. Veamos ahora más de cerca las diferentes formas en que se pueden relacionar dos tablas.

Datos relacionales y relaciones de tablas

Para continuar explorando el mundo de las bases de datos relacionales, analicemos más a fondo las tres tablas lógicas que hemos estado viendo

hasta ahora. Para hacer la vida más fácil, démosles nombres ahora: la tabla que contiene profesores es profesor (necesitamos renombrarla); la tabla que contiene departamentos es departamento; y el último es asignatura. ¡No hay sorpresas aquí! Afortunadamente, estas tablas implementan los tipos más comunes de relaciones que existen entre tablas, las relaciones de uno a muchos y de muchos a muchos, por lo que tienes la oportunidad de aprender sobre ellas.

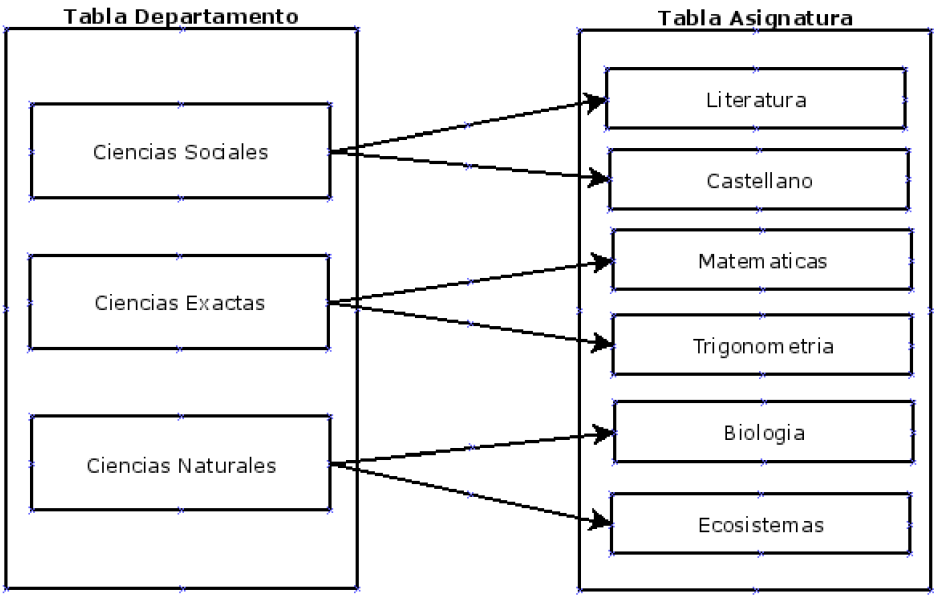
Renombremos nuestra tabla **profesores** a **profesor** para seguir las convenciones.

```
ALTER TABLE profesores RENAME TO profesor;
```

Nota: Existen algunas variaciones de estos dos tipos de relación que explicaremos, así como la relación uno a uno que es menos popular. En la relación uno a uno, cada fila de una tabla coincide exactamente con una fila de la otra. Por ejemplo, en una base de datos que permitiera que los pacientes fueran asignados a camas, ¡sería de esperar que hubiera una relación uno a uno entre los pacientes y las camas! Los sistemas de bases de datos no admiten la aplicación de este tipo de relación, porque tendría que agregar registros coincidentes en ambas tablas al mismo tiempo. Además, se pueden unir dos tablas con una relación uno a uno para formar una sola tabla.

Relaciones uno a muchos (one-to-many)

La relación de uno a varios ocurre cuando un registro de una tabla se puede asociar con varios registros de la tabla relacionada, pero no al revés. En nuestro caso, esto sucede para la relación departamento-asignatura. Un departamento específico puede contener cualquier número de categorías, pero cada categoría pertenece exactamente a un departamento. La figura siguiente representa mejor la relación de uno a varios entre departamentos y categorías.



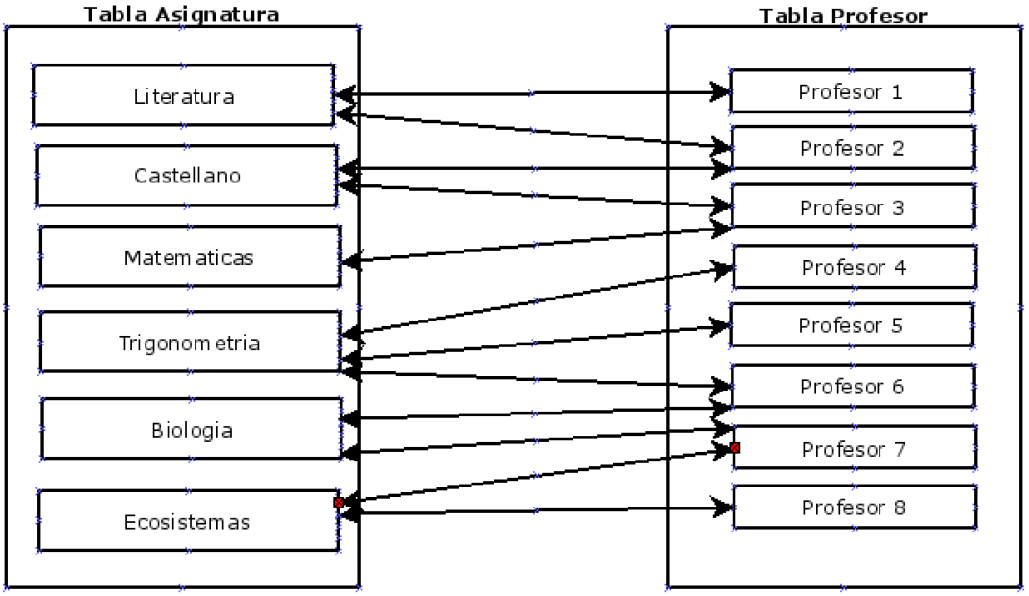
La relación de uno a muchos se implementa en la base de datos agregando una columna adicional en la tabla en el "lado muchos" de la relación, que hace referencia a la columna de ID de la tabla en el lado de la relación. En pocas palabras, en la tabla de asignaturas, tendrá una columna adicional (llamada **departamento_id**) que contendrá el ID del departamento al que pertenece la asignatura.

Relaciones de muchos a muchos (many-to-many)

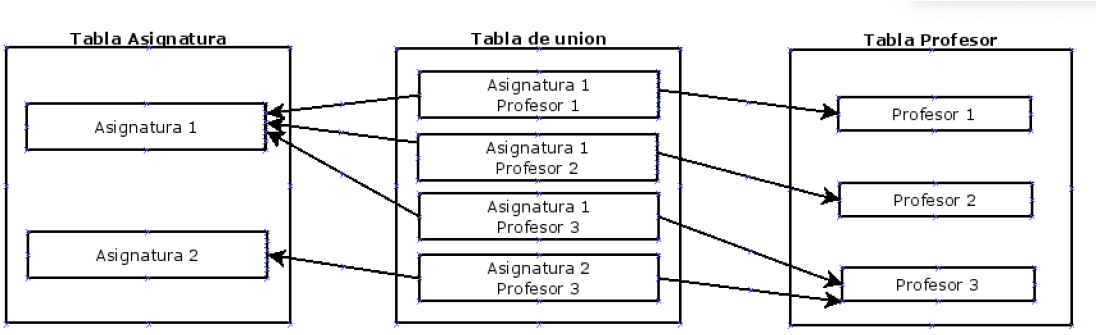
El otro tipo común de relación es la relación de muchos a muchos o varios a varios. Este tipo de relación se implementa cuando los registros en ambas tablas de la relación pueden tener múltiples registros coincidentes en la otra. En nuestro escenario, esto sucede para las tablas de profesores y asignaturas, porque sabemos que un profesor puede existir en más de una asignatura (un profesor con muchas asignaturas) y una asignatura puede tener más de un profesor (una asignatura con muchos profesores).

Esto sucede porque decidimos antes que un profesor podría estar en más de una asignatura. Si un profesor solo pudiera pertenecer a una asignatura, tendría otra relación de uno a varios, como la que existe entre departamentos y asignaturas (donde una asignatura no puede pertenecer a más de un departamento).

Representaremos esta relación con una imagen, al igual que como lo hicimos para la relación anterior:



Aunque lógicamente la relación de muchos a muchos ocurre entre dos tablas, las bases de datos no tienen los medios para implementar físicamente este tipo de relación usando solo dos tablas, por lo que hacemos trampa agregando una tercera tabla a la mezcla. Esta tercera tabla, llamada tabla de unión (también conocida como tabla de vinculación o tabla asociada) y dos relaciones de uno a muchos ayudarán a lograr la relación de muchos a muchos. La tabla de unión se utiliza para asociar profesores y asignaturas, sin restricciones sobre cuántos profesores pueden existir para una asignatura o cuántas asignaturas se pueden agregar a un profesor. La figura muestra el papel de la tabla de unión.



Tenga en cuenta que cada registro de la tabla de unión vincula una categoría con un profesor. Puede tener tantos registros como desee en la tabla de unión, vinculando cualquier asignatura a cualquier profesor. La tabla de unión contiene dos campos, cada uno de los cuales hace referencia a la llave primaria de una de las dos tablas vinculadas. En nuestro caso, la tabla de unión contendrá dos campos: un campo **asignatura_id** y un campo **profesor_id**.

Cada registro de la tabla de unión constará de un par de ID de profesor y asignatura (**profesor_id**, **asignatura_id**), que se utilizará para asociar un profesor en particular con una asignatura en particular. Al agregar más registros a la tabla **profesor_asignatura**, puede asociar un profesor con más asignaturas o una asignatura con más profesores, implementando eficazmente la relación de muchos a muchos.

Debido a que la relación de muchos a muchos se implementa utilizando una tercera tabla que hace la conexión entre las tablas vinculadas, no es necesario agregar campos adicionales a las tablas relacionadas de la forma en que agregamos el **departamento_id** a la tabla de asignatura para implementar la relación de uno a muchos.

No existe una convención de nomenclatura definitiva para usar en la tabla de unión. La mayoría de las veces está bien unir los nombres de las dos tablas vinculadas; en este caso, la tabla de unión se llama asignatura_profesor.

Aplicación de relaciones de tabla mediante llaves externas

Las relaciones entre tablas se pueden hacer cumplir físicamente en la base de datos utilizando restricciones **FOREIGN KEY**, o simplemente llaves externas o claves foráneas.

Ya aprendimos acerca de la restricción **PRIMARY KEY** o llave principal. Las llaves externas, por otro lado, ocurren entre dos tablas: la tabla en la que se define la llave externa (la tabla de referencia) y la tabla a la que hace referencia la clave externa (la tabla referenciada).

Una llave externa es una columna o combinación de columnas que se utiliza para imponer un vínculo entre los datos de dos tablas (generalmente representa una relación de uno a varios). Las llaves

externas se utilizan como método para garantizar la integridad de los datos y para establecer una relación entre tablas.
Para hacer cumplir la integridad de la base de datos, las llaves externas, aplican ciertas restricciones. A diferencia de las restricciones **PRIMARY KEY** y **UNIQUE** que aplican restricciones a una sola tabla, la restricción **FOREIGN KEY** aplica restricciones tanto en las tablas de referencia como en las referenciadas. Por ejemplo, si aplica la relación de uno a varios entre las tablas de departamento y de categoría mediante una restricción FOREIGN KEY, la base de datos incluirá esta relación como parte de su integridad. No le permitirá agregar una categoría a un departamento inexistente, ni le permitirá eliminar un departamento si hay categorías que pertenecen a él.

Creación y llenado de nuevas tablas de datos

Ya renombramos la tabla profesores como **profesor** para seguir las convenciones (no usar plurales). Es hora de completar el resto de nuestras tablas.

```
CREATE TABLE departamento (
departamento_id int,
nombre VARCHAR(100),
descripcion VARCHAR(1000),
PRIMARY KEY (departamento_id)
);

INSERT INTO departamento (departamento_id, nombre, descripcion)
VALUES
(1, 'Ciencias Sociales', 'Ramas de la ciencia relacionadas con la sociedad y el comportamiento humano.'),
(2, 'Ciencias Exactas', 'Disciplinas que se basan en la observación y experimentación para crear conocimientos y cuyos contenidos pueden sistematizarse a partir del lenguaje matemático.'),
(3, 'Ciencias Naturales', 'Ciencias que tienen por objeto el estudio de la naturaleza, siguiendo la modalidad del método científico conocida como método empírico-analítico.');

CREATE TABLE asignatura
(
```

```
asignatura_id int NOT NULL,
departamento_id int NOT NULL,
nombre VARCHAR(100) NOT NULL,
descripcion VARCHAR(1000),
PRIMARY KEY (asignatura_id)
);
```

Insertamos registros en la tabla de asignaturas:

```
INSERT INTO asignatura (
asignatura_id, departamento_id, nombre, descripcion)
VALUES
(1, 1, 'Literatura', 'Arte de la expresión verbal'),
(2, 1, 'Castellano', 'Lengua romance procedente del latín hablado'),
(3, 2, 'Matemáticas', 'Ciencia formal que, partiendo de axiomas y siguiendo el razonamiento lógico, estudia las propiedades y relaciones entre entidades abstractas como números, figuras geométricas, iconos, glifos, o símbolos en general'),
(4, 2, 'Trigonometría', 'Rama de la matemática, cuyo significado etimológico es la medición de los triángulos'),
(5, 3, 'Biología', 'Rama de la ciencia que estudia los procesos naturales de los organismos vivos, considerando su anatomía, fisiología, evolución, desarrollo, distribución y relaciones'),
(6, 3, 'Ecosistema', 'Sistema biológico constituido por una comunidad de organismos vivos (biocenosis) y el medio físico donde se relacionan (biotopo)');
```

Si por alguna razón desea borrar alguna tabla, por ejemplo, con el fin de crearla de nuevo para experimentar, puede ejecutar:

```
DROP TABLE <nombre de la tabla>;
```

Ahora llenaremos algunas filas de nuestra tabla asignatura:

```
CREATE TABLE profesor_asignatura (
profesor_id INT NOT NULL,
asignatura_id INT NOT NULL,
PRIMARY KEY (profesor_id, asignatura_id)
);

INSERT INTO profesor_asignatura (profesor_id, asignatura_id) VALUES (1, 1), (1, 2), (2, 3), (2, 4), (3, 5), (3, 6), (4, 6), (5, 4), (6, 1), (6, 2), (6, 3), (6, 4), (6, 5), (6, 6), (7, 3), (7, 4), (8, 1);
```

Ahora realicemos cambios en nuestra tabla profesor para continuar con las convenciones de nombres que hemos adoptado.

```
ALTER TABLE profesor RENAME COLUMN id TO profesor_id;
```

Agregamos la llave primaria

```
ALTER TABLE profesor ADD PRIMARY KEY (profesor_id);
```

2.9.- Consultas de selección con tablas relacionadas

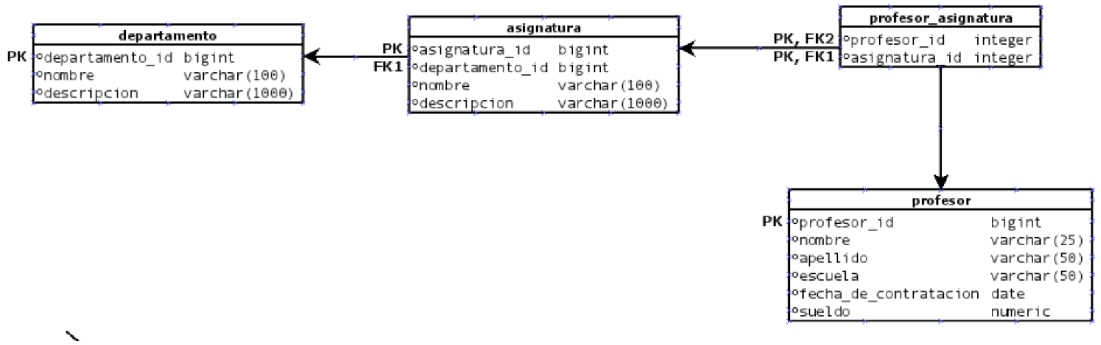
Ahora tenemos un esquema de tablas que podemos relacionar y consultar según nuestras necesidades, por ejemplo, si queremos saber a qué departamento pertenece la asignatura 'Castellano'.

```
SELECT departamento.nombre FROM departamento
INNER JOIN asignatura
ON asignatura.departamento_id = departamento.departamento_id
WHERE asignatura.nombre = 'Castellano';
```

Esta consulta entrega lo siguiente:

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

	nombre
▶	Ciencias Sociales



O si queremos saber en qué departamento(s) trabaja el profesor Pablo Rojas

```
SELECT d.nombre
FROM departamento d
INNER JOIN asignatura a
ON a.departamento_id = d.departamento_id
INNER JOIN profesor_asignatura pa
ON pa.asignatura_id = a.asignatura_id
INNER JOIN profesor p
ON p.profesor_id = pa.profesor_id
WHERE p.nombre = 'Pablo' AND p.apellido = 'Rojas';
```

El resultado obtenido será:

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

	nombre
▶	Ciencias Exactas
	Ciencias Exactas

Puedes comprobar el resultado siguiendo las queries y las tablas de forma visual/manual.

2.10.- Integridad referencial

La integridad referencial se refiere a la precisión y consistencia de los datos dentro de una relación.

En las relaciones, los datos están vinculados entre dos o más tablas. Esto se logra haciendo que la llave externa (en la tabla asociada) haga referencia a un valor de llave principal (en la tabla principal). Debido a esto, debemos asegurarnos de que los datos de ambos lados de la relación permanezcan intactos.

Por lo tanto, la integridad referencial requiere que, siempre que se use un valor de llave externa, debe hacer referencia a una llave primaria válida existente en la tabla principal.

En nuestras tablas no hemos observado que, por ejemplo, exista una asignatura que haga referencia al departamento_id = 4, solo tenemos 3 departamentos: Ciencias Sociales, Ciencias Exactas y Ciencias Naturales. Referenciar un departamento con id = 4 generaría un récord huérfano.

La falta de integridad referencial en una base de datos puede llevar a que se devuelvan datos incompletos, normalmente sin indicación de error. Esto podría resultar en la "pérdida" de registros en la base de datos, porque nunca se devuelven en consultas o informes.

También podría dar lugar a que aparezcan resultados extraños en informes (como productos sin una empresa asociada). O peor aún, podría resultar en que los clientes no reciban los productos por los que pagaron o que un paciente del hospital no reciba el tratamiento correcto, o un equipo de socorro en casos de desastre que no reciba los suministros o la información correctos.

2.11.- Queries anidadas

Una subconsulta a veces se anida dentro de otra consulta. Normalmente, se utiliza para un cálculo o una prueba lógica que proporciona un valor o

un conjunto de datos que se pasarán a la parte principal de la consulta. Su sintaxis no es inusual: simplemente encerramos la subconsulta entre paréntesis y la usamos donde sea necesario. Por ejemplo, podemos escribir una subconsulta que devuelve varias filas y trata los resultados como una tabla en la cláusula FROM de la consulta principal. O podemos crear una subconsulta escalar que devuelva un solo valor y usarlo como parte de una expresión para filtrar filas a través de las cláusulas WHERE, IN y HAVING. Esos son los usos más comunes de las subconsultas.

Como ejemplo una vez más buscaremos en qué departamento(s) trabaja el profesor Pablo Rojas:

```
SELECT d.nombre
FROM departamento d
WHERE d.departamento_id IN (
SELECT departamento_id FROM asignatura a
WHERE a.asignatura_id IN (
SELECT asignatura_id FROM profesor_asignatura pa
WHERE pa.profesor_id IN (
SELECT p.profesor_id FROM profesor p
WHERE p.nombre = 'Pablo' AND p.apellido = 'Rojas'
)
)
);
```

El resultado obtenido es:

2.12.- Queries con distintos tipos de JOIN (INNER, LEFT, OUTER)

En lo que se refiere a **JOIN(s)**, hasta ahora hemos utilizado **INNER JOIN** en nuestras consultas. Ahora veremos las distintas opciones de esta cláusula.

Hay más de una forma de unir tablas en SQL, y el tipo de unión que usará depende de cómo desee recuperar los datos. La siguiente lista describe los diferentes tipos de combinaciones. Al revisar cada uno, es útil pensar en dos tablas una al lado de la otra, una a la izquierda de la palabra clave JOIN y la otra a la derecha. A continuación de la lista, se muestra un ejemplo basado en datos de cada combinación:

- **JOIN** Devuelve filas de ambas tablas donde se encuentran valores coincidentes en las columnas unidas de ambas tablas. La sintaxis alternativa es **INNER JOIN**.
- **LEFT JOIN** Devuelve todas las filas de la tabla de la izquierda más las filas que coinciden con los valores de la columna unida de la tabla de la derecha. Cuando una fila de la tabla de la izquierda no tiene una coincidencia en la tabla de la derecha, el resultado no muestra valores de la tabla de la derecha.
- **RIGHT JOIN** Devuelve todas las filas de la tabla derecha más las filas que coinciden con los valores clave en la columna clave de la tabla izquierda. Cuando una fila de la tabla de la derecha no tiene una coincidencia en la tabla de la izquierda, el resultado no muestra valores de la tabla de la izquierda.
- **FULL JOIN** Devuelve cada fila de ambas tablas y coincide con las filas; luego une las filas donde coinciden los valores de las columnas unidas. Si no hay ninguna coincidencia para un valor en la tabla izquierda o derecha, el resultado de la consulta contiene una fila vacía para la otra tabla. En MySQL no existe este comando, no así en otros motores de bases de datos.

Para explicar de forma más clara vamos a "ensuciar" un poco nuestra base de datos e insertar una fila adicional y sin relaciones en la tabla departamento y otra en la tabla asignatura (violando la integridad referencial solo para ejemplificar).

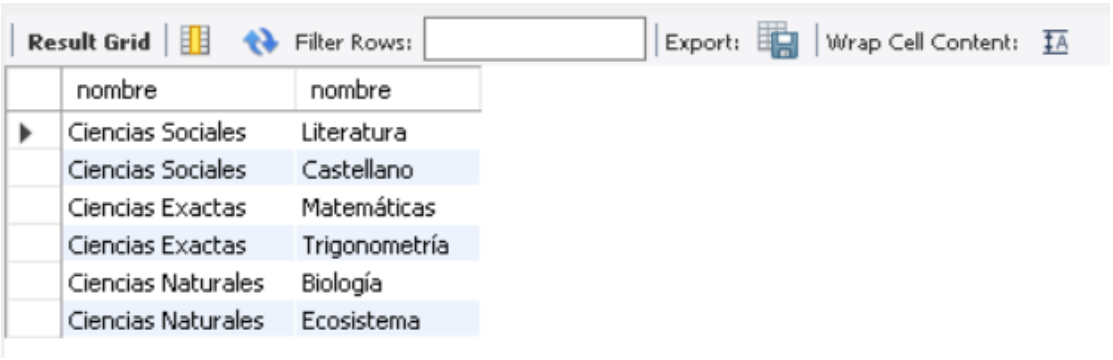
```
INSERT INTO departamento (departamento_id, nombre, descripcion)
VALUES (4, 'Ciencias Especiales', 'Ramas de la ciencia que son especiales.');

INSERT INTO asignatura (asignatura_id, departamento_id, nombre,
descripcion) VALUES (7, 10, 'Especial', 'Asignatura Especial');
```

Usaremos un par de tablas de nuestra base de datos para demostrar alternativas.

Hasta ahora hemos usado **JOIN (INNER JOIN)**; este tipo puede ser pensado como la "Intersección" de ambas tablas. Ejemplo para recordar que asignaturas corresponden a que departamento podemos ejecutar:

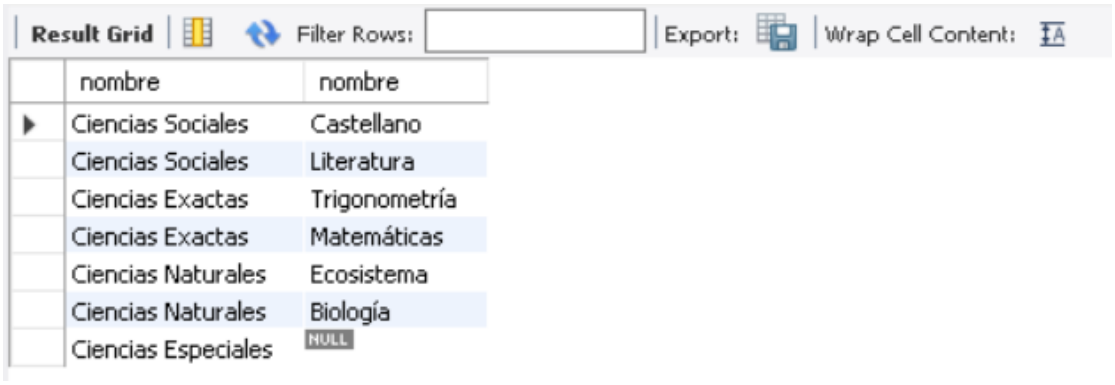
```
SELECT d.nombre, a.nombre FROM
departamento d JOIN asignatura a
ON d.departamento_id = a.departamento_id;
```



nombre	nombre
Ciencias Sociales	Literatura
Ciencias Sociales	Castellano
Ciencias Exactas	Matemáticas
Ciencias Exactas	Trigonometría
Ciencias Naturales	Biología
Ciencias Naturales	Ecosistema

LEFT JOIN y RIGHT JOIN: A diferencia de **JOIN**, las palabras clave **LEFT JOIN** y **RIGHT JOIN** devuelven todas las filas de una tabla y muestran filas en blanco de la otra tabla si no se encuentran valores coincidentes en las columnas unidas. Primero veamos **LEFT JOIN** en acción, esta puede ser pensada como la "Intersección más lo que está a la izquierda".

```
SELECT d.nombre, a.nombre FROM
departamento d LEFT JOIN asignatura a
ON d.departamento_id = a.departamento_id;
```



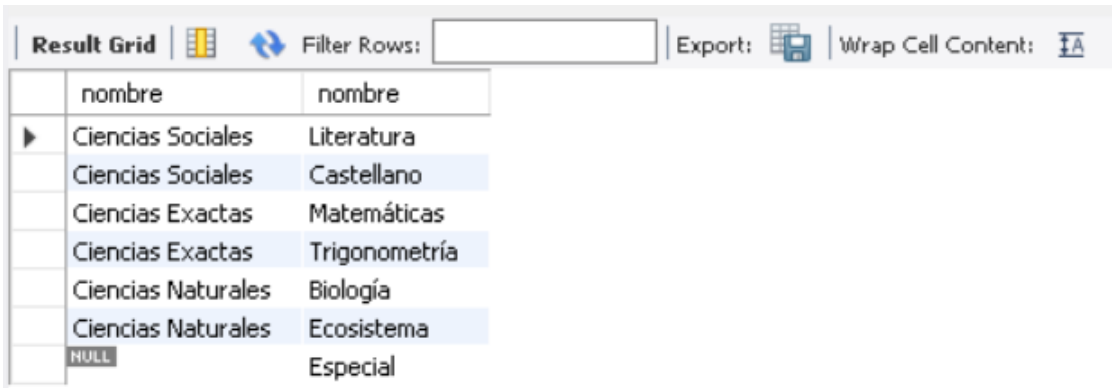
nombre	nombre
Ciencias Sociales	Castellano
Ciencias Sociales	Literatura
Ciencias Exactas	Trigonometría
Ciencias Exactas	Matemáticas
Ciencias Naturales	Ecosistema
Ciencias Naturales	Biología
Ciencias Especiales	NULL

Nota: Vemos que el departamento dummy Ciencias Especiales también aparece en la consulta.

RIGHT JOIN es similar solo que este puede ser pensado como la "Intersección más lo que está a la derecha".

```
SELECT d.nombre, a.nombre FROM
departamento d RIGHT JOIN asignatura a
ON d.departamento_id = a.departamento_id;
```

Obtenemos:

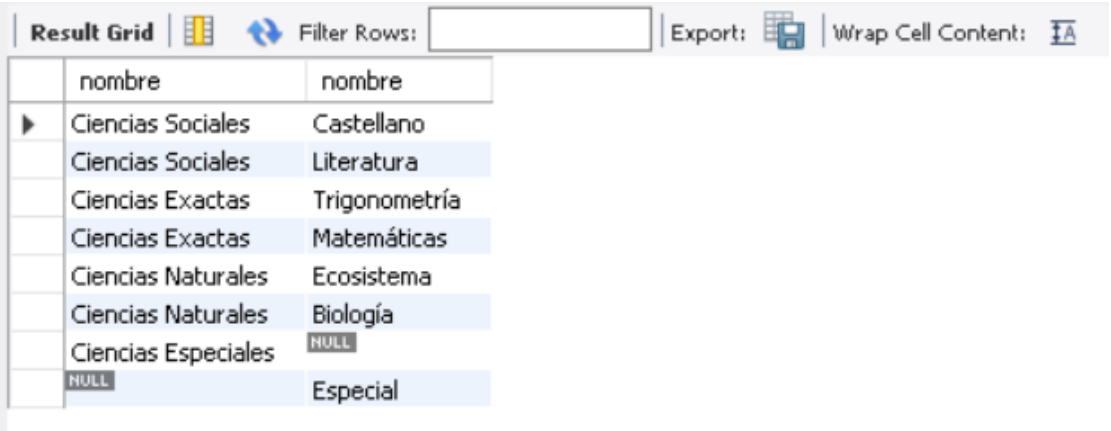


nombre	nombre
Ciencias Sociales	Literatura
Ciencias Sociales	Castellano
Ciencias Exactas	Matemáticas
Ciencias Exactas	Trigonometría
Ciencias Naturales	Biología
Ciencias Naturales	Ecosistema
NULL	Especial

Finalmente, tal como se dijo antes, FULL OUTER JOIN no existe en el motor de bases de datos en estudio. Si se desea simular en una consulta, se puede usar un comando UNION que junte el resultado de un LEFT JOIN con el resultado de RIGHT JOIN.

```
SELECT d.nombre, a.nombre FROM
departamento d LEFT JOIN asignatura a
ON d.departamento_id = a.departamento_id
UNION
SELECT d.nombre, a.nombre FROM
departamento d RIGHT JOIN asignatura a
ON d.departamento_id = a.departamento_id;
```

Con la consulta anterior se obtiene lo siguiente:



nombre	nombre
Ciencias Sociales	Castellano
Ciencias Sociales	Literatura
Ciencias Exactas	Trigonometría
Ciencias Exactas	Matemáticas
Ciencias Naturales	Ecosistema
Ciencias Naturales	Biología
Ciencias Especiales	NULL
NULL	Especial

Anexo: Referencias

1.- ¿Qué es una base de datos?
Referencia: https://www.hn.cl/blog/para-que-sirven-la-bases-de-datos/

2.- MySQL: Consulta SELECT
Referencia:
https://www.anerbarrena.com/mysql-select-consultas-base-datos-5426/

3.- Funciones en MySQL
Referencia:
https://dev.mysql.com/doc/refman/8.0/en/date-and-time-functions.html

4.- Funciones de agregación en MySQL
Referencia:
https://www.campusmvp.es/recursos/post/Fundamentos-de-SQL-Agrupaciones-y-funciones-de-agregacion.aspx

5.- ¿Qué es un modelo de datos?
Referencia:
https://sites.google.com/site/modelamientodebasesdedatos/mysql-workbench

6.- Querys anidadas en MySQL
Referencia: https://www.javatpoint.com/mysql-join

7.- Procedimientos almacenados en MySQL
Referencia:
https://www.neoguias.com/procedimientos-almacenados-mysql/

8.- Procedimientos, excepciones y cursores
Referencia:
http://tisbddocs.dlsi.ua.es/inicio/sql/practicas-sql-espanol/procedimientos

9.- Instalación de MySQL en Windows 10
https://www.solvetic.com/tutoriales/article/9129-instalar-mysql-8-0-22-server-en-windows-10/

https://desarrolloweb.com/articulos/2408.php

https://www.todopostgresql.com/como-crear-base-de-datos-en-postgresql/#:~:text=En%20PostgreSQL%20existen%20dos%20formas,de%20la%20instalaci%C3%B3n%20de%20PostgreSQL.

https://www.todopostgresql.com/crear-usuarios-en-postgresql/

https://www.linuxito.com/programacion/337-como-listar-tablas-y-bases-de-datos-en-postgres