SQL

TABLA DE CONTENIDO

1. INTRODUCCIÓN

Veremos los conceptos básicos de SQL y del modelo relacional.

SQL (Structured Query Language) es un lenguaje utilizado para consultar bases de datos relacionales u obtener datos desde una base de datos.

Los **datos** son una colección de hechos en forma de palabras, números e incluso imágenes. Los datos son utilizados prácticamente en todos lados. Su banco almacena datos acerca suyo, como su nombre, dirección, teléfono, etc. Su tarjeta de crédito y su cuente de paypal también almacenan datos suyos.

Los datos son importantes, por lo que necesitan:

- Estar asegurados.
- Ser almacenados.
- Poder ser accedidos rápidamente.

La respuesta para lograr lo anterior es una base de datos.

Una **base de datos** :

- Es un repositorio de datos.
- Es un programa que almacena datos.
- Provee funcionalidades para agregar, modificar y consultar datos.

Hay diferentes clases de bases de datos con diferentes requerimientos.

Los datos pueden ser almacenados de forma distinta. Cuando se almacenan en forma tabular, los datos son organizados en tablas, como columnas y filas. Esa es una base de datos relacional. Las columnas tienen propiedades acerca de cada elemento, como nombre, apellido, mail, etc. Una tabla es una colección de ítems relacionados, como una lista de empleados o de autores. En una base relacional, pueden formarse relaciones entre tablas.

Un conjunto de herramientas de software para los datos en una base de datos se llama **sistema de administración de base de datos DBMS (Database Management System).**

Los términos base de datos, servidor de base de datos, sistema de base de datos y DBMS a menudo se usan de forma intercambiable.

Para bases relacionales, el DBMS es llamado **RDBMS** (Relational DBMS).

El RDBMS controla el acceso a los datos, su organización y alamcenamiento.

Ejemplos de bases relacionales son:

- MySQL
- Oracle Databases
- DB2 Warehouse on Cloud

Los comandos básicos son:

- Crear una tabla.
- Insertar datos en una tabla.
- Seleccionar datos de una tabla.
- Actualizar datos en una tabla.
- Borrar datos de una tabla.

2. INTRODUCCIÓN A BASES DE DATOS EN LA NUBE

Una **base de datos en la nube** es un servicio de base de datos construido y accedido a través de una plataforma en la nube. Brinda muchas de las funcionalidades de las bases de datos tradicionales con el agregado de la flexibilidad de la computación en la nube.

Los usuarios instalan software SW en una infraestructura en la nube para implementar su base de datos.

Las ventajas de usar bases de datos en la nube son:

- Fácil de usar
  - Los usuarios pueden acceder a su base de datos virtualmente desde cualquier lugar, utilizando la API de los vendors o una interfaz web.
- Escalabilidad
  - Pueden expandir sus necesidades de almacenamiento en tiempo de ejecución para adaptarse a los cambios en las necesidades; las organizaciones sólo pagan por lo que usan.
- Disaster Recovery
  - Los datos son respaldados en servidores remotos.

Ejemplos de bases de datos en la nube son:

- IBM Db2 on Cloud
- Compose for PostgreSQL
- Oracle Database Cloud
- Microsoft Azure Cloud
- Amazon Relational Database Services

Ellas pueden ejecutarse en la nube, ya sea en una máquina virtual o como servicio dependiendo del vendor.

Los **servicios de bases de datos**

- Son abstracciones lógicas para administrar cargas de trabajo en una base de datos.
- Cada servicio representa una carga de trabajo con atributos comunes, umbrales de nivel de servicio y prioridades.
  - El agrupamiento se basa en atributos de trabajo que pueden incluir:

- La función de la aplicación a ser utilizada.
- La prioridad de la ejecución para la función de aplicación.
- La clase del job a ser administrado.
-

Una **instancia de la base de datos** en la nube opera como un servicio que maneja todas las solicitudes de la aplicación para trabajar con los datos y cualquiera de las bases de datos administradas por esa instancia.

La instancia de servicio de la base de datos es el objetivo de la solicitud de conexión desde las aplicaciones. Aquí usaremos Python. Cuando una conexión es completada, el código Python envía sentencias SQL a través de la conexión a la base de datos. La instancia de la base de datos resuelve las sentencias SQL en operaciones sobre los datos y objetos en la base de datos. Cualquier dato recuperado es retornado a la aplicación.

Los detalles de una conexión a la base de datos suelen incluir:

- host name
- port number
- database name
- user ID
- password

3. SENTENCIAS BÁSICAS

3.1. CREATE

La mayoría de las personas que usan bases de datos utilizan 5 sentencias simples:

- Crear una tabla.
- Insertar datos.
- Seleccionar datos.
- Actualizar datos.
- Borrar datos.

Estas sentencias caen en 2 categorías:

- **Data Definition Language DDL** (lenguaje de definición de datos).
  - Se usan para definir, cambiar o borrar datos.
- **Data Manipulation Language DML** (lenguaje de manipulación de datos).
  - Se usan para leer y modificar datos.

Basados en las entidades book y author crearemos tablas usando el nombre de la entidad. Los atributos de entidad serán las columnas de la tabla. Deben asignarse tipos de datos a los atributos (char, varchar….). El atributo author_id será la clave primaria, que asegura no puedan existir valores duplicados.
Una tabla se crea con la sentencia:

CREATE TABLE <nombre tabla>

CREATE TABLE es una sentencia DDL. Las sentencias DDL se utilizan para crear objetos de base de datos.

Ejemplo 1.
Crear una tabla “COUNTRY”:

create table COUNTRY(

```
        ID int NOT NULL,
         CCODE char(2),
         NAME varchar(60),
         PRIMARY KEY(ID)
);
```

Ejemplo 2.

Borrar la tabla COUNTRY.

```
drop table COUNTRY;
```

3.2. SELECT

Para ver los datos, usamos la sentencia SELECT, que es del tipo DML.

La sentencia SELECT es llamada consulta, y la salida que obtenemos, conjunto resultado.

Pueden seleccionarse todos los registroso o sólo algunos de ellos si se utiliza la cláusula WHERE.

Pueden seleccionarse todas las consultas, o sólo algunas.

Ejemplo 1.

Seleccionar los países con id<5

```
select * from COUNTRY where ID < 5;
```

3.3. COUNT, DISTINCT y LIMIT

COUNT es una función incorporada que recupera el número de filas que coinciden con un criterio. Por ejemplo, para obtener el número total de filas en una tabla dada escribimos:

```
select COUNT(*) from <nombre de la tabla>
```

Si por ejemplo tenemos una tabla llamada MEDALS que tiene una columna COUNTRY y queremos el número de filas para el cual el participante es de CANADA escribimos:

select COUNT(COUNTRY) from MEDALS where COUNTRY=’CANADA’.

DISTINCT se usa para quitar valores duplicados de un conjunto resultado. Por ejemplo, para recuperar valores únicos en una columna escribimos:

select DISTINCT <nombre de columna> from <nombre de la tabla>.

En la tabla MEDALS un país puede recibir múltiples medallas de oro. Para recuperar la lista de países que recibieron medallas escribimos:

select DISTINCT COUNTRY from MEDALS where MEDALTYPE=’GOLD’

LIMIT se usa para restringir el número de filas recuperadas de la base de datos. Por ejemplo, recuperar los primeros 10 registros en una tabla:

select * from <nombre de la tabla> LIMIT 10

Por ejemplo, recuperemos sólo 5 registros en la tabla MEDALS para el año 2018:

select * from MEDALS where YEAR=2018 LIMIT 5

3.4. INSERT

Para insertar datos en una tabla, usamos la sentencia INSERT, que es del tipo DML.
Las datos en las tablas pueden ingresarse una fila a la vez o varias filas a la vez.

3.5. UPDATE Y DELETE

Para actualizar datos se usa la sentencia UPDATE, que es del tipo DML.

La sintaxis es:
UPDATE [TableName] SET [ColumnName] = [Value] WHERE [Condition]

Las filas se borran con la sentencia DELETE, que es del tipo DML.
La sintaxis es:

DELETE FROM [TableName] WHERE [Condition]

Las filas a ser removidas serán especificadas en la condición WHERE. Si no especifica la clásula WHERE, todas las filas serán borradas.

4. MODELOS DE INFORMACIÓN Y DE DATOS

La siguiente figura ilustra la relación entre modelo información y de datos.

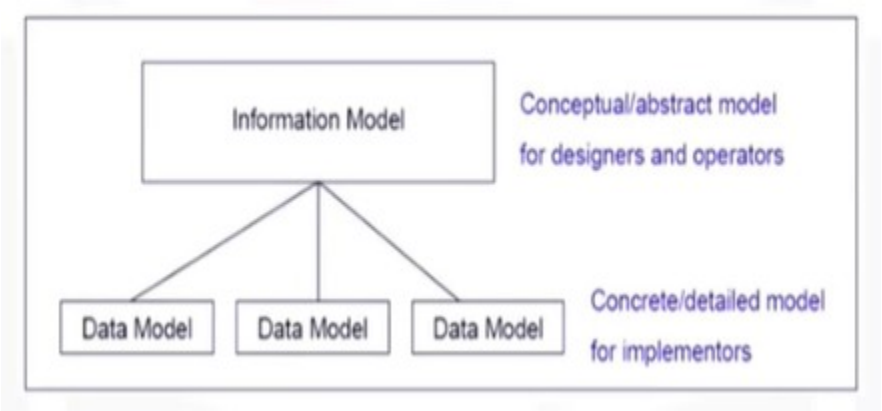


Figura. Blueprint de un sistema de administración de bases de datos.

Un **modelo de información** es una representación formal de entidades que incluye sus propiedades, relaciones y las operaciones que pueden ser realizados sobre ellos. Las entidades a ser modeladas, pueden ser del mundo real, como una biblioteca.

Los modelos de datos e información son diferentes y tienen diferentes propósitos.

Un **modelo de información** se ubica en un nivel conceptual y define relaciones entre objetos. Los modelos de datos son definidos en un nivel más concreto, más específico e incluye detalles. Un modelo de datos es el blueprint de cualquier sistema de bases de datos.

Hay diferentes tipos de modelos de información:

- Jerárquico
- Relacional
- Entidad-Relación

El **modelo jerárquico** organiza los datos en una estructura de árbol.

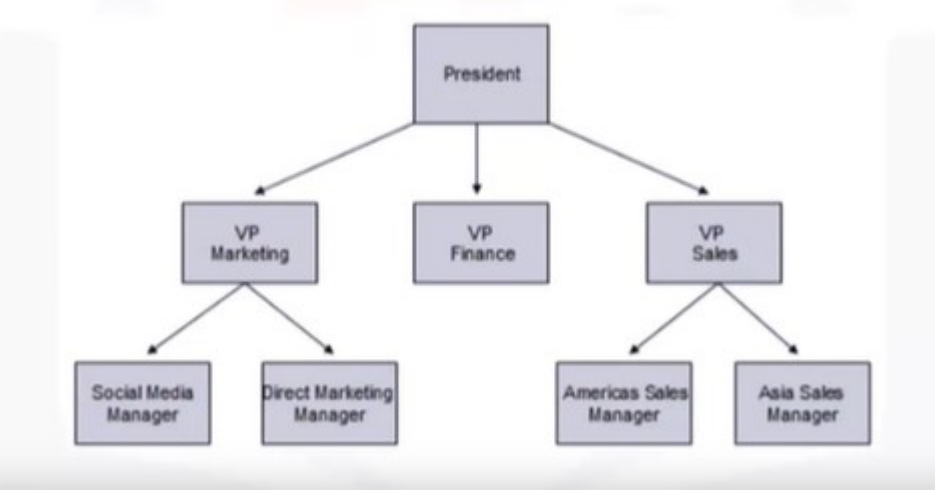


Figura. Modelo jerárquico.

La raíz del árbol es el nodo padre, seguido por los nodos hijos. Un nodo hijo no puede tener más de un padre, pero un padre puede tener muchos nodos hijos.

El **modelo relacional** permite independencia de los datos. Los datos son almacenados en tablas. Esto provee independencia de datos lógica y física e independencia de almacenamiento física.

También tenemos el **modelo entidad-relación ER**. Usando la base de datos simplificada de una biblioteca como ejemplo, la siguiente figura muestra un diagrama entidad relación (ERD) que representa entidades llamadas tablas y sus relaciones.

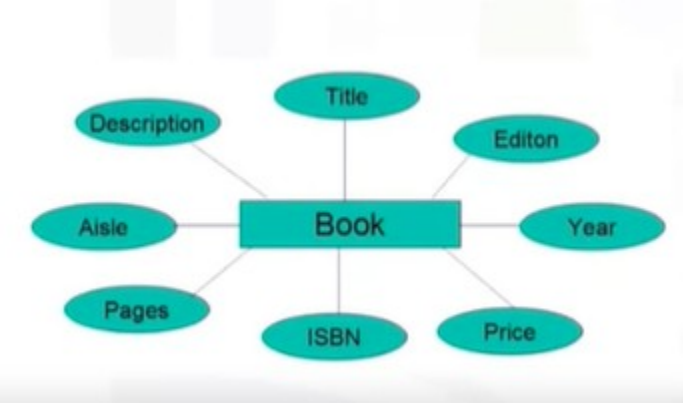


Figura. Modelo entidad-relación.

Tenemos autores que escriben libros, gente que los lleva prestados, varias copias de cada libro, etc.

Un modelo ER:

- Propone pensar una base de datos como una colección de entidades.

- Es utilizado como una herramienta para desarrollar bases relacionales.
- Las entidades son objetos que existen independientemente de cualquier otra entidad en la base de datos.

Es simple convertir una diagrama ER en una colección de tablas.

- Los bloques de construcción de una diagrama ER son las entidades y los atributos.
- Las entidades tienen atributos, que son los elementos de datos que caracterizan a la entidad.
    - Los atributos nos dicen más acerca de la entidad.
- En un diagrama ER, una entidad se dibuja como un rectángulo, y los atributos como óvalos. Las entidades pueden ser sustantivos, personas, lugares o cosas.

Usando el modelo de la biblioteca, un libro (book) es un ejemplo de una entidad. Los atributos serían el título, el año en el que fue escrito, etc. Los atributos se conectan a exactamente una entidad.

La entidad se convierte en una tabla en la base de datos y los atributos en las columnas de la tabla.

Continuando con el modelo del la biblioteca, los libros son escritos por autores (author), book es una entidad y author es una entidad.

La entidad author tiene atributos, como el apellido (last name), ciudad (city), etc. Incluye un “author ID” para identificar unívocamente al autor.

La entidad autor se vuelve una tabla en la base de datos y sus atributos sus columnas.

5. TIPOS DE RELACIONES

Veremos los tipos de relaciones entre entidades.

Los bloques de construcción de una relación son:

- Entidades
- Conjuntos de relaciones
- Crows foot notation

Las entidades se representan mediante rectángulos.

Los conjuntos de relaciones se representan mediante un diamante, con líneas conectando las entidades asociadas.

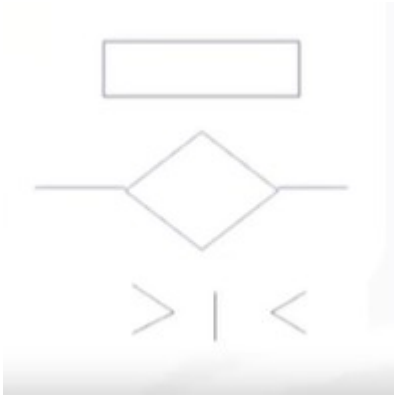


Figura.

Se usan diferentes técnicas para representar relaciones. Aquí usaremos las crows foot notations. Algunos son el símbolo de mayor, el de menor y una línea vertical.

Un ejemplo de diagrama entida-relación ER para una entidad “book” podría verse así:

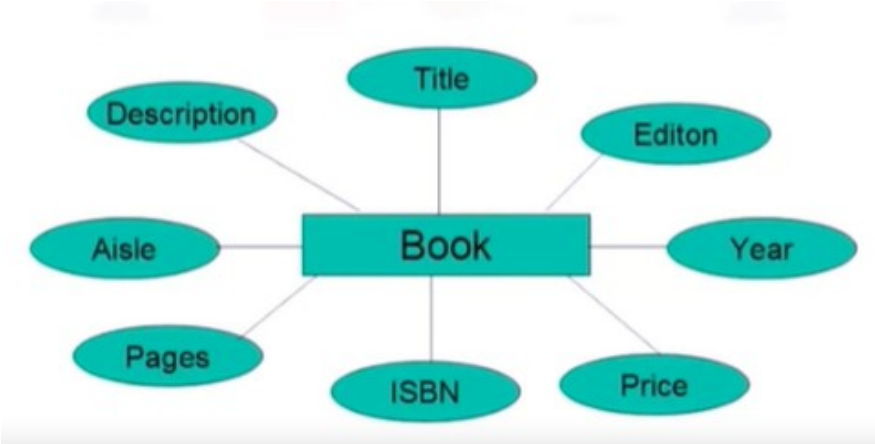


Figura. Modelo entidad-relación para Book.

El libro es dibujado como un rectángulo y los atributos como óvalos.

Los atributos:

- Son propiedades de la entidad, por ejemplo título, año, etc.
- Están conectados a exactamente una entidad.

Para la entidad “author” el diagrama podría verse así:

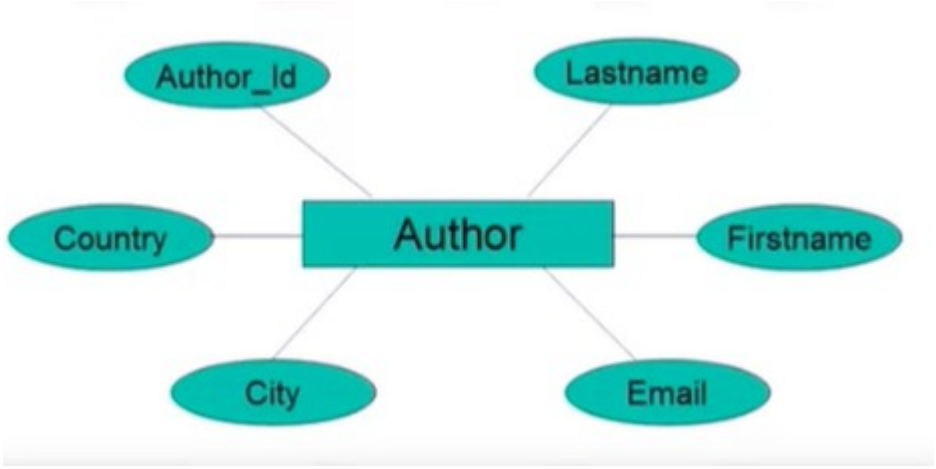


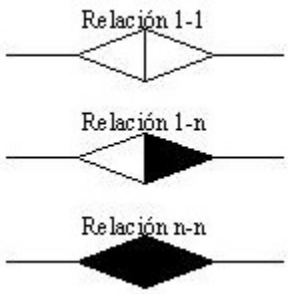
Figura. Modelo entidad-relación para Author.

Veamos cómo se relacionan las entidades book y author entre sí.

Un libro debe ser escrito por al menos 1 autor, y muchos autores pueden escribir 1 libro. A su vez 1 autor puede escribir solamente 1 libro, o muchos. En ambos casos hay una relación entre ellos.

Los diferentes tipos de relaciones son:

- 1..1
  - Las entidades que intervienen se asocian una a una.
    - Por ejemplo, la entidad HOMBRE, la entidad MUJER y entre ellos la relación matrimonio.
- 1..N
  - Una ocurrencia de una entidad está asociada con muchas (N) de la otra
    - Por ejemplo, la entidad EMPRESA, la entidad TRABAJADOR y entre ellos la relación TRABAJA-EN.
- N..M
  - Cada ocurrencia en cualquiera de las 2 entidades puede estar asociada con muchas de la otra y viceversa.
    - Por ejemplo, la entidad ALUMNO, la entidad EMPRESA y entre ellos la relación MATRICULA.



6. MAPEANDO ENTIDADES A TABLAS

Vamos a ver cómo las entidades se transforman en tablas del modelo relacional.

Los diagramas ER son fundamentales en el diseño de una base de datos.

En el diseño de bases relacionales, se comienza con un diagrama ER y luego este diagrama se convierte a las tablas.

El mapeo es simple, las entidades se convierten en tablas, y los atributos en columnas de la tabla.

7. CONCEPTOS DEL MODELO RELACIONAL

Veremos conceptos del modelo relacional.

El modelo relacional se basa en un modelo matemático, sus bloques de construcción son:

- Relaciones
- Conjuntos

El modelo relacional se basa en el concepto de relación.

Una **relación** es un concepto matemático basado en la idea de conjuntos.

Un **conjunto** es una colección sin orden de elementos diferentes.

Una **base de datos relacional** es un conjunto de relaciones.

Una relación a veces es el término matemático para una tabla.

Una tabla es una combinación de filas y columnas.

Una relación está compuesta de 2 partes:

- El esquema relacional (relational schema)
- La instancia relacional (relational instance)

Un **esquema relacional** especifica el nombre de una relación y sus atributos. Por ejemplo, para la entidad Author, “Author” es el nombre de la relación. Author_ID es un atributo, al igual que lastname, firstname, email, city y country. Esto constituye el esquema relacional.

Una **instancia relacional** es una tabla compuesta de atributos o columnas y tuplas o filas. Las columnas son los atributos o campos. Las filas son tuplas.

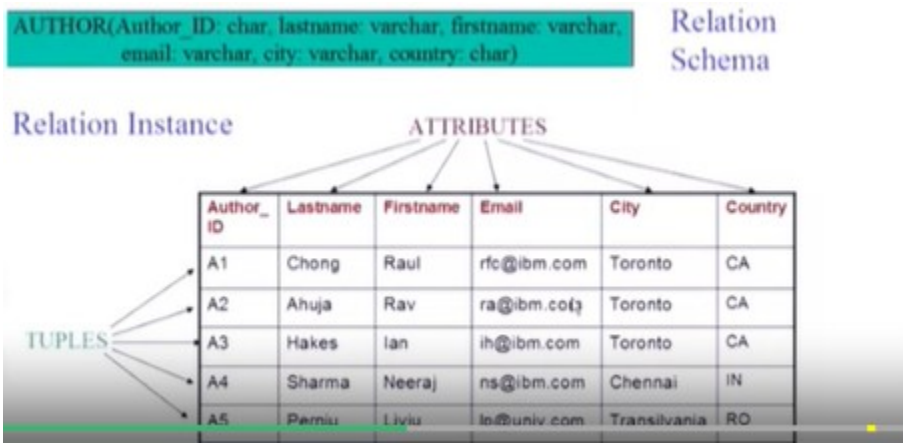


Figura. Esquema, instancia relacional, tuplas y atributos.

El **grado** hace referencia a la cantidad de atributos o columnas en una relación.

La **cardinalidad** se refiere a la cantidad de tuplas o filas.

En el ejemplo de la figura previa, el grado es 6, ya que hay 6 columnas, y la cardinalidad es 5, ya que hay 5 filas.

Un esquema SQL es identificado mediante un nombre e incluye un identificador de autorización para indicar la cuenta o el usuario al que pertenece.

Los elementos del esquema incliuyen:

- Tablas
- Restricciones
- Vistas
- Dominios
- Otros

8. UTILIZANDO PATRONES DE STRINGS Y RANGOS

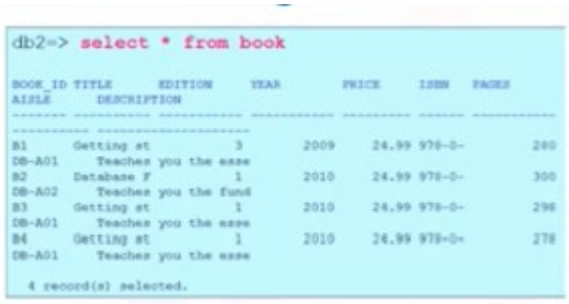
Veremos cómo simplificar una sentencia SELECT usando patrones de string, rangos y conjuntos de valores.

El prop{osito de un sistema de administración de bases de datos no es simplemente almacenar los datos, sino también facilitar la recuperación de los mismos.

En su forma más simple, una sentencia SELECT es:

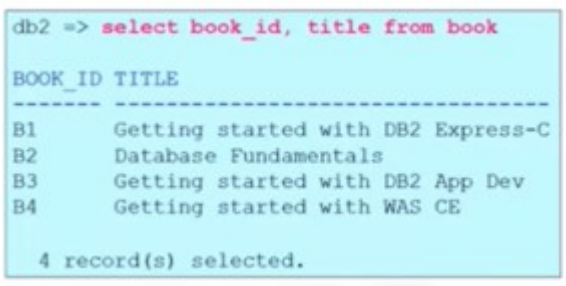
SELECT * FROM <nombre de la tabla>

Basado en el modelo simple de librería y en la tabla book, la sentencia “select * from book” da un conjunto resultado de 4 filas.

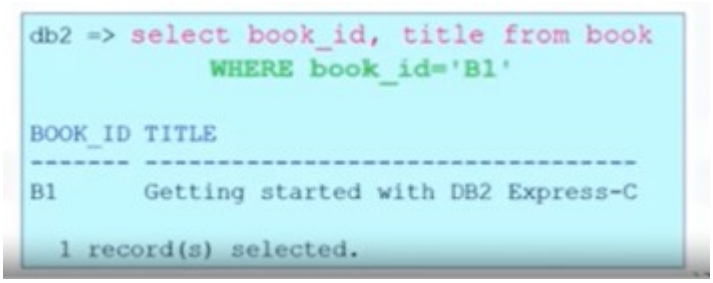


Todas las filas de datos para todas las columnas en la tabla book son desplegadas.

También puede recuperar un subconjunto de columnas solamente, como book_ID y title.



O puede restringir el conjutno resultado usando la cláusula WHERE. Por ejemplo, puede seleccionar el título del linro cuyo book_ID es B1.



Qué pasa si no sabemos qué valor exacto especificar en la cláusula WHERE? La cláusula WHERE siempre requiere un predicado, que es una condición que se evalúa para true, false o desconocido. Pero, y si no sabemos cuál es exactamente el predicado? Por ejemplo, no podemos recordar el nombre del autor, pero recordamos que su primer nombre empieza con R. Podemos usar patrones de strings para buscar filas de datos que coinciden con la condición.

Veamos algunos ejemplos.

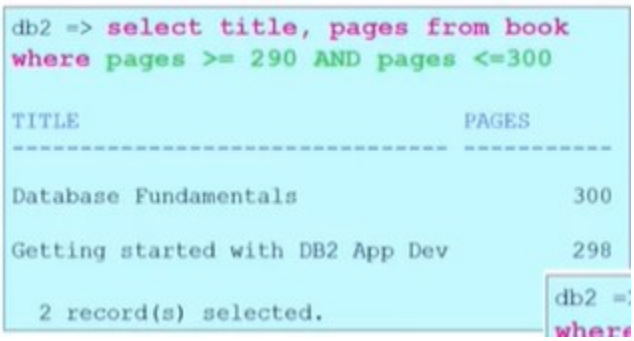
Si no recordamos el nombre del autor, pero sabemos empieza con R, usamos la cláusula WHERE con el predicado like. El predicado like se usa en una cláusula WHERE para buscar un patrón en una columna. El signo de porcentaje se usa para definir las letras faltantes y puede estar ubicado antes del patrón, después o tanto antes como después. En el ejemplo lo usamos antes del patrón, que es la letra R.

El signo de porcentaje se llama **wildcard**. Un wildcard es utilizado para sustituir otros caracteres.

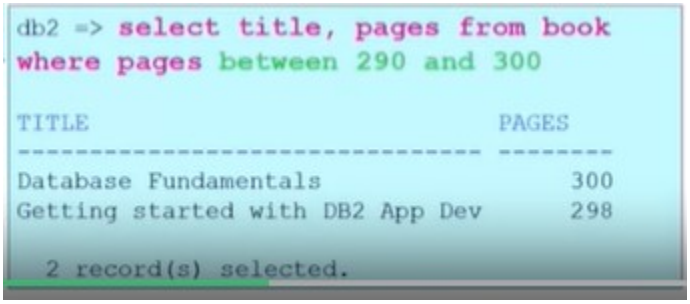
Así, la sentencia:

SELECT firstname FROM author WHERE firstname LIKE ‘R%’

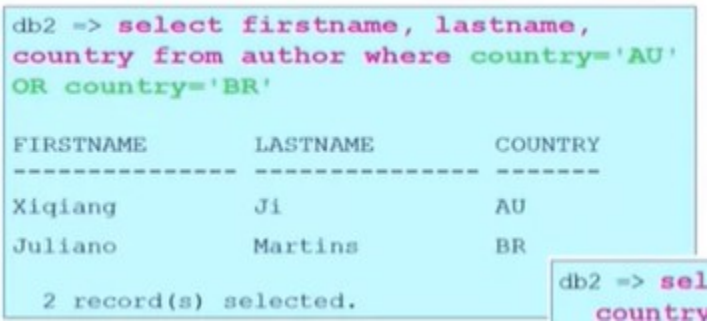
devolverá todas las filas en la tabla “author” para las que el primer nombre del autor comience con R.
En el ejemplo, se retornan 2 filas, para los autores Raul y Rav.
Qué pasa si queremos recuperar la lista de libros cuyo número de páginas es mayor a 290 pero menor a 300?
Podríamos escribir la siguiente sentencia SELECT:



Pero, hay una forma más sencilla, utilizando un rango numérico. En vez de usar los operadores de comparación mayor y menor, usamos el operador “between and”, que compara 2 valores. Los valores en el rango son inclusivos. En este caso, la consulta es así:



En algunos casos, hay valores de datos que no pueden ser agrupados en rangos. Por ejemplo, si queremos saber de qué países son los autores. Si queremos recuperar los autores que son de Australia o Brasil, podríamos escribir la sentencia SELECT con la cláusula WHERE repitiendo los valores de los países.

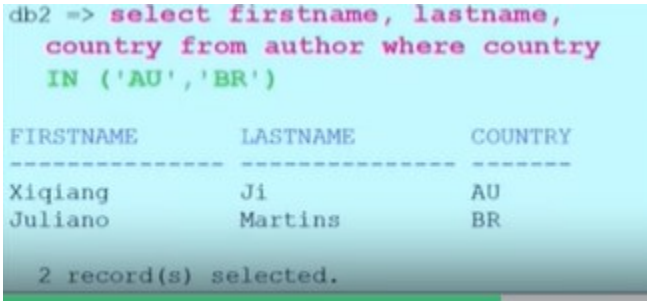


Sin embargo, qué pasa si también queremos recuperar los autores de Canadá, India y China?

La cláusula WHERE se volvería muy larga repitiendo cada condición.

En su lugar podemos usar el operador IN.

El operador IN permite especificar un conjutno de valores en una clásula WHERE. Este operador toma una lista de expresiones contra los cuales comparar, en este caso Australia o Brasil.



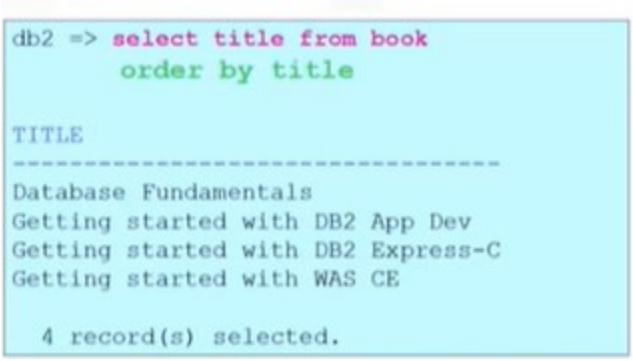
9. ORDENANDO CONJUNTOS DE RESULTADOS

Veremos cómo ordenar el conjunto resultado ya sea en orden ascendente o descendente y explicaremos cómo indicar qué columna usar para el orden.

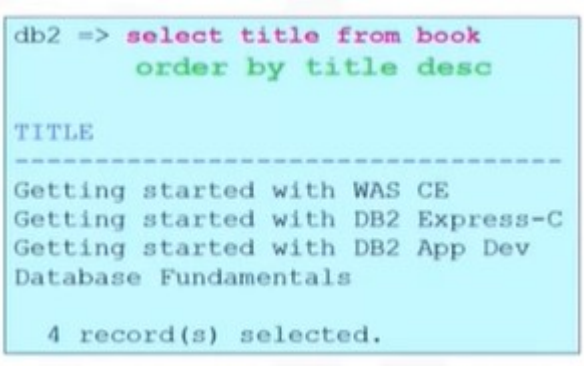
Si ejecutamos “select * from book” obtenemos todas las filas de datos para todas las columnas. También podemos listar solamente los títulos. Sin embargo, el resultado no parece tener ningún orden. Desplegar los resultados en orden alfabético haría más conveniente el resultado. Para lograr esto, usamos la cláusula **order by**.

La cláusula order by se usa en una consulta para ordenar el conjunto resultado mediante la columna especificada.

En este ejemplo, usamox order by en la columna title para ordenar el conjunto resultado.

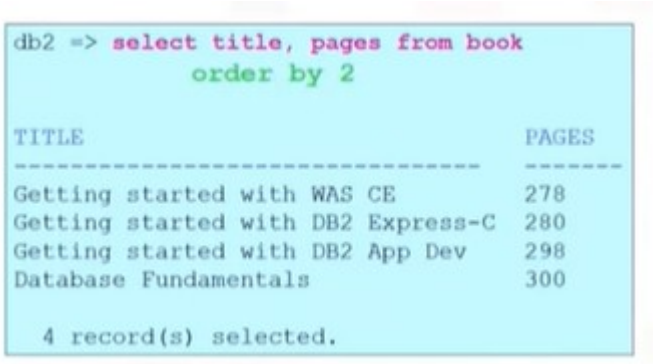


Por defecto, el conjunto resultado es ordenado en orden ascendente. Para ordenarlo en orden descendente, usamos la palabra clave **desc**.



Ahora, el resultado es ordenado acorde a la columna especificada (title en el ejemplo) en orden descendente.

Otra forma de especificar la columna por la cual ordenar es indicar el número de secuencia de la columna. Por ejemplo:



En lugar de especificar la columna “pages” se utilizó el número 2. En la sentencia SELECT, la segunda columna especificada en la lista de columnas es “pages”, por lo que el orden se basa en los valores de la columna “pages”. En este caso, la columna “pages” indica el número de páginas en el libro. Puede verse que el conjunto resultado es ordenado en orden ascendente por el número de páginas.

10. AGRUPANDO EL CONJUNTO DE RESULTADOS

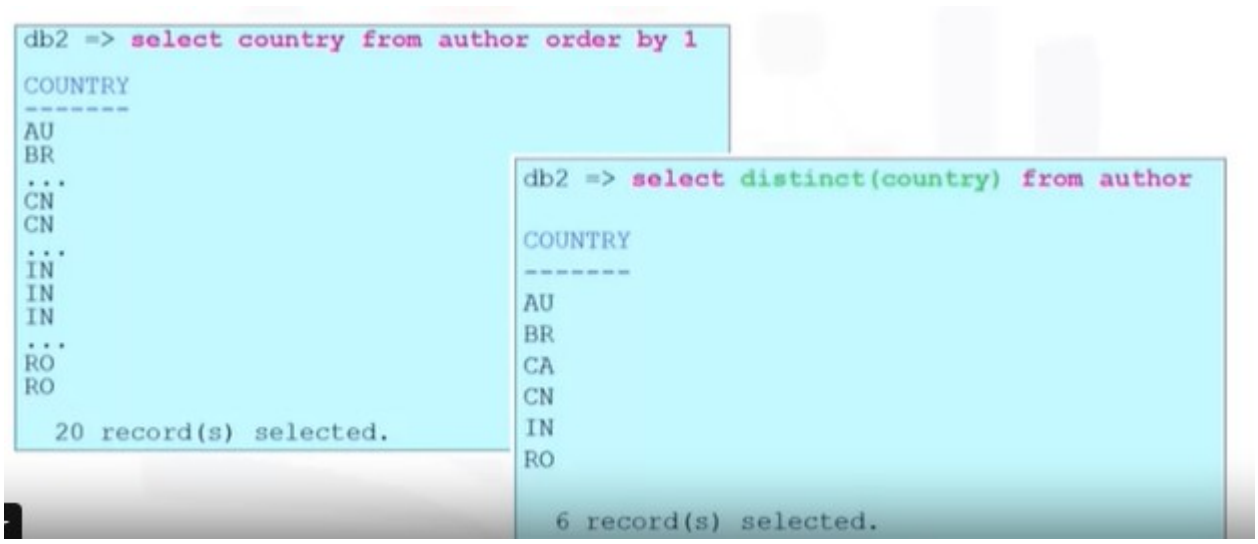
Veremos:

- Cómo ordenar y agrupar la forma en la que el conjunto de resultados es desplegado.
- Cómo eliminar duplicados.
- Cómo restringir aún más el conjunto de resultados.

A veces, una sentencia select devuelve un conjunto que tiene duplicados.

Basados en nuestro modelo de la la librería, en la tabla “author” la columna “country” contiene el código país de 2 letras de donde proviene el autor.

Si seleccionamos solamente la columna “country”, obtenemos una lista de todos los países. Por ejemplo "select country from author order by one”. El resultado lista los países a los que pertenecen los autores, ordenados alfabéticamente por “country”. En este caso el resultado muestra 20 filas, una por cada uno de los 20 autores. Pero algunos de los autores pueden ser del mismo país, por lo que el resultado contiene duplicados. Sin embargo, todo lo que necesitamos es una lista de los países de dónde provienen los autores. En este caso, los duplicados no tienen sentido. Para eliminar duplicados, usamos la palabra clave **distinct**.

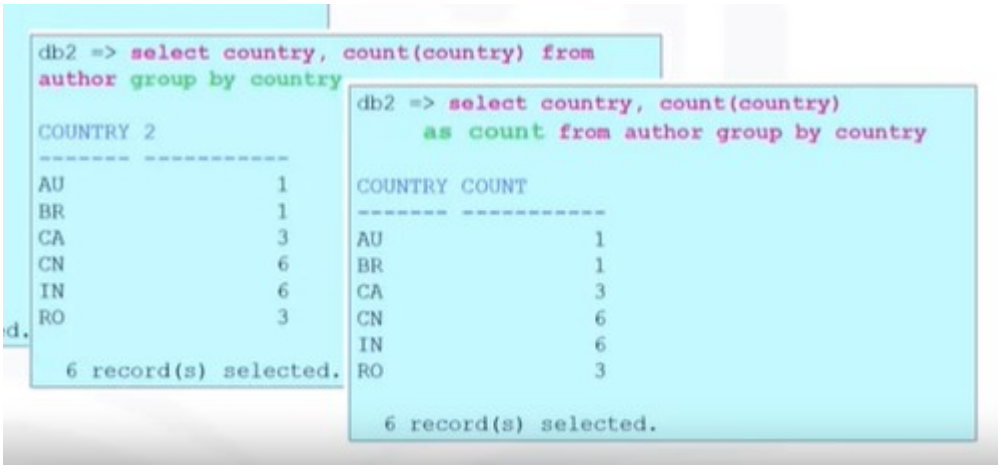


Usar “distinct” reduce el conjunto a 6 filas.

Qué pasa si además queremos saber cuántos autores provienen del mismo país? Para desplegar el conjunto resultado que liste el país y el número de autores que provienen de él, le agregamos la cláusula **group by** a la sentencia select.

La cláusula “group by” agrupa resultados en subconjuntos que tienen valores coincidentes para una o más columnas.

En este ejemplo, los países son agrupados y luego contados utilizando la función “count”.



Observe el encabezado de la columna para la segunda columna y el conjunto resultado. El valor numérico “2” se despliega como nombre de columna porque el nombre de columna no está disponible directamente en la tabla. La segunda columna en el conjunto resultado fue calculada mediante la función “count”.

En lugar de usar la columna llamada “2” podemos asignarle un nombre de columna al resultado. Hacemos esto usando la palabra clave “as”. En el ejemplo, cambiamos el nombre de la columna deducida “2” a “Count”. Esto simplifica el entendimiento del conunto resultado.

Ahora que ya tenemos la cuenta de autores de los diferentes países, podemos restringir aún más el número de filas pasando condiciones.

Por ejemplo, podemos chequear si hay más de 4 autores provenientes del mismo país. Para establecer una condición para una clásula “group by” usamos la palabra clave **having**.

La cláusula “having” se utiliza en combinación con “group by”. Es importante notar que la cláusula “where” es para el conjunto resultado completo, pero “having” trabaja solamente con “group by”.

Para saber si hay más de 4 autores provenientes del mismo país usamos la siguiente consulta:

```
db2 => select country, count(country)
as count from author group by country
having count(country) > 4

COUNTRY COUNT
------- -----------
CN                6
IN                6

  2 record(s) selected.
```

Sólo los países que tengan 5 o más autores serán listados en el conjunto resultado. En este ejemplo, esos países son China, con 6 autores, e India, también con 6 autores.

11. FUNCIONES INCORPORADAS

Veremos las funciones incorporadas.

Si bien es posible obtener primero los datos de la base de datos y luego realizar operaciones sobre ellos desde su aplicación, la mayoría de las bases de datos vienen con funciones incorporadas.

Estas funciones:

- Pueden ser incluidas en sentencias SQL, permitiendo realizar operaciones sobre los datos dentro de la propia base de datos.
- Pueden reducir significativamente la cantidad de datos que necesitan ser recuperados de la base de datos.
    - Es decir, reducen tanto el tráfico en la red como el uso del ancho de banda.

Cuando se trabaja con data sets grandes, puede ser más rápido usar funciones incorporadas, en vez de recuperar los datos en su aplicación y luego ejecutar funciones sobre los datos recuperados.

También es posible que un usuario cree sus propias funciones, las así llamadas funciones definidas por el usuario, pero en este capítulo estudiaremos las funciones incorporadas.

En esta lección consideraremos la tabla “petsale” en una base de datos para una tienda de mascotas. La tabla registra los detalles de las transacciones de ventas e incluye las columnas:

- ID
- animal
- quantity
- sale price
- sale date

Hemos poblado la tabla con varias columnas de datos, como se muestra en la figura.

PETSALE

ID INTEGER	ANIMAL VARCHAR(20)	QUANTITY INTEGER	SALEPRICE DECIMAL(6, 2)	SALEDATE DATE
1	Cat	9	450.09	2018-05-29
2	Dog	3	666.66	2018-06-01
3	Dog	1	100.00	2018-06-04
4	Parrot	2	50.00	2018-06-04
5	Dog	1	75.75	2018-06-10
6	Hamster	6	60.60	2018-06-11
7	Cat	1	44.44	2018-06-11
8	Goldfish	24	48.48	2018-06-14
9	Dog	2	222.22	2018-06-15

Entonces, qué son las funciones agregadas y de columna?

Una **función agregada** toma una colección de valores livianos, tales como los valores en una columna, como entrada y retorna un único valor o null.

Ejemplos de funciones agregadas son:

- sum (suma)
- minimun (mínimo)
- maximum (máximo)
- average (promedio)

Veamos algunos ejemplos basados en la tabla petsale.

Sum se utiliza para sumar todos los valores en una columna, por ejemplo, para sumar todos los valores en la columna “price” podría escribir:

```
SELECT SUM(SALEPRICE) FROM PETSALE
```

Cuando usa una función agregada, a la columna en el conjunto resultado le es dado un número. Es posible nombrar explícitamente la columna resultado. Supongamos que a la columna de salida del ejemplo anterior queremos llamarla SUM_OF_SALEPRICE, entonces escribrimos:

select SUM(SALEPRICE) as SUM_OF_SALEPRICE from petsale

Observe el uso de “as” en el ejemplo previo.
Minimun es utilizada para obtener el valor mínimo.
Maximum es utilizada para obtener el valor máximo. Por ejemplo, para obtener la máxima cantidad vendida de cualquier animal en una única transacción se escribe:

select MAX(QUANTITY) from petsale

Las funciones agregadas pueden ser aplicadas a un subconjunto de los datos en lugar de a la columna entera. Por ejemplo, para obtener el mínimo ID para los perros se escribe:

select MINIMUM(ID) from petsale where animal equals dog

average se usa para devolver el valor medio. Por ejemplo, para especificar el valor promedio del precio de una venta (sale price) se escribe:

select AVERAGE(SALEPRICE) from PETSALE

Observe que podemos realizar operaciones matemáticas engte columnas, y luego aplicar funciones agregadas sobre ellas. Por ejemplo, para calcular el valor de venta promedio por perro se escribe:

select AVERAGE(SALEPRICE divided by QUANTITY) from PETSALE where animal equals dog

En este caso los precios de venta tienen distintas unidades, por lo que primero dividimos el precio por la cantidad.
Ahora veamos funciones escalares y de strings.

Las **funciones escalares** realizan operaciones sobre valores individuales. Por ejemplo, para redondear hacia arriba o hacia abajo en la columna precio al entero más cercano se secribe:

select ROUND (SALEPRICE) from PETSALE

Hay una clase de funciones escalares llamadas **funciones de strings**, que se usan para realizar operaciones sobre strings (valores chat y varchar).
Por ejemplo, para recuperar el largo de cada valor en la columna “animal” escribimos:

select LENGTH(ANIMAL) from PETSALE

Las funciones uppercase y lowecase devuelven los valores en mayúsuculas y minúsculas respectivamente para strings. Por ejemplo, para recuperar valores en mayúsuculas de los animales se escribe:

select UPPERCASE(ANIMAL) from PETSALE

Las funciones escalares pueden utilizarse en la cláusula where. Por ejemplo, para obtener los valores en minúsucula de la columna “animal” para los gatos, se escribe:

select * from PETSALE where LOWERCASE(ANIMAL) equals cat

Este tipo de sentencia es útil para hacer coincidir valores en la cláusula where, cuando no se está seguro si los valores están almacenados en mayúsucula, minúsucula o de forma mixta en la tabla.
También se puede tener una función que opere sobre la salida de otra. Por ejemplo, para obtener los valores únicos de la columna “animal” en mayúsucula se escribe:

select DISTINCT(UPPERCASE(ANIMAL)) from PETSALE

12. FUNCIONES DE FECHA Y HORA

La mayoría de las bases de datos contienen tipos especiales de datos para fechas y horas.
Db2 contiene los tipos:

- Fecha
- Hora
- Timestamp (marca de tiempo)

En Db2 se tiene:

- 8 dígitos para las fechas: año, mes y día.
- 6 dígitos para la hora: horas, minutos y segundos.
- 20 dígitos para un timestamp: año, mes, día, hora, minutos, segundos y microsegundos.

Existen funciones para extraer el día, mes, día del mes, día de la semana, día del año, semana, hora y segundo.
Veamos algunos ejemplos.
La función day se usa para extraer la parte que corresponde al día en una fecha. Por ejemplo, para obtener la parte que corresponde al día en cada venta que involucre a un gato se escribe:

select DAY(SALEDATE) from PETSALE where ANIMAL equals cat

Las funciones date y time pueden utilizarse en la cláusula where. Por ejemplo, para obtener el número de ventas que ocurrieron durante el mes de mayo (mes 5) se escribe:

select COUNT(*) from PETSALE where MONTH(SALEDATE) = ‘05’

También pueden realizarse operaciones aritméticas. Por ejemplo, supongamos que queremos saber cuál fue la fecha 3 días después de cada venta porque cada orden necesita ser procesada dentro de 3 días. La consulta sería: cuál fecha es 3 días después de cada fecha de venta? Para ello se escribe:

```
select(SALEDATE + 3 DAYS) from PETSALE
```

También hay disponibles registros especiales para la fecha y hora actuales. Por ejemplo, encontrar cuántos días han pasado desde cada venta hasta ahora:

```
select (CURRENT_DATE – SALEDATE) from PETSALE
```

13. SUBCONSULTAS Y SELECT ANIDADOS

Aprenderemos como escribir subconsultas.

Las **subconsultas** (o sub-selects) son como las consultas regulares pero se ubican dentro de paréntesis y anidadas dentro de otras consulta. Esto permite realizar consultas más poderosas.

A continuación se muestra un ejemplo de consulta anidada:

```
select COLUMN1 from TABLE
  where COLUMN2 = (select MAX(COLUMN2) from TABLE)
```

Puede verse que la sub-consulta se encuentra dentro de la cláusula where de otra consulta.

Considere la tabla de empleados.

EMPLOYEES

EMP_ID	F_NAME	L_NAME	SSN	B_DATE	SEX	ADDRESS	JOB_ID	SALARY	MANAGER_ID	DEP_ID
E1001	John	Thomas	123456	1976-01-09	M	5631 Rice, OakPark,IL	100	100000	30001	2
E1002	Alice	James	123457	1972-07-31	F	980 Berry ln, Elgin,IL	200	80000	30002	5
E1003	Steve	Wells	123458	1980-08-10	M	291 Springs, Gary,IL	300	50000	30002	5

La tabla contiene varias columnas, incluyendo:

- ID de empleado
- Nombre
- Apellido
- Salario

Veremos algunos ejemplos que involucren esta tabla, considerando escenarios que involucren el uso de sub-consultas.

Supongamos que queremos recuperar la lista de empleados que ganan más que el salario promedio. Podríamos usar la siguiente consulta:

select * from employees where salary > AVG(salary)

Pero eso resultará en el siguiente error:

```
SQL0120N Invalid use of an aggregate function or
OLAP function.SQLCODE=-120, SQLSTATE=42903
```

El error indica el uso inválido de una función de agregación. Una de las limitantes de las funciones de agregación incorporadas, es que no siempre pueden evaluarse dentro de una cláusula where. Entonces, para evaluar una función como average en la cláusula where, podemos usar una sub-consulta como la siguiente:

```
select EMP_ID, F_NAME, L_NAME, SALARY
       from employees
       where SALARY <
       (select AVG(SALARY) from employees);
```

Observe que la función de agregación es evaluada en la primer parte de la subconsulta., permitiéndonos superar la limitación de evaluarla directamente en la cláusula where.
La sub-consulta no tiene por qué ir en la cláusula where, puede ir en otras partes, como en la lista de columnas a ser seleccionadas. Tales subconsultas se llaman **expresiones de columna**.
Veamos ahora un ejemplo con expresiones de columna. Supongamos que queremos comparar el salario de cada empleado con el salario promedio. Podríamos intentar la siguiente consulta:

```
select EMP_ID, SALARY, AVG(SALARY) AS AVG_SALARY
       from employees ;
```

Pero ejecutarla dará un error indicando que no hay cláusula by especificada.

```
select EMP_ID, SALARY, AVG(SALARY) AS AVG_SALARY
       from employees ;
```

Lo que hacemos entonces es usar la función de agregación en una sub-consulta ubicada en la lista de columnas. Por ejemplo:

```
select EMP_ID, SALARY,
       ( select AVG(SALARY) from employees )
                 AS AVG_SALARY
       from employees ;
```

Otra opción es realizar la consulta como parte de la cláusula FROM. Subconsultas como estas son llamadas a veces **expresiones de tabla o tabla derivada**, ya que la consulta exterior usa los resultados de la subconsulta como fuente de datos.
Veamos cómo crear una expresión de tabla que contenga información no sensible del empleado:

```
select * from
       ( select EMP_ID, F_NAME, L_NAME, DEP_ID
                from employees) AS EMP4ALL ;
```

La tabla derivada en la subconsulta no incluye campos sensibles como la fecha de nacimiento o el salario. Este ejemplo es trivial y fácilmente podríamos haber incluido las columnas en la consulta exterior. Sin embargo, las tablas derivadas han probado ser poderosas en situaciones más complejas que involucren trabajar con múltiples tablas.

14. TRABAJANDO CON MULTIPLES TABLAS

Veremos cómo escribir consultas que acceden a más de una tabla.

Hay varias formas de acceder a múltiples tablas en una misma consulta, a saber:

- Subconsultas
- JOIN implícito
- Operadores JOIN (INNER JOIN, OUTER JOIN, etc)

Aquí veremos las primeras 2 opciones.

Consideraremos las tablas employees (empleados) y departments (departments).

EMPLOYEES:

EMP_ID	F_NAME	L_NAME	SSN	B_DATE	SEX	ADDRESS	JOB_ID	SALARY	MANAGER_ID	DEP_ID
E1001	John	Thomas	123456	1976-01-09	M	5631 Rice, OakPark,IL	100	100000	30001	2
E1002	Alice	James	123457	1972-07-31	F	980 Berry ln, Elgin,IL	200	80000	30002	5
E1003	Steve	Wells	123458	1980-08-10	M	291 Springs, Gary,IL	300	50000	30002	5

DEPARTMENTS:

DEPT_ID_DEP	DEP_NAME	MANAGER_ID	LOC_ID
5	Software Development	30002	L0002
7	Design Team	30003	L0003

Usemos subconsultas para trabajar con múltiples tablas.

Si queremos recuperar solamente los registros de la tabla de empleados para los cuales el ID de departamento existe en la tabla “departments” podemos usar la siguiente subconsulta:

```
select * from employees
       where DEP_ID IN
       ( select DEPT_ID_DEP from departments );
```

Aquí la consulta externa accede a la tabla de empleados y la subconsulta en la tabla de departamentos es utilizada para filtrar el conjunto resultado de la consulta exterior.
Digamos que queremos solamente la lista de empleados de una ubicación específica. No tenemos información de ubicación en la tabla de empleados, pero la tabla de departamentos tiene una columna llamada “location ID”. Podemos entonces usar una subconsulta desde la tabla de departamentos como entrada para la de empleados como sigue:

```
select * from employees
       where DEP_ID IN
       ( select DEPT_ID_DEP from departments
                where LOC_ID = 'L0002' );
```

Ahora, recuperemos el ID y nombre de departamento para aquellos empleados que ganen más de $70.000. Para hacerlo necesitamos una subconsulta en la tabla de empleados que satisfaga el criterio del salario, y alimentarla como entrada a una consulta exterior en la tabla de departamentos para obtener la información del departamento coincidente.

```
select DEPT_ID_DEP, DEP_NAME from departments
 where DEPT_ID_DEP IN
  ( select DEP_ID from employees
    where SALARY > 70000 ) ;
```

También podemos acceder a múltiples tablas especificándolas en la cláusula FROM de la consulta.
Considere el siguiente ejemplo:

select * from employees, departments

Aquí especificamos 2 tablas en la cláusula FROM. Esto resulta en un join (unión), pero observe que no estamos usando explícitamente el operador join. El join resultante se llama **full join** o **join cartesiano**, porque cada fila de la primera tabla es unida (joined) con cada fila de la segunda. En el resultado, habrán más filas que en ambas tablas individualmente.
Podemos usar operandos adicionales para limitar el conjunto resultado. Veamos un ejemplo donde limitamos el conjunto resultado a unas pocas filas al hacer coincidir los IDs de departamento:

```
select * from employees, departments
       where employees.DEP_ID =
             departments.DEPT_ID_DEP;
```

Observe que en la cláusula where prefijamos el nombre de la columna con el nombre de la tabla. Esto es para cuantificar completamente el nombre de la columna, ya que diferentes tablas podrían tener columnas con exactamente el mismo nombre.
Como los nombres de las tablas pueden ser largos, a menudo usamos alias para los nombres de las tablas:

```
select * from employees E, departments D
       where E.DEP_ID = D.DEPT_ID_DEP;
```

Aquí definimos el alias E para la tabla employees y D para la tabla departments, y luego usamos los alias en la cláusula where.
Si queremos ver el nombre del departamento para cada empleado, escribimos:

```
select employees.EMP_ID, departments.DEPT_NAME
       from employees E, departments D
       where E.DEP_ID = D.DEPT_ID_DEP;
```

Del mismo modo que lo anterior, los nombres de columna pueden ser prefijados con alias:

```
select E.EMP_ID, D.DEP_ID_DEP from
       employees E, departments D
       where E.DEP_ID = D.DEPT_ID_DEP
```

15. RESTRICCIONES DEL MODELO RELACIONAL

En todos los negocios, los datos deben adherirse a ciertas restricciones o reglas.

Una clave primaria PK (Primary Key) identifica cada fila en una tabla.

Una clave foránea FK (Foreign Key) es un conjunto de columnas que hace referencia a una PK de otra tabla.

Una tabla que contiene una PK que es relacionada a al menos una FK es llamada tabla padre.

Una tabla que contiene una o más Fks es llamada tabla dependiente o hija.

Veremos ahora las 6 restricciones del modelo relacional.

Los datos deben cumplir ciertas restricciones o reglas, las restricciones nos ayudan a implementar las reglas del negocio.

En el modelo relacional, la integridad de datos puede lograrse usando reglas de integridad o restricciones.

Las siguientes 6 restricciones se definen en el modelo relacional:

- Entity integrity constraint, **restricción de integridad de la entidad**
  - Para identificar cada tupla en una relación, ésta debe tener una PK. La PK es un valor único que identifica cada tupla o fila en una tabla, ésta es la restricción de integridad de la entidad.
  - Los términos PK, key constraint o unique constraint son utilizados indistintamente.
  - Esta restricción evita tener valores duplicados en una tabla.
  - Para implementar estas restricciones, se usan índices.
  - La restricción de integridad de la entidad establece que ningún atributo que participe en la PK de una relación puede tener valores null.
- Referencial Integrity constraint, **restricción de integridad referencial**
  - Define relaciones entre tablas y asegura que estas relaciones permanezcan válidas.
- Semantic integrity constraint, **restricción de identidad semántica**
  - Se refiere a la exactitud del significado de los datos.
- Domain constraint, **restricción de dominio**
  - Especifica los valores permisibles para un atributo dado.
- Null constraint, **restricción null**

    - Especifica que el atributo no puede ser null.
- Check constraint, **restricción de chequeo** f
    - Fuerza la integridad de dominio limitando los valores que son aceptados por un atributo.

La validez de los datos es forzada usando una combinación de Pks y Fks.

PK

Si un esquema de relación tiene más de una clave, cada clave se llama clave candidata. Una de ellas es designada como PK y las otras son llamadas claves secundarias.

Reglas para PKs:

- El valor de la PK debe ser único para cada instancia de la entidad.
- No puede haber valores faltantes (no se acepta NULL). Si la clave está compuesta por muchos atributos, ninguno de ellos puede ser NULL.
- Es inmutable (una vez creada, no puede ser modificada. Si consiste de múltiples atributos, ninguno puede ser modificado.

Integridad Semántica

Asegura que los datos que ingresan a una fila reflejen un valor permisible para la misma. El valor debe pertenecer al dominio, o a un conjunto de valores permitidos para esa columna. Por ejemplo, la columna “cantidad” sólo permite números.

Restricciones semánticas

Las restricciones semánticas son restricciones que no pueden expresarse directamente en los esquemas del modelo de datos. También son llamadas **reglas basadas en aplicación o reglas del negocio**. Son reglas adicionales especificadas por los usuarios o los DBA. Por ejemplo, una clase puede tener un máximo de 30 estudiantes o el salario de un empleado no puede exceder el de un gerente.

Las restricciones de dominio especifican que dentro de una tupla el valor de cada atributo debe ser un elemento del dominio de ese atributo.

Los data types asociados con el dominio incluyen:

- Integer (short integer, integer, long integer).
- Real numbers (float and double precision float).
- Caracteres.
- Booleanos.
- Strings de largo fijo y variable
- Date, time y timestamp.
- Dinero.
- Otros.

Otros posibles valores de dominio podrían ser un sub-rango de valores de un data type o un enumerado donde los valores son explícitamente listados.

16. ACCEDER A BASES DE DATOS UTILIZANDO PYTHON

Las bases de datos son herramientas poderosas para los científicos de datos.

En este y los siguientes capítulos aprenderá a:

- Crear tablas, cargar y consultar datos usando SQL desde una Jupyter Notebook.
- Analizar datos.
- Crear una instancia en la nube, conectarse a una base de datos, consultar datos desde una base de datos usando SQL y analizar datos usando Python.
- Conectar una aplicación Pyhton a una base de datos.
- Describir APIs SQL.

Recordemos algunas ventajas de utilizar Python:

- Provee numerosas herramientas para data science. Algunos de los paquetes más populares son NumPy, Pandas, Matplotlib y SciPy.
- Es fácil de aprender.
- Es open source.
- Soporta sistemas de bases relacionales.

Escribir código Python para acceder a bases de datos se hace fácil debido a la presencia de la database API de Python, comúnmente llamada **DB API**.

Las notebooks son muy populares en data science porque se ejecutan en un ambiente que permite crear y compartir documentos que contienen código vivo, ecuaciones, visualizaciones y textos explicativos.

La interfaz de una notebook es un ambiente virtual usado para la programación.

Ejemplos de interfaces de notebook son:

- Maple worksheet
- Matlab notebook
- Ipython Jupyter
- R Markdown
- Apache Zeppelin
- Apache Spark notebook
- Databricks cloud

En este módulo usaremos las Jupyter notebooks.

Algunas de las ventajas de usar Jupyter notebooks son:

- Soporte para más de 40 lenguajes de programación, incluyendo Python, Julia, R y Scala.
- Las notebooks pueden ser compartidas con otros vía mail, Dropbox, GitHub y el Jupyter Notebook viewer.
- Su código puede producir HTML interactivo, imágenes, videos lay tech y tipos personalizados.
- Puede aprovechar herramientas de big data como Apache Spark de Python, R y Scala, y explorar esos mismos datos con pandas, scikit-learn, ggplot2 y TensorFlow.

Así es como un usuario típicamente accede a bases de datos usando Python en una Jupyter Notebook: Hay un mecanismo mediante el cual el programa Python se comunica con la DBMS. El código Python se conecta a la base de datos utilizando llamadas a la API.

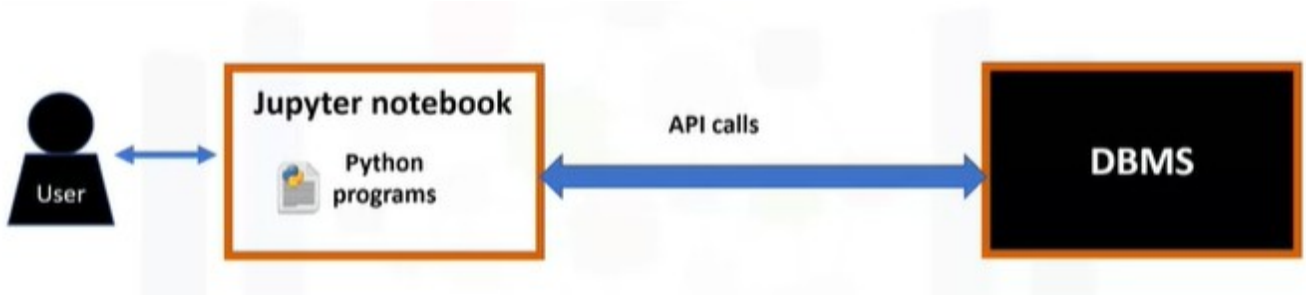


Figura. Acceder a una base de datos usando Python en una Jupyter Notebook.

Una **API** (Application Programming Interface) es un conjunto de funciones que puede llamar para obtener acceso a algún tipo de servicios.

La **API SQL** consiste de llamadas a funciones de librerías.

Para pasar sentencias SQL al DBMS, un programa llama a funciones en la API y a otros funciones para recuperar resultados de consultas e información de status del DBMS.

La operación básica de una API SQL típica se ilustra en la figura.

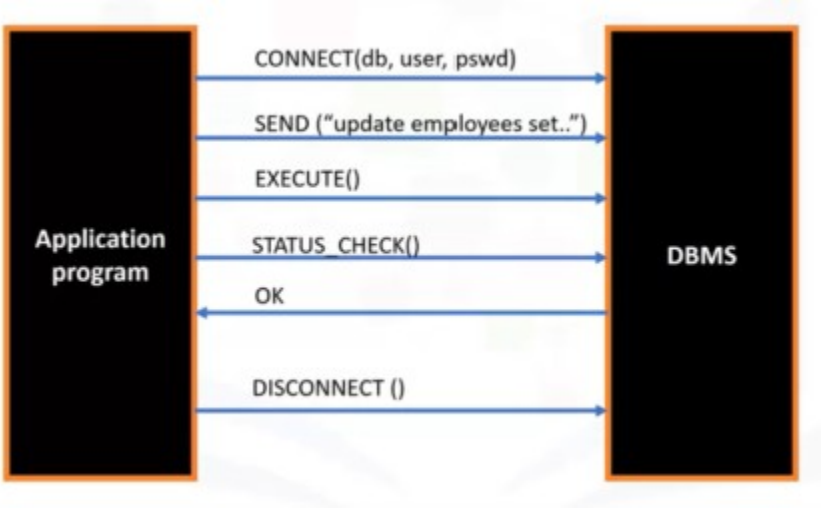


Figura. Operación básica de una API SQL.

- El programa comienza su acceso a la base de datos con una o más llamadas a la API que conectan el programa a la DBMS.
- Para enviar la sentencia SQL al DBMS, el programa construye la sentencia como texto en un buffer y luego realiza llamadas a la API para pasar el contenido del buffer al DBMS.
- El programa realiza llamadas a la API para chequear el status de la solicitud al DBMS y para manejar errores.
- El programa termina su acceso a la base de datos con una llamada a la API que desconecta de la base de datos.

Veamos conceptos básicos acerca de algunas APIs propietarias utilizadas por DBMSs populares.

Cada sistema de base de datos tiene su propia librería. La tabla siguiente muestra una lista de algunas aplicaciones y sus correspondientes APIs SQL.

Application or Database	SQL API
MySQL	MySQL C API
PostgreSQL	psycopg2
IBM DB2	ibm_db
SQL Server	dblib API
Database access for Microsoft Windows OS	ODBC
Oracle	OCI
Java	JDBC

Figura. Aplicaciones populares y sus correspondientes APIs SQL.

MySQL C API provee acceso de bajo nivel al cliente MySQL y habilita a programas en C a acceder al contenido de la base de datos.

Psycopg2 conecta aplicaciones Python para bases de datos PostgreSQL.

IBM_DB API se usa para conectar aplicaciones Python a bases de datos IBM DB2.

dblib API es utilizada para conectarse a bases de datos SQL Server.

ODBC es utilizada para acceder a bases de datos Microsoft Windows OS.

OCI es utilizado por bases de datos Oracle.

JDBC es utilizado por aplicaciones Java.

17. ESCRIBIENDO CÓDIGO UTILIZANDO LA DB API

Escribiremos programas utilizando una Jupyter Notebook.

Hay un mecanismo por el cual el código Python se comunica con el DBMS.

El código Python se conecta a la base de datos realizando llamadas a DB-API. DB-API es una API Python estándar para acceder a bases de datos relacionales. Es un estándar que permite escribir un único programa para que trabaje con múltiples tipos de bases de datos relacionales en lugar de escribir un programa separado para cada una. Así, si aprende las funciones de la DB-API, podrá aplicar ese conocimiento para utilizarlo con cualquier base de datos con Python.

Algunas ventajas de usar la DB-API son:

- Es fácil de implementar y entender.
    - La API ha sido definida para alentar la similaridad entre los módulos de Python que se usan para acceder a bases de datos.
- Logra consistencia, lo que lleva a módulos más fáciles de entender.
- El código es generalmente más portable a través de bases de datos y tiene un alcance más amplio de conectividad.

Cada sistema de bases de datos tiene su propia librería. La tabla siguiente muestra algunas bases de datos y sus correspondientes DB-APIs para conectarse a aplicaciones Python.

Database	DB API
DB2 Warehouse on Cloud	Ibm_db
Compose for MySQL	MySQL Connector/Python
Compose for PostgreSQL	psycopg2
Compose for MongoDB	PyMongo

Figura. Bases de datos y DB-APIs para Python.

Los 2 conceptos principales en la DB-API Python son:

- **Objetos de conexión** (connection objetcs)
- **Objetos de consulta** (query objects)

Se usan **objetos de conexión** para conectarse a una base de datos y administrar sus transacciones.

Los **objetos cursor** se usan para ejecutar consultas.

Usted abre un objeto cursor y luego ejecuta consultas. El cursor funciona de forma similar a un cursor en un sistema de procesamiento de texto donde se desplaza hacia abajo en su conjunto resultado e introduce sus datos en la aplicación.

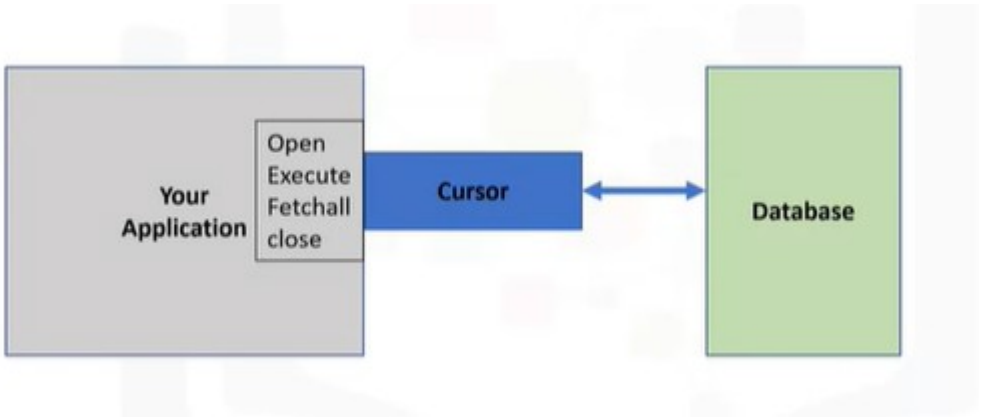
Los cursores se usan para escanear a través de los resultados de una base de datos.

Algunos métodos utilizados con los objetos cursor son:

- cursor()
  - Devuelve un nuevo cursor utilizando la conexión.
- commit()
  - Se utiliza para confirmar cualquier transacción pendiente para la base de datos.
- rollback()
  - Provoca que la base de datos vuelva hacia el comienzo de cualquier transacción que tenga pendiente.

- close()
    - Es utilizado para cerrar una conexión a una base de datos.

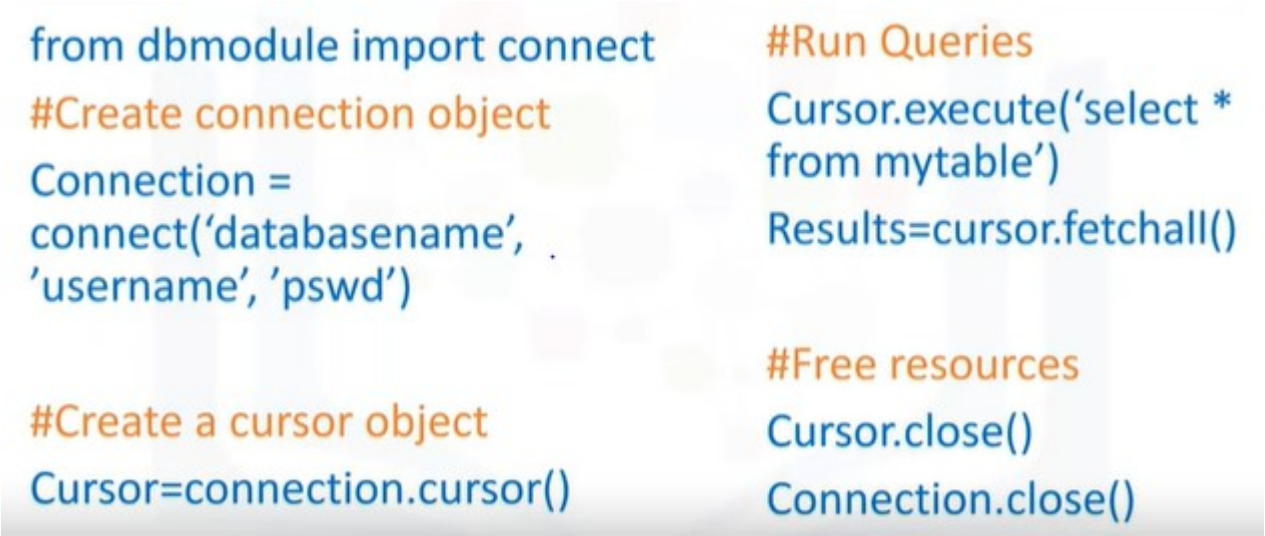
Estos objetos representan un cursor de base de datos el cual es utilizado para administrar el contenido de una operación de recuperación.
Los cursores creados por la misma conexión no están aislados, es decir, cualquier cambio realizado por un cursor es inmediatamente visible para los otros.
Los cursores creados desde diferentes conexiones pueden o no estar aislados dependiendo de cómo esté implementado el soporte de transacciones.
Un cursor de una base de datos es una estructura de control que permite recorrer los registros de una base de datos.



Un cursor se comporta de forma similar a un nombre de archivo o manejador de archivo en un lenguaje de programación. Así como un programa abre un archivo y accede a su contenido, se abre un cursor para ganar acceso a los resultados de la consulta. Similarmente así como un programa cierra el archivo para finalizar el acceso a su contenido, el cursor se cierra para dar fin al acceso a los resultados de la consulta. Así como un programa sigue rastro de la posición actual dentro de un archivo abierto, el cursor sigue rastro de la posición actual del programa dentro de los resultados de la consulta.
Estudiemos una aplicación Python que usa DB-API para consultar una base de datos.

- Primero, importa su módulo de base de datos usando la API de conexión para ese módulo.

- Para abrir una conexión a la base de datos, usa la función de conexión y le pasa los parámetros, entre ellos nombre de la base de datos, usuario y password.
- La función de conexión retorna un objeto de conexión.
- Luego, crea un objeto cursor sobre el objeto conexión.
- El cursor es utilizado para ejecutar consultas y obtener resultados.
- Luego de ejecutar las consultas usando el cursor, también usamos el cursor para buscar los resultados de la consulta.
- Cuando el sistema realizó todas las consultas, libera los recursos cerrando la conexión.
    - Es importante cerrar las conexiones para evitar que conecciones que no estén siendo utilizadas consuman recursos.



18. CONECTANDOSE A UNA BASE DE DATOS UTILIZANDO IBM_DB API

En este capítulo:

- Entenderemos la API ibm_db.
- Veremos qué credenciales son necesarias para conectarse a una base de datos utilizando Python.

También veremos cómo conectarse a la base de datos IBM DB2 warehouse usando código Python escrito en una Jupyter Notebook.
La API ibm_db provee una variedad de funciones Python útiles para acceder y manipular datos en un servidor de base de datos IBM incluyendo funciones para:

- Conectarse a la base de datos.
- Preparar y emitir sentencias SQL.
- Obtener filas de conjuntos de resultados.
- Llamar a procedimientos almacenados.
- Confirmar y volver atrás transacciones.
- Manejar errores.
- Recuperar metadata.

La API ibm_db usa el IBM Data Server Driver para ODBC y APIs CLI para conectarse a IBM, DB2 e Informix.
Importamos la librería ibm_db en nuestra aplicación Python:

import ibm_db

Conectarse al warehouse DB2 requiere la siguiente información:

- Nombre del driver.
- Nombre de la base de datos.
- Nombre DNS del host o dirección IP.
- Puerto del host.
- Protocolo de conexión.
- ID de usuario.
- Password de usuario.

```
dsn_driver = "IBM DB2 ODBC DRIVER"
dsn_database = "BLUDB"            # e.g. "BLUDB"
dsn_hostname = "dashdb-entry-yp-dal10-01.services.dal.bluemix.net" # e.g.: "awh-yp-small03.services.dal.bluemix.net"
dsn_port = "50001"                # e.g. "50000"
dsn_protocol = "TCPIP"            # i.e. "TCPIP"
dsn_uid = "dash****"        # e.g. "dash104434"
dsn_pwd = "*********"       # e.g. "7dBZ3jWt9xN6$o0JiX!m"
```

Aquí un ejemplo:

```
#Create database connection
dsn = (
    "DRIVER={{IBM DB2 ODBC DRIVER}};"
    "DATABASE={0};"
    "HOSTNAME={1};"
    "PORT={2};"
    "PROTOCOL=TCPIP;"
    "UID={3};"
    "PWD={4};").format(dsn_database, dsn_hostname, dsn_port, dsn_uid, dsn_pwd)

try:
    conn = ibm_db.connect(dsn, "", "")
    print ("Connected!")

except:
    print ("Unable to connect to database")
```

```
Connected!
```

Creamos un objeto de conexión DSN que almacena las credenciales de conexión.

La función de conexión de la API ibm_db será utilizada para crear una conexión no persistente.

El objeto DSN se pasa como parámetro a la función de conexión.

Si se ha establecido conexión con la base de datos el código retorna conectado así como la salida, de lo contrario mostrará que no puede conectarse.

Luego liberamos todos los recursos cerrando la conexión.

```
In [8]: ibm_db.close(conn)

Out[8]: True
```

19. CREAR UNA INSTANCIA DE BASE DE DATOS EN LA NUBE

A continuación se muestran las claves de conexión resultantes de crear una instancia de Db2 en IBM Cloud.


```
{
  "db": "BLUDB",
  "dsn": "DATABASE=BLUDB;HOSTNAME=dashdb-txn-sbox-yp-dal09-
10.services.dal.bluemix.net;
PORT=50000;PROTOCOL=TCPIP;UID=vcx43363;PWD=6ph4z6-8w0hwmrk8;",
  "host": "dashdb-txn-sbox-yp-dal09-10.services.dal.bluemix.net",
  "hostname": "dashdb-txn-sbox-yp-dal09-10.services.dal.bluemix.net",
  "https_url": "https://dashdb-txn-sbox-yp-dal09-10.services.dal.bluemix.net",
  "jdbcurl": "jdbc:db2://dashdb-txn-sbox-yp-dal09-10.services.dal.bluemix.net:50000/BLUDB",
  "parameters": {
    "role_crn": "crn:v1:bluemix:public:iam::::serviceRole:Manager"
  },
  "password": "6ph4z6-8w0hwmrk8",
  "port": 50000,
  "ssldsn": "DATABASE=BLUDB;HOSTNAME=dashdb-txn-sbox-yp-dal09-
10.services.dal.bluemix.net;PORT=50001;PROTOCOL=TCPIP;UID=vcx43363;PWD=6ph4z6-
8w0hwmrk8;Security=SSL;",
  "ssljdbcurl": "jdbc:db2://dashdb-txn-sbox-yp-dal09-10.services.dal.bluemix.net:50001/
BLUDB:sslConnection=true;",
  "uri": "db2://vcx43363:6ph4z6-8w0hwmrk8@dashdb-txn-sbox-yp-dal09-
10.services.dal.bluemix.net:50000/BLUDB",
  "username": "vcx43363"
}
```


De una forma más clara:

```
dsn_hostname = "dashdb-txn-sbox-yp-dal09-10.services.dal.bluemix.net"
dsn_uid = "vcx43363"
dsn_pwd = "6ph4z6-8w0hwmrk8"
dsn_driver = "{IBM DB2 ODBC DRIVER}"
dsn_database = "BLUDB"
dsn_port = "50000"
dsn_protocol = "TCPIP"
```

```
#Replace the placeholder values with your actual Db2 hostname, username, and password:
dsn_hostname = "YourDb2Hostname" # e.g.: "dashdb-txn-sbox-yp-dal09-
04.services.dal.bluemix.net"
dsn_uid = "YourDb2Username"        # e.g. "abc12345"
```

```
dsn_pwd = "YoueDb2Password"      # e.g. "7dBZ3wWt9XN6$o0J"

dsn_driver = "{IBM DB2 ODBC DRIVER}"
dsn_database = "BLUDB"           # e.g. "BLUDB"
dsn_port = "50000"               # e.g. "50000"
dsn_protocol = "TCPIP"           # i.e. "TCPIP"
```

20. CREAR TABLAS, CARGAR Y CONSULTAR DATOS

En esta sección crearemos tablas, cargaremos y consultaremos datos usando Python.

Para este ejemplo, usaremos DB2 Warehouse como base de datos.

Primero obtenemos un recurso de conexión conectándonos a la base de datos usando el método connect de ibm_dbapi.

Hay diferentes formas de crear tablas en DB2 Warehouse:

- Usando la consola web provista por DB2 Warehouse
- Desde cualquier ambiente SQL, R o Python.

Veamos cómo hacerlo desde una aplicación Python.

Aquí un ejemplo de una tabla de una base de datos de camiones comerciales.

Serial No	Model	Manufacturer	Engine Size	Class
A1234	Lonestar	International Trucks	Cummins ISX15	Class 8
B5432	Volvo VN	Volvo Trucks	Volvo D11	Heavy Duty Tractor Class 8
C5674	Kenworth W900	Kenworth Truck Company	Caterpillar C9	Class 8

Veamos cómo crear la tabla “trucks” usando Python.

Para crear una tabla, usamos la función:

ibm_db.exec_inmediate

Los parámetros de la función son:

- connection
  - Que es un recurso de conexión válido retornado desde la función ibm_dbconnect o la función ibm_dbpconnect.
- statement
  - Un string que contiene la declaración.
- options

- Parámetro opcional que incluye un diccionario que especifica el tipo de cursor a retornar para conjuntos de resultados.

Aquí el código para **crear una tabla** llamada “trucks”.

```
stmt = ibm_db.exec_immediate(conn,
"CREATE TABLE Trucks(
serial_no varchar(20) PRIMARY KEY NOT NULL,
model VARCHAR(20) NOT NULL,
manufacturer VARCHAR(20) NOT NULL,
Engine_size VARCHAR(20) NOT NULL,
Truck_Class VARCHAR(20) NOT NULL)“
)
```

Usamos la función ibm_dbexec_inmediate de la ibm_dbapi.

- El recurso de conexión que fue creado pasa el primer parámetro a esta función.
- El próximo parámetro es la sentencia, que es el create table utilizado para crear la tabla.

La nueva tabla creada tendrá 5 columnas, y serial_no será la clave primaria.
Ahora veamos cómo **cargar datos**.

```
stmt = ibm_db.exec_immediate(conn,
"INSERT INTO Trucks(serial_no, model,manufacturer,Engine_size, Truck_Class)
VALUES('A1234','Lonestar','International Trucks','Cummins ISX15','Class 8');")
```

Usamos la función ibm_dbexec_inmediate de la ibm_dbapi.

- El recurso de conexión que fue creado antes se pasa como primer argumento.
- El siguiente parámetro es la sentencia, que es el insert para insertar datos en la tabla “trucks”.

Una nueva fila es agregada a la tabla “trucks”.
De modo similar, podemos agregar más filas.
Ahora veamos cómo **recuperar datos.**

```
In [10]: stmt = ibm_db.exec_immediate(conn,"SELECT * FROM Trucks")

         ibm_db.fetch_both(stmt)

Out[10]: {0: 'A1234',
          1: 'Lonestar',
          'MANUFACTURER': 'International Trucks',
          3: 'Cummins ISX15',
          'SERIAL_NO': 'A1234',
          'ENGINE_SIZE': 'Cummins ISX15',
          'MODEL': 'Lonestar',
          'TRUCK_CLASS': 'Class 8',
          2: 'International Trucks',
          4: 'Class 8'}
```

Nuevamente usamos la función ibm_dbexec_inmediate de la ibm_dbapi.

- El recurso de conexión que fue creado antes se pasa como primer argumento.
- El próximo parámetro es la sentencia, que es el select.

El código Python retorna la salida, que muestra los datos.
Puede chequear si la salida retornada es correcta mirando la consola de DB2 Warehouse.
Veamos ahora cómo usar Pandas para recuperar datos de las tablas. Pandas es una librería que contiene estructuras de datos de alto nivel y herramientas de manipulación diseñadas para realizar análisis de datos de forma rápida y sencilla con Python.

Cargamos los datos desde la tabla trucks en un data frame llamado DF. Un data frame representa una hoja de cálculo tabular, como una estructura de datos, que contiene una colección ordenada de columnas, pudiendo tener cada una un tipo de valor diferente.

```
In [19]: import pandas
         import ibm_db_dbi
         pconn = ibm_db_dbi.Connection(conn)
         df = pandas.read_sql('SELECT * FROM Trucks', pconn)
         df
```

Out[19]:

	SERIAL_NO	MODEL	MANUFACTURER	ENGINE_SIZE	TRUCK_CLASS
0	A1234	Lonestar	International Trucks	Cummins ISX15	Class 8
1	B5432	Volvo VN	Volvo Trucks	Volvo D11	Heavy Duty Class 8
2	C5674	Kenworth W900	Kenworth Truck Co	Caterpillar C9	Class 8

21. ANALIZANDO DATOS CON PYTHON

Veremos conceptos básicos relacionados a realizar análisis exploratorio sobre datos.

Estudiaremos en un ejemplo cómo almacenar datos usando DB2 Warehouse en la nube y luego usaremos Python para realizar análisis de datos.

Usaremos el menú nutricional de MacDonalds como datos.

Aunque McDonads es conocido por sus productos de comida rápida, como hamburguesas y papas fritas, la compañía ha agregado a su menú ensaladas, pescado, smoothies y fruta.

McDonalds le brinda un análisis de nutrición a sus clientes para ayudarlos a balancear su comida.

El dataset utilizado ha sido obtenido de Kaggle.

Necesitamos crear una tabla en Db2 Warehouse para almacenar el dataset de menú de información nutricional que usaremos.

Usaremos la consola provista por Db2 Warehouse.

Hay 4 pasos involucrados en cargar datos en una tabla:

- Source (fuente)
- Target (objetivo)
- Define (definir)
- Finalize (finalizar)

Primero cargamos la hoja de cálculo en la consola. Luego seleccionamos el esquema objetivo y se le dará la opción de cargar los datos en una tabla existente o crear una nueva. Cuando elige crear una nueva, tiene la opción de especificar el nombre de la tabla. Luego verá una previa de los datos, donde también puede definir columnas y data types. Revise la configuración y comience a cargar. Cuando termina la carga, puede ver estadísticas de los datos.

Db2 Warehouse le permite analizar datos usando analíticas in-database, APIs, Rstudio o Python.

Los datos han sido cargados en una base relacional.

Puede ejecutar scripts Python que recuperen datos desde y escriban en la base de datos. Puede usar los scripts para generar modelos estadísticos basados en los datos en su base y

graficar los resultados de estos modelos. Aquí usaremos scripts Python que se ejecutarán dentro de una Jupyter Notebook.

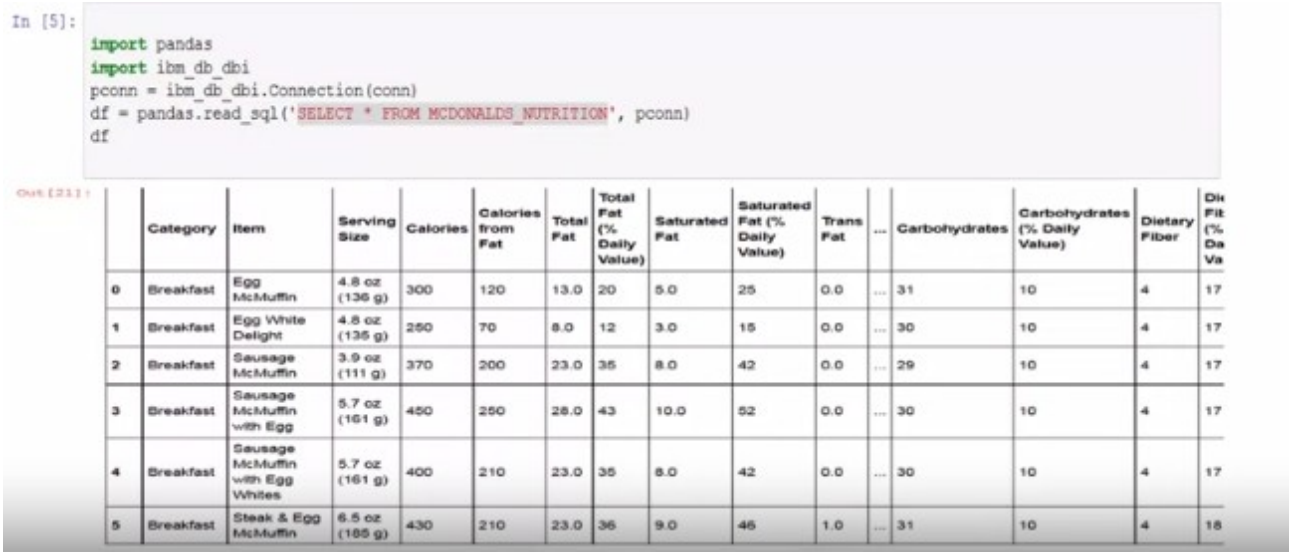
Luego de obtener un recurso de conexión, conectándose a la base de datos, usando el método de conexión ibm_dbapi, usamos la sentencia SQL select count para verificar el número de filas que han sido cargadas en la tabla creada.

La figura siguiente muestra una captura de la salida.

```
In [ ]: ### Verify Loaded Data Using SQL

In [7]: stmt = ibm_db.exec_immediate(conn,"SELECT count(*) FROM MCDONALDS_NUTRITION")

        ibm_db.fetch_both(stmt)

Out[7]: {0: '260', '1': '260'}
```

La salida obtenida es de 260, como el número de filas que se ven en la consola de Db2 Warehouse.

Veamos cómo usar Pandas para recuperar datos de las tablas



```
In [5]:
import pandas
import ibm_db_dbi
pconn = ibm_db_dbi.Connection(conn)
df = pandas.read_sql('SELECT * FROM MCDONALDS_NUTRITION', pconn)
df
```

Out[21]:

	Category	Item	Serving Size	Calories	Calories from Fat	Total Fat	Total Fat (% Daily Value)	Saturated Fat	Saturated Fat (% Daily Value)	Trans Fat	...	Carbohydrates	Carbohydrates (% Daily Value)	Dietary Fiber	Di Fi (% Da Va
0	Breakfast	Egg McMuffin	4.8 oz (136 g)	300	120	13.0	20	5.0	25	0.0	...	31	10	4	17
1	Breakfast	Egg White Delight	4.8 oz (135 g)	250	70	8.0	12	3.0	15	0.0	...	30	10	4	17
2	Breakfast	Sausage McMuffin	3.9 oz (111 g)	370	200	23.0	35	8.0	42	0.0	...	29	10	4	17
3	Breakfast	Sausage McMuffin with Egg	5.7 oz (161 g)	450	250	28.0	43	10.0	52	0.0	...	30	10	4	17
4	Breakfast	Sausage McMuffin with Egg Whites	5.7 oz (161 g)	400	210	23.0	35	8.0	42	0.0	...	30	10	4	17
5	Breakfast	Steak & Egg McMuffin	6.5 oz (185 g)	430	210	23.0	36	9.0	46	1.0	...	31	10	4	18

Cargamos los datos en un data frame DF usando el método read_sql. A este método se le pasan:

- La consulta SQL
- El objeto de conexión

Podemos ver las primeras filas del data frame que creamos.

Ahora es tiempo de aprender acerca de los datos.

Pandas está equipado con métodos estadísticos y matemáticos.

Usemos el método “describe” para ver un resumen estadísticos de los datos.

```
In [34]: df.describe(include='all')
```

Out[34]:

	Category	Item	Serving Size	Calories	Calories from Fat	Total Fat	Total Fat (% Daily Value)	Saturated Fat	Saturated Fat (% Daily Value)	Trans Fat	...	Carbohydrates	Ca (% Va
count	260	260	260	260.000000	260.000000	260.000000	260.000000	260.000000	260.000000	260.000000	...	260.000000	26
unique	9	260	107	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	Na
top	Coffee & Tea	Nonfat Caramel Mocha (Large)	16 fl oz cup	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	Na
freq	95	1	45	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	Na
mean	NaN	NaN	NaN	368.269231	127.096154	14.165385	21.815385	6.007692	29.965385	0.203846	...	47.346154	15
std	NaN	NaN	NaN	240.269886	127.875914	14.205998	21.885199	5.321873	26.639209	0.429133	...	28.252232	9.4
min	NaN	NaN	NaN	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.0
25%	NaN	NaN	NaN	210.000000	20.000000	2.375000	3.750000	1.000000	4.750000	0.000000	...	30.000000	10
50%	NaN	NaN	NaN	340.000000	100.000000	11.000000	17.000000	5.000000	24.000000	0.000000	...	44.000000	15
75%	NaN	NaN	NaN	500.000000	200.000000	22.250000	35.000000	10.000000	48.000000	0.000000	...	60.000000	20
max	NaN	NaN	NaN	1880.000000	1060.000000	118.000000	182.000000	20.000000	102.000000	2.500000	...	141.000000	47

Vemos:

- Que hay 260 observaciones o ítems de comida.
- Que hay 9 categorías únicas de ítems de comida.
- Un resumen estadístico, con información como la frecuencia, mediana, desviación estándar, etc. para los 260 ítems a para diferentes variables.
    - Por ejemplo, el valor máximo para grasa (fat) es 118.

Investiguemos estos datos un poco más allá.

Intentemos entender uno de los nutrientes en los ítems de comida, el sodio.

- Una fuente importante de sodio es la sal de mesa.

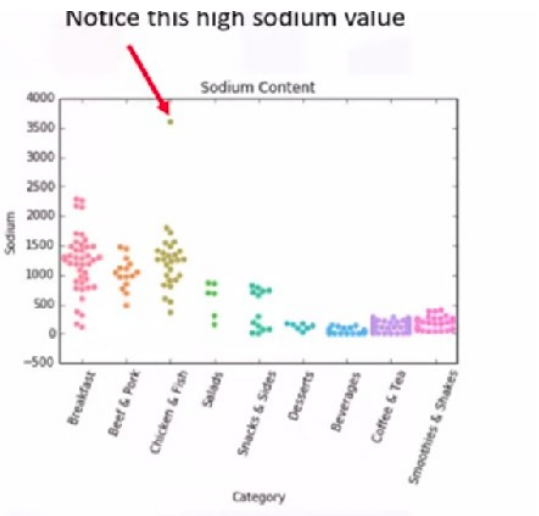
- Un americano promedio come 5 o más cucharadas de salta cada día. Esto cerca de 20 veces más que lo que el cuerpo necesita.
- El sodio se encuentra naturalmente en los alimentos, pero mucho de él es agregado durante el procesamiento y la preparación.
- Muchas comidas no tienen un sabor salado, pero son altas en sodio.
- Grandes cantidades de sodio pueden estar escondidas en enlatados y procesados.
- El sodio controla el balance de fluidos en nuestro cuerpo y mantiene el volumen y la presión en la sangre.
- Comer mucho sodio puede elevar la presión sanguínea y provocar retención de fluidos, lo que puede llevar a problemas de salud.
- Cuando se limita el sodio en una dieta, lo ideal es comer menos de 2000 miligramos por día.

Ahora, usando el dataset de McDonald's hagamos magia para responder la pregunta:

Qué alimento tiene el máximo contenido de sodio?

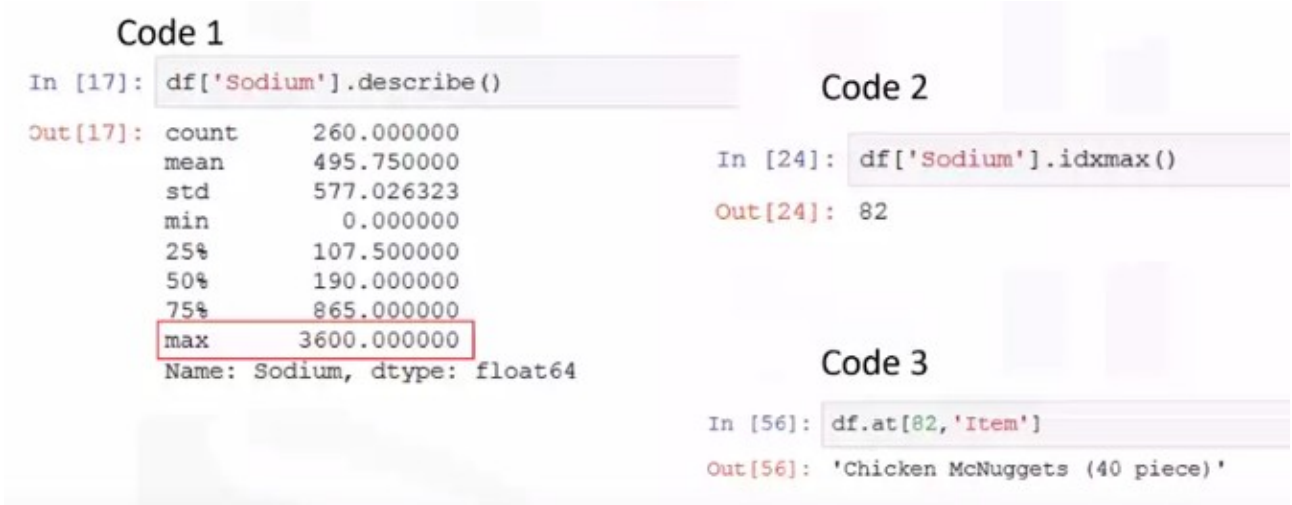
Primero usamos visualización para explorar el contenido de sodio de los alimentos. Usamos el método swarm provisto por el paquete Seaborn; con él podemos podemos crear un **gráfico de dispersión categórico,** con las categorías en el eje X, y el sodio en el eje Y.

```
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

### Categorical scatterplots

plot = sns.swarmplot(x="Category", y='Sodium', data=df)
plt.setp(plot.get_xticklabels(), rotation=70)
plt.title('Sodium Content')
plt.show()
```



La figura muestra así los valores de sodio para diferentes alimentos por categoría.

Observe que hay un valor muy alto, de alrededor de 3600. Exploremos un poco más este valor tan alto e identifiquemos para qué alimentos en el menú se da. Usemos Python para encontrar qué alimentos en el menú tienen el máximo contenido de sodio.



Para chequear los valores de los niveles de sodio, usamos el código que en la figura anterior se muestra como “Code 1”. El método describe se usa para entender el resumen de estadísticas asociadas con el sodio. Observe que el valor máximo del sodio es de 3600.

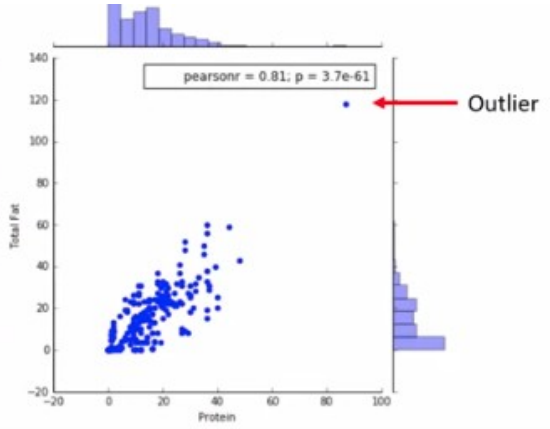
Ahora exploremos la fila asociada con el valor máximo de sodio como se muestra en “Code 2”. Usamos el método idmax para computar los valores de los índices para los cuales se da el valor máximo de sodio. Vemos que la salida es 82.

Ahora encontremos el nombre del ítem asociado con el ítem 82 del data frame. Utilizamos el “Code 3”. Así, obtenemos que el ítem en el menú que tiene el mayor contenido de sodio es “Chicken McNuggets, 40 pieces”.

Las visualizaciones son muy útiles para el análisis exploratorio de datos inicial. Pueden ayudarnos a entender relaciones, patrones y outliers en los datos.

Creemos un gráfico de dispersión con las proteínas en el eje X y la grasa total en las Y. Los **gráficos de dispersión** muestran las relaciones entre 2 variables con un punto para cada observación. Para hacer esto podemos usar la función jointplot provista por el paquete Seaborn.

```
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

plot = sns.jointplot(x="Protein", y='Total Fat', data=df)
plot.show()
```



El gráfico tiene una forma interesante. Muestra la correlación entre las 2 variables: proteína y grasa. La **correlación** es una medida de la asociación entre 2 variables, y tiene un valor entre 1 y -1. Los puntos en el gráfico de dispersión están cercanos a una línea recta en la dirección positiva, por lo que tenemos una correlación positiva entre las 2 variables.

Arriba en el gráfico, vemos los valores de la correlación de Pearson: 0.81 y de la significancia de la correlación, denotada como Pwhich, tiene un valor que muestra que las variables están ciertamente correlacionadas.

El gráfico muestra 2 histogramas: uno en la parte superior y el otro del lado derecho. El histograma en la parte superior corresponde a la variable “protein” y el de la derecha a “fat”.

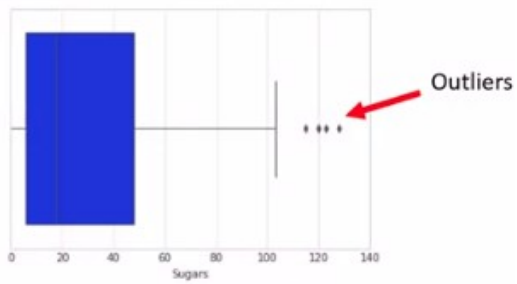
También notamos que hay un punto en el gráfico fuera del patrón general. Es un posible outlier.

Ahora veamos cómo podemos visualizar los datos con diagramas de caja (box plots).

Los **diagramas de caja** son gráficos que indican la distribución de una o más variables. A través de las líneas y puntos podemos encontrar asimetrás y outliers.

Creemos un diagrama de caja para el azúcar. Usaremos la función box plot del paquete Seaborn.

```
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

plot = sns.set_style("whitegrid")
ax = sns.boxplot(x=df["Sugars"])
plot.show()
```



Vemos que el valor promedio es de azúcar es de 30 gramos. También detectamos posibles outliers que indican alimentos con valores extremos de azúcar. Existen alimentos que tienen un contenido de azúcar de 128 gramos. Los caramelos pueden estar entre ellos.

22. JOIN OVERVIEW

Veremos cómo hacer un join entre 2 tablas.

Un select simple recupera datos de una o más columnas de una única tabla. El próximo nivel de complejidad es recuperar datos de 2 o más tablas, lo que lleva a múltiples posbilidades acerca de cómo el conjunto resultado es generado.

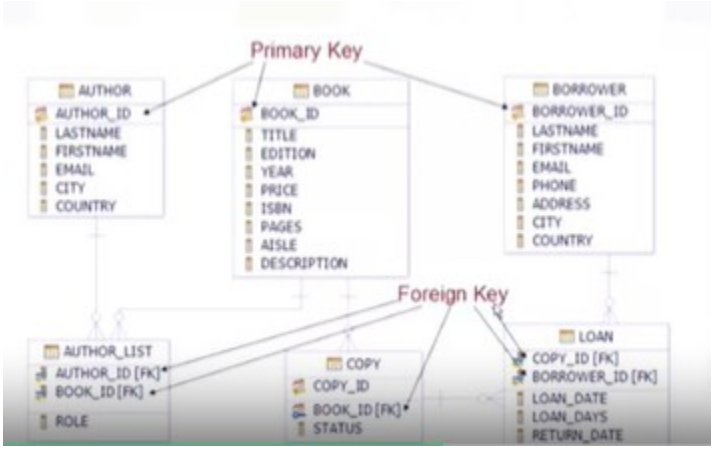
Para combinar datos de 2 tablas, usamos el operador JOIN. Un JOIN combina las filas de 2 o más tablas basado en cierta relación entre ellas.

Consideremos las entidades book y author. El diagrama ER siguiente representa el modelo relacional para las entidades book y author así como para otras entidades, tales como borrower, loan, copy y author_list.

En este modelo, la información se divide en diferentes tablas. Si quiere saber a qué prestatario (borrower) tiene una copia de un libro a préstamo (loan) debe obtener información de 3 tablas: borrower, loan y copy. Aquí es dónde necesita un JOIN.

Lo primero que necesita es identificar la relación entre las tablas. Esto es, qué columna o columnas en cada tabla serán utilizadas como enlace entre las tablas.

En el siguiente diagrama, observe que author_ID, book_ID, borrower_ID y copy_ID tienen el ícono de clave primaria.



Una clave primaria identifica de forma única cada fila en una tabla.

Observe que algunos atributos tienen marcado [FK] a su costado. Esto identifica una clave foránea (Foreign Key). Una clave foránea es un conjunto de columnas que hace referencia a una clave primaria en otra tabla.

Entonces, si quiere saber qué prestatario tiene un libro a préstamo, necesita reunir información de las tablas borrower y loan. Necesitará borrower_ID en ambas tablas.

SQL ofrece diferentes tipos de joins, dependiendo de lo que se busque. Por ejemplo, puede extraer los datos que corresponden a la intersección de ambas tablas o seleccionar la combinación de todos los datos de las tablas involucradas.

22.1. INNER JOIN

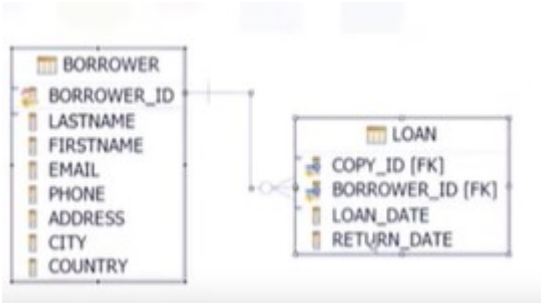
Un join combina las filas de 2 o más tablas basado en relaciones entre las columnas de esas tablas.

Hay 2 tipos de joins:

- Inner joins

- Outer joins

Un **inner join** coincide los resultados de 2 o más tablas y despliega solamente el conjunto resultado que coincide con el criterio especificado en la consulta.
Trabajaremos con el modelo de la biblioteca.
Si queremos chequear los nombres de las personas que tienen un libro prestado, observamos que la información se divide en 2 tablas. Debemos encontrar los nombres de las personas de la tabla borrower que están listados en la tabla loan. Debemos identificar las relaciones entre las tablas, para hacerlo miramos sus columnas. Aquí lo hacemos coincidiendo por borrower_ID, ya que dicha columna existe en la tabla borrower como clava primaria y en la tabla loan como clave foránea.



Usamos el operador inner join para hacer coincidir las ocurrencias de borrower_ID en la tabla borrower con las de borrower_ID en loan.
La consulta es la siguiente:

```
SELECT B.BORROWER_ID, B.LASTNAME, B.COUNTRY,
    L.BORROWER_ID, L.LOAN_DATE
FROM BORROWER B INNER JOIN LOAN L
    ON B.BORROWER_ID = L.BORROWER_ID
```

En la cláusula FROM identificamos la tabla borrower como B y la tabla loan como L.

Seleccionamos ciertos campos con la condición de que borrower_ID en la tabla borrower sea igual al borrower_ID en la tabla loan.
Observe los prefijos en las columnas, ya sean B o L. En SQL se les llama alias y brindan una escritura más sencilla.

Aquí el conjunto resultado:

BORROWER_ID	LASTNAME	COUNTRY	BORROWER_ID	LOAN_DATE
D1	SMITH	CA	D1	11/24/2010
D2	SANDLER	CA	D2	11/24/2010
D3	SOMMERS	CA	D3	11/24/2010
D4	ARDEN	CA	D4	11/24/2010
D5	XIE	CA	D5	11/24/2010

Se muestran las filas de ambas tablas que tienen el mismo borrower_ID.
Hasta ahora vimos cómo combinar 2 tablas, qué pasa si debemos combinar 3 o más tablas?
Simplemente se agregan más joins a la sentencia.
Por ejemplo, queremos saber qué prestatarios tienen un libro prestado, pero también qué copia del libro tienen. Así es cómo se relacionan las 3 tablas:

En este caso, hacemos un join de las tablas 2 a 2. Primero hacemos un join entre las tablas borrower y loan donde coinciden el borrower_ID, y luego entre las tablas loan y copy donde coinciden copy_ID.

```
SELECT B.LASTNAME, L.COPY_ID, C.STATUS
FROM BORROWER B
    INNER JOIN LOAN L ON B.BORROWER_ID = L.BORROWER_ID
    INNER JOIN COPY C ON L.COPY_ID = C.COPY_ID
```

Aquí el resultado:

LASTNAME	COPY_ID	STATUS
SMITH	C1	LOANED
SANDLER	C6	LOANED
SOMMERS	C7	LOANED
ARDEN	C8	LOANED
XIE	C13	LOANED

22.2. LEFT OUTER JOIN

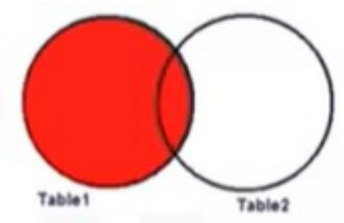
Un outer join es una forma especializada de join.

Hay 3 tipos de outer join:

- Left
- Right
- Full

Aquí estudiaremos el Left Outer Join o simplemente Left Join.

Observe el siguiente diagrama:



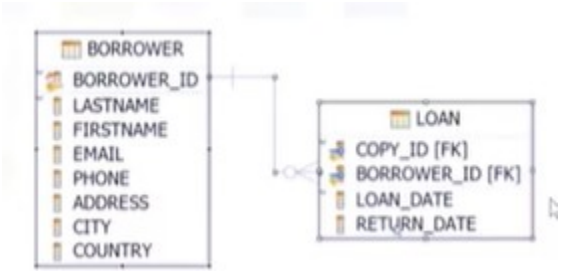
Los términos izquierda y derecha hacen referencia a la tabla en el lado izquierdo y derecho del diagrama respectivamente.

En este diagrama, la tabla1 es la izquierda.

Un **left join** (join izquierdo) hace coincidir los resultados de 2 tablas y despliega todas las filas de la tabla izquierda y combina la información con las filas de la tabla derecha que coinciden con el criterio especificado en la consulta.

Volvamos al ejemplo de la biblioteca.

Queremos chequear el status de todas las personas para ver qué libros tienen prestados. Esta información se divide en 2 tablas: Borrower y Loan. Estas tablas tienen la columna Borrower_ID en común, que es PK (Primary Key) en Borrower y FK en Loan.

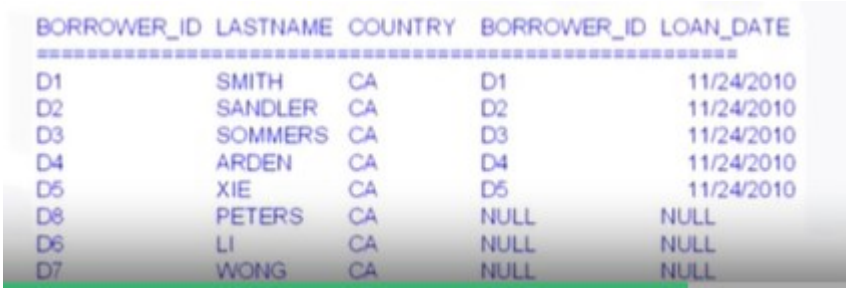


En un Outer Join, la primera tabla especificada en la cláusula FROM es referida como la tabla izquierda y la restante tabla como derecha.

La consulta es la siguiente:

```
SELECT B.BORROWER_ID, B.LASTNAME, B.COUNTRY,
    L.BORROWER_ID, L.LOAN_DATE
FROM BORROWER B LEFT JOIN LOAN L
    ON B.BORROWER_ID = L.BORROWER_ID
```

En este ejemplo, la tabla borrower es la primera tabla especificada en la cláusula FROM, por lo que es la tabla izquierda, y la tabla loan es la derecha.
En la cláusula FROM, borrower es listada en el lado izquierdo del operador join, por tanto se seleccionarán todas las filas de la tabla borrower y se combinarán con los contenidos de la tabla loan basado en el criterio especificado en la consulta.
Aquí el resultado:



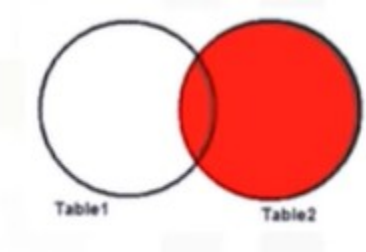
BORROWER_ID	LASTNAME	COUNTRY	BORROWER_ID	LOAN_DATE
D1	SMITH	CA	D1	11/24/2010
D2	SANDLER	CA	D2	11/24/2010
D3	SOMMERS	CA	D3	11/24/2010
D4	ARDEN	CA	D4	11/24/2010
D5	XIE	CA	D5	11/24/2010
D8	PETERS	CA	NULL	NULL
D6	LI	CA	NULL	NULL
D7	WONG	CA	NULL	NULL

El resultado muestra cada Borrower_ID de la tabla borrower y el “loan date” para ese prestatario.
Observe que hay un loan_date para las primeras 5 filas: D1 a D5, sin embargo, para las últimas 3 muestra valores null. Un valor null indica un valor desconocido. Cuando usamos un left join, si en la tabla derecha no existe un valor correspondiente, se retorna null. En este ejemplo, los prestatarios Peters, Li y Wong, cuyos borrower_IDs son D8, D6 y D7 nunca sacaron un libro a préstamo, por tanto no hay un borrower_ID correspondiente en la tabla loan. El left join despliega todas las filas de la tabla izquierda (borrower en este caso) y combina la información con las de la tabla derecha (loan). La tabla borrower contiene filas para los prestatarios D8, D6 y D7, pero la tabla loan no, así, el resultado muestra valores null para ellos.

22.3. RIGHT OUTER JOIN

Estudiaremos el right outer join, o simplemente right join.

Mirando el diagrama siguiente, los términos izquierda y derecha, hacen referencia a las tablas del lado izquierdo y derecho respectivamente. En el diagrama, la tabla 1 es la izquierda.



Un right join hace coincidir los resultados de 2 tablas, y despliega todas las filas de la tabla derecha combinadas con información de la tabla izquierda que coinciden con el criterio especificado en la consulta.

El conjunto resultado de un right join son todas las filas de ambas tablas que coinciden con el criterio especificado en la consulta más aquellas que no coinciden de la tabla derecha.

Volvamos al modelo de la biblioteca.

Queremos conocer el estado de todos los libros en préstamo. Esta información se divide en 2 tablas, “loan” que incluye los borrowers ID y loan_date y “borrower” que incluye los borrowers ID, first_name y last_name. El enlace entre ambas tablas es borrower_ID, que es PK en borrower y FK en loan.

En un outer join, la primera tabla especificada en la consulta es referida como la tabla izquierda y la restante como la derecha.

La consulta es la siguiente:

```
SELECT B.BORROWER_ID, B.LASTNAME, B.COUNTRY,
    L.BORROWER_ID, L.LOAN_DATE
FROM BORROWER B RIGHT JOIN LOAN L
    ON B.BORROWER_ID = L.BORROWER_ID
```

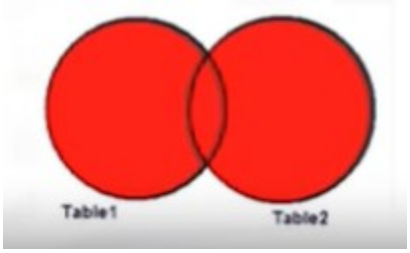
En este ejemplo, borrower es la tabla izquierda y loan la derecha.

En la cláusula FROM, la tabla loan es listada del lado derecho del operador join, por tanto seleccionaremos todas las filas de loan y las combinaremos con los contenidos de borrower basados en el criterio especificado en la consulta.
Aquí el resultado:

BORROWER_ID	LASTNAME	COUNTRY	BORROWER_ID	LOAN_DATE
D1	SMITH	CA	D1	11/24/2010
D2	SANDLER	CA	D2	11/24/2010
D3	SOMMERS	CA	D3	11/24/2010
D4	ARDEN	CA	D4	11/24/2010
D5	XIE	CA	D5	11/24/2010

22.4. FULL OUTER JOIN

Estudiaremos el full outer join o simplemente full join.
La palabra clave full join retorna todas las filas de ambas tablas. Esto es, todas las filas de la tabla izquierda y todas las de la derecha.
Las convenciones izquierda y derecha son las mismas que en los casos anteriores.



Volvamos al modelo de la biblioteca.
Queremos seleccionar todas las filas de la tabla borrower y todas las de la tabla loan. El enlace entre ambas tablas es borrower_ID.
La consulta es la siguiente:

```
SELECT B.BORROWER_ID, B.LASTNAME, B.COUNTRY,
    L.BORROWER_ID, L.LOAN_DATE
FROM BORROWER B FULL JOIN LOAN L
    ON B.BORROWER_ID = L.BORROWER_ID
```

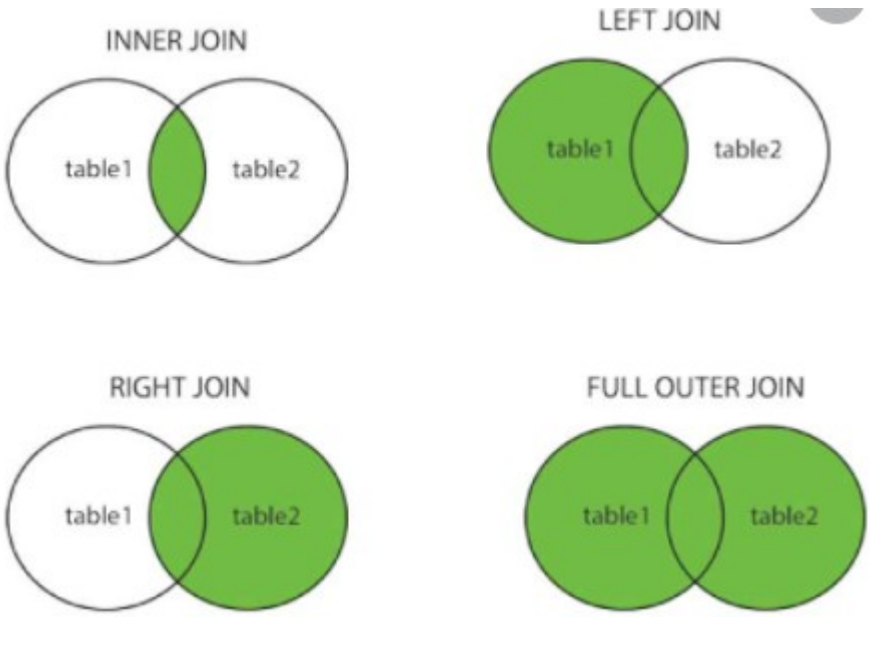
En el full join, seleccionamos todas las filas de la tabla borrower y todas las de loan.
Aquí el resultado:

BORROWER_ID	LASTNAME	COUNTRY	BORROWER_ID	LOAN_DATE
D1	SMITH	CA	D1	11/24/2010
D2	SANDLER	CA	D2	11/24/2010
D3	SOMMERS	CA	D3	11/24/2010
D4	ARDEN	CA	D4	11/24/2010
D5	XIE	CA	D5	11/24/2010
D8	PETERS	CA	NULL	NULL
D6	LI	CA	NULL	NULL
D7	WONG	CA	NULL	NULL

El conjunto resultado muestra todas las filas de borrower y todas las de loan.
Hay 8 filas en total. Todos los registros de borrower son listados con los datos correspondientes en loan. Observe que las últimas 3 filas, borrower_ID y loan_date en la tabla loan no tienen un borrower_ID que les corresponda, por lo que se devuelve null.

22.5. RESUMEN DE JOINS

En el siguiente diagrama se muestran los distintos tipos de joins:



23. TRABAJANDO CON DATASETS REALES

Muchos de los datasets reales están disponibles vía archivos CSV. Estos últimos son archivos que contienen valores típicamente separados por comas, aunque pueden usarse otros separadores.

En esta sección usaremos un archivo llamado DOGS,csv.

Aunque es un dataset ficticio que contiene nombres de perros y sus razas, lo usaremos para ilustrar conceptos que pueden aplicarse con datasets reales.

Un ejemplo de DOGS.csv se muestra aquí:

```
Id,Name of Dog,Breed (dominant breed if not pure breed)
1,Wolfie,German Shepherd
2,Fluffy,Pomeranian
3,Huggy,Labrador
```

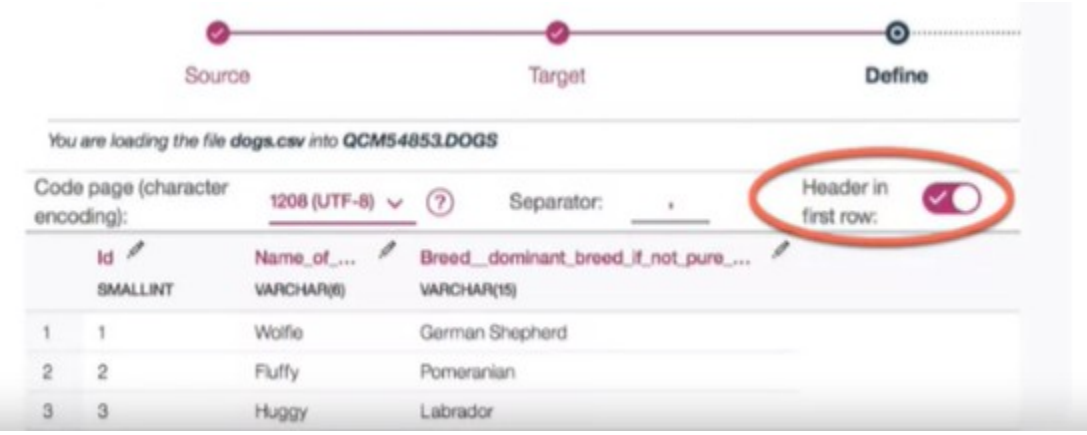
La primera columna contiene etiquetas que se mapean a nombres de columna en una tabla.

En DOGS.csv:

- La primera fila contiene el nombre de 3 atributos:
  - ID: que a su vez tiene los valores 1,2 y 3.
  - Nombre del perro: en este caso son Wolfie, Fluffy y Huggy.
  - La raza: la raza dominante en caso de no ser puro; tiene los valores German Sheperd, Pomeranian y Labrador.

Los archivos CSV pueden tener una primera fila o encabezado que contenga los nombres de los atributos o no.

Si está cargando los datos en la base de datos usando la herramienta visual en la consola, asegúrese que el encabezado en la primera fila esté habilitado. Esto hará un mapeo entre los nombres de atributos de la primer fila del CSV y los nombres de columnas en la tabla de la base de datos, y el resto de las filas en filas de datos en la tabla.



Observe que los nombres de columnas por defecto no siempre son amigables con la base de datos, y si ese es el caso, quizás quiera editarlos antes de que la tabla sea creada.
Ahora hagamos una consulta en la columna de nombres que estén con minúsculas o mixtos (una combinación de minúsuculas y mayúsuculas). Asumamos que cargamos el archivo DOGS.csv usando los nombres de columna por defecto desde el CSV. Si queremos recuperar el contenido de la columna ID usando la consulta:

select id from DOGS

obtendremos un error indicando que id no es válido. Esto es porque el parser de la base de datos asume nombres en mayúscula por defecto, mientras que cuando cargamos el CSV, el nombre de la columna era “Id”, que es un formato mixto.
En este caso, lo que necesitamos es especificar el nombre de la columna entre comillas dobles:

select “Id” from DOGS

Ahora veremos cómo consultar nombres de columnas que tengan espacios y otros caracteres.
En un CSV si el nombre de la columna contiene espacios, por defecto la base de datos los mapea a guiones bajos (_). Por ejemplo, cambiaría “Name of Dog” por Name_of_Dog“.

Otros caracteres especiales, como paréntesis también podrían llegar a mapearse a guiones bajos. Por tanto, cuando escriba una consulta asegúrese de usar un formateo apropiado como se muestra en este ejemplo:

select “Id”,”Name_of_Dog”,”Breed__dominant_breed_if_not_pure_breed_” from dogs

Observe el doble guión bajo entre Breed y dominant.
Es importante notar el guión bajo final luego de la palabra “breed” cerca del final de la consulta. Este se usa en lugar del paréntesis de cierre.
Cuando usa comillas en Jupyter notebooks, podría estar emitiendo consultas en una notebook asignándolas primero a variables Python. En tales casos, si su consulta contiene por ejemplo comillas dobles, para especificar una columna con nombre de tipo mixto, podría diferenciar las comillas usando comillas simples para las variables Python que envuelvan la consulta SQL, y dobles para los nombres de columna. Por ejemplo:

Qué pasa si se necesitan usar comillas simples dentro de la consulta, por ejemplo, para especificar un valor dentro de una cláusula where? En este caso puede usar la barra invertida como carácter de escape:

```
selectQuery = 'select * from dogs
      where "Name_of_Dog"=\'Huggy\' '
```

Si tiene consultas muy largas, podría ser útil dividir la consulta en múltiples líneas para mejorar la legibilidad. En Python notebooks puede usar la barra invertida para indicar la continuación a la siguiente fila como en el ejemplo:

```
%sql select "Id", "Name_of_Dog", \
    from dogs \
    where "Name_of_Dog"='Huggy'
```

Tenga presente que obtendrá un error si divide el código en múltiples líneas en una notebook Python en caso de no utilizar la barra invertida.

Cuando utiliza SQL magic, puede usar %% en la primer línea de una celda en Jupyter Notebooks. Esto implica que el resto del contenido de la celda será interpretado por SQL magic. Por ejemplo:

```
%%sql
select "Id", "Name_of_Dog",
    from dogs
    where "Name_of_Dog"='Huggy'
```

Cuando se utiliza %% sql, la barra invertida no es necesaria al final de cada línea.

Cómo restringimos el número de columnas recuperadas? Es una buena pregunta, ya que una tabla puede contener miles o incluso millones de filas, y usted quizás sólo quiera ver una muestra de los datos o ver algunas columnas para saber qué tipo de datos contiene la tabla. Podría verse tentado a ejecutar “select * from tableName” para recuperar los datos en un dataframe de Pandas y aplicarle la función head a él. Pero esto podría demorar mucho, pues la consulta puede demorar. En su lugar, puede restringir los resultados usando la cláusula limit. Por ejemplo, use la siguiente consulta para recuperar las primeras 3 filas de la tabla census:

select * from census limit 3

24. OBTENIENDO DETALLES DE TABLAS Y COLUMNAS

Cómo obtenemos una lista de tablas en una base de datos?

Los sistemas de bases de datos típicamente contienen un sistema o catálogo de tablas, de dónde puede listar las tablas y obtener sus propiedades. En Db2 el catálogo se llama “syscat tables”, en SQL Server “information schema tables” y en Oracle “all tables” o “user tables”.

Para obtener una lista de tablas en Db2 ejecute:

```
select * from syscat.tables
```

La sentencia anterior puede devolver demasiadas tablas, incluyendo tablas del sistema, por lo que es mejor filtrar el resultado como se muestra aquí (donde debe remplazar ABC12345 con su propio usuario DB2):

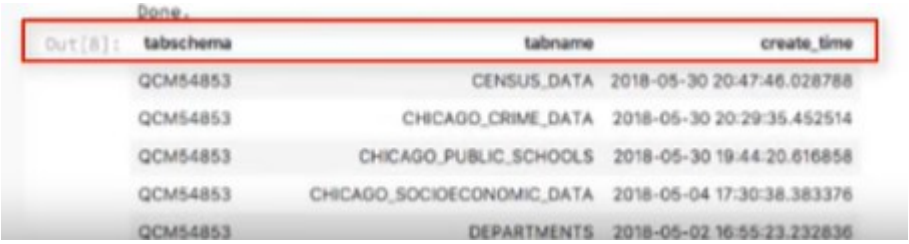
```
select TABSCHEMA, TABNAME, CREATE_TIME
    from syscat.tables
    where tabschema= 'ABC12345'
```

Cuando hace select * from syscat_tables obtiene todas las propiedades de la tabla. A veces, sólo nos interesan algunas, como el tiempo de creación.

Digamos que creó varias tablas con nombres similares, por ejemplo dog1, dog_test, etc. Pero sólo quiere saber cuál de estas tablas fue la última en crearse, puede entonces ejecutar:

```
select TABSCHEMA, TABNAME, CREATE_TIME
    from syscat.tables
    where tabschema='QCM54853'
```

La salida contendrá el nombre del esquema, el nombre de la tabla y el tiempo de creación para todas las tablas en su esquema.



Done.

Out[8]:

tabschema	tabname	create_time
QCM54853	CENSUS_DATA	2018-05-30 20:47:46.028788
QCM54853	CHICAGO_CRIME_DATA	2018-05-30 20:29:35.452514
QCM54853	CHICAGO_PUBLIC_SCHOOLS	2018-05-30 19:44:20.616858
QCM54853	CHICAGO_SOCIOECONOMIC_DATA	2018-05-04 17:30:38.383376
QCM54853	DEPARTMENTS	2018-05-02 16:55:23.232836

Ahora veamos cómo obtener una lista de columnas en una tabla. Puede ejecutar la consulta:

```
select * from syscat.columns
        where tabname = 'DOGS'
```

A saber en MySQL alcanza con ejecutar el comando “show columns from dogs”.
Si quiere saber propiedades específicas como la longitud del datatype, en DB2 puede escribir:

```
select distinct(name), coltype, length
        from sysibm.syscolumns
        where tbname = 'DOGS'
```

Aquí vemos los resultados de recuperar propiedades de una columna, para una tabla real llamada “Chicago_Crime_Data” desde una Jupyter Notebook.

name	coltype	length
Arrest	VARCHAR	5
Beat	SMALLINT	2
Block	VARCHAR	35
Case_Number	VARCHAR	8
Community_Area	DECIMAL	4
Date	VARCHAR	22
Description	VARCHAR	46
District	DECIMAL	4
Domestic	VARCHAR	5