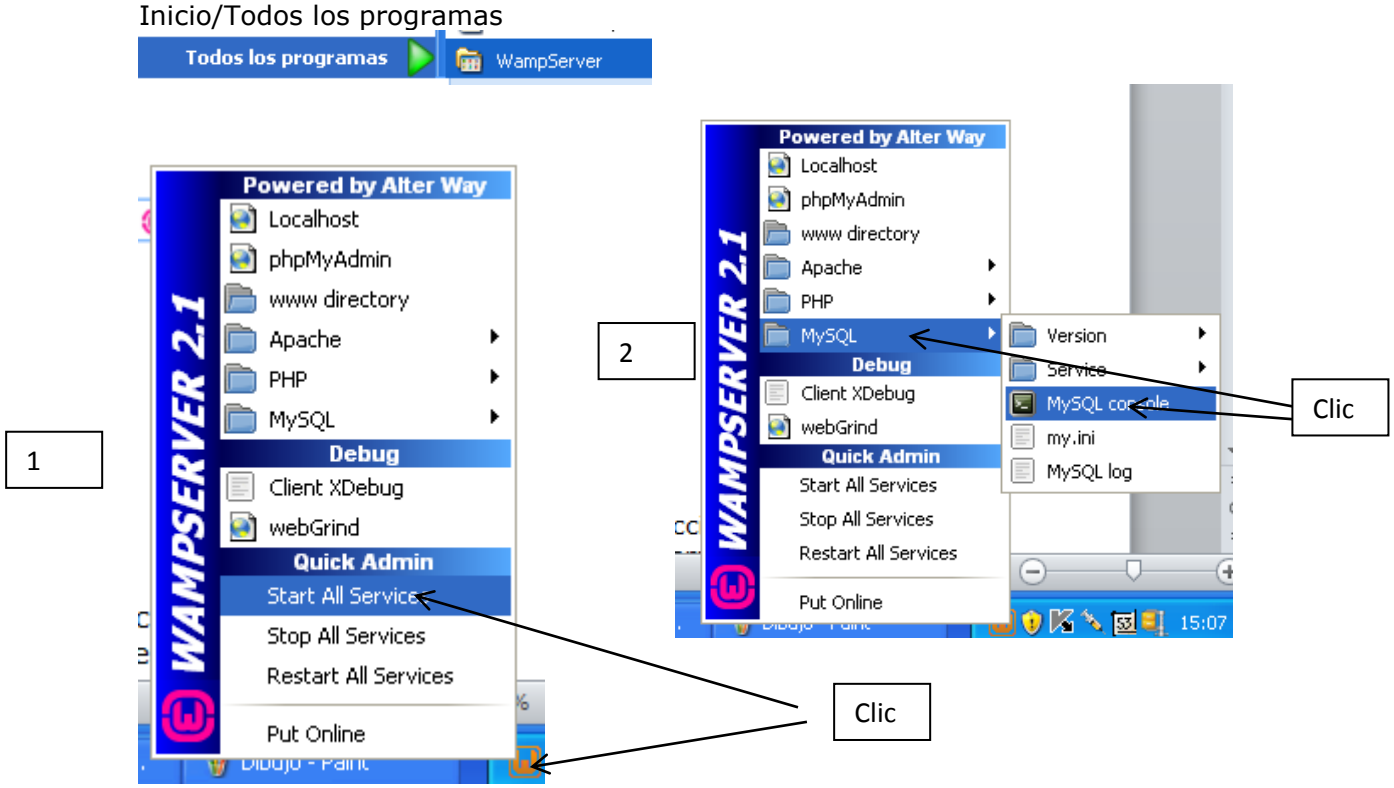
GUIA PARA LA MANIPULACION DE DATOS EN UNA BASE DE DATOS A TRAVÉS DE MYSQL

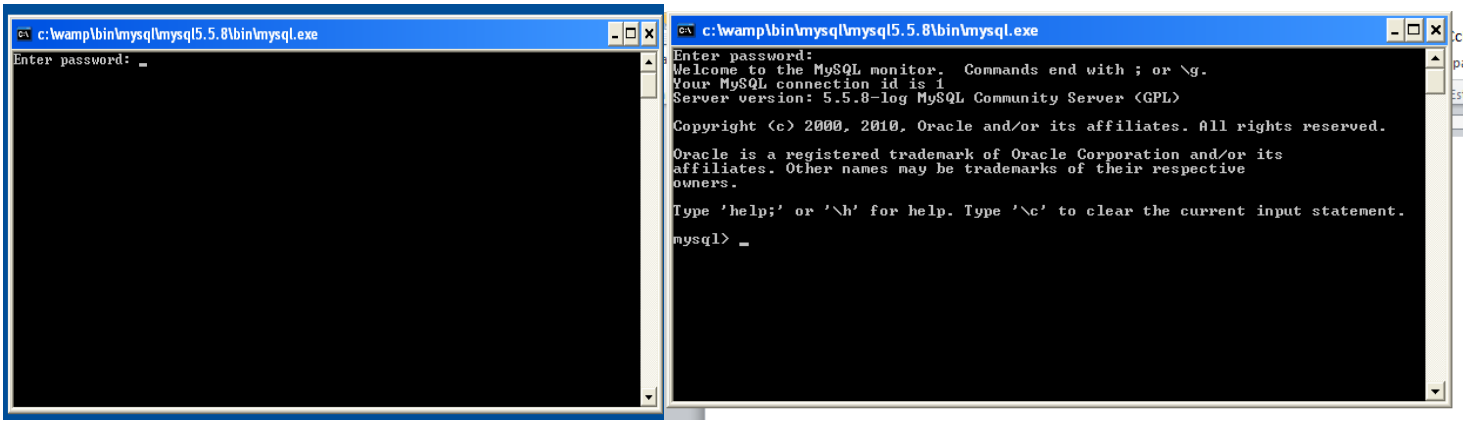
En esta guía continuaremos con el modelamiento de la base de datos a partir del diagrama entidad-relación, además de esto realizaremos la conexión a la base de datos y generaremos la base de datos con las tablas. Con las cuales realizaremos una serie de consultas a los datos establecidos en estas. Para esta actividad usaremos el lenguaje SQL como estrategia de manipulación de datos.

Para empezar debemos usar el aplicativo wampserver de manera que podamos utilizar el computador como un servidor, para esto debemos hacer lo siguiente:

Inicio/Todos los programas



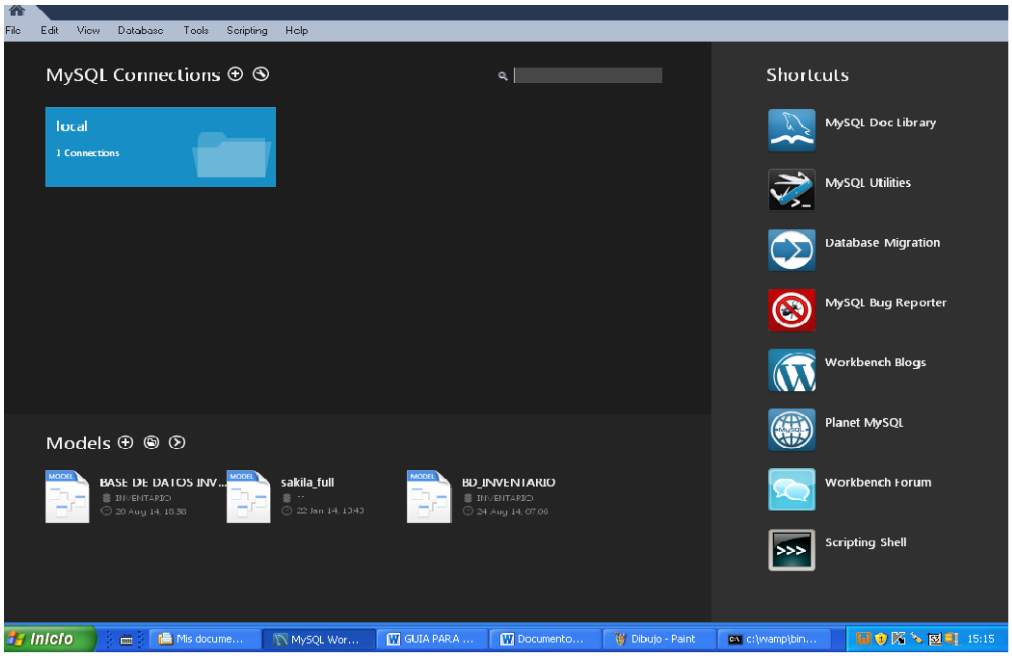

PRESIONAR ENTER EN ESTE PANTALLAZO Y APARECERA LOS SIGUIENTE:




Que indica que ya está conectado como servidor a MySQL.

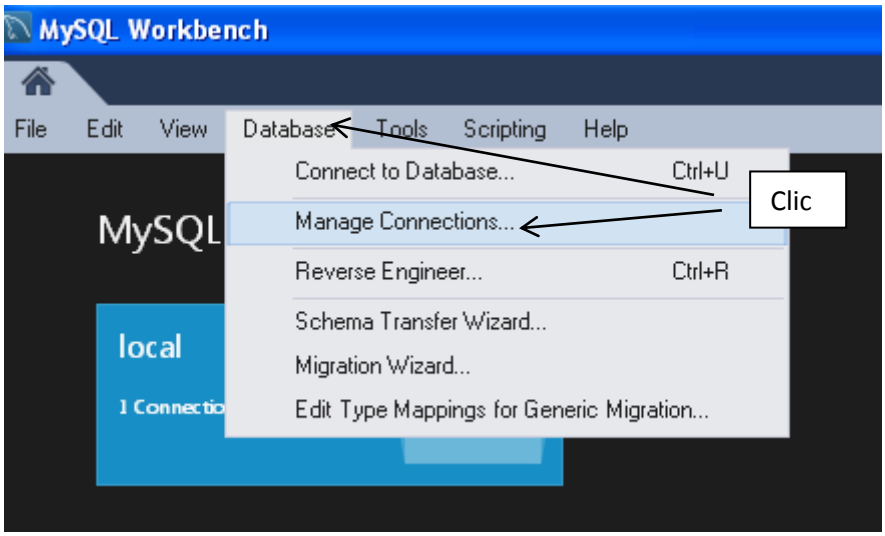
Luego de este procedimiento, pasamos a abrir el aplicativo:

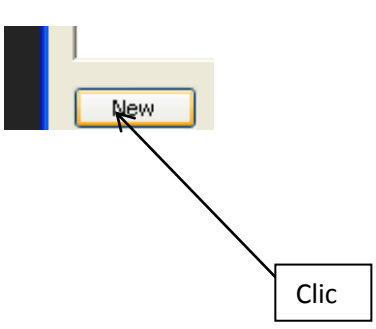
GUIA PARA LA MANIPULACION DE DATOS EN UNA BASE DE DATOS A TRAVÉS DE MYSQL

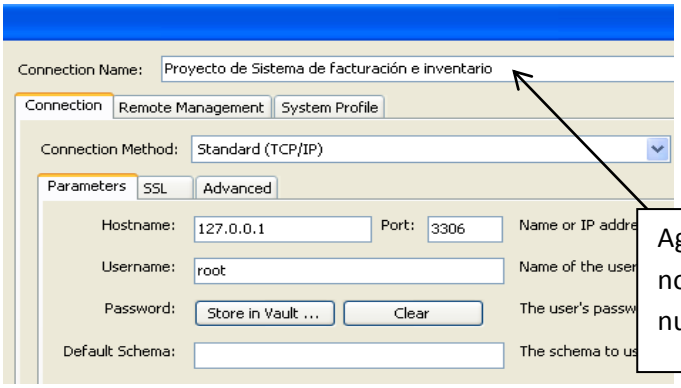


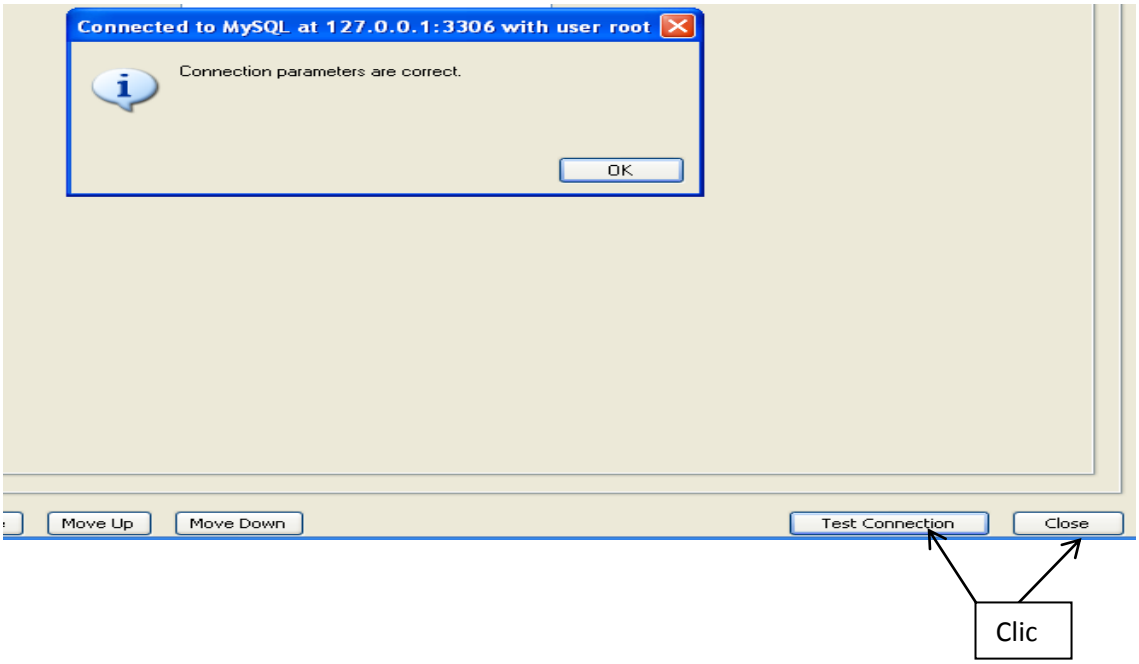

**A**

Para el trabajo de manipulación de datos en la base de Datos **Proyecto de Sistema de facturación e inventario** procedemos a crear una conexión a la base de datos para esto realizamos los siguientes procedimientos:

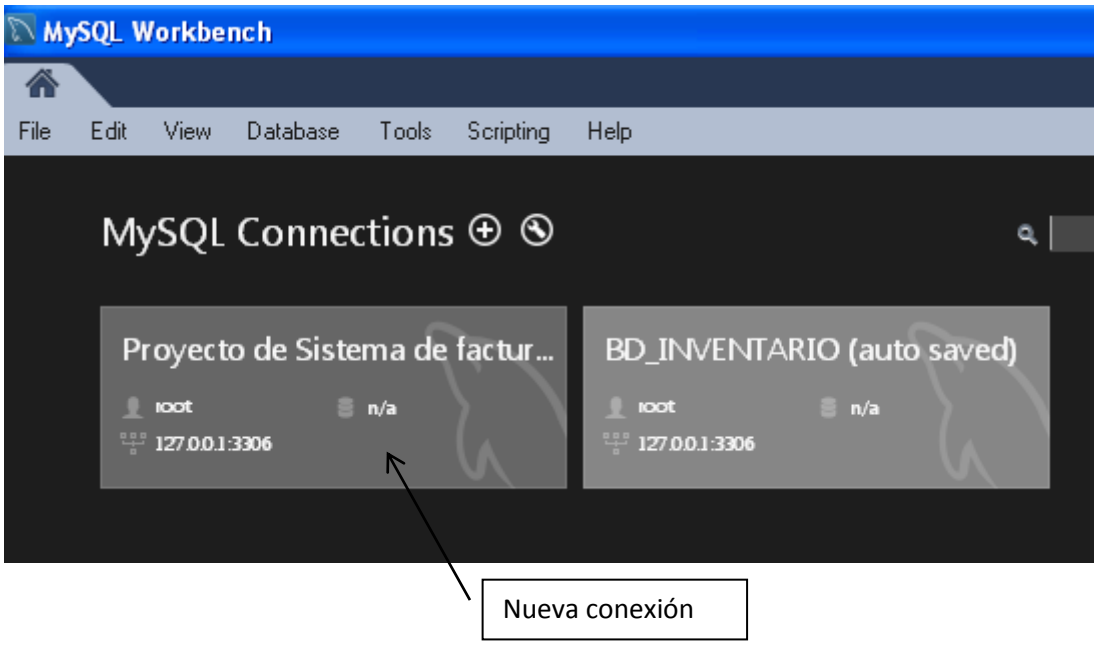


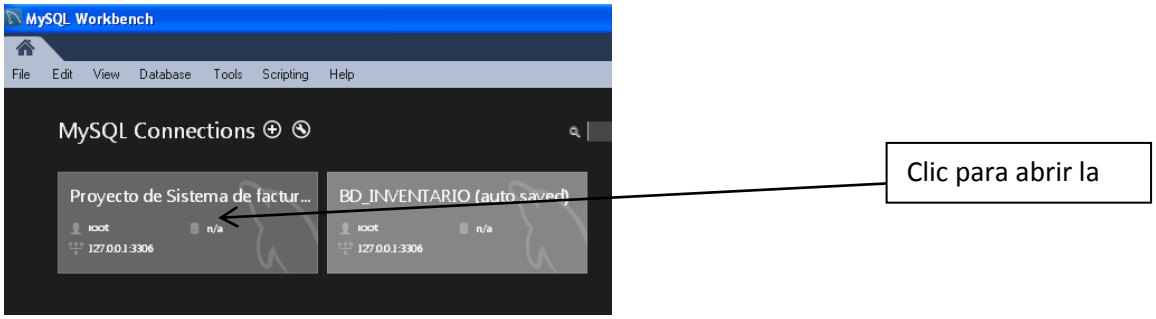



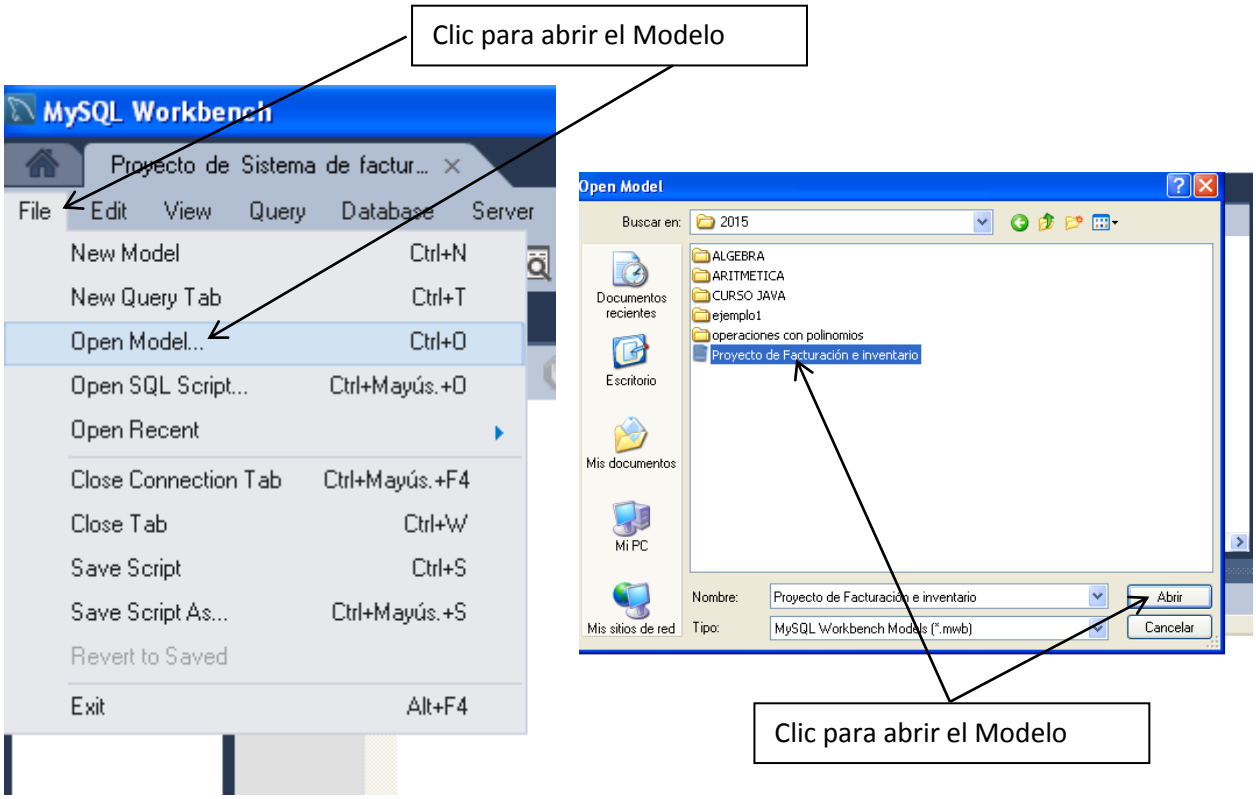



Luego cerramos y abrimos nuevamente el MySQL Workbench y verificamos que aparezca la nueva conexión. Como se muestra a continuación:




Abrimos la conexión Proyecto de Sistema de facturación e inventario y abrimos el modelo como se muestra a continuación:

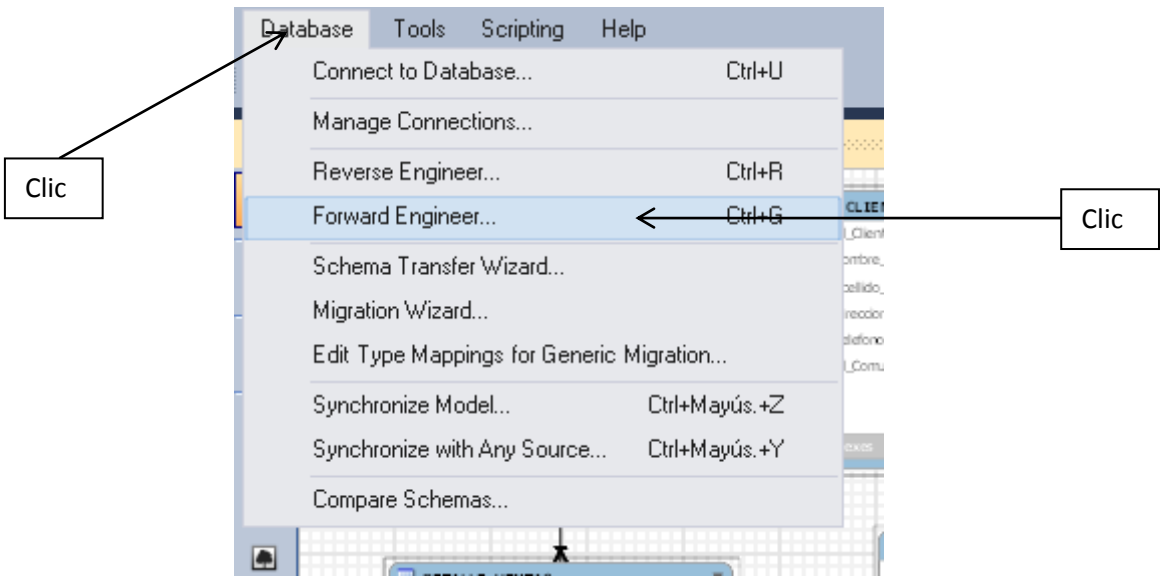




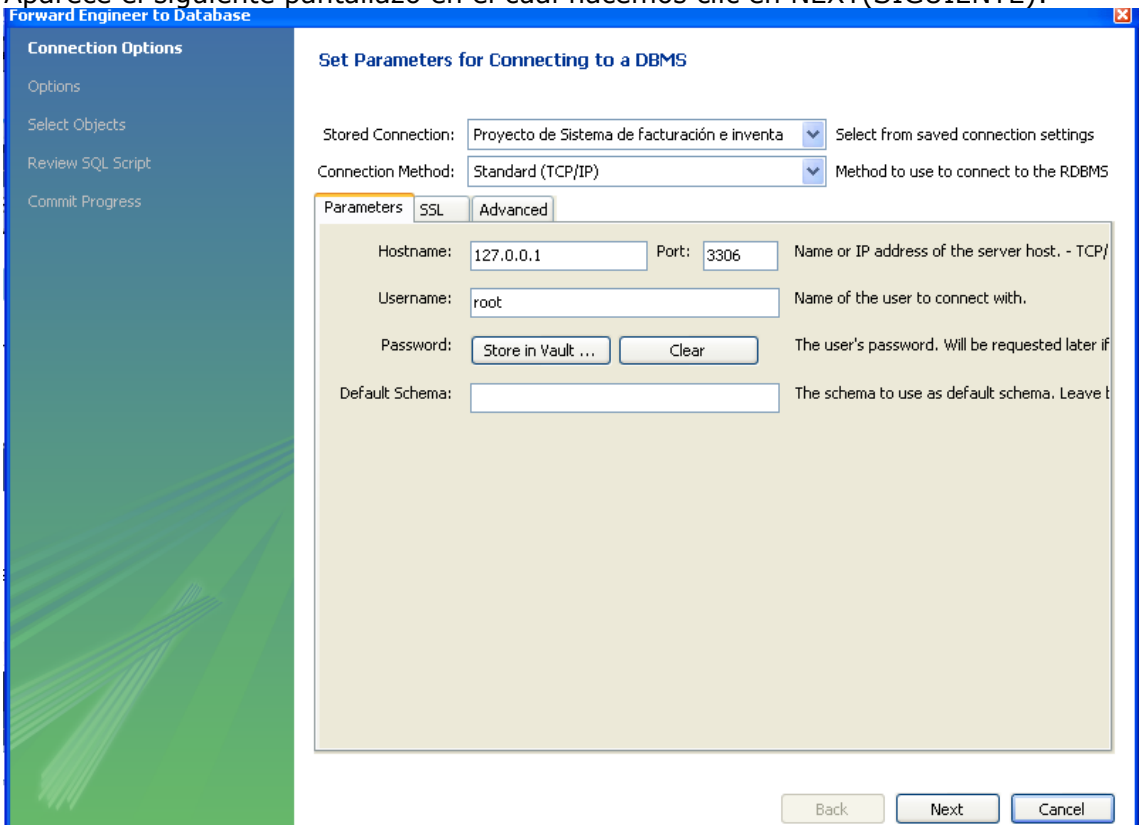

Para la definición de la base de datos utilizamos el MySQL Workbench como un recurso de diseño a través del cual creamos la base de datos y diseñamos el diagrama entidad-relación y realizamos el proceso de inserción de datos en algunas tablas. Todo este proceso fue sencillo, sin la necesidad de escribir instrucciones en SQL como Create---, Insert----, etc. La ventaja de dicho aplicativo es que nos permite generar el código tanto del diagrama entidad relación, como de los registros ingresados a través de la herramienta Insertar.

En este caso necesitamos generar el código o el script del diagrama Entidad-Relación de la Base de Datos Proyecto de Sistema de facturación e inventario. Debido a esto es necesario para poder hacer las consultas o Querys a los datos en las tablas.

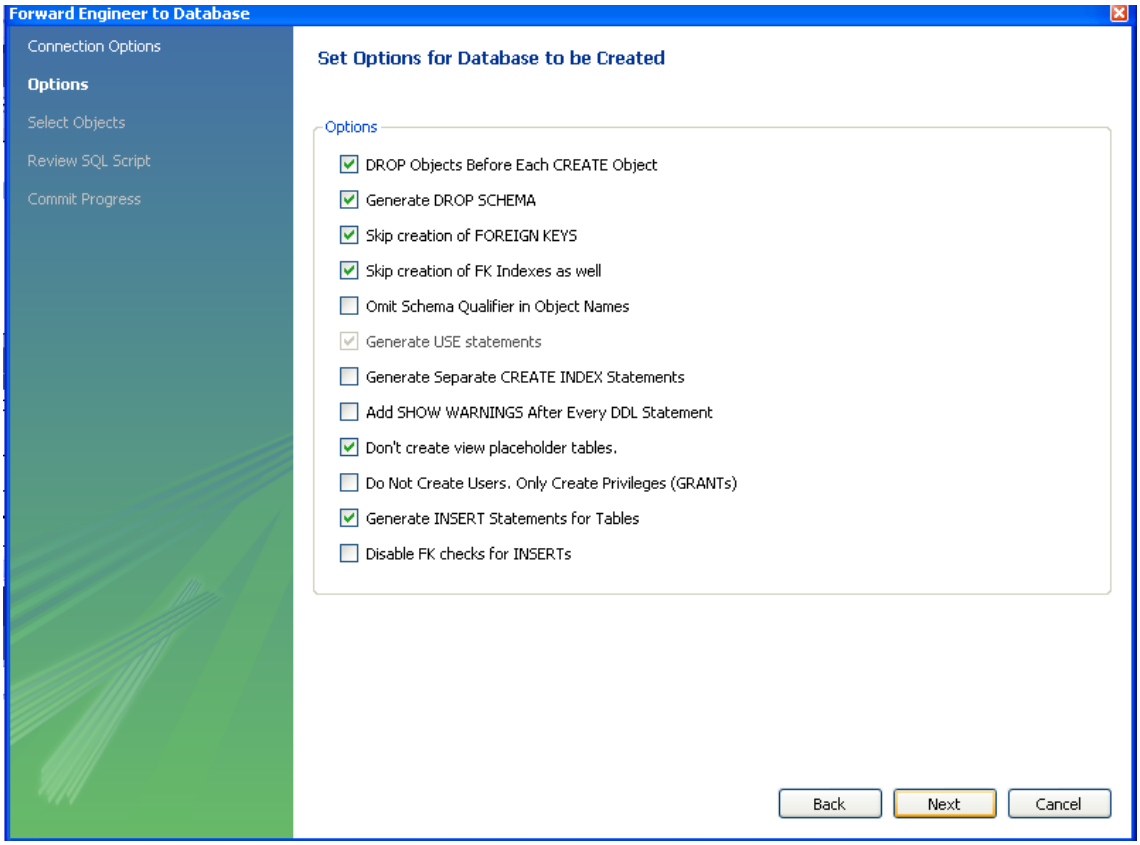
Para esto realizamos el siguiente procedimiento:



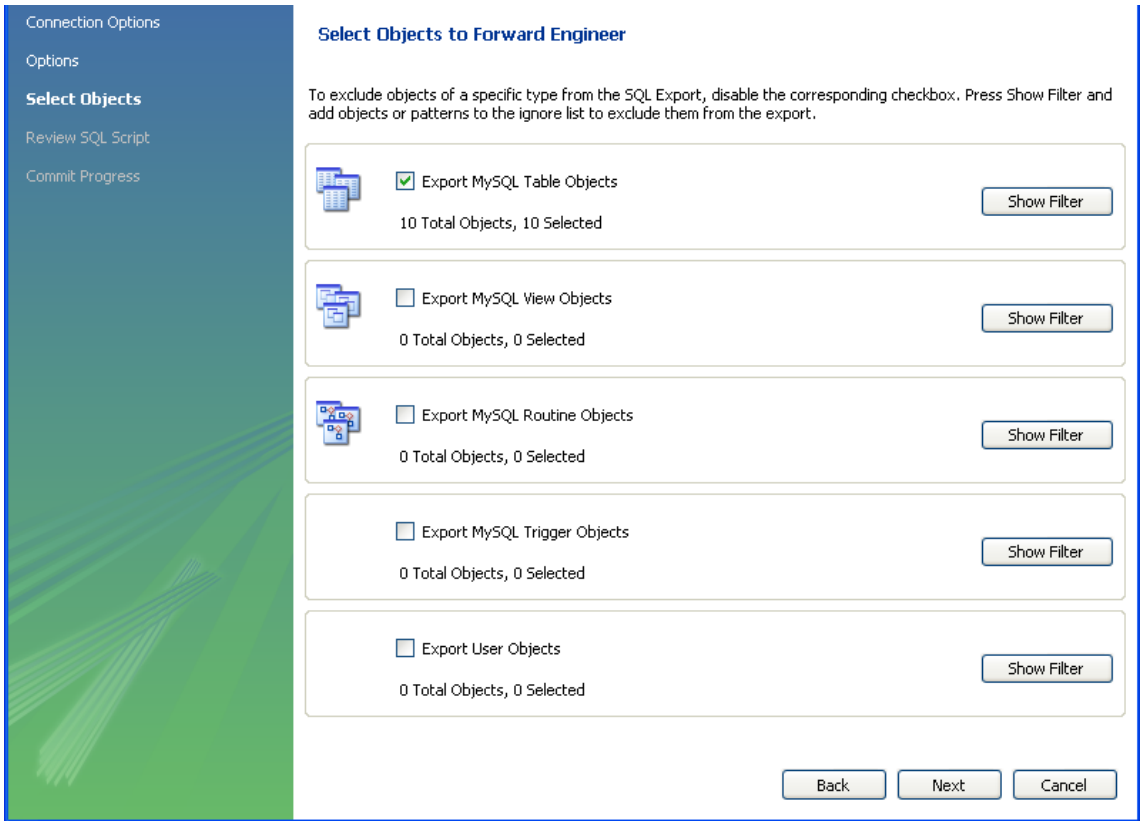
Aparece el siguiente pantallazo en el cual hacemos clic en NEXT(SIGUIENTE):



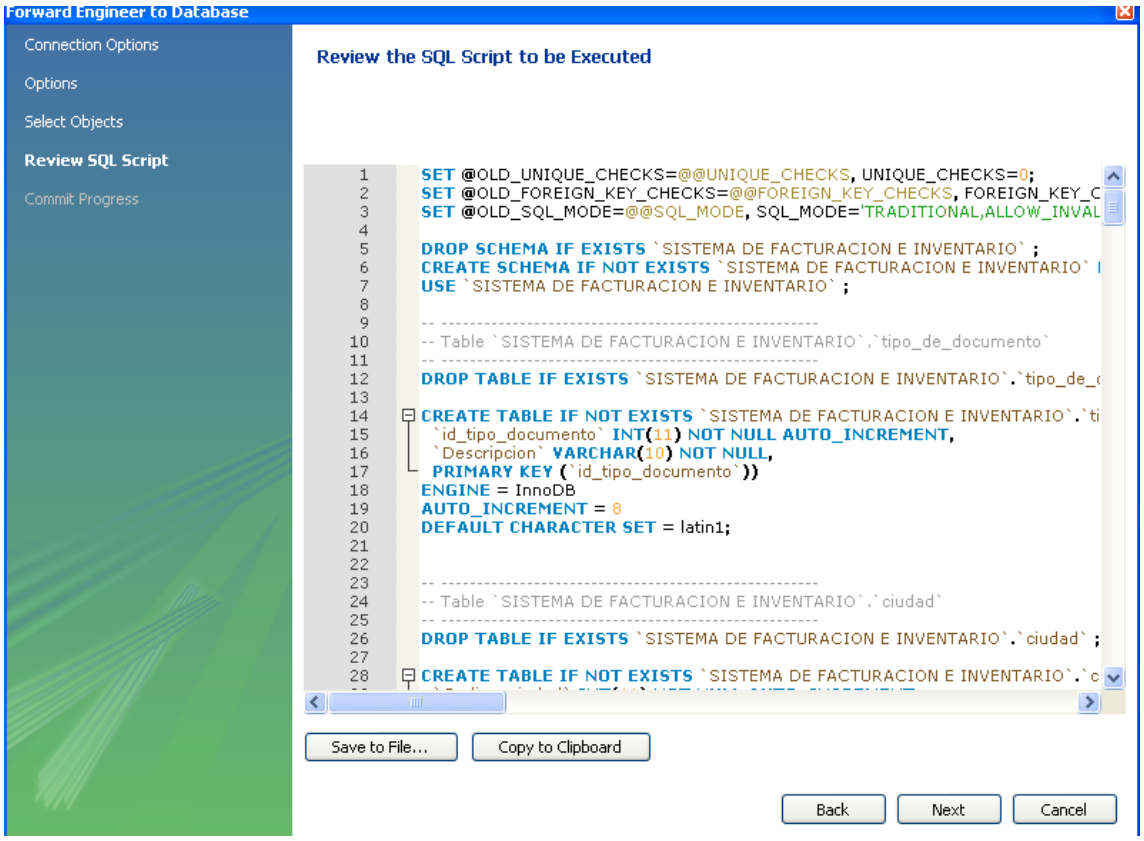

En dicho pantallazo seleccionamos las siguientes opciones y hacemos clic en NEXT




Verificamos la exportación de la base de datos con las 10 tablas establecidas y clic en NEXT

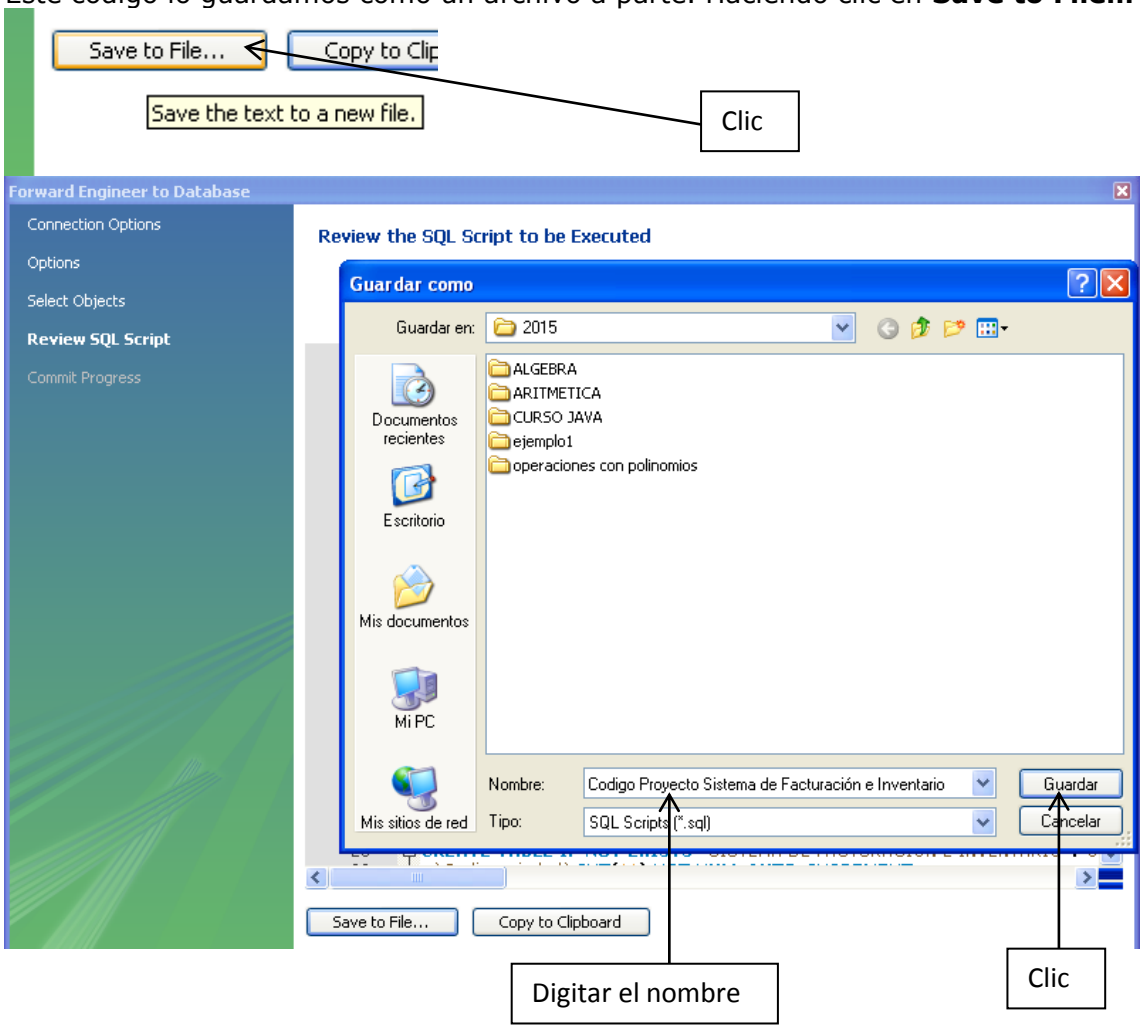



Con lo que aparece este pantallazo con el código



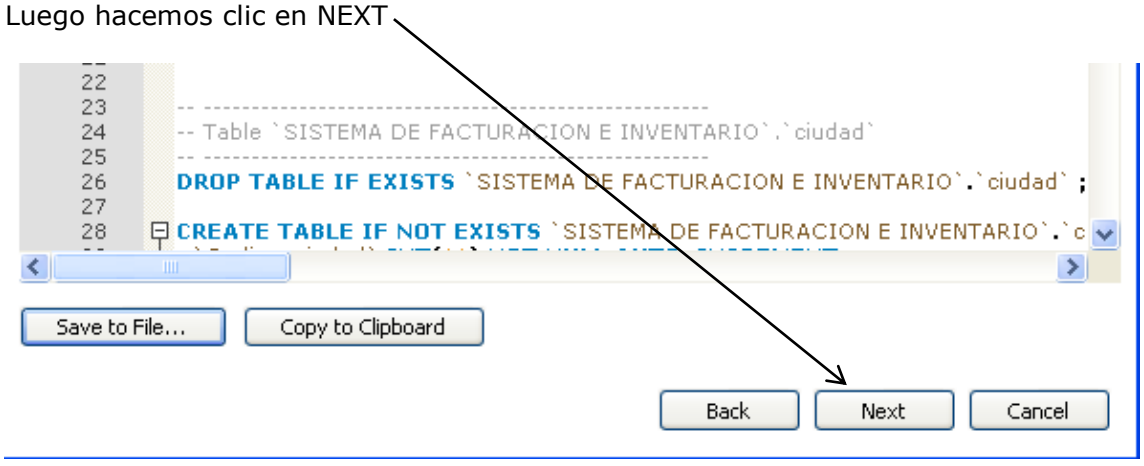

En este pantallazo observamos el archivo con todo el código generado a partir del diagrama entidad-relación.

Este código lo guardamos como un archivo a parte. Haciendo clic en **Save to File...**

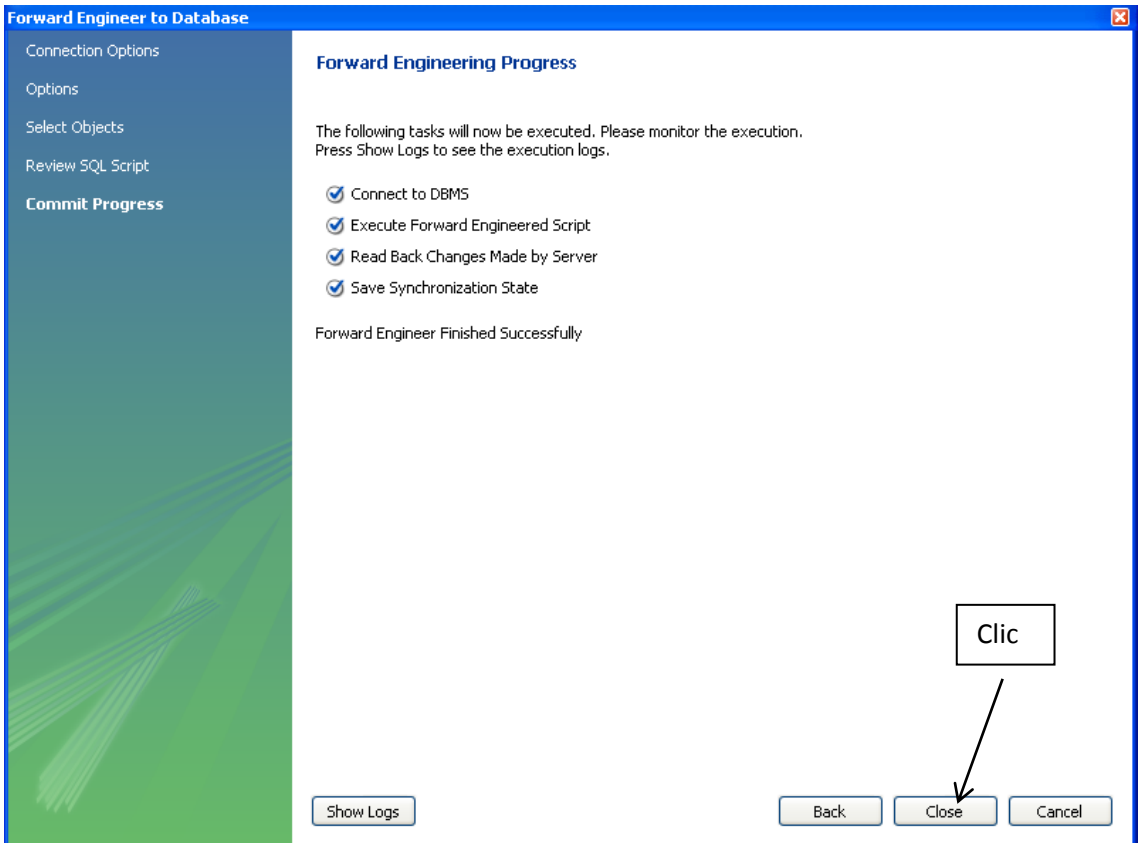



De esta manera generamos el Scripts de la base de datos, el cual es necesario para generar las consultas sobre los datos.

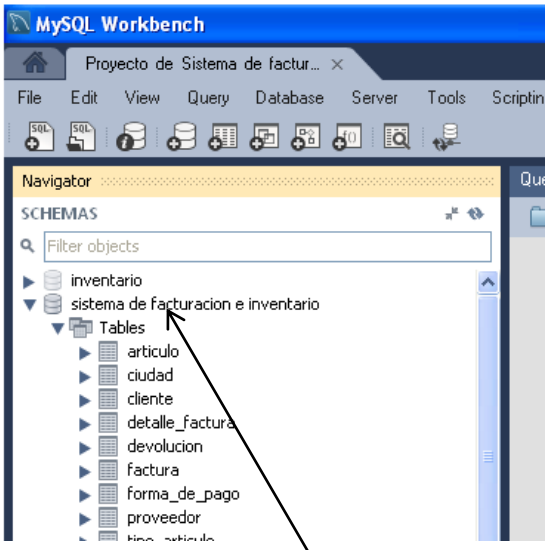
Luego hacemos clic en NEXT




**Se genera el siguiente pantallazo terminado así el proceso de modelación y generación de la base de datos.**




Para terminar con este proceso hacemos clic en **Close** y luego cerramos **MySQL** y lo abrimos nuevamente la conexión y verificamos la existencia de la base de datos. Como se muestra a continuación:

Con este procedimiento terminamos la definición de datos y continuamos con la manipulación de datos. Aunque este proceso ya lo habíamos iniciado en el momento en que comenzamos a insertar registros en las tablas.

Realmente con este proceso es que iniciaremos el trabajo de consultas o Querys con MySQL.

Antes de iniciar se indicara alguna terminología o comandos necesarios para el trabajo con el SQL.

**TERMINOLOGIA SQL**

SQL (Lenguaje de Consulta Estructurado)

El SQL (Structure Query Language), es un lenguaje de consulta estructurado establecido claramente como el lenguaje de alto nivel estándar para sistemas de base de datos relacionales. Los responsables de publicar este lenguaje como estándar, fueron precisamente los encargados de publicar estándar, la ANSI (Instituto Americano de Normalización) y la ISO (organismo Internacional de Normalización). Es por lo anterior que este lenguaje lo vas a encontrar en cualquiera de los DBMS relacionales que existen en la actualidad, por ejemplo, ORACLE, SYBASES, SQL SERVER por mencionar algunos.

El SQL agrupa tres tipos de sentencias con objetivos particulares, en los siguientes lenguajes:

- ✓ Lenguaje de Definición de Datos (DDL, Data Definiton Language)
- ✓ Lenguaje de Manipulación de Datos (DML, Data Management Language)
- ✓ Lenguaje de Control de Datos (DCL, Data Control Language)

A continuación se describen cada uno de los lenguajes:

**Lenguaje de Definición de Datos (DDL, Data Definiton Language)**

Grupo de sentencias del SQL que soportan la definición y declaración de los objetos de la base de datos. Objetos tales como: la base de datos misma(DATABASE), las tablas(TABLE), las Vistas (VIEW), los índices (INDEX), los procedimientos almacenados (PROCEDURE), los disparadores (TRIGGER),

Reglas (RULE), Dominios (Domain) y Valores por defecto (DEFAULT).

**CREATE,**

**ALTER y**

**DROP**

**Lenguaje de Manipulación de Datos (DML, Data Management Language)**

Grupo de sentencias del SQL para manipular los datos que están almacenados en las bases de datos, a nivel de filas (tuplas) y/o columnas (atributos). Ya sea que se requiera que los datos sean modificados, eliminados, consultados o que se agregaren nuevas filas a las tablas de las base de datos.

**INSERT,**

**UPDATE,**

**DELETE y**

**SELECT**

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

Grupo de sentencias del SQL para controlar las funciones de administración que realiza el DBMS, tales como la atomicidad y seguridad.

**COMMIT TRANSACTION,**

**ROLLBACK TRANSACTION,**

**GRANT**

**REVOKE**

**Sentencia CREATE**

Dentro del Lenguaje de Definición (DL) del SQL, la sentencia CREATE permiten la definición o creaciòn de muchos objetos de la base de datos tales como: tablas (esquemas), índices, vistas, dominios, ligaduras de integridad y procedimientos.

En esta oportunidad veremos las sentencias correspondientes a la creación de los esquemas o lo que es lo mismo las tablas que contendrán los datos de la base de datos, La sentencia CREATE TABLE.

La sentencia CREATE TABLE, define el nombre de la tabla, las columnas con su tipo de datos, las ligaduras de intengridad que vigilan el valor que se guarde como dato en las columnas o atributos sean llaves o no.

**Sintaxis:**

```
CREATE TABLE nombre_tabla
(
campo1 tipo dato [NULL/NOT NULL] | CHECK (expresiónLógica) | [
DEFAULT expresiónConstante],
campo2 tipo dato [NULL/NOT NULL] | CHECK (expresiónLógica) | [
DEFAULT expresiónConstante ],
campo-N,
PRIMARY KEY(campo_llave),
FOREIGN KEY (campo_llave) REFERENCES tabla2 (campo_llave-tabla2)
)
```

## Eliminación de Tablas:

La sentencia para eliminar una tabla y por ende todo los objetos asociados con esa tabla como: las vistas, disparadores, etc., DROP TABLE, donde **r** es el nombre de una tabla existente.

**DROP TABLE r**

Por ejemplo si deseamos eliminar a la tabla ARTICULO_PANAMA, revisemos haber si esta ya existe, de la siguiete manera

SELECT * from ARTICULO_PANAMA

Posteriormente una vez veriifcada que la tabla existe, se procede a borrar la tabla, escribiendo lo siguiente en l ainterface de consultas del SQL:

### Sentencia ALTER

Después que una tabla ha sido utilizada durante algún tiempo, los usuarios suelen descubrir que desean almacenar información adicional con respecto a las tablas. Por ejemplo en la base de datos **VENTAS**, se podría desear:

- ✓ Añadir el nombre y numero de de una persona de contacto a cada fila de la tabla CLIENTES para contactar a los clientes.
- ✓ Añadir una columna de punto de reorden mínimo en la tabla ARTICULO, para que la base de datos pueda alertar automáticamente cuando la cantidad o stock de un producto en particular esta por debajo de lo optimo para la venta.

Por lo general, esta sentencia ALTER TABLE se utiliza sobre tablas que ya poseen desde cientos a miles de filas por ser tablas de un sistemas de Base de Datos que ya esta en producción.

**Consultas Simples**

El corazón o poder del Lenguaje SQL es el poder hacer consultas de cualquier tipo a la base de datos en forma no procedural. La sentencia SELECT es muy poderosa y ampliamente rica en sus cláusulas y variantes permitiendo la capacidad de atender en poco tiempo a consultas complejas sobre la base de datos. Está en el especialista desarrollador de aplicaciones conocerlo a profundidad para explotar las bondades y virtudes.

Gracias a esta sentencia es que se pueden realizar todas las operaciones del Algebra Relacional, tratadas en la sección 3.2 y 3.3 del modulo III. En esta sección veremos la sintaxis de como se utiliza.

La sentencia SELECT, obtiene filas de la base de datos y permite realizar la selección de una o varias filas o columnas de una o varias tablas.

La sintaxis completa de la instrucción SELECT es compleja, pero la voy resumir como sigue (los corchetes cuadrados indican que la cláusula no es obligatoria):

SELECT nombres de las columnas

[INTO nueva Tabla destino para resultados del select_]

FROM origenTabla

[WHERE condición de Búsqueda]

[GROUP BY nombres de columnas por la cual Agrupar]

[HAVING condiciónBúsqueda para Group By ]

[ORDER BY nombre de columnas [ASC | DESC] ]

También se puede unir estas sentencias con otras por el operador UNION entre consultas para combinar sus resultados en un sola tabla de resultados sin nombre.

Veamos cales son las funciones de cada una de las cláusulas de la sentencia SELECT.

La cláusula SELECT: Se usa para listar las columnas de las tablas que se desean ver en el resultado de una consulta. Además de las columnas se pueden listas columnas a calcular por el SQL cuando actué la sentencia. Esta cláusula no puede omitirse.

La cláusula FROM: Lista las tablas que deben ser analizadas en la evaluación de la expresión de la cláusula WHERE y de donde se listarán las columnas enunciadas en el SELECT. Esta cláusula no puede omitirse.

La cláusula WHERE: establece criterios de selección de ciertas filas en el resultado de la consulta gracias a las condiciones de búsqueda. Si no se requiere condiciones de búsqueda puede omitirse y el resultado de la consulta serán todas las filas de las tablas enunciadas en el FROM.

La cláusula GROUP BY: especifica una consulta sumaria. En vez de producir una fila de resultados por cada fila de datos de la base de datos, una consulta sumaria agrupa todas las filas similares y luego produce una fila sumaria de resultados para cada grupo de los nombres de columnas enunciado en esta cláusula. En otras palabras, esta cláusula permitirá agrupar un conjunto de columnas con valores repetidos y utilizar las funciones de agregación sobre las columnas con valores no repetidas. Esta cláusula puede omitirse.

La cláusula HAVING: le dice al SQL que incluya sólo ciertos grupos producidos por la cláusula GROUP BY en los resultados de la consulta. Al igual que la cláusula WHERE, utiliza una condición de búsqueda para especificar los grupos deseados. La cláusula HAVING es la encargada de condicionar la selección de los grupos en base a los valores resultantes en las funciones agregadas utilizadas debidas que la cláusula WHERE condiciona solo para la selección de filas individuales. Esta cláusula puede omitirse.

La cláusula ORDER BY: permitirá establecer la columna o columnas sobre las cuales las filas resultantes de la consulta deberán ser ordenadas. Esta cláusula puede

**LOS COMANDOS MÁS USADOS EN SQL SON:**

# SELECT

¿Para qué utilizamos los comandos SQL? El uso común es la selección de datos desde tablas ubicadas en una base de datos. Inmediatamente, vemos dos palabras claves: necesitamos **SELECT** la información (Columna) **FROM** una tabla. (Note que la tabla es un contenedor que reside en la base de datos donde se almacena la información. Por lo tanto tenemos la estructura SQL más básica:

```
SELECT * FROM "nombre_tabla";
```

Para ilustrar el ejemplo anterior, suponga que queremos que se nos muestren todos los registros de la tabla: Cliente de la base de datos Sistema de facturación e inventario

Tabla ***CLIENTE***

cliente - Table

Table Name: cliente Schema: SISTEMA DE FACTURACION

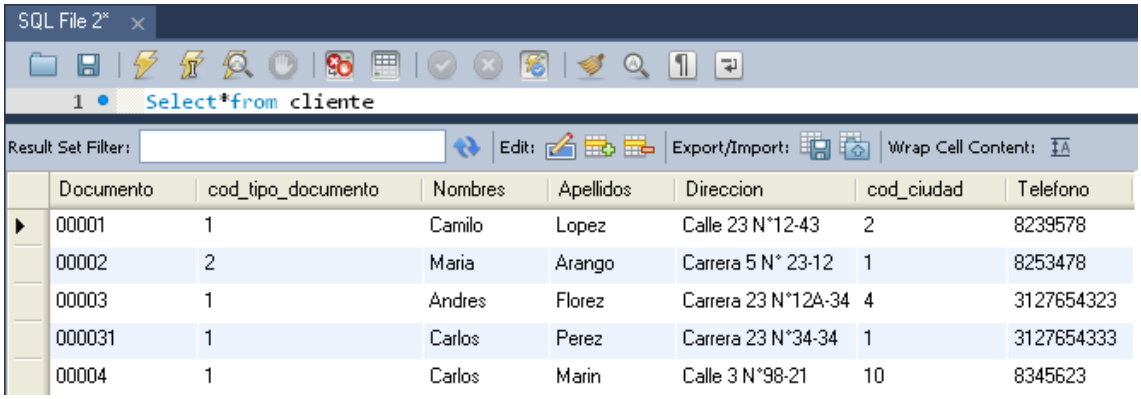
Result Set Filter: Edit: Export/Import: Wrap Cell Content:

| Documento | cod_tipo_documento | Nombres | Apellidos | Direccion | cod_ciudad | Telefono |
|---|---|---|---|---|---|---|
| 000031 | 1 | Carlos | Perez | Carrera 23 N°34-34 | 1 | 3127654333 |
| 00001 | 1 | Camilo | Lopez | Calle 23 N°12-43 | 2 | 8239578 |
| 00002 | 2 | Maria | Arango | Carrera 5 N° 23-12 | 1 | 8253478 |
| 00003 | 1 | Andres | Florez | Carrera 23 N°12A-34 | 4 | 3127654323 |
| 00004 | 1 | Carlos | Marin | Calle 3 N°98-21 | 10 | 8345623 |
| 00005 | 1 | Elvira | Orozco | Centro | 15 | 3219843543 |
| 00006 | 1 | Kevin I | Ayala | Carrera 23 N°4B-23 | 1 | 8354624 |
| 00007 | 1 | Angela | Hoyos | Calle 4 N°23-34 | 3 | 3217654300 |

En esta observamos varias columnas con datos. De la cual consultaremos toda la tabla

**SELECT * FROM CLIENTE;**

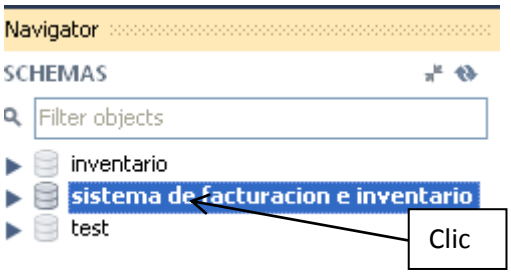
Obteniendo como Resultado:



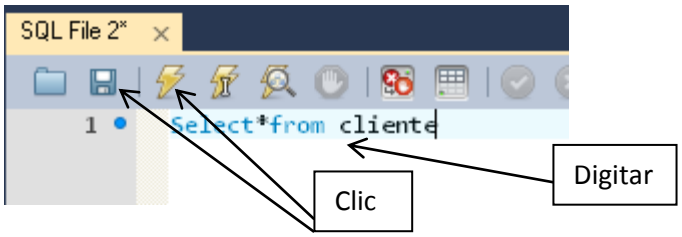

| Documento | cod_tipo_documento | Nombres | Apellidos | Direccion | cod_ciudad | Telefono |
|---|---|---|---|---|---|---|
| 00001 | 1 | Camilo | Lopez | Calle 23 N°12-43 | 2 | 8239578 |
| 00002 | 2 | Maria | Arango | Carrera 5 N° 23-12 | 1 | 8253478 |
| 00003 | 1 | Andres | Florez | Carrera 23 N°12A-34 | 4 | 3127654323 |
| 000031 | 1 | Carlos | Perez | Carrera 23 N°34-34 | 1 | 3127654333 |
| 00004 | 1 | Carlos | Marin | Calle 3 N°98-21 | 10 | 8345623 |

Veamos el procedimiento a realizar en MySQL Workbench:

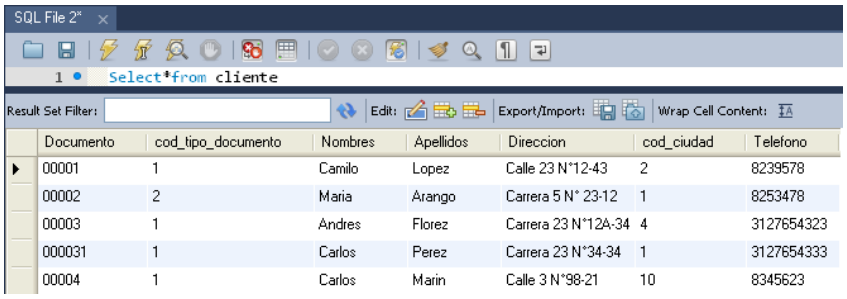
Seleccionamos la base de datos




Digitamos en el Query la instrucción y Hacemos clic en Rayo para generar la consulta y procedemos a guardarla




Generando la siguiente consulta

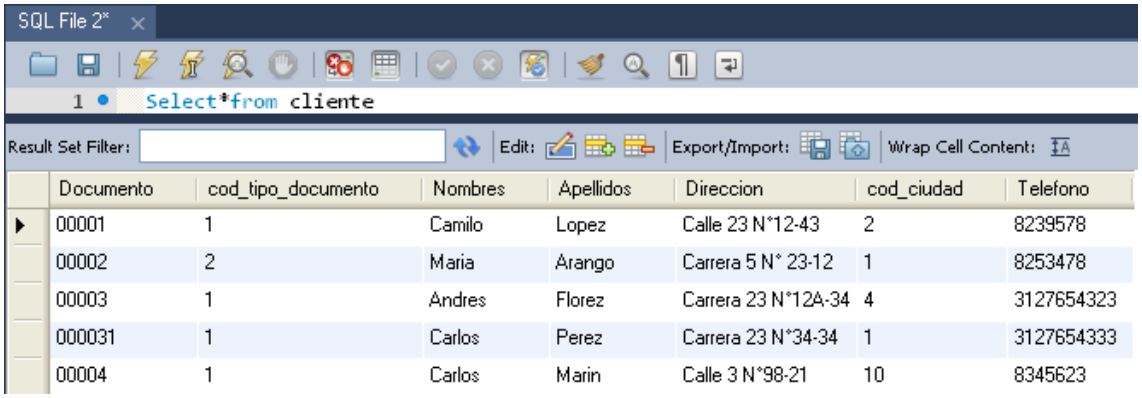



| Documento | cod_tipo_documento | Nombres | Apellidos | Direccion | cod_ciudad | Telefono |
|---|---|---|---|---|---|---|
| 00001 | 1 | Camilo | Lopez | Calle 23 N°12-43 | 2 | 8239578 |
| 00002 | 2 | Maria | Arango | Carrera 5 N° 23-12 | 1 | 8253478 |
| 00003 | 1 | Andres | Florez | Carrera 23 N°12A-34 | 4 | 3127654323 |
| 000031 | 1 | Carlos | Perez | Carrera 23 N°34-34 | 1 | 3127654333 |
| 00004 | 1 | Carlos | Marin | Calle 3 N°98-21 | 10 | 8345623 |

**SELECT "nombre_columna" FROM "nombre_tabla";**

Para ilustrar el ejemplo anterior, suponga que tenemos la tabla: Cliente de la base de datos **Proyecto de Sistema de facturación e inventario**

Tabla ***CLIENTE***

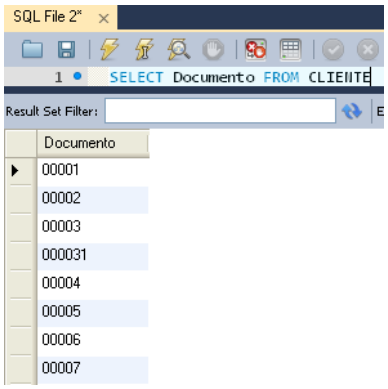



| Documento | cod_tipo_documento | Nombres | Apellidos | Direccion | cod_ciudad | Telefono |
|---|---|---|---|---|---|---|
| 00001 | 1 | Camilo | Lopez | Calle 23 N°12-43 | 2 | 8239578 |
| 00002 | 2 | Maria | Arango | Carrera 5 N° 23-12 | 1 | 8253478 |
| 00003 | 1 | Andres | Florez | Carrera 23 N°12A-34 | 4 | 3127654323 |
| 000031 | 1 | Carlos | Perez | Carrera 23 N°34-34 | 1 | 3127654333 |
| 00004 | 1 | Carlos | Marin | Calle 3 N°98-21 | 10 | 8345623 |

En esta observamos varias columnas con datos. De la cual seleccionaremos como ejemplo la columna correspondiente al **Documento**

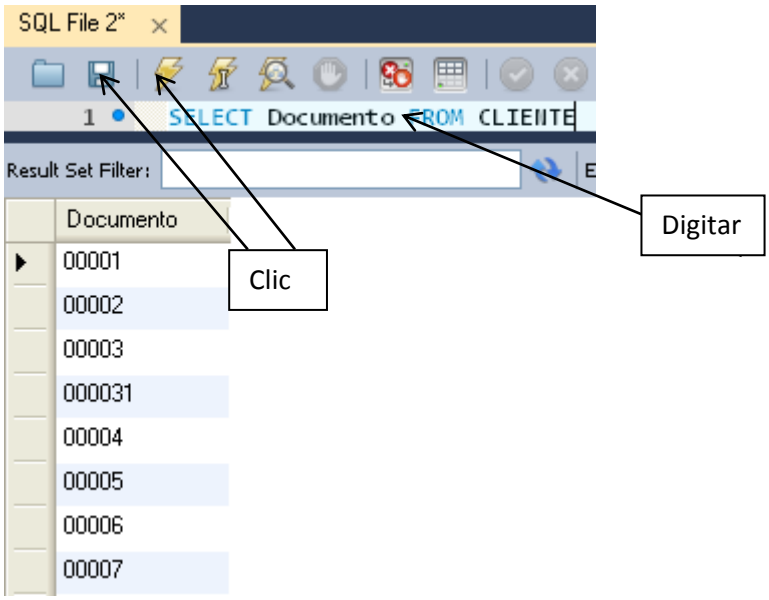
**SELECT Id_Cliente FROM CLIENTE;**

Obteniendo como Resultado:



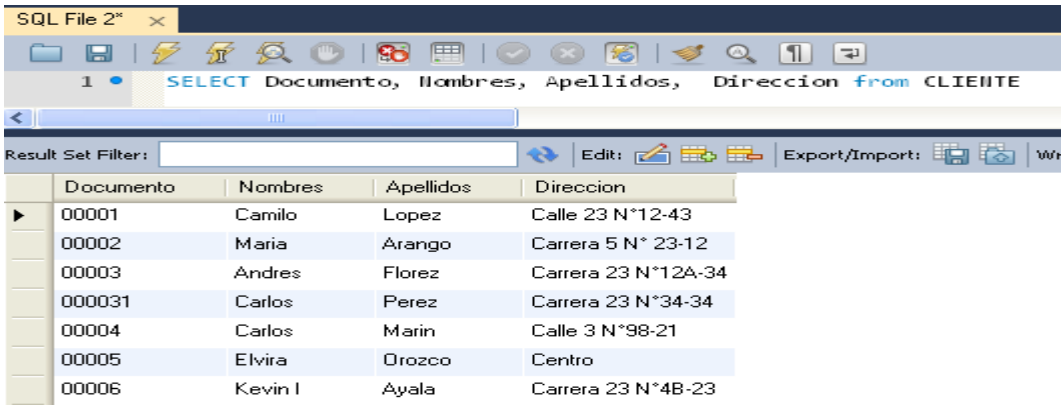

| Documento |
|---|
| 00001 |
| 00002 |
| 00003 |
| 000031 |
| 00004 |
| 00005 |
| 00006 |
| 00007 |

Veamos el procedimiento a realizar en MySQL Workbench




Pueden seleccionarse los nombres de columnas múltiples, así como también los nombres de tablas múltiples.

Ver ejemplo:




**EJERICICIO 1**

Usando los comandos **Select** y **From** del Lenguaje **SQL** genere las siguientes consultas:

**Consulta 1:** De la tabla **CLIENTE** nombres, apellidos y el teléfono de los clientes

**Consulta 2:** De la tabla **ARTICULO** en Id del articulo, descripcion y el precio_venta

**Consulta 4:** De la tabla **PROVEEDOR** el No_documento, cod_tipo_documento, Nombre, Apellido, Nombre_comercial.

**Consulta 5:** De la tabla **DETALLE_FACTURA** el cod_factura, total

# WHERE

Luego, podríamos desear seleccionar condicionalmente los datos de una tabla. Por ejemplo, podríamos desear sólo mostrar el precio de venta mayores a 50000. Para ello, utilizamos la palabra clave **WHERE**. La sintaxis es la siguiente:

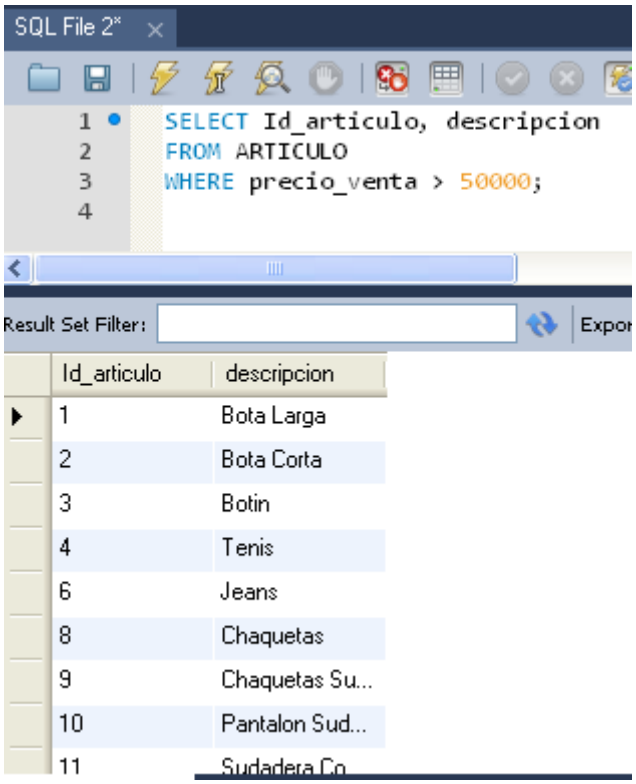
```
SELECT "nombre_columna"
FROM "nombre_tabla"
WHERE "condición";
```

Por ejemplo, para seleccionar todos los precio_venta mayores a 50000 y el Id_articulo y la descripcion en la tabla **ARTICULO**

Ingresamos,

```
SELECT Id_articulo, descripcion
FROM ARTICULO
WHERE precio_venta > 50000;
```

Resultado:




**EJERCICIO 2**

**Consulta 6:** De la tabla **CLIENTE** Consultar el Documento, el nombre, apellido y el teléfono de los clientes del cod_ciudad 1, 2 y 3

**Consulta 7:** De la tabla **PROVEEDOR** Consultar el No_Documento, Nombre, Apellido de los proveedores con cod_tipo_documento = 4

# Order By

Hasta ahora, hemos visto cómo obtener datos de una tabla utilizando los comandos **SELECT** y **WHERE**. Con frecuencia, sin embargo, necesitamos enumerar el resultado en un orden particular. Esto podría ser en orden ascendente, en orden descendente, o podría basarse en valores numéricos o de texto. En tales casos, podemos utilizar la palabra clave **ORDER BY** para alcanzar nuestra meta.

La sintaxis para una instrucción **ORDER BY** es la siguiente:

```
SELECT "nombre_columna"
FROM "nombre_tabla"
[WHERE "condición"]
ORDER BY "nombre_columna" [ASC, DESC];
```

[ ] significa que la instrucción **WHERE** es opcional. Sin embargo, si existe una cláusula **WHERE**, viene antes de la cláusula **ORDER BY ASC** significa que los resultados se mostrarán en orden ascendente, y **DESC** significa que los resultados se mostrarán en orden descendente. Si no se especifica ninguno, la configuración predeterminada es **ASC**.

Es posible ordenar por más de una columna. En este caso, la cláusula **ORDER BY** anterior se convierte en

```
ORDER BY "nombre1_columna" [ASC, DESC], "nombre2_columna" [ASC, DESC]
```

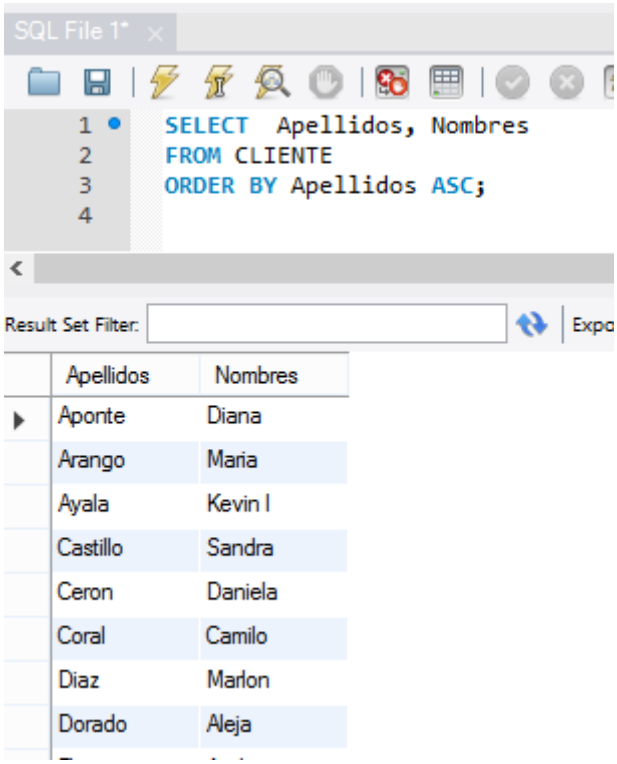
Suponiendo que elegimos un orden ascendente para ambas columnas, el resultado se clasificará en orden ascendente según la columna 1. Si hay una relación para el valor de la columna 1, se clasificará en orden ascendente según la columna 2.

Por ejemplo, podríamos mostrar en orden alfabético de acuerdo al nombre o apellido los datos de **CLIENTE**

Ingresamos,

```
SELECT  Apellidos, Nombres
FROM CLIENTE
ORDER BY Apellidos ASC;
```

Resultado:




| Apellidos | Nombres |
|---|---|
| Aponte | Diana |
| Arango | Maria |
| Ayala | Kevin I |
| Castillo | Sandra |
| Ceron | Daniela |
| Coral | Camilo |
| Diaz | Marlon |
| Dorado | Aleja |

**EJERCICIO 3**

**Consulta 8:** De la tabla **CLIENTE** Consultar el Documento, el nombre, apellidos y el teléfono de los clientes y ordenar alfabéticamente los datos por nombre en forma descendente.

**Consulta 9:** De la tabla **ARTICULO** Consultar el id_articulo, descripción y el precio de venta. Y ordenar los datos en forma ascendente a partir del precio de venta.

# And Or

En esta parte de la guía continuaremos usando la palabra clave **WHERE** la cual utilizamos anteriormente. En este caso la utilizaremos para seleccionar datos condicionalmente desde una tabla. Esta condición puede ser una condición simple, o puede ser una condición compuesta. Las condiciones compuestas están formadas por múltiples condiciones simples conectadas por **AND (y)** u **OR(o)**. No hay límites en el número de condiciones simples que pueden presentarse en una sola instrucción SQL.

La sintaxis de una condición compuesta es la siguiente:

```
SELECT "nombre_columna"
FROM "nombre_tabla"
WHERE "condición simple"
{[AND|OR] "condición simple"}+;
```

{}+ significa que la expresión dentro de las llaves ocurrirá una o más veces. Note que **AND** u **OR** pueden utilizarse indistintamente. Además, podemos utilizar el símbolo paréntesis ( ) para indicar el orden de la condición.

Por ejemplo, podemos seleccionar todos los **id_articulo**, **descripción** y el **precio de venta** menor que 70000 **O** mayor que 30000. Tabla ***ARTICULO***,

Tabla ***ARTICULO***

Table Name: articulo  Schema: **SISTEMA DE FACTURACION E INVENTARIO**

Edit: Export/Import: Wrap Cell Content:

| id_articulo | descripcion | precio_venta | precio_costo | stock | cod_tipo_articulo | cod_proveedor | fecha_ingreso |
|---|---|---|---|---|---|---|---|
| 1 | Bota Larga | 100000 | 80000 | 20 | 19 | 00003-A | 2012-11-03 |
| 2 | Bota Corta | 70000 | 50000 | 55 | 19 | 00003-A | 2012-10-11 |
| 3 | Botin | 150000 | 130000 | 18 | 19 | 00003-A | 2012-09-11 |
| 4 | Tenis | 120000 | 100000 | 8 | 20 | 00002-2 | 2012-10-02 |
| 5 | Zandalias | 50000 | 30000 | 59 | 21 | 00001-2 | 2012-11-01 |
| 6 | Jeans | 90000 | 70000 | 18 | 24 | 00004-4 | 2012-10-02 |
| 7 | Camisas | 50000 | 30000 | 12 | 22 | 00003-A | 2012-10-20 |
| 8 | Chaquetas | 70000 | 50000 | 8 | 24 | 00004-A | 2012-11-02 |
| 9 | Chaquetas S... | 60000 | 40000 | 20 | 23 | 00002-2 | 2012-10-23 |
| 10 | Pantalon Sud... | 60000 | 40000 | 17 | 23 | 00002-2 | 2012-10-23 |
| 11 | Sudadera Co... | 100000 | 80000 | 20 | 23 | 00002-2 | 2012-10-23 |
| 12 | Camisetas | 30000 | 20000 | 30 | 24 | 00005-F | 2012-10-03 |

Ingresamos,

```
SELECT id_articulo, descripción, precio_venta
FROM ARTICULO
WHERE  precio_venta < 70000
OR (precio_venta < 70000 AND precio_venta > 30000);
```

Resultado:

SQL File 1*

```
1  SELECT id_articulo, descripcion, precio_venta
2  FROM ARTICULO
3  WHERE  precio_venta < 70000
4  OR (precio_venta < 70000 AND precio_venta > 30000);
5
6
```

Result Set Filter: Edit: Export/Import:

| id_articulo | descripcion | precio_venta |
|---|---|---|
| 5 | Zandalias | 50000 |
| 7 | Camisas | 50000 |
| 9 | Chaquetas Sudadera | 60000 |
| 10 | Pantalon Sudadera | 60000 |
| 12 | Camisetas | 30000 |

**EJERCICIO 4**

**CONSULTA 10:** de la Tabla **ARTICULO**, consultar **id articulo**, **descripción** y el **precio de venta.** Del **articulo** con **precio de venta** Menores a **100000** y Mayores que **50000**

**CONSULTA 11:** de la Tabla **ARTICULO**, consultar **id articulo**, **descripción** y el **precio de venta.** Del **articulo** con **precio de venta** iguales a **50000** y Mayores que **50000**

**CONSULTA 12:** de la Tabla **ARTICULO**, consultar **id articulo**, **descripción** y el **precio de compra.** Del **articulo** con **precio de compra** Menores a **130000** y Mayores que **90000**

**CONSULTA 13:** de la Tabla **ARTICULO**, consultar **id articulo**, **descripción** y el **precio de compra.** Del **articulo** con **precio de compra** iguales a **80000** y Mayores que **80000**

**CONSULTA 14:** de la Tabla **detalle_factura,** consultar el **cod_factura**, **cod_articulo** y el **total.** Donde el **total** es Mayor que **100000**

# In

En SQL, hay dos usos de la palabra clave **IN**, y esta sección introduce aquél relacionado con la cláusula **WHERE**. Cuando se lo utiliza en este contexto, sabemos exactamente el valor de los valores regresados que deseamos ver para al menos una de las columnas. La sintaxis para el uso de la palabra clave **IN** es la siguiente:

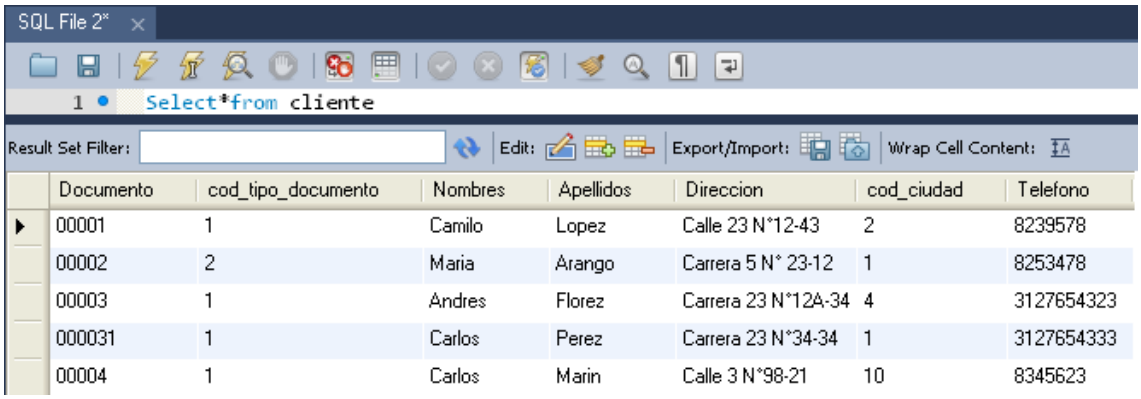
```
SELECT "nombre_columna"
FROM "nombre_tabla"
WHERE "nombre_columna" IN (''valor1', ''valor2', ...);
```

El número de valores en los paréntesis pueden ser uno o más, con cada valor separado por comas. Los valores pueden ser números o caracteres. Si hay sólo un valor dentro del paréntesis, este comando es equivalente a

```
WHERE "nombre_columna" = 'valor1'
```

Por ejemplo, podríamos desear seleccionar documento, nombres, apellidos, donde el cod_tipo_documento es uno en la Tabla ***CLIENTE***

Tabla ***CLIENTE***

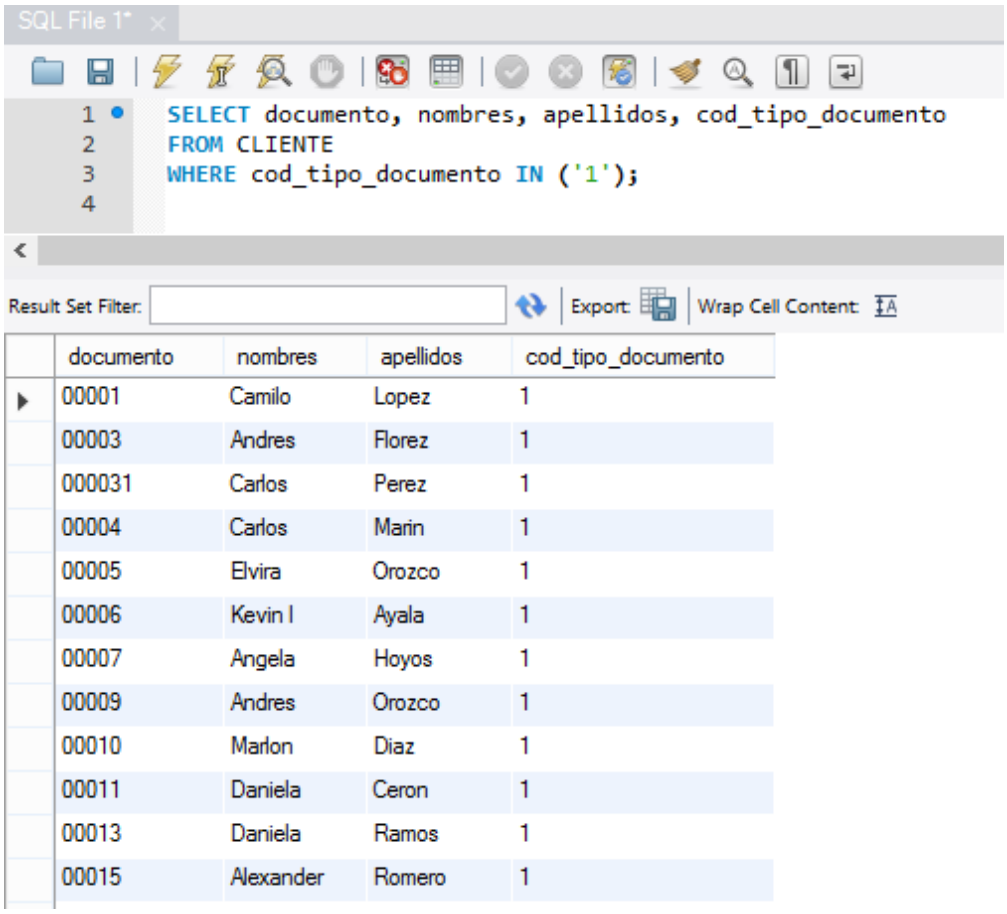



| Documento | cod_tipo_documento | Nombres | Apellidos | Direccion | cod_ciudad | Telefono |
|---|---|---|---|---|---|---|
| 00001 | 1 | Camilo | Lopez | Calle 23 N°12-43 | 2 | 8239578 |
| 00002 | 2 | Maria | Arango | Carrera 5 N° 23-12 | 1 | 8253478 |
| 00003 | 1 | Andres | Florez | Carrera 23 N°12A-34 | 4 | 3127654323 |
| 000031 | 1 | Carlos | Perez | Carrera 23 N°34-34 | 1 | 3127654333 |
| 00004 | 1 | Carlos | Marin | Calle 3 N°98-21 | 10 | 8345623 |

Ingresamos,

```
SELECT documento, nombres, apellidos, cod_tipo_documento
FROM CLIENTE
WHERE cod_tipo_documento IN ('1');
```

Resultado:




| documento | nombres | apellidos | cod_tipo_documento |
|---|---|---|---|
| 00001 | Camilo | Lopez | 1 |
| 00003 | Andres | Florez | 1 |
| 000031 | Carlos | Perez | 1 |
| 00004 | Carlos | Marin | 1 |
| 00005 | Elvira | Orozco | 1 |
| 00006 | Kevin I | Ayala | 1 |
| 00007 | Angela | Hoyos | 1 |
| 00009 | Andres | Orozco | 1 |
| 00010 | Marlon | Diaz | 1 |
| 00011 | Daniela | Ceron | 1 |
| 00013 | Daniela | Ramos | 1 |
| 00015 | Alexander | Romero | 1 |

EJERCICIO 5

**CONSULTA 15:** de la Tabla **CLIENTE**, consultar los datos de los clientes con código ciudad uno

**CONSULTA 16:** de la Tabla **CLIENTE**, consultar los datos de los clientes con cód_tipo_documento igual a 2

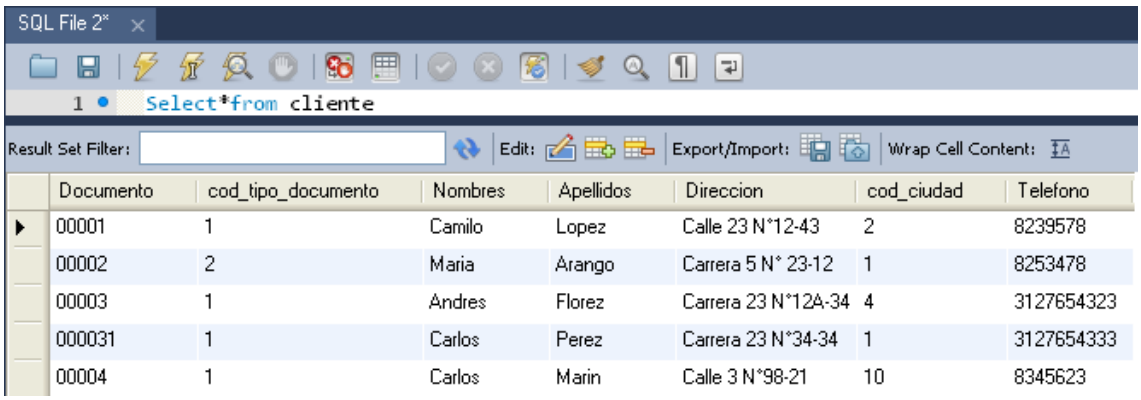
# BETWEEN

Mientras que la palabra clave **IN** ayuda a las personas a limitar el criterio de selección para uno o más valores discretos, la palabra clave **BETWEEN** permite la selección de un rango. La sintaxis para la cláusula **BETWEEN** es la siguiente:

```
SELECT "nombre_columna"
FROM "nombre_tabla"
WHERE "nombre_columna" BETWEEN 'valor1' AND 'valor2';
```

Esto seleccionará todas las filas cuya columna tenga un valor entre 'valor1' y 'valor2'.

Por ejemplo, podríamos desear seleccionar la visualización del documento, nombres, apellidos. Donde el documento este entre 00004 y 00009, en la Tabla ***Cliente***,

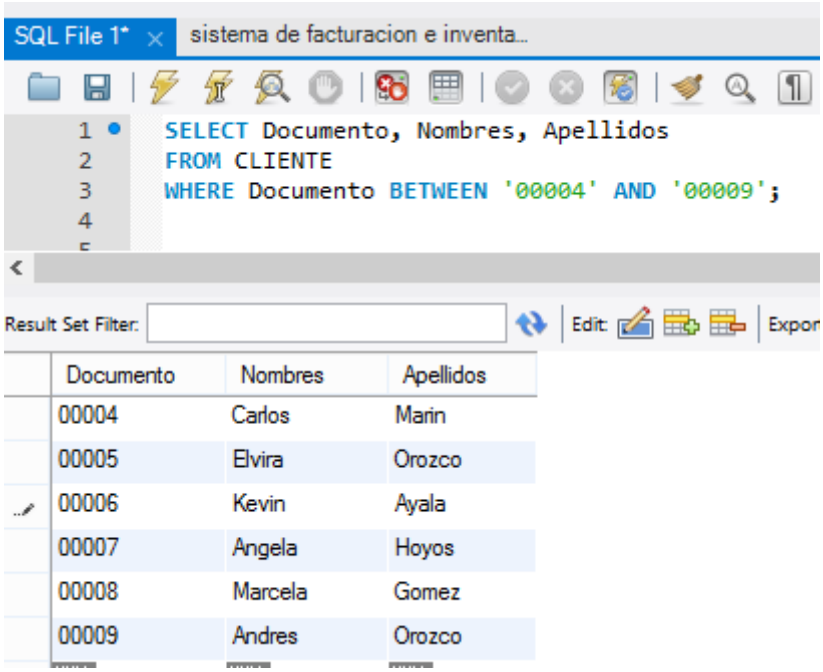



| Documento | cod_tipo_documento | Nombres | Apellidos | Direccion | cod_ciudad | Telefono |
|---|---|---|---|---|---|---|
| 00001 | 1 | Camilo | Lopez | Calle 23 N°12-43 | 2 | 8239578 |
| 00002 | 2 | Maria | Arango | Carrera 5 N° 23-12 | 1 | 8253478 |
| 00003 | 1 | Andres | Florez | Carrera 23 N°12A-34 | 4 | 3127654323 |
| 000031 | 1 | Carlos | Perez | Carrera 23 N°34-34 | 1 | 3127654333 |
| 00004 | 1 | Carlos | Marin | Calle 3 N°98-21 | 10 | 8345623 |

Ingresamos,

```
SELECT Documento, Nombres, Apellidos
FROM CLIENTE
WHERE Documento BETWEEN '00004' AND '00009';
```

Resultado:




| Documento | Nombres | Apellidos |
|---|---|---|
| 00004 | Carlos | Marin |
| 00005 | Elvira | Orozco |
| 00006 | Kevin | Ayala |
| 00007 | Angela | Hoyos |
| 00008 | Marcela | Gomez |
| 00009 | Andres | Orozco |

EJERCICIO 6

**CONSULTA 17:** de la Tabla **CLIENTE**, consultar el documento, nombres, apellidos. Donde el documento este entre 00010 y 00020

**CONSULTA 18:** de la Tabla **CIUDAD**, consultar los datos de las ciudades desde la Codigo_ciudad 6 hasta Codigo_ciudad 10.

# Like

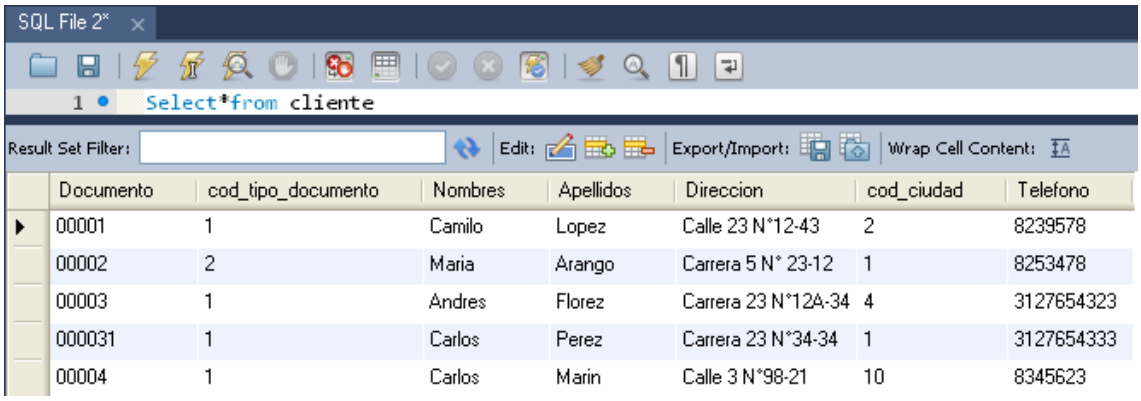
**LIKE** es otra palabra clave que se utiliza en la cláusula **WHERE**. Básicamente, **LIKE** le permite hacer una búsqueda basada en un patrón en vez de especificar exactamente lo que se desea (como en **IN**) o determinar un rango (como en **BETWEEN**). La sintaxis es la siguiente:

```
SELECT "nombre_columna"
FROM "nombre_tabla"
WHERE "nombre_columna" LIKE {patrón};
```

{patrón} generalmente consiste en comodines. Aquí hay algunos ejemplos:

- 'A_Z': Toda línea que comience con 'A', otro carácter y termine con 'Z'. Por ejemplo, 'ABZ' y 'A2Z' deberían satisfacer la condición, mientras 'AKKZ' no debería (debido a que hay dos caracteres entre A y Z en vez de uno).
- 'ABC%': Todas las líneas que comienzan con 'ABC'. Por ejemplo, 'ABCD' y 'ABCABC' ambas deberían satisfacer la condición.
- '%XYZ': Todas las líneas que terminan con 'XYZ'. Por ejemplo, 'WXYZ' y 'ZZXYZ' ambas deberían satisfacer la condición.
- '%AN%': : Todas las líneas que contienen el patrón 'CA' en cualquier lado. Por ejemplo, 'CAMILO y CARLOS' ambos deberían satisfacer la condición.

Digamos que tenemos la siguiente tabla:



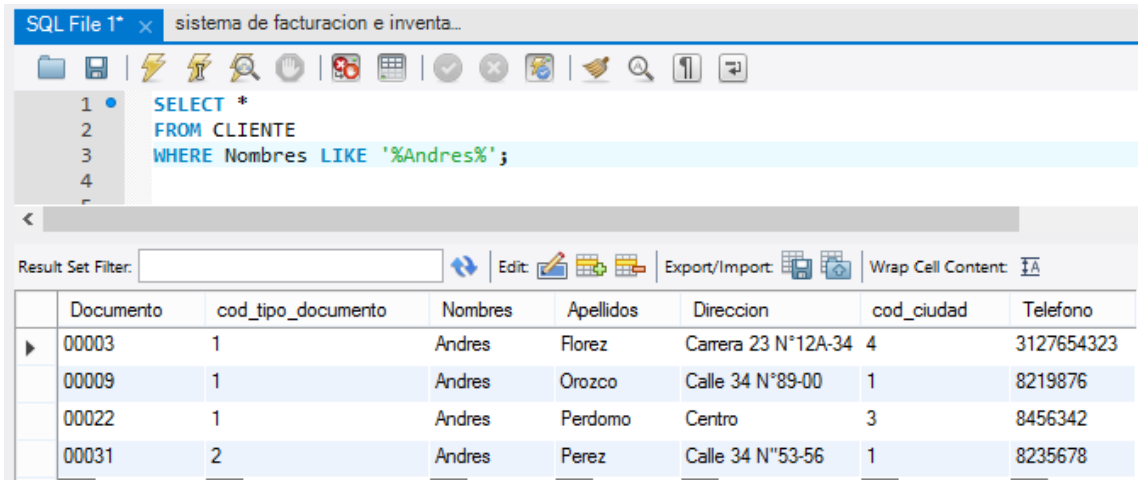

| Documento | cod_tipo_documento | Nombres | Apellidos | Direccion | cod_ciudad | Telefono |
|---|---|---|---|---|---|---|
| 00001 | 1 | Camilo | Lopez | Calle 23 N°12-43 | 2 | 8239578 |
| 00002 | 2 | Maria | Arango | Carrera 5 N° 23-12 | 1 | 8253478 |
| 00003 | 1 | Andres | Florez | Carrera 23 N°12A-34 | 4 | 3127654323 |
| 000031 | 1 | Carlos | Perez | Carrera 23 N°34-34 | 1 | 3127654333 |
| 00004 | 1 | Carlos | Marin | Calle 3 N°98-21 | 10 | 8345623 |

Tabla ***CLIENTE***

Deseamos encontrar todos los clientes cuyos nombres contengan 'ANDRES'. Para hacerlo, ingresamos,

```
SELECT *
FROM CLIENTE
WHERE Nombres LIKE '%Andres%';
```

Resultado:




| Documento | cod_tipo_documento | Nombres | Apellidos | Direccion | cod_ciudad | Telefono |
|---|---|---|---|---|---|---|
| 00003 | 1 | Andres | Florez | Carrera 23 N°12A-34 | 4 | 3127654323 |
| 00009 | 1 | Andres | Orozco | Calle 34 N°89-00 | 1 | 8219876 |
| 00022 | 1 | Andres | Perdomo | Centro | 3 | 8456342 |
| 00031 | 2 | Andres | Perez | Calle 34 N°53-56 | 1 | 8235678 |

EJERCICIO 7

**CONSULTA 19:** de la Tabla **CLIENTE**, consultar los datos de los clientes cuyo apellido es "**Pérez"**.

**CONSULTA 20:** de la Tabla **PROVEEDOR**, consultar los datos de los proveedores cuyo nombre comercial contiene la palabra "**Jeans"**.

**CONSULTA 21:** de la Tabla **CLIENTE**, consultar los datos de los clientes cuya dirección es "**Centro"**.

## Nota: con estas consultas terminamos la primera parte del tema de consultas.

## En la siguiente guía trabajaremos con los comandos de:

- **Funciones (AVG, COUNT, MAX, MIN, SUM)**
- **Group By**
- **Having**
- **Alias**
- **Join**
- **Outer Join**
- **Concatenar Funcion**
- **Substring Funcion**

Además de esto hablaremos de la Matriz Crud

**Yadir Alexander Agudelo Durango**
**Ingeniero de Sistemas**
Universidad de Medellín