



## Nivel II de Diseño de Bases de Datos

Enero, 2016



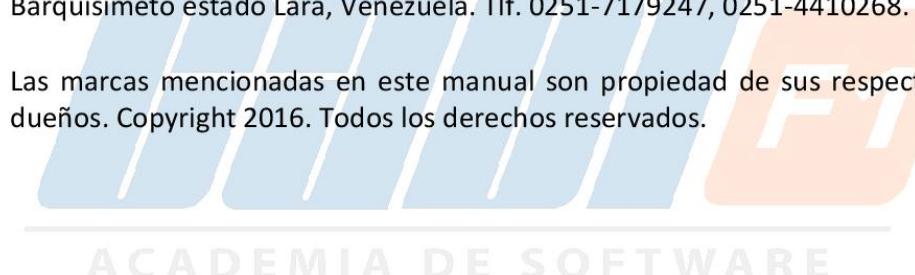
## Objetivos del curso

- Aprender a usar las instrucciones SQL
- Manipular datos con insert, update y delete
- Consultas básicas y avanzadas
- Funciones de agregado
- Sub consultas

## Acerca de este manual

Este manual pertenece al Centro de Asesoramiento y Desarrollo Informático C.A. (CADIF1). Para obtener más información sobre este u otros cursos visite nuestra sitio Web [www.cadif1.com](http://www.cadif1.com), escribanos a la dirección de correo [cadi@cadif1.com](mailto:cadi@cadif1.com) o visítenos en nuestra sede ubicada en la Av. Pedro León Torres con calle 59, Centro Comercial Sotavento, piso 2 oficina 27, Barquisimeto estado Lara, Venezuela. Tlf. 0251-7179247, 0251-4410268.

Las marcas mencionadas en este manual son propiedad de sus respectivos dueños. Copyright 2016. Todos los derechos reservados.





## Contenido del curso

### Capítulo 1. Introducción al Sql

- 1.1.- Concepto.
- 1.2.- Tipos de Instrucciones Sql.
- 1.3.- Instrucciones DDL.
- 1.4.- Instrucciones DML.

### Capítulo 2. Instrucciones Ddl. Parte 1

- 2.1.- Create Database.
- 2.2.- Create Table.

### Capítulo 3. Instrucciones Ddl. Parte 2

- 3.1.- Alter Table.
- 3.2.- Drop.
- 3.3.- Truncate.

### Capítulo 4. Insert

- 4.1.- Sintaxis.
- 4.2.- Insertar Con Campos Auto Incrementales.
- 4.3.- Insertar Varios Registros.

### Capítulo 5. Update

- 5.1.- Sintaxis.

### Capítulo 6. Delete

- 6.1.- Sintaxis.
- 6.2.- Eliminar Algunos Registros.

### Capítulo 7. Select

- 7.1.- Sintaxis.

- 7.2.- Alias de Campos.
- 7.3.- Ordenamiento.

## Capítulo 8. Where. Parte 1

- 8.1.- Sintaxis.
- 8.2.- Operadores Básicos.
- 8.3.- Operadores Avanzados.

## Capítulo 9. Where. Parte 2

- 9.1.- Valores Nulos.
- 9.2.- Limit.
- 9.3.- Distinct.

## Capítulo 10. Join

- 10.1.- Inner Join.
- 10.2.- Union.

## Capítulo 11. Funciones Agregadas. Parte 1

- 11.1.- Max.
- 11.2.- Min.

## Capítulo 12. Funciones Agregadas. Parte 2

- 12.1.- Count.
- 12.2.- Sum.
- 12.3.- Avg.

## Capítulo 13. Group by

- 13.1.- Group By.
- 13.2.- Having.

## Capítulo 14. Sub Consultas. Parte 1

- 14.1.- Concepto.
- 14.2.- Sub Consultas de Un Escalar.

## Capítulo 15. Sub Consultas. Parte 2

- 15.1.- Exist Y Not Exist.
- 15.2.- All.
- 15.3.- ANY, SOME, IN.



## Capítulo 1. INTRODUCCIÓN AL SQL

### 1.1.- Concepto

El lenguaje de consulta estructurado o SQL (por sus siglas en inglés Structured Query Language) es un lenguaje declarativo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones en ellas. El SQL es el lenguaje por excelencia de los diversos sistemas de gestión de bases de datos relacionales surgidos a partir del año 1979 y fue por fin estandarizado en 1986 por el ANSI, dando lugar a la primera versión estándar de este lenguaje, el "SQL-86" o "SQL1". Al año siguiente este estándar es también adoptado por la ISO.

Sin embargo, este primer estándar no cubría todas las necesidades de los desarrolladores e incluía funcionalidades de definición de almacenamiento que se consideró suprimirlas. Así que, en 1992, se lanzó un nuevo estándar ampliado y revisado del SQL llamado "SQL-92" o "SQL2".

En la actualidad el SQL es el estándar de facto de la inmensa mayoría de los SGBD comerciales. Y, aunque la diversidad de añadidos particulares que incluyen las distintas implementaciones comerciales del lenguaje es amplia, el soporte al estándar SQL-92 es general y muy amplio.

El ANSI SQL sufrió varias revisiones y agregados a lo largo del tiempo:

Año	Nombre	Alias	Comentarios
1986	SQL 86	SQL 87	Primería publicación hecha por ANSI. Confirmada por ISO en 1987.
1989	SQL-89		Revisión menor.
1992	SQL 92	SQL 2	Revisión mayor
1999	SQL 1999	SQL2000	Se agregaron expresiones regulares, consultas recursivas (para relaciones jerárquicas), triggers y algunas características orientadas a objetos.
2003	SQL 2003		Introdujo algunas características de XML, cambios en las funciones, estandarización del objeto sequence y de las columnas autonumericas. (Ver Eisenberg et al.: <a href="#">SQL:2003 Has Been Published</a> .)
2005	SQL 2005		ISO/IEC 9075-14:2005 Define las maneras en las cuales el SQL se puede utilizar conjuntamente con XML. Define maneras de importar y guardar datos XML en una base de datos SQL, manipularlos dentro de la base de datos y publicando el XML y los datos SQL convencionales en forma XML. Además, proporciona facilidades que permiten a las aplicaciones integrar dentro de su código SQL el uso de XQuery, lenguaje de consulta XML publicado por el W3C (World Wide Web Consortium) para acceso concurrente a datos ordinarios SQL y documentos XML.
2008	SQL 2008		Permite el uso de la cláusula ORDER BY fuera de las definiciones de los cursor. Incluye los disparadores del tipo INSTEAD OF. Añado la sentencia TRUNCATE. (Ver <a href="#">[1]</a> .)

### 1.2.- Tipos de Instrucciones Sql

SQL es un lenguaje declarativo de "alto nivel" o "de no procedimiento" que, gracias a su fuerte base teórica y su orientación al manejo de conjuntos de registros —y no a registros individuales— permite una alta productividad en codificación y la orientación a objetos. De esta forma, una sola sentencia puede equivaler a uno o más programas que se utilizarían en un lenguaje de bajo nivel orientado a registros. Las instrucciones SQL están clasificadas en 2 grupos:

- El LDD: Lenguaje de Definición de Datos de SQL (Data Definition Language DDL) proporciona comandos para la definición de esquemas de relación, borrado de relaciones y modificaciones de los esquemas de relación.
- El LMD: Lenguaje interactivo de Manipulación de Datos de SQL (Data Manipulation Language DML) incluye lenguajes de consultas basados tanto en álgebra relacional como en cálculo relacional de tuplas.
- El LCD: Lenguaje de Consulta de Datos (Data Query Language DQL) del SQL proporciona el comando para la consulta de datos en las tablas.
- El Lenguaje de Control de Datos proporciona comandos para ayudar al administrador a controlar la seguridad y los accesos a los datos.

### 1.3.- Instrucciones DDL

El lenguaje de definición de datos (en inglés Data Definition Language, o DDL), es el que se encarga de la modificación de la estructura de los objetos de la base de datos. Incluye órdenes para modificar, borrar o definir las tablas en las que se almacenan los datos de la base de datos. Existen cuatro operaciones básicas:

- CREATE: crear bases de datos y tablas.
- ALTER: modificar tablas existentes.
- DROP: eliminar bases de datos, tablas, índices, etc.
- TRUNCATE: vaciar tablas.

#### 1.4.- Instrucciones DML

Un lenguaje de manipulación de datos (Data Manipulation Language, o DML en inglés) es un lenguaje proporcionado por el sistema de gestión de base de datos que permite a los usuarios llevar a cabo las tareas de consulta o manipulación de los datos, organizados por el modelo de datos adecuado.

El lenguaje de manipulación de datos es el más popular del SQL, debido a que es usado para recuperar y manipular datos en una base de datos relacional. Posee 4 instrucciones que permiten hacer las operaciones básicas sobre los registros de una tabla, a lo que se le denomina CRUD (por sus siglas en inglés de Create, Read, Update y Delete). Las instrucciones del DML son:

- SELECT: para consultar datos sobre las tablas de la base de datos.
- INSERT: para agregar registros a una tabla de la base de datos.
- UPDATE: para actualizar o modificar los registros de una tabla.
- DELETE: para eliminar uno o varios registros de las tablas.



## Capítulo 2. INSTRUCCIONES DDL. PARTE 1

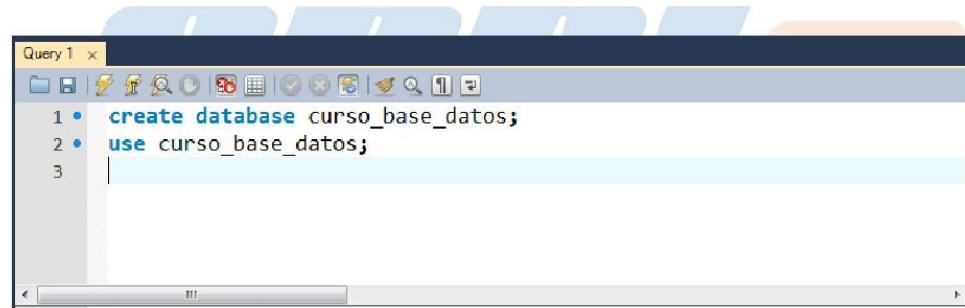
### 2.1.- Create Database

Este comando permite crear una nueva base de datos en el DBMS. La sintaxis es la siguiente:

```
CREATE DATABASE mibasededatos;
```

Luego de crear la base de datos, si queremos ejecutar instrucciones SQL sobre esa base de datos nueva, debemos ejecutar la instrucción:

```
USE mibasededatos;
```



### 2.2.- Create Table

Este comando permite crear tablas en una base de datos del DBMS. Su sintaxis es la siguiente:

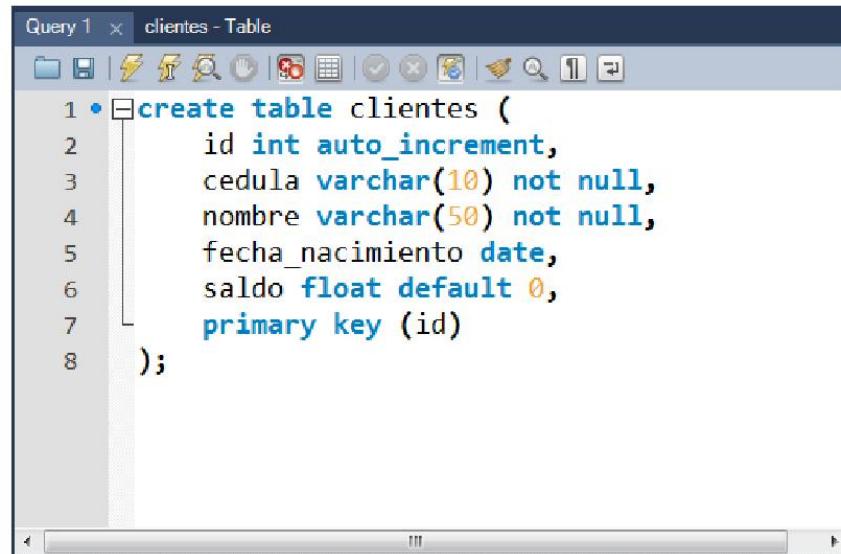
```
CREATE TABLE nombre (campo1 tipodato [, campo2 tipodato,...][  
primary key (campo1, [campos2] )]
```

Cada campo lleva su respectivo tipo de dato. El tipo de dato dependerá del DBMS, ya que cada uno puede tener tipos de datos con nombres particulares. Se puede indicar la clave primaria con la cláusula PRIMARY KEY,

la cual debe hacer referencia a uno de los campos definidos en la tabla. Veamos un ejemplo:

```
Query 1 ×
create table clientes (
    cedula varchar(10),
    nombre varchar(50),
    fecha_nacimiento date,
    saldo float,
    primary key (cedula)
)
```

La mayoría de los DBMS permite indicar de alguna manera cuales campos no permitirán valores nulos: NOT NULL indicará que al insertar un registro en esta tabla, este campo no puede estar sin valor. También permiten indicar cuando un campo su valor será auto incremental, es decir, que se generará automáticamente cada vez que se inserta un nuevo registro. El siguiente es un ejemplo en el que el campo "id" es un entero y es auto incremental. Para que sea un campo auto incremental debe ser la clave primaria de la tabla:



The screenshot shows a MySQL Workbench interface with a query editor titled "Query 1" and a results tab titled "clientes - Table". The query editor contains the following SQL code:

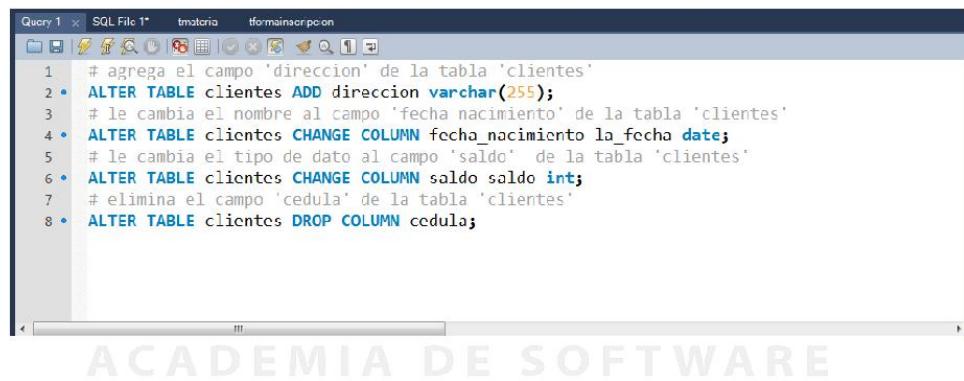
```
1 • □create table clientes (
2     id int auto_increment,
3     cedula varchar(10) not null,
4     nombre varchar(50) not null,
5     fecha_nacimiento date,
6     saldo float default 0,
7     primary key (id)
8 );
```



## Capítulo 3. INSTRUCCIONES DDL. PARTE 2

### 3.1.- Alter Table

Este comando permite modificar la estructura de una tabla. Se pueden agregar/quitar/renombrar campos a una tabla, modificar el tipo de un campo, agregar/quitar índices a una tabla, etc. Tiene 3 cláusulas: ADD COLUMN (para agregar un nuevo campo), DROP COLUMN (para eliminar un campo), CHANGE COLUMN (para cambiar propiedades de un campo: nombre, tipo de dato, etc). La sintaxis de alter table es la siguiente:



The screenshot shows a MySQL Workbench interface with a query editor titled 'Query 1'. The code entered is as follows:

```
Query 1 | SQL File 1* | Materia | tformainscripcion
1 # agrega el campo 'direccion' de la tabla 'clientes'
2 • ALTER TABLE clientes ADD direccion varchar(255);
3 # le cambia el nombre al campo 'fecha nacimiento' de la tabla 'clientes'
4 • ALTER TABLE clientes CHANGE COLUMN fecha_nacimiento la_fecha date;
5 # le cambia el tipo de dato al campo 'saldo' de la tabla 'clientes'
6 • ALTER TABLE clientes CHANGE COLUMN saldo saldo int;
7 # elimina el campo 'cedula' de la tabla 'clientes'
8 • ALTER TABLE clientes DROP COLUMN cedula;
```

ACADEMIA DE SOFTWARE

### 3.2.- Drop

La instrucción DROP se utiliza para eliminar tablas y base de datos del DBMS. Su sintaxis es muy simple también:

```
DROP mitabla;  
DROP DATABASE mibase;
```

Se debe tener especial precaución al utilizar este comando, debido a que al eliminar una tabla se eliminan todos los registros de la misma y su recuperación no es posible, a menos que tenga un respaldo. Al eliminar una base de datos se debe tener la misma precaución. Esta operación es irreversible.

### 3.3.- Truncate

Este comando trunca todo el contenido de una tabla. La ventaja sobre el comando DROP, es que si se quiere borrar todo el contenido de la tabla, es mucho más rápido, especialmente si la tabla es muy grande. La desventaja es que TRUNCATE sólo sirve cuando se quiere eliminar absolutamente todos los registros. El comando TRUNCATE borra la tabla y la vuelve a crear y no ejecuta ninguna transacción. La sintaxis es muy sencilla:

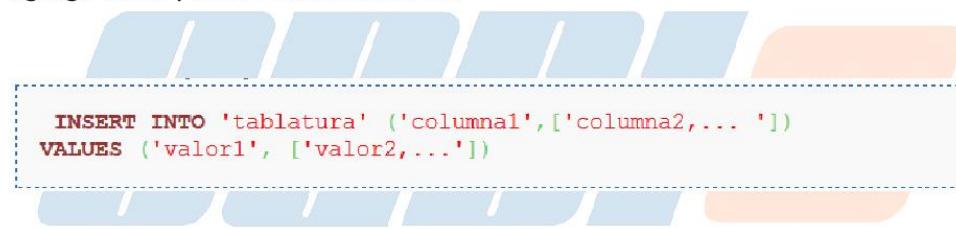
```
TRUNCATE tabla;
```



## Capítulo 4. INSERT

### 4.1.- Sintaxis

Una sentencia INSERT de SQL agrega uno o más registros a una (y sólo una) tabla en una base de datos relacional. Su forma general se puede visualizar en la imagen siguiente. Las cantidades de columnas y valores deben ser iguales. Si una columna no se especifica, le será asignado el valor por defecto. Los valores especificados (o implícitos) por la sentencia INSERT deberán satisfacer todas las restricciones aplicables, por ejemplo: violar la clave primaria (valor repetido del campo clave), que un campo sea NOT NULL o que no coincidan los tipos de datos, un valor que no satisfaga una clave foránea. Si ocurre un error de sintaxis o si alguna de las restricciones es violada, no se agrega la fila y se devuelve un error.

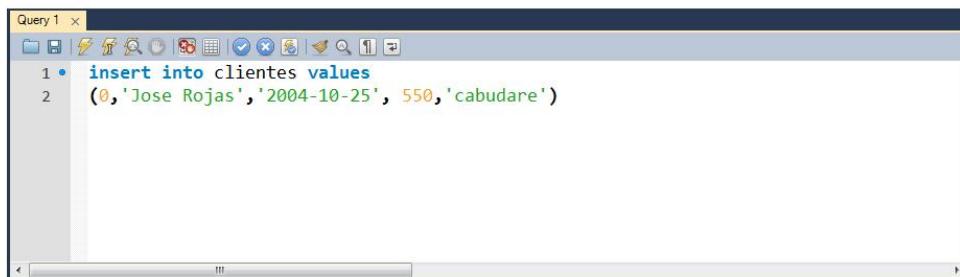


### 4.2.- Insertar Con Campos Auto Incrementales

Si en la tabla hay un campo auto numérico no se coloca en la sentencia insert debido a que el DBMS generará su valor automáticamente. El siguiente es un ejemplo de cómo insertar una fila en una tabla:

```
Query 1
1 • insert into clientes (la_fecha, saldo, direccion, cedula, nombre ) values
2 ('2004-10-25', 550, 'cabudare', 'Jose Rojas')
```

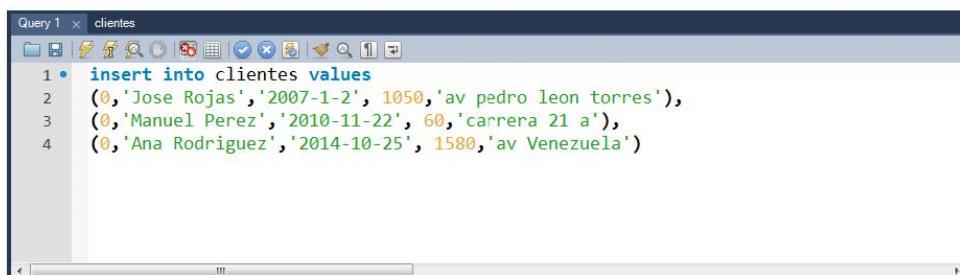
En ocasiones cuando la tabla tiene muchos campos, se puede colocar sólo los valores. En este caso, se deben colocar todos los valores y deben colocarse en el mismo orden en que están los campos en la definición de la tabla. Si la tabla posee algún campo auto numérico debe colocarse el valor 0 (cero) para que el DBMS pueda generar el valor nuevo, si se coloca un valor distinto de 0 (cero), se le intentará asignar al campo ese valor. Se Debe tener cuidado debido a que un campo auto numérico debe ser la clave primaria de la tabla, por lo tanto, al asignar valor nosotros mismo se podría violar la clave primaria. Por ejemplo:



```
Query 1 ×
clients
1 • insert into clientes values
2 (0,'Jose Rojas','2004-10-25', 550,'cabudare')
```

#### 4.3.- Insertar Varios Registros

También se pueden agregar varios registros en una misma instrucción INSERT. El siguiente es un ejemplo de cómo insertar varios registros:



```
Query 1 × clients
1 • insert into clientes values
2 (0,'Jose Rojas','2007-1-2', 1050,'av pedro leon torres'),
3 (0,'Manuel Perez','2010-11-22', 60,'carrera 21 a'),
4 (0,'Ana Rodriguez','2014-10-25', 1580,'av Venezuela')
```

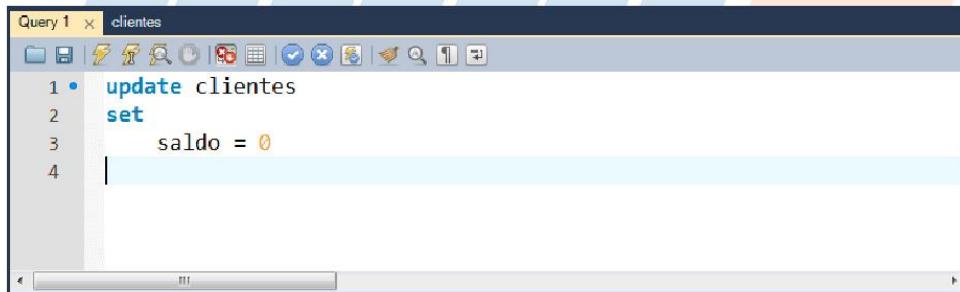
## Capítulo 5. UPDATE

### 5.1.- Sintaxis

Una sentencia UPDATE de SQL es utilizada para modificar los valores de un conjunto de registros existentes en una tabla. Si no se coloca la cláusula WHERE serán actualizados todos los registros de la tabla. Los nuevos valores que se le asignan a los campos deben cumplir con las restricciones (como con el INSERT). La forma general de la instrucción UPDATE es la siguiente:

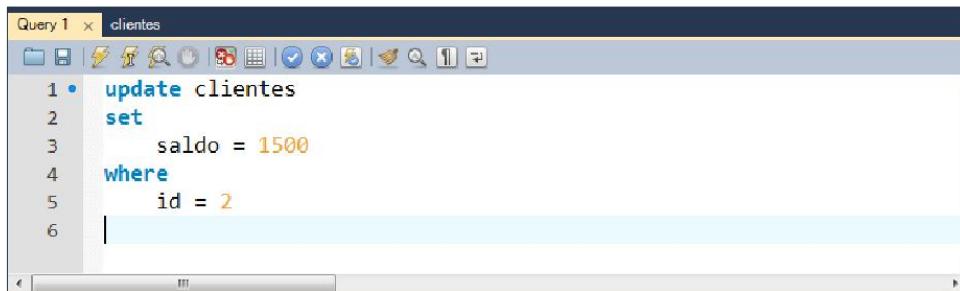
```
UPDATE TABLE_NAME SET column_name = VALUE [, column_name = VALUE ...] [WHERE condition]
```

Veamos algunos ejemplos del uso de UPDATE, donde a todos los registros de la tabla "clientes" se les asigna el valor 0 (cero) en el campo "saldo":



```
Query 1 x clientes
update clientes
set
    saldo = 0
```

En el siguiente ejemplo se le asigna el valor 1500 al campo "saldo" del registro cuyo "id" es igual a 2. Si no hay algún registro con este "id" no sucede nada:



```
Query 1 x clientes
update clientes
set
    saldo = 1500
where
    id = 2
```

## Capítulo 6. DELETE

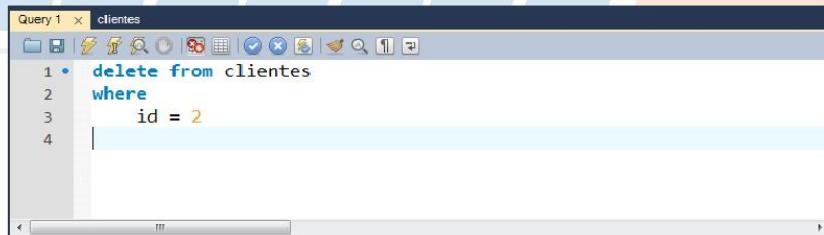
### 6.1.- Sintaxis

Una sentencia DELETE de SQL borra uno o más registros existentes en una tabla. Si no se coloca la cláusula WHERE serán eliminados todos los registros. En este caso sería más eficiente usar la instrucción TRUNCATE vista en el capítulo anterior. La forma general del DELETE es la siguiente:

```
DELETE FROM nombre_tabla [WHERE condicion];
```

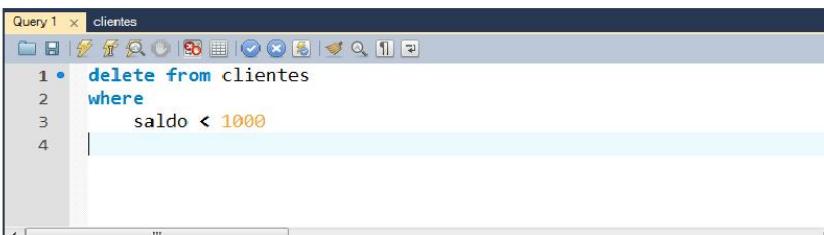
### 6.2.- Eliminar Algunos Registros

En el siguiente ejemplo del uso del DELETE, se elimina el registro de la tabla "clientes" cuyo "id" sea igual a 2. Si el "id" es la clave primaria se eliminará un solo registro, a menos, que no haya algún registro en la tabla que cumpla la condición, en cuyo caso, no pasa nada:



```
Query 1 x clientes
1 • delete from clientes
2   where
3     id = 2
4 |
```

En el siguiente ejemplo se eliminan todos los registros que cumplan con la condición de tener el campo "saldo" menor a 1000:



```
Query 1 x clientes
1 • delete from clientes
2   where
3     saldo < 1000
4 |
```

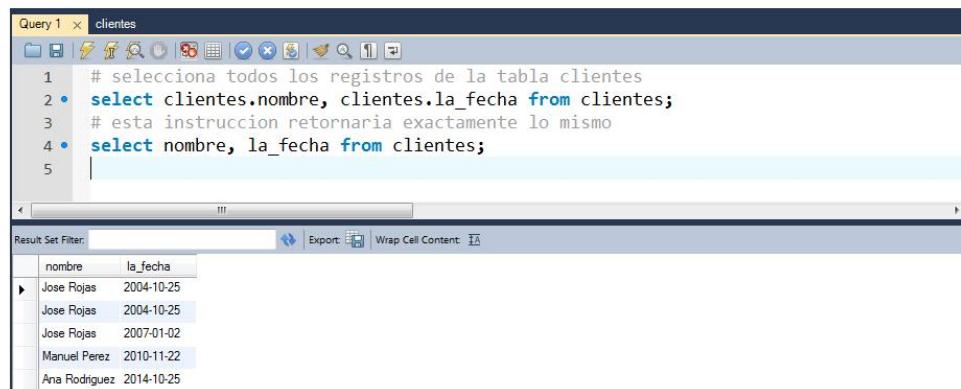
## Capítulo 7. SELECT

### 7.1.- Sintaxis

La sentencia SELECT permite consultar los datos almacenados en una tabla o varias tablas de la base de datos. La forma general de la instrucción SELECT es la siguiente:

```
SELECT [ALL | DISTINCT ]
       <nombre_campo> [{,<nombre_campo>}]
  FROM <nombre_tabla>|<nombre_vista>
       [{,<nombre_tabla>}|<nombre_vista>]
[WHERE <condicion> [{ AND|OR <condicion>}]]
[GROUP BY <nombre_campo> [{,<nombre_campo>}]]
[HAVING <condicion>[{ AND|OR <condicion>}]]
[ORDER BY <nombre_campo>|<indice_campo> [ASC | DESC]
       [{,<nombre_campo>}|<indice_campo> [ASC | DESC ]]]]
```

Se colocan los nombres de los campos que se quieren consultar, en el orden en que se desea que sean mostrados. A los nombres de los campos se le antepone el nombre de la tabla, pero esto es útil solamente si la sentencia incluye varias tablas, de lo contrario es innecesario. Veamos un ejemplo de como hacer un SELECT:



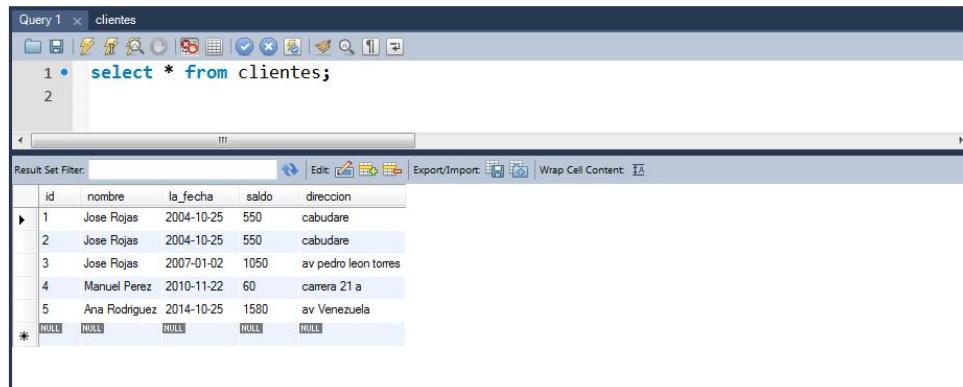
The screenshot shows the MySQL Workbench interface. The top window is titled "Query 1" and has a tab labeled "clientes". The code area contains the following SQL queries:

```
1 # selecciona todos los registros de la tabla clientes
2 • select clientes.nombre, clientes.la_fecha from clientes;
3 # esta instruccion retornaria exactamente lo mismo
4 • select nombre, la_fecha from clientes;
```

The results grid below shows the output of the second query:

nombre	la_fecha
Jose Rojas	2004-10-25
Jose Rojas	2004-10-25
Jose Rojas	2007-01-02
Manuel Perez	2010-11-22
Ana Rodriguez	2014-10-25

Si una tabla tiene muchos campos y se desea verlos todos, se usan un asterisco "\*". Por defecto los datos son devueltos en el orden en que estan en la base de datos. El siguiente es un ejemplo:



The screenshot shows a MySQL Workbench interface with a query editor titled "Query 1" and a results table. The query is:

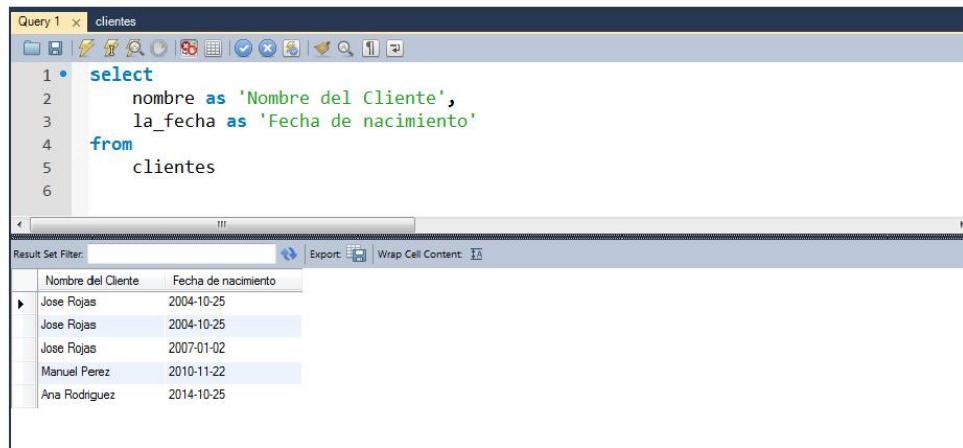
```
1 • select * from clientes;
```

The results table has columns: id, nombre, la\_fecha, saldo, and direccion. The data is:

	id	nombre	la_fecha	saldo	direccion
1	1	Jose Rojas	2004-10-25	550	cabudare
2	2	Jose Rojas	2004-10-25	550	cabudare
3	3	Jose Rojas	2007-01-02	1050	av pedro leon torres
4	4	Manuel Perez	2010-11-22	60	camera 21 a
5	5	Ana Rodriguez	2014-10-25	1580	av Venezuela
*	NULL	NULL	NULL	NULL	NULL

## 7.2.- Alias de Campos

Se le pueden colocar sobre nombres a los campos de la consulta resultante. Esto es especialmente útil cuando los nombres de los campos no son representativos o en la consulta se incluyen varias tablas y campos con el mismo nombre en estas tablas. El siguiente es un ejemplo de como usar la cláusula "AS":



The screenshot shows a MySQL Workbench interface with a query editor titled "Query 1" and a results table. The query is:

```
1 • select
2     nombre as 'Nombre del Cliente',
3     la_fecha as 'Fecha de nacimiento'
4   from
5     clientes
6
```

The results table has columns: Nombre del Cliente and Fecha de nacimiento. The data is:

	Nombre del Cliente	Fecha de nacimiento
1	Jose Rojas	2004-10-25
2	Jose Rojas	2004-10-25
3	Jose Rojas	2007-01-02
4	Manuel Perez	2010-11-22
5	Ana Rodriguez	2014-10-25

### 7.3.- Ordenamiento

Si necesita que los datos estén en un orden específico debemos usar la cláusula ORDER BY. La cláusula ORDER BY es la instrucción que nos permite especificar el orden en el que serán devueltos los datos. Podemos especificar la ordenación ascendente o descendente a través de las palabras clave ASC y DESC. El valor predeterminado es ASC si no se especifica al hacer la consulta. Veamos un ejemplo:

The screenshot shows the MySQL Workbench interface. The top window is titled "Query 1" and contains the following SQL code:

```
1 • select
2      *
3      from
4        clientes
5      order by saldo
```

The bottom window shows the result set of the query. The table has columns: id, nombre, la\_fecha, saldo, and direccion. The data is as follows:

	id	nombre	la_fecha	saldo	direccion
4	4	Manuel Perez	2010-11-22	60	carrera 21 a
1	1	Jose Rojas	2004-10-25	550	cabudare
2	2	Jose Rojas	2004-10-25	550	cabudare
3	3	Jose Rojas	2007-01-02	1050	av pedro leon torres
5	5	Ana Rodriguez	2014-10-25	1580	av Venezuela
*	HULL	HULL	HULL	HULL	HULL

Si dos registros cumplen con el mismo valor del campo colocado en el ORDER BY, se puede utilizar un segundo campo, de forma tal que sea un segundo criterio de ordenamiento. Veamos un ejemplo:

Query 1 x clientes

```
1 • select
2      *
3  from
4      clientes
5  order by nombre desc , saldo asc
6
```

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

	id	nombre	la_fecha	saldo	direccion
▶	4	Manuel Perez	2010-11-22	60	carrera 21 a
	2	Jose Rojas	2004-10-25	330	cabudare
	1	Jose Rojas	2004-10-25	550	cabudare
	3	Jose Rojas	2007-01-02	1050	av pedro leon torres
	5	Ana Rodriguez	2014-10-25	1580	av Venezuela
*	HULL	HULL	HULL	HULL	HULL



## Capítulo 8. WHERE. PARTE 1

### 8.1.- Sintaxis

La cláusula WHERE es la instrucción que permite filtrar el resultado de una sentencia SELECT (aunque también se usa en las instrucciones delete y update como ya se ha mostrado). Habitualmente no se desea obtener toda la información existente en la tabla, sino que se quiere obtener sólo los datos que resulten útiles en un momento. La cláusula WHERE filtra los datos antes de ser devueltos por la consulta. Cuando en la Cláusula WHERE se necesita incluir un campo tipo texto, se debe colocar el valor entre comillas simples. Para entender el uso del WHERE se muestra un ejemplo de una tabla llamada "ordenes", con los siguientes registros:

The screenshot shows the MySQL Workbench interface. A query window titled 'Query 1' contains the SQL command: 'SELECT \* FROM ordenes;'. Below the query window is a results table with the following data:

	id	fecha_ordenes	precio_ordenes	id_cliente
1	1	2010-09-23	1120	1
2	2	2007-02-21	1990	2
3	3	2006-06-09	400	1
4	4	2006-04-01	700	1
5	5	2005-03-30	2120	3
6	6	2011-11-17	160	2
*	NULL	NULL	NULL	NULL
	ordenes	3	x	

Para el WHERE se pueden usar los operadores relacionales tradicionales: =, >, <, >=, <=, <>. Veamos un ejemplo:

The screenshot shows the MySQL Workbench interface. In the top tab bar, 'Query 1' is selected, followed by 'cliente' and 'ordenes'. The main area displays a SQL query:

```
1 •  SELECT
2      *
3  FROM
4      ordenes
5  where
6      precio_ordenes > 1500
```

Below the query is a results grid with the following data:

	id	fecha_ordenes	precio_ordenes	id_cliente
▶	2	2007-02-21	1990	2
	5	2005-03-30	2120	3
*	NULL	NULL	NULL	NULL

At the bottom of the window, there are 'Apply' and 'Cancel' buttons.

## 8.2.- Operadores Básicos

Se pueden combinar expresiones lógicas con los operadores AND Y OR, donde se pueden usar expresiones más complejas. Cualquier expresión booleana que retorne verdadero o falso es válida. Si es necesario se deben usar paréntesis. Veamos un ejemplo:

```
Query 1  cliente  ordenes x
1 •  SELECT
2      *
3      FROM
4          ordenes
5      WHERE
6          precio_ordenes < 1500 AND id_cliente = 1

Result Set Filter: Edit Export/Import:
id fecha_ordenes precio_ordenes id_cliente
1 2010-09-23    1120           1
3 2006-06-09    400            1
4 2006-04-01    700            1
* NULL          NULL           NULL

ordenes 5 x Apply Cancel
```

### 8.3.- Operadores Avanzados

Existen otros operadores muy particulares, ellos son:

- BETWEEN: para verificar si un valor está entre 2 valores (uno mínimo y uno máximo).
- IN: para verificar si un valor está en un conjunto de valores.
- LIKE: para verificar si un valor es similar a otro.

Veamos un ejemplo del uso operador de "BETWEEN", donde se consultan los registros cuyo "precio\_ordenes" está entre 500 y 1500:

The screenshot shows the MySQL Workbench interface with a query editor and a results grid. The query is:

```
1 *   SELECT
2       *
3   FROM
4       ordenes
5   WHERE
6       precio_ordenes between 500 and 1500
```

The results grid displays the following data:

	id	fecha_ordenes	precio_ordenes	id_cliente
▶	1	2010-09-23	1120	1
	4	2006-04-01	700	1
*	NULL	NULL	NULL	NULL

Veamos un ejemplo del operador IN, donde se consultan los registros que tengan en el campo "id\_cliente" los valores 1 y 3. Entre los paréntesis pueden haber tantos valores como sean necesarios:

The screenshot shows the MySQL Workbench interface with a query editor and a results grid. The query is:

```
1 *   SELECT
2       *
3   FROM
4       ordenes
5   WHERE
6       id_cliente IN (1 , 3)
```

The results grid displays the following data:

	id	fecha_ordenes	precio_ordenes	id_cliente
▶	1	2010-09-23	1120	1
	3	2006-06-09	400	1
	4	2006-04-01	700	1
	5	2005-03-30	2120	3
*	NULL	NULL	NULL	NULL

Veamos un ejemplo del uso de LIKE:

The screenshot shows the MySQL Workbench interface. The top window is titled "Query 1" and contains the following SQL code:

```
1 •   SELECT
2      *
3      FROM
4          ordenes
5      WHERE
6          fecha_ordenes like "201%"
```

Below the code is a result set table with the following data:

	id	fecha_ordenes	precio_ordenes	id_cliente
▶	1	2010-09-23	1120	1
▶	6	2011-11-17	160	2
*	NULL	NULL	NULL	NULL

The bottom window is titled "ordenes 15" and has "Apply" and "Cancel" buttons.

## Capítulo 9. WHERE. PARTE 2

### 9.1.- Valores Nulos

NULL indica que un valor es desconocido o que aún no existe un valor asignado dentro de un campo de una tabla de una base de datos. NULL no pertenece a ningún dominio de datos (no pertenece a los enteros, ni a los booleanos, ni a los flotantes, etc), se puede considerar como un marcador que indica la ausencia de un valor. NULL no debe confundirse con un valor 0, ya que el valor 0 pertenece a algún tipo de dato (entero o flotante) mientras que, como ya se mencionó, NULL es la falta de un dato.

Para determinar si un campo tiene valor NULL no se puede utilizar el operador "`=`". Se debe utilizar el operador "`IS NULL`" en la cláusula "`WHERE`". Para determinar si no es nulo se debe usar el operador "`IS NOT NULL`".

### 9.2.- Limit

En ocasiones necesitamos solo una cantidad limitada registros de la consulta, por ejemplo, los 5 últimos registros que cumplen una condición. Para esta tarea, el estandar de SQL no define una cláusula, por lo que es necesario consultar la documentación del DBMS para conocer la forma específica de lograr este objetivo. Algunos DBMS tienen la cláusula "`TOP N`", donde N es el número de los N primeros registros que resultan de la consulta. Se coloca como parte del "`SELECT`". Otros DBMS utilizan la cláusula "`LIMIT N`" (Mysql es uno de ellos), que se coloca al final de la instrucción completa. Veamos un ejemplo:

```
Query 1 cliente ordenes x
SELECT *
FROM
    ordenes
ORDER BY ID DESC
LIMIT 2
```

	id	fecha_ordenes	precio_ordenes	id_cliente
▶	6	2011-11-17	160	2
	5	2005-03-30	2120	3
*	NULL	NULL	NULL	NULL

### 9.3.- Distinct

#### ACADEMIA DE SOFTWARE

Por defecto, un select se trae todos los registros que cumplan con la condición especificada en el "WHERE", pero en ocasiones existen registros repetidos y necesitamos los registros sin repeticiones. Para especificar que no se desean las filas repetidas, se utiliza la cláusula "DISTINCT" justo después de la palabra "SELECT". Veamos un ejemplo:

The image shows two side-by-side MySQL query windows. Both queries are identical:

```
Query 1 | ordenes x
1 • SELECT
2     fecha_ordenes, id_cliente
3 FROM
4     ordenes;
```

The left window displays all rows from the 'ordenes' table, including duplicates. The right window displays only unique combinations of 'fecha\_ordenes' and 'id\_cliente', effectively using the `SELECT DISTINCT` keyword.

fecha_ordenes	id_cliente
2010-09-23	1
2007-02-21	2
2006-06-09	1
2006-04-01	1
2005-03-30	3
2011-11-17	2
2007-02-21	2
2006-06-09	1
2005-03-30	3

fecha_ordenes	id_cliente
2010-09-23	1
2007-02-21	2
2006-06-09	1
2006-04-01	1
2005-03-30	3
2011-11-17	2



## Capítulo 10. JOIN

### 10.1.- Inner Join

La sentencia JOIN en SQL permite combinar registros de dos o más tablas en una base de datos relacional. Hay tres tipos de JOIN: interno (INNER), externo (OUTER) y cruzado (cross). En el curso usaremos el tipo interno solamente.

La cláusula INNER JOIN calcula el producto cruzado de todos los registros; así cada registro en la tabla A es combinado con cada registro de la tabla B; pero sólo permanecen aquellos registros en la tabla combinada que satisfacen las condiciones que se especifiquen. Este es el tipo de JOIN más utilizado, por lo que es considerado el tipo de combinación predeterminado.

Para entender su uso de INNER JOIN, veamos un ejemplo: La tabla "cliente" contiene el "nombre" de los clientes y el "id\_cliente" que representa la clave primaria. Mientras que la tabla "ordenes" contiene la "fecha\_ordenes", el "precio" y el "id\_empleado" que indica a qué "cliente" pertenece la orden. El campo "id\_cliente" es la clave foránea. Existen "cliente" que no tienen "ordenes", igualmente, existen "ordenes" asociadas a "clientes" que no existen en la tabla. Veamos las 2 tablas:

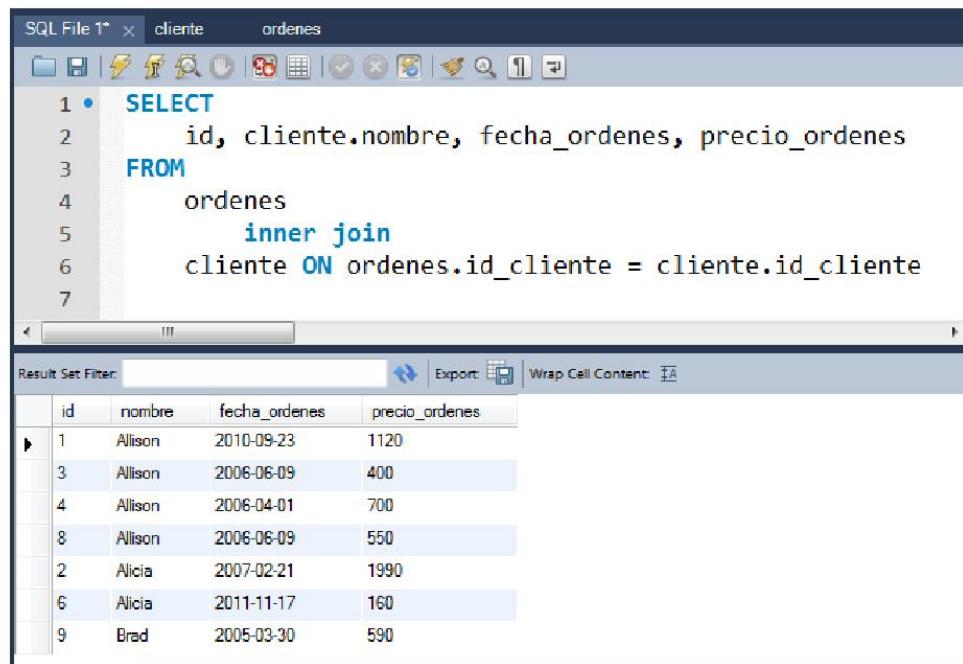
The screenshot shows two separate SQL developer sessions. The left session, titled 'SQL File 1\* cliente', contains the query: 'SELECT \* FROM cliente;'. Its results show a table with columns 'id\_cliente' and 'nombre', containing five rows: 1 (Alison), 2 (Alicia), 3 (Brad), 4 (Peter), and 5 (Steven). The right session, titled 'SQL File 1\* ordenes', contains the query: 'SELECT \* FROM ordenes;'. Its results show a table with columns 'id', 'fecha\_ordenes', 'precio\_ordenes', and 'id\_cliente', containing nine rows. The data includes various dates like 2010-09-23, prices like 1120, and client IDs like 1, 2, 3, 4, 5, 6, 7, 8, and 9.

id_cliente	nombre
1	Alison
2	Alicia
3	Brad
4	Peter
5	Steven

id	fecha_ordenes	precio_ordenes	id_cliente
1	2010-09-23	1120	1
2	2007-02-21	1990	2
3	2006-06-09	400	1
4	2006-04-01	700	1
5	2005-03-30	2120	8
6	2011-11-17	160	2
7	2007-02-21	1650	7
8	2006-06-09	550	1
9	2005-03-30	590	3

Con la siguiente consulta, se toman todos los registros de la tabla "clientes" y encuentran todas las combinaciones en la tabla "ordenes". La sentencia JOIN compara los valores en la columna "id\_cliente" en ambas tablas. Cuando no existe esta correspondencia entre algunas combinaciones, éstas no se muestran; es decir, que si el "id\_cliente" en la tabla "ordenes" no coincide con los id de la tabla "cliente", no se mostrará la orden la tabla resultante. Veamos el ejemplo:



The screenshot shows a SQL editor window titled "SQL File 1" with tabs for "cliente" and "ordenes". The code area contains the following SQL query:

```
1 •   SELECT
2       id, cliente.nombre, fecha_ordenes, precio_ordenes
3   FROM
4       ordenes
5       inner join
6           cliente ON ordenes.id_cliente = cliente.id_cliente
7
```

The results pane displays a table with the following data:

	id	nombre	fecha_ordenes	precio_ordenes
▶	1	Allison	2010-09-23	1120
	3	Allison	2006-06-09	400
	4	Allison	2006-04-01	700
	8	Allison	2006-06-09	550
	2	Alicia	2007-02-21	1990
	6	Alicia	2011-11-17	160
	9	Bred	2005-03-30	590

Los clientes "Peter" y "Steven" no son presentados en los resultados ya que ninguno de éstos tiene registros correspondientes en la otra tabla. Las "ordenes" con id 5 y 7 no aparecen porque tienen "id\_cliente" que no coincide con ningún registro de la tabla "cliente".

### 10.2.- Union

Combina los resultados de dos o más consultas en un solo conjunto de resultados que incluye todas las filas que pertenecen a las consultas de la

unión. La operación UNION es distinta de la utilización de combinaciones de columnas de dos tablas. La forma general de la cláusula UNION es la siguiente:

```
SELECT campo1, campo2, campon FROM tabla1
```

```
UNION
```

```
SELECT campo1, campo2, campon FROM tabla2
```

```
ORDER BY campon
```

A continuación se muestran las reglas básicas para combinar los conjuntos de resultados de dos consultas con UNION:

- El número y el orden de las columnas debe ser el mismo en todas las consultas.
- Los tipos de datos deben ser compatibles.

"Union" retorna los registros no repetidos de las dos consultas. Si que queremos ver todos los registros, incluyendo los repetidos, debemos usar "UNION ALL". También están las cláusulas "INTERCEPT" que retorna la intersección de los registros de una consulta con la de la otra (los que están en la consulta 1 y en la consulta 2) y el "EXCEPT" que retorna los registros que están en la consulta 1 pero no están en la consulta 2. Otra consideración es que la cláusula "order by" se coloca al final.

## Capítulo 11. FUNCIONES AGREGADAS. PARTE 1

### 11.1.- Max

La función MAX() retorna el máximo valor de la columna seleccionada. En SQL la sintaxis utilizada es de la siguiente manera:

```
SELECT MAX(nombre_columna) FROM nombre_tabla;
```

Veamos un ejemplo donde se visualiza el uso de la función MAX, en el cual se consulta la orden cuyo campo "precio\_orden" sea el máximo de todas las ordenes de la tabla:

The screenshot shows two MySQL Workbench windows. The left window, titled 'Query 1', contains the SQL statement: 'SELECT \* FROM ordenes;'. Its results pane displays a table with columns: id, fecha\_orden, precio\_orden, and id\_cliente. The data is as follows:

id	fecha_orden	precio_orden	id_cliente
1	2010-09-23	1120	1
2	2007-02-21	1990	2
3	2006-06-09	400	1
4	2006-04-01	700	1
5	2005-03-30	2120	3
6	2011-11-17	160	2

The right window, also titled 'Query 1', contains the SQL statement: 'select MAX(precio\_orden) from ordenes;'. Its results pane shows a single row with the value '2120'.

### 11.2.- Min

La función MIN() retorna el mínimo valor de la columna seleccionada. En SQL la sintaxis utilizada es de la siguiente manera:

```
SELECT MIN(nombre_columna) FROM nombre_tabla;
```

Veamos un ejemplo donde se visualiza el uso de la función MIN, en el cual se consulta la orden cuyo campo "precio\_ordenes" sea el mínimo de todas las ordenes de la tabla:

The screenshot shows two MySQL Workbench windows. The left window displays the results of the query `SELECT * FROM ordenes;`, which retrieves all columns for 6 rows from the 'ordenes' table. The right window displays the results of the query `select MIN(precio_ordenes) from ordenes;`, which retrieves the single value 160.

id	fecha_ordenes	precio_ordenes	id_cliente
1	2010-09-23	1120	1
2	2007-02-21	1990	2
3	2006-06-09	400	1
4	2006-04-01	700	1
5	2005-03-30	2120	3
6	2011-11-17	160	2

Result 8 x Read Only

ACADEMIA DE SOFTWARE

## Capítulo 12. FUNCIONES AGREGADAS. PARTE 2

### 12.1.- Count

La función COUNT() retorna el número de filas según los criterios que especificaron. Hay varias formas de usar la función "COUNT":

- count(campo): retorna el número de valores que se encuentran en la columna especificada. Los valores NULL no se cuentan.
- count(\*) : retorna el número de registros de una tabla.
- count(distinct columna): retorna el número de valores distintos a la columna especificada.

Veamos un ejemplo de cómo usar el COUNT( \* ) con el siguiente ejemplo, donde se muestran cuantos registros hay en la tabla ordenes:

The screenshot shows the MySQL Workbench interface with two queries.   
**Query 1:** A result set showing the 'ordenes' table with columns: id, fecha\_ordenes, precio\_ordenes, and id\_cliente. The data consists of 6 rows.   
**Query 2:** A result set showing the execution of the SQL statement: `select count(*) from ordenes`. The result is a single row with the value 6.

	id	fecha_ordenes	precio_ordenes	id_cliente
1	2010-09-23	1120	1	
2	2007-02-21	1990	2	
3	2006-06-09	400	1	
4	2006-04-01	700	1	
5	2005-03-30	2120	3	
6	2011-11-17	160	2	

Veamos un ejemplo de como usar el COUNT( DISTINCT campo) con el siguiente ejemplo, donde se muestran cuantos clientes diferentes hay en las ordenes:

```

Query 1 cliente ordenes
1 • SELECT * FROM ordenes;

Result Set Filter: Edit Wrap Cell Content: A
id fecha_ordenes precio_ordenes id_cliente
1 2010-09-23 1120 1
2 2007-02-21 1990 2
3 2006-06-09 400 1
4 2006-04-01 700 1
5 2005-03-30 2120 3
6 2011-11-17 160 2
NULL NULL NULL NULL
ordenes 3 Apply Cancel

Query 1 x cliente ordenes
1 • select
2 count(distinct id_cliente)
3
4 from
5 ordenes

Result Set Filter: Export Wrap Cell Content: A
count(distinct id_cliente)
3

```

## 12.2.- Sum

La función SUM() retorna la suma total de una columna numérica. En SQL la sintaxis utilizada es de la siguiente manera:

```
SELECT SUM(nombre_columna) FROM nombre_tabla;
```

Veamos un ejemplo donde se visualiza el uso de la función SUM, en el cual se suman el campo "precio\_ordenes" de todas las ordenes de la tabla:

```

Query 1 cliente ordenes
1 • SELECT * FROM ordenes;

Result Set Filter: Edit Wrap Cell Content: A
id fecha_ordenes precio_ordenes id_cliente
1 2010-09-23 1120 1
2 2007-02-21 1990 2
3 2006-06-09 400 1
4 2006-04-01 700 1
5 2005-03-30 2120 3
6 2011-11-17 160 2
NULL NULL NULL NULL
ordenes 3 Apply Cancel

Query 1 x cliente ordenes
1 • select
2 sum(precio_ordenes)
3
4 from
5 ordenes

Result Set Filter: Export Wrap Cell Content: A
sum(precio_ordenes)
6490

```

### 12.3.- Avg

La función AVG() retorna el valor promedio de una columna numérica. En SQL la sintaxis es de la siguiente manera:

```
SELECT AVG(nombre_columna) FROM nombre_tabla;
```

Veamos un ejemplo donde se visualiza el uso de la función AVG, en el cual se calcula el promedio de "precio\_ordenes" de todas las ordenes de la tabla:

The screenshot shows two windows of Oracle SQL Developer. The left window, titled 'Query 1 cliente ordenes', contains the following code and result:

```
1 • SELECT * FROM ordenes;
```

id	fecha_ordenes	precio_ordenes	id_cliente
1	2010-09-23	1120	1
2	2007-02-21	1990	2
3	2006-06-09	400	1
4	2006-04-01	700	1
5	2005-03-30	2120	3
6	2011-11-17	160	2
*	NULL	NULL	NULL

The right window, also titled 'Query 1 cliente ordenes', contains the following code and result:

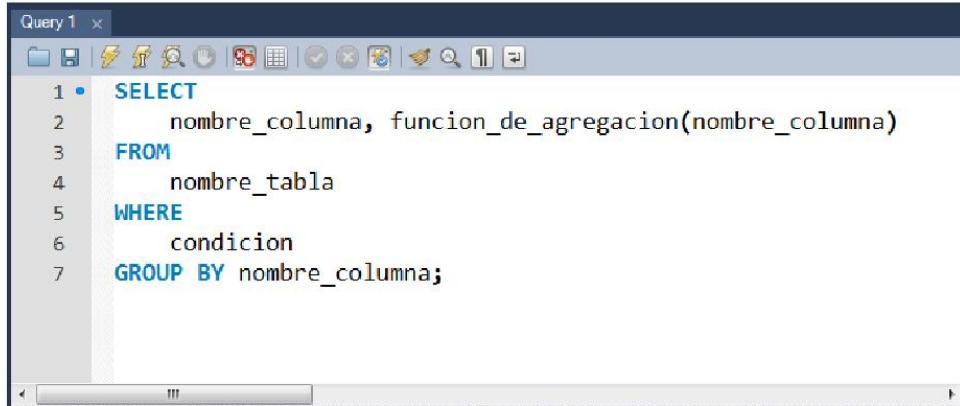
```
1 • select
2     avg(precio_ordenes)
3     from
4     ordenes
```

avg(precio_ordenes)
1081.6667

## Capítulo 13. GROUP BY

### 13.1.- Group By

La instrucción GROUP BY se utiliza en conjunción con las funciones de agregado para agrupar el conjunto de resultados de una o más columnas. La cláusula "GROUP BY" se coloca al final de la instrucción. La forma general de usar el GROUP BY es la siguiente:



The screenshot shows a MySQL query editor window titled "Query 1". The code area contains the following SQL statement:

```
1 • SELECT
2     nombre_columna, funcion_de_agregacion(nombre_columna)
3     FROM
4     nombre_tabla
5     WHERE
6     condicion
7     GROUP BY nombre_columna;
```

El siguiente ejemplo de como se utiliza el GROUP BY, en el cual se usa una función de agregado ( SUM ) para sumar los "precios\_ordenes" agrupados por "id\_cliente". El resultado será la suma total de las ordenes de cada cliente que esta en la tabla "ordenes":

```

Query 1 cliente ordenes
1 • SELECT * FROM ordenes;

Result Set Filter: Edit Export
id fecha_orden precio_orden id_cliente
1 2010-09-23 1120 1
2 2007-02-21 1990 2
3 2006-06-09 400 1
4 2006-04-01 700 1
5 2005-03-30 2120 3
6 2011-11-17 160 2
NULL NULL NULL NULL
ordenes 3 x Apply Cancel

Query 1 cliente ordenes
1 • SELECT
2 id_cliente, SUM(precio_orden)
3 FROM
4 ordenes
5 GROUP BY id_cliente

Result Set Filter: Export
id_cliente SUM(precio_orden)
1 2220
2 2150
3 2120
Result 12 x Read Only !

```

Se puede combinar el GROUP BY con el WHERE. Veamos un ejemplo:

```

Query 1 cliente ordenes
1 • SELECT * FROM ordenes;

Result Set Filter: Edit Export
id fecha_orden precio_orden id_cliente
1 2010-09-23 1120 1
2 2007-02-21 1990 2
3 2006-06-09 400 1
4 2006-04-01 700 1
5 2005-03-30 2120 3
6 2011-11-17 160 2
NULL NULL NULL NULL
ordenes 3 x Apply Cancel

Query 1 cliente ordenes
1 • SELECT
2 id_cliente, SUM(precio_orden)
3 FROM
4 ordenes
5 WHERE
6 fecha_orden > "2006-01-01"
7 GROUP BY id_cliente

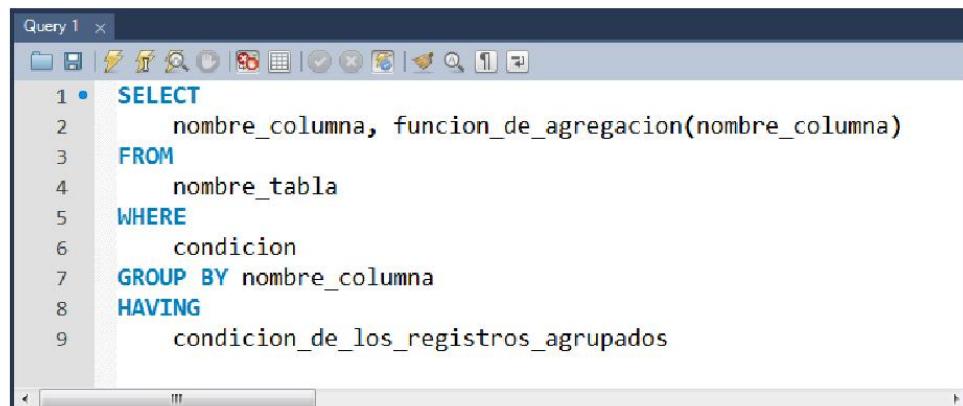
Result Set Filter: Export
id_cliente SUM(precio_orden)
1 2220
2 2150
Result 13 x Read Only !

```

### 13.2.- Having

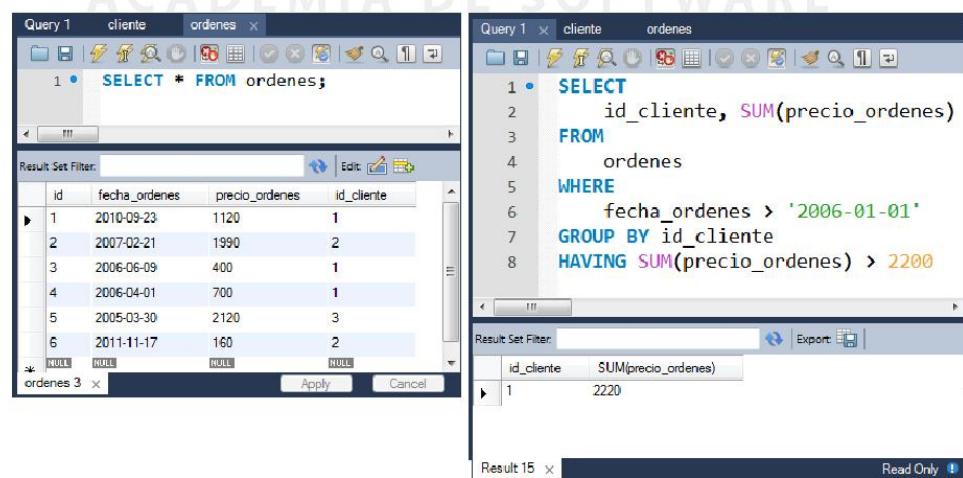
Especifica una condición que debe cumplirse para que los datos sean devueltos por la consulta. Su funcionamiento es similar al de WHERE pero aplicado al conjunto de resultados devueltos por la consulta. Debe aplicarse siempre junto a GROUP BY y la condición debe estar referida a los campos contenidos en ella. La cláusula HAVING se utiliza puesto que la palabra clave

WHERE no puede utilizarse con las funciones de agregado en sus condiciones.  
En SQL la sintaxis que se utiliza es de la siguiente manera:



```
Query 1
1 • SELECT
2     nombre_columna, funcion_de_agregacion(nombre_columna)
3     FROM
4         nombre_tabla
5     WHERE
6         condicion
7     GROUP BY nombre_columna
8     HAVING
9         condicion_de_los_registros_agrupados
```

Veamos un ejemplo de como se utiliza el HAVING, en el cual se usa una función de agregado ( SUM ) para sumar los "precios\_ordenes" agrupados por "id\_cliente". El resultado será la suma total de las ordenes de cada cliente que esta en la tabla "ordenes" cuya suma sea mayor a 2200:



The screenshot shows two queries in MySQL Workbench. The left pane displays the raw data from the 'ordenes' table, while the right pane shows the corresponding SQL query and its results.

**Left pane (Raw Data):**

```
Query 1 cliente ordenes
1 • SELECT * FROM ordenes;
```

	id	fecha_ordenes	precio_ordenes	id_cliente
1	1	2010-09-23	1120	1
2	2	2007-02-21	1990	2
3	3	2006-06-09	400	1
4	4	2006-04-01	700	1
5	5	2005-03-30	2120	3
6	6	2011-11-17	160	2

**Right pane (SQL and Results):**

```
Query 1 x cliente ordenes
1 • SELECT
2     id_cliente, SUM(precio_ordenes)
3     FROM
4         ordenes
5     WHERE
6         fecha_ordenes > '2006-01-01'
7     GROUP BY id_cliente
8     HAVING SUM(precio_ordenes) > 2200
```

id_cliente	SUM(precio_ordenes)
1	2220

Result 15 x Read Only !

## Capítulo 14. SUB CONSULTAS. PARTE 1

### 14.1.- Concepto

Una sub consulta es una consulta que está anidada típicamente dentro de un SELECT, ya sea en la lista de campos o en el WHERE, o incluso otra sub consulta. Una sub consulta es una sentencia SELECT que aparece dentro de otra sentencia SELECT. Una sub consulta tiene la misma sintaxis que una sentencia SELECT normal exceptuando que aparece encerrada entre paréntesis.

Una sub consulta puede retornar un escalar (un valor único), un registro, una columna o una tabla (uno o más registros de una o más columnas). Éstas se llaman consultas de escalar, columna, registro y tabla.

Las principales ventajas de sub consultas son:

- Permiten consultas estructuradas de forma que es posible aislar cada parte de un comando.
- Proporcionan un modo alternativo de realizar operaciones que de otro modo necesitarían joins y uniones complejos.
- Son, en la opinión de mucha gente, leíbles. De hecho, fue la innovación de las sub consultas lo que dio a la gente la idea original de llamar a SQL “Structured Query Language.”

### 14.2.- Sub Consultas de Un Escalar

En su forma más sencilla, una sub consulta es una sub consulta escalar que retorna un único valor. Una sub consulta escalar es un operando simple, y puede usarse prácticamente en cualquier sitio en que un valor de columna o literal sea legal. Se puede utilizar en el SELECT o en el WHERE. La forma general de la sub consulta en el SELECT es la siguiente:

```
SELECT campo1, campo2, (SUB CONSULTA) FROM tabla2
```

La forma general de la sub consulta de un escalar en el WHERE es la siguiente:

```
SELECT campo1, campo2 FROM tabla WHERE campo3 OPERADOR (SUB CONSULTA)
```

Donde (SUB CONSULTA) es una instrucción SELECT que retorne un solo valor, típicamente se utilizan funciones agregadas de SQL.

Veamos un ejemplo en el uso de sub consultas en el SELECT. En el ejemplo se puede ver que para cada registro de la tabla "clientes" se muestra el número de registros de la tabla "ordenes":

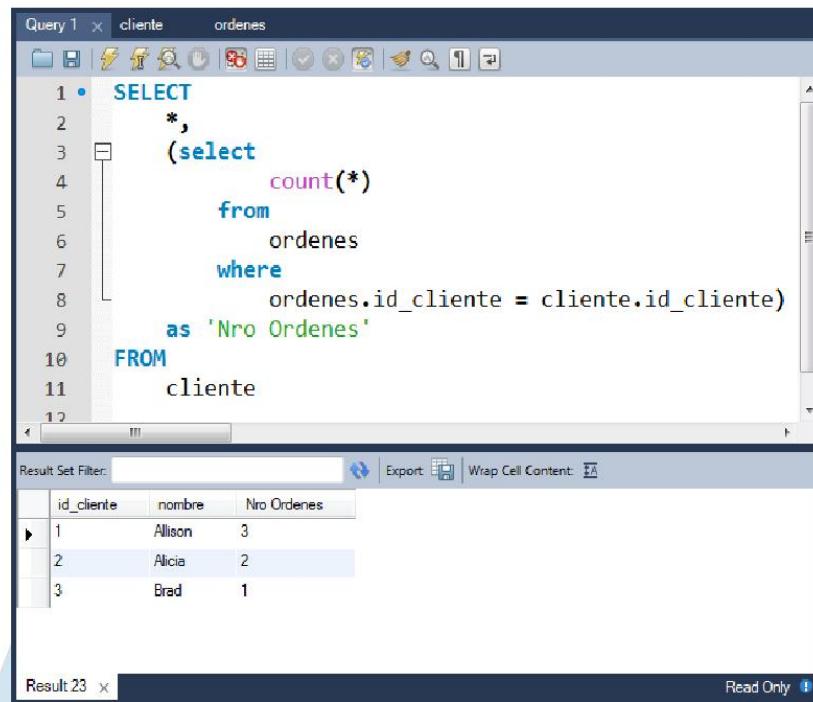
The screenshot shows the MySQL Workbench interface. The top window is titled 'Query 1' and contains the following SQL code:

```
1 •  SELECT
2      *
3      (select
4          count(*)
5          from
6              ordenes)
7      as 'Nro Ordenes'
8  FROM
9      cliente C
```

The bottom window is titled 'Result 21' and displays a table with three rows of data:

	id_cliente	nombre	Nro Ordenes
1	1	Allison	6
2	2	Alicia	6
3	3	Brad	6

Pero supongamos que deseamos ver las cantidades de "ordenes" de cada "cliente". En este caso debemos usar un WHERE en la sub consulta, que haga referencia a uno de los campos de la tabla "cliente" que es donde se está haciendo la consulta principal:



The screenshot shows a MySQL Workbench interface with two tabs: 'cliente' and 'ordenes'. A query is being run in the 'cliente' tab:

```

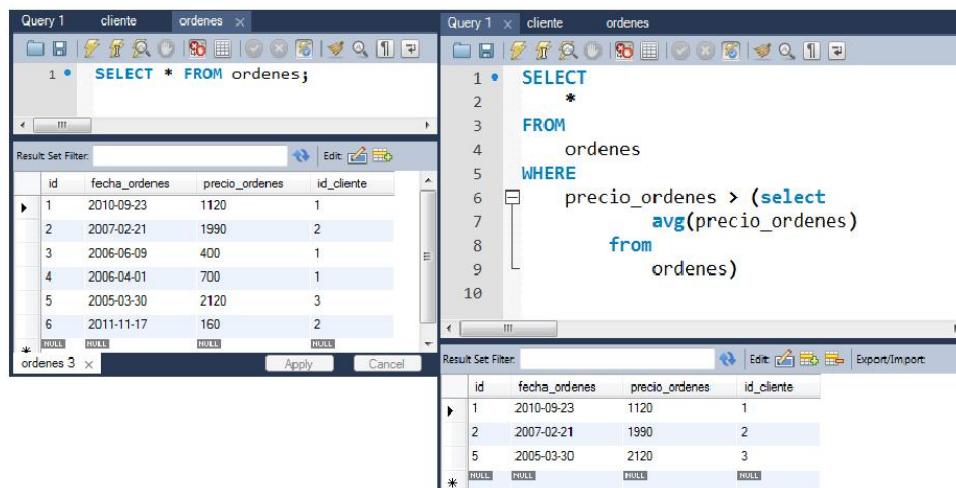
1 • SELECT
2     *,
3     (
4         select
5             count(*)
6         from
7             ordenes
8         where
9             ordenes.id_cliente = cliente.id_cliente
10        as 'Nro Ordenes'
11    FROM
12        cliente

```

The result set shows three rows:

	id_cliente	nombre	Nro Ordenes
1	Allison	3	
2	Alicia	2	
3	Brad	1	

Veamos un ejemplo de una sub consulta en el WHERE, en la que se consultan las "ordenes" cuyo "precio\_orden" sea mayor que el promedio de precio de todas las ordenes:



The screenshot shows two MySQL Workbench queries. The left query retrieves all columns from the 'ordenes' table:

```
1 • SELECT * FROM ordenes;
```

The right query uses a subquery in the WHERE clause to filter orders where the price is greater than the average price:

```

1 • SELECT
2     *
3     FROM
4         ordenes
5     WHERE
6         precio_orden > (select
7             avg(precio_orden)
8             from
9                 ordenes)

```

The results show three rows from the 'ordenes' table:

	id	fecha_orden	precio_orden	id_cliente
1	1	2010-09-23	1120	1
2	2	2007-02-21	1990	2
3	5	2005-03-30	2120	3

## Capítulo 15. SUB CONSULTAS. PARTE 2

### 15.1.- Exist Y Not Exist

Si una subconsulta retorna algún registro, entonces EXISTS subquery es TRUE, y NOT EXISTS subquery es FALSE. Por ejemplo:

```
SELECT c1 FROM t1 WHERE EXISTS (SELECT * FROM t2);
```

Una consulta usando EXISTS se puede escribir equivalente a una consulta IN o ANY. NOT EXISTS funciona de forma contraria a EXISTS. La cláusula WHERE NOT EXISTS se cumple si la subconsulta no devuelve ninguna fila. Típicamente se utiliza en la sub consulta una sentencia WHERE que hace referencia a un campo de la consulta principal. Por ejemplo:

```
SELECT c1 FROM t1 WHERE EXISTS (SELECT * FROM t2 WHERE t1.c1=t2.c2);
```

En este ejemplo se están consultando registros de la tabla t1 si existe algún registro en la tabla t2 donde el campo t1.c1 sea igual al campo t2.c2.

En el siguiente ejemplo, se tiene la tabla "clientes" y la tabla "ordenes". La tabla "ordenes" posee una clave foránea llamada "id\_clientes", pero se puede notar que hay varios registros en la tabla "clientes" donde el campo "id\_cliente" posee valores que no están en la tabla "ordenes", es decir, hay clientes con ordenes y clientes sin ordenes. Veamos las tablas:

The image shows two separate database windows side-by-side. The left window, titled 'SQL File 1\*' and 'cliente', contains the SQL query 'SELECT \* FROM cliente;'. Its result set is a table with columns 'id\_cliente' and 'nombre', containing five rows of data: Allison, Alicia, Brad, Peter, and Steven. The right window, also titled 'SQL File 1\*' and 'ordenes', contains the SQL query 'SELECT \* FROM ordenes;'. Its result set is a table with columns 'id', 'fecha\_ordenes', 'precio\_ordenes', and 'id\_cliente', containing nine rows of data.

id_cliente	nombre
1	Allison
2	Alicia
3	Brad
4	Peter
5	Steven
*	NULL

id	fecha_ordenes	precio_ordenes	id_cliente
1	2010-09-23	1120	1
2	2007-02-21	1990	2
3	2006-06-09	400	1
4	2006-04-01	700	1
5	2005-03-30	2120	8
6	2011-11-17	160	2
7	2007-02-21	1650	7
8	2006-06-09	550	1
9	2005-03-30	590	3

Si queremos consultar los "clientes" que tienen "ordenes" usando la cláusula EXISTS, se escribe la siguiente instrucción:

The image shows a single SQL editor window with the title 'SQL File 1\*' and tabs 'cliente' and 'ordenes'. It displays the following SQL code:

```

1 • SELECT
2   *
3   FROM
4     cliente
5   where
6     EXISTS(
7       SELECT
8         *
9         FROM
10        ordenes
11      WHERE
12        cliente.id_cliente = ordenes.id_cliente)

```

The result set is a table with columns 'id\_cliente' and 'nombre', containing three rows of data: Allison, Alicia, and Brad.

id_cliente	nombre
1	Allison
2	Alicia
3	Brad
*	NULL

Si por el contrario, consultar los "clientes" que NO tienen "ordenes" usando la cláusula NOT EXISTS, se escribe la siguiente instrucción:

The screenshot shows a MySQL Workbench interface. The SQL editor tab contains the following query:

```
SQL File 1* x cliente ordenes
1 •  SELECT
2      *
3  FROM
4      cliente
5  where
6      NOT EXISTS( SELECT
7          *
8      FROM
9          ordenes
10     WHERE
11         cliente.id_cliente = ordenes.id_cliente)
```

The results pane displays a table with two rows:

	id_cliente	nombre
▶	4	Peter
▶	5	Steven
*	NULL	NULL

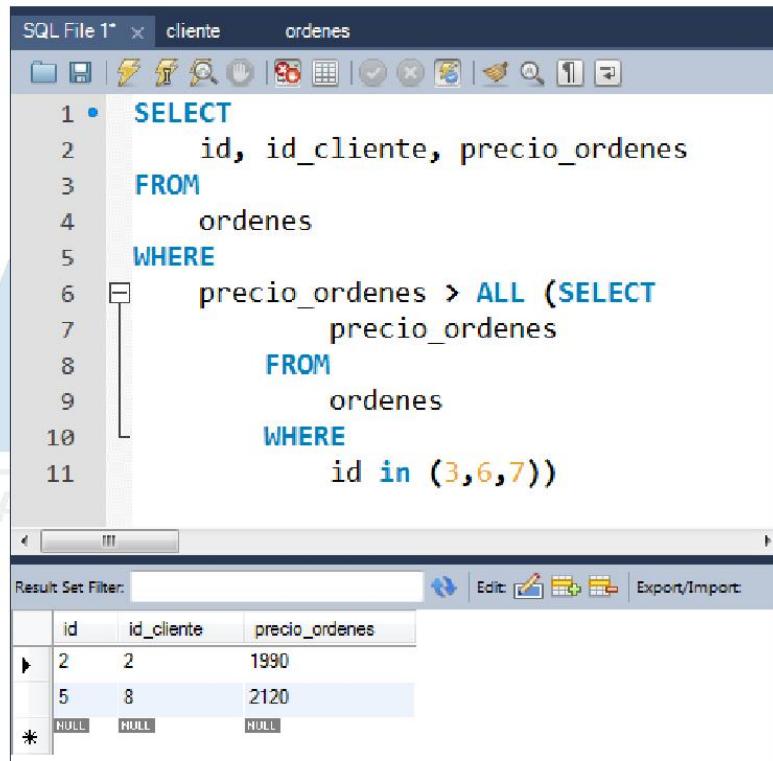
### 15.2.- All

La palabra ALL, que debe seguir a un operador de comparación, significa "return TRUE si la comparación es TRUE para ALL todos los valores en la columna que retorna la subconsulta." La forma general del uso del ALL es la siguiente:

```
SELECT s1 FROM t1 WHERE s1 OPERADOR ALL (SUBCONSULTA)
```

donde "OPERADOR" es cualquier operador válido (<,>,=,<>,in, etc). La consulta retornará los registros en los cuales la condición se cumpla para todos los registros que retorna la sub consulta.

Veamos un ejemplo de como usar la cláusula ALL, donde se muestran las ordenes cuyo precio se mayor que el precio de todas ordenes que tengan id 3,6 y 7:



The screenshot shows a SQL developer interface with a query editor and a results grid. The query is:

```
1 • SELECT
2     id, id_cliente, precio_ordenes
3     FROM
4         ordenes
5     WHERE
6         precio_ordenes > ALL (SELECT
7             precio_ordenes
8             FROM
9                 ordenes
10            WHERE
11                id in (3,6,7))
```

The results grid shows two rows of data:

	id	id_cliente	precio_ordenes
▶	2	2	1990
▶	5	8	2120
*	NULL	NULL	NULL

### 15.3.- ANY, SOME, IN

Cuando la sub consulta retorna uno o más registros se utiliza en el WHERE. Hay tres operadores que son equivalentes que se utilizan en este caso: ANY, IN , SOME. La palabra clave ANY , que debe seguir a un operador de comparación, significa "return TRUE si la comparación es TRUE para ANY

(cualquiera) de los valores en la columna que retorna la subconsulta." Por ejemplo:

```
SELECT s1 FROM t1 WHERE s1 > ANY (SELECT s1 FROM t2);
```

Este ejemplo describe: seleccione el campo "s1" de la tabla "t1" cuyo campo "s1" sea mayor que cualquiera de los registros de la sub consulta que obtiene el campo "s1" de la tabla "t2". Es importante destacar que la sub consulta debe tener un solo campo en la lista de campos.

Veamos un ejemplo del uso del operador IN:

The screenshot displays two MySQL Workbench windows. The left window, titled 'cliente ordenes', contains a query 'SELECT \* FROM ordenes;' and its results. The results show six rows of data with columns: id, fecha\_orden, precio\_orden, and id\_cliente. The right window, also titled 'cliente ordenes', contains a more complex query using an IN operator. This query selects from the 'cliente' table where the 'id\_cliente' is in a subquery that filters 'orden' records where 'fecha\_orden' is greater than '2007-01-01'. The results of this query show two client records: Allison and Alicia.

id	fecha_orden	precio_orden	id_cliente
1	2010-09-23	1120	1
2	2007-02-21	1990	2
3	2006-06-05	400	1
4	2006-04-01	700	1
5	2005-03-30	2120	3
6	2011-11-17	160	2

id_cliente	nombre
1	Allison
2	Alicia