FACULTAD DE INGENIERIA MECÁNICA Y ELECTRICA
LICENCIATURA EN INGENIERIA EN SISTEMAS COMPUTACIONALES

Manual de Practicas

SQL

Materia:

Base de Datos

Semestre: 04

Maestra:

M. en C. Martha Elizabeth Evangelista Salazar

Agosto 2013
2da. Edición

UNIVERSIDAD DE COLIMA

Rector
M. en C. José Eduardo Hernández Nava

Secretario General
Mtro. Christian Jorge Torres Ortiz Zermeño

Coordinador General de Docencia
Dra. Martha Alicia Ochoa Echeverría

Director General de Educación Superior
Dr. Carlos Eduardo Monroy Galindo

FACULTAD DE INGENIERÍA MECÁNICA Y ELÉCTRICA

Director del Plantel
M. en C. José Luis Álvarez Flores

Coordinador Académico
M. en I. Salvador Barragán González

Titular de la materia
M. en C. Martha Elizabeth Evangelista Salazar

Ing. Brenda Cerrato Abdalá / Encargado del Centro de Cómputo

Presentación

La gestión de las bases de datos ha evolucionado desde una aplicación informática especializada hasta convertirse en parte esencial de los entornos informáticos modernos. Por lo tanto, los conocimientos acerca de los sistemas de bases de datos se han convertido en una parte imprescindible de la información en informática.

Propósito del manual.

El alumno conocerá los fundamentos de la teoría de base de datos para aplicarlos en el diseño, desarrollo, implementación y administración en el uso de los manejadores de bases de datos.

Requerimientos:

- Computadora portátil o de escritorio
- Windows 97 o superior
- ACCESS 2007 o superior

CONTENIDO TEMÁTICO

1.- LENGUAJES DE CONSULTA COMERCIALES (SQL)

Un lenguaje de consulta comercial proporciona una interfaz más amigable al usuario. Un ejemplo de este tipo de lenguaje es el SQL, (Structured Query Languaje, Lenguaje de Consulta Estructurado).

Las partes más importantes del SQL son:

- LDD: Lenguaje de definición de datos (que nos permite crear las estructuras)
- LMD: Lenguaje de manipulación de datos (que nos permite tener acceso a las estructuras para suprimir, modificar e insertar)

1.1- TIPOS BÁSICOS DE DOMINIOS:

- **char(n).-** Una cadena de caracteres de longitud fija, con una longitud n especificada por el usuario. También se puede utilizar la palabra completa **character.**
- **varchar(n)**.- Una cadena de caracteres de longitud variable con una longitud máxima n especificada por el usuario. La forma completa, **character varying** es equivalente.
- **int**.- Para especificar números enteros. La palabra **integer** es equivalente.
- **numeric(p,d)**.- Un número de coma fijo, cuya precisión la especifíca el usuario. El número está formado por ***p*** dígitos, ***d*** pertenece a la parte decimal. Ejemplo: numeric(3,1) permite el número 33.9, más no 0.96, 3.98, 444.5, etc.
- **Smallint**.- Enteros cortos .

1.2.- ESTRUCTURA BÁSICA:

La estructura básica de una expresión en SQL contiene 3 partes: ***Select, From y Where.***

- La cláusula ***Select*** se usa para listar los atributos que se desean en el resultado de una consulta.
- ***From,*** Lista las relaciones que se van a examinar en la evaluación de la expresión.
- ***Where***, es la definición de las condiciones a las que puede estar sujeta una consulta.

La consulta típica de SQL tiene la siguiente forma:

Select A1,A2,A3...An
From r1,r2,r3...rm
Where Condición(es)

Donde:

A1,A2,A3...An: Representan a cada atributo(s) o campos de las tablas de la base de datos relacional.
R1,r2,r3....rm: Representan a la(s) tabla(s) involucradas en la consulta.
Condición: Es el enunciado que rige el resultado de la consulta.

Si se omite la cláusula ***Where,*** la condición es considerada como verdadera, la lista de atributos (A1,A2..An) puede sustituirse por un asterisco (*), para seleccionar todos los atributos de todas las tablas que aparecen en la cláusula **From.**

2.- FUNCIONAMIENTO DEL SQL.

El SQL forma el producto cartesiano de las tablas involucradas en la cláusula From, cumpliendo con la condición establecida en la orden Where y después proyecta el resultado con la orden Select.

Para la realización de los siguientes ejercicios es necesario contar con las siguientes tablas de la entidad bancaria:

Tabla cliente

Nombre_cliente	Calle_cliente	Ciudad_cliente
Abril	Preciado	Valsaín
Amo	Embajadores	Arganzuela
Badorrey	Delicias	Valsaín
Fernández	Jazmín	León
Gómez	Carretas	Cercedas
González	Arenal	La granja
López	Mayor	Peguerinos
Pérez	Carretas	Cerceda
Rodríguez	Yeserías	Cádiz
Rupérez	Ramblas	León
Santos	Mayor	Peguerinos
Valdivieso	Goya	Vigo

Tabla cuenta

Numero_cuenta	Nombre_sucursal	saldo
C-101	Centro	500
C-102	Navacerrada	400
C-201	Galapagar	900
C-215	Becerril	700
C-217	Galapagar	750
C-222	Moralzarzal	700
C-305	Collado Mediano	350

Tabla impositor

Nombre_cliente	Número_ cuenta
Abril	C-305
Gómez	C-215
González	C-101
González	C-201
López	C-102
Rupérez	C-222
Santos	C-217

Tabla sucursal

Nombre_sucursal	Ciudad_sucursal	activos
Becerril	Aluche	400
Centro	Arganzuela	9000
Collado Mediano	Aluche	8000
Galapagar	Arganzuela	7100
Moralzarzal	La granja	2100
Navacerrada	Aluche	1700
Navas de la Asunción	Alcalá de Henares	300
Segovia	Cerceda	3700

Tabla préstamo

Número_ préstamo	Nombre_sucursal	importe
P-11	Collado mediano	900
P-14	Centro	1,500
P-15	Navacerrada	1,500
P-16	Navacerrada	1,300
P-17	Centro	1,000
P-23	Moralzarzal	2,000
P-93	Becerril	500

Tabla prestatario

Nombre_cliente	Número_ préstamo
Gómez	P-11
Sotoca	P-14
López	P-15
Fernández	P-16
Santos	P-17
Gómez	P-23
Pérez	P-93

2.1.- La cláusula SELECT- El resultado de la cláusula **select** es, por supuesto, una relación (se mostrará todos los atributos que se encuentren en esa cláusula).

Ejemplo: Obtener el nombre de todas las sucursales de la relación préstamo.

select nombre_sucursal
from préstamo

El resultado es una relación consistente en un único atributo con el encabezado nombre_sucursal.

nombre_sucursal
Collado_mediano
Centro
Navacerrada
Navacerrada
Centro
Moralzarzal
Becerril

En el caso en los que se desea forzar la eliminación de los valores duplicados, se inserta la palabra clave **distinct** después de **select.**

select **distinct** nombre_sucursal
from préstamo

nombre_sucursal
Collado_mediano
Centro
Navacerrada
Moralzarzal
Becerril

Se permite utilizar la cláusula **all** para especificar de manera explícita que no se eliminen los duplicados.

```
select all nombre_sucursal
from préstamo
```

El símbolo asterisco (*) se puede utilizar para denotar “todos los atributos”. Ejemplo: Mostrar el contenido de la tabla préstamo.

```
select numero_prestamo, nombre_sucursal, importe
from préstamo
```

ó

```
select *
from préstamo
```

La cláusula **select** puede contener también expresiones aritméticas que contengan los operadores +,-,*,/ y operen sobre constantes o atributos de las tuplas. Por ejemplo, la consulta.

```
select numero_prestamo, nombre_sucursal, importe*10
from préstamo
```

devolverá una relación que es igual a la de préstamo, salvo que el atributo importe estará multiplicado por 10.

SQL también proporciona tipos de datos especiales, tales como varias formas del tipo date (fecha) y permite que varias funciones aritméticas operen sobre esos tipos.

2.2.- La cláusula WHERE- A continuación se muestra el uso de la clausula where: Ejemplo.

Obtener todos los préstamos de la sucursal Navacerrada con importe superior a 1200.00

```
select numero_prestamo
from préstamo
where nombre_sucursal=”Navacerrda” and importe>1200.00
```

SQL utiliza las conectivas lógicas **and, or** y **not** (en lugar de los símbolos matemáticos $\wedge,\vee,\neg$) en la clausula where. También se pueden utilizar los operadores de comparación <,<=,>,>=,<> y = . También incluye el operador de comparación **between** (se encuentra entre un rango de comparación). De forma análoga, se puede utilizar el operador de comparación **not between.**

Ejercicios:

1.- Obtener el número de préstamo de los préstamos con importe entre 900.00 y 1,500.00.

```
select numero_préstamo
from préstamo
where importe >=900.00 and importe <=1,500.00
```

ó

```
select numero_préstamo
from préstamo
where importe between 900.00 and 1,500.00
```

2.- Obtener el nombre de la sucursal cuyos activos no estén entre 7,000.00 y 9,000.00.

3.- Obtener el número de cuenta de la sucursal “Galapagar” con saldo superior a 800.00.

2.3.- La cláusula FROM- Define por sí misma un producto cartesiano de las relaciones que aparecen en la cláusula.

Ejercicios:

1.- Para todos los clientes que tienen un préstamo en el banco, obtener el nombre, el número de préstamo y su importe.

```
select nombre_cliente, prestatario.numero_préstamo, importe
from prestatario, préstamo
where prestatario.numero_prestamo = préstamo.numero_prestamo
```

2.- Determinar el nombre, el número de préstamo y sus importes de todos los prestamos de la sucursal Navacerrada.

```
select  nombre_cliente, prestatario.numero_préstamo, importe
```

from prestatario, préstamo
where prestatario.numero_prestamo = préstamo.numero_prestamo **and** nombre_sucursal=’Navacerrada’.

2.4.- Operación renombramiento

SQL proporciona un mecanismo para renombrar tanto relaciones como atributos. Utiliza la cláusula **AS** de la siguiente forma:

nombre-antiguo **AS** nombre_nuevo

para el caso de atributos

select nombre_cliente, prestatario.numero_préstamo **AS** id_prestamo, importe
from prestatario, préstamo
where prestatario.numero_prestamo = préstamo.numero_prestamo

para el caso de relaciones

select nombre_cliente, T.numero_préstamo, S.importe
from prestatario AS T, préstamo AS S
where T.numero_prestamo = S.numero_prestamo

2.5- Operaciones con cadena de caracteres

SQL especifica la cadena de caracteres encerrándolas entre comillas simples, ‘Navacerrada’.

La operación más utilizada sobre la comparación de cadenas de caracteres es el operador **like**. Para la descripción de los patrones se utilizan dos caracteres especiales:

- Tanto por ciento (%).- El carácter % coincide con cualquier subcadena de caracteres.
- El subrayado (_).- El _ coincide con cualquier carácter

Los caracteres en minúsculas difieren de los caracteres en mayúsculas y viceversa

- ‘Nava%’ coincide con cualquier cadena de caracteres que empiece con “Nava”.

- '%cer%' conincide con cualquier cadena que contenga "cer" como subcadena. Ejemplo "Navacerrada", "Becerril", "Cáceres", etc.
- ' _ _ _' coincide con cualquier cadena que contenga tres caracteres.
- _ _ _%' coincide con cualquier cadena que contenga al menos tres caracteres.

Ejemplo: Determinar el nombre de todos los clientes cuya dirección contenga la subcadena de caracteres 'Mayor'

```
select nombre_cliente
from cliente
where calle_cliente like '%Mayor%'
```

SQL permite buscar discordancias en lugar de concordancias utilizando el operador de comparación **not like.**

2.6.- Orden en la presentación de las tuplas.

SQL permite cierto control sobre el orden en el cual se presentan las tuplas de una relación. La cláusula **order by** hace que las tuplas se presenten en un orden especificado. (De manera predeterminada la orden **order by** coloca los elementos en orden ascendente).

Ejemplo: Obtener una relación en orden alfabético de todos los clientes que tienen un préstamo en la sucursal "Navacerrada"

```
select distinct nombre_cliente
from prestatario, préstamo
where prestatario.numero_prestamo = préstamo.numero_prestamo and
      nombre_sucursal='Navacerrada'.
order by nombre_cliente
```

Para especificar el tipo de ordenación se puede especificar **desc** para ordenar de manera descendente ó **asc** para ordenar ascendentemente.

Colocar toda la relación préstamo en orden descendente según su importe, se ordena de manera ascendente según su préstamo.

```
select *
from préstamo
order by importe desc, numero_prestamo asc
```

Lo orden **order by** solo se utilizará cuando sea estrictamente necesario (porque ordenar un gran número de tuplas puede resultar costoso).

3.- OPERACIONES SOBRE CONJUNTOS.

Las operaciones de SQL **union, intersect** y **except** corresponden con las operaciones del algebra relacional $\cup$, $\cap$ y -. Las relaciones que participen en las operaciones han de ser compatibles, esto es, deben tener el mismo conjunto de atributos.

3.1.- La operación unión.- La operación **unión** elimina los valores duplicados automáticamente

Ejercicio: Encontrar los clientes del banco que ***tienen un*** préstamo, ***una cuenta o las dos*** cosas en el banco.

```
(select nombre_cliente
from impositor)
union
(select nombre_cliente
from prestatario)
```

Si desea conserver todos los duplicados hay que escribir **unión all** en lugar de **unión.**

```
(select nombre_cliente
from impositor)
unión all
(select nombre_cliente
from prestatario)
```

3.2.- La operación intersección.- La operación **intersección (intersect)** elimina los valores duplicados automáticamente

Ejercicio: Encontrar los clientes del banco que tienen ***tanto un*** préstamo ***como una*** cuenta en el banco

```
(select nombre_cliente
from impositor)
intersect
(select nombre_cliente
from prestatario)
```

Si desea conserver todos los duplicados hay que escribir **intersect all** en lugar de **intersect.**

```
(select nombre_cliente
from impositor)
intersect all
```

```
(select  nombre_cliente
from     prestatario)
```

3.3.- La operación excepto.- La operación **excepto (except)** elimina los valores duplicados automáticamente

Ejercicio: Encontrar los clientes del banco que tienen ***una*** cuenta ***pero no un*** prestamo en el banco

```
(select  nombre_cliente
from     impositor)
except
(select  nombre_cliente
from     prestatario)
```

Si desea conserver todos los duplicados hay que escribir **except all** en lugar de **except.**

```
(select  nombre_cliente
from     impositor)
except all
(select  nombre_cliente
from     prestatario)
```

4.- SUBCONSULTAS ANIDADAS

Existen dos formas para realizar consultas: Join de Querys y Subquerys. Cuando en la sentencia **From** colocamos los nombres de las tablas separados por comas se dice que efectuamos una consulta de la forma **Join de Querys**, en este caso se requiere anteponer el nombre de la tabla y un punto al nombre del atributo. En el Join de Querys el resultado que se produce con las tablas que intervienen en la consulta es la concatenación de las tablas, en donde los valores de una columna de la primera tabla coinciden con los valores de una segunda tabla, la tabla de resultado tiene una fila por cada valor coincidente que resulte de las dos tablas originales.

Para ejemplificar esto, consideremos 2 tablas: Tabla1 y Tabla2, entonces:

C1	C2	C3
A	AAA	10
B	BBB	45
C	CCC	55
D	DDD	20
E	EEE	20
F	FFF	90
G	GGG	15
H	HHH	90

CA	CB
35	R
10	S
65	T
20	U
90	V
90	W
75	X
90	Y
35	Z

Resultado de la operación Join:

C1	C2	C3	CA	CB
A	AAA	10	10	S
D	DDD	20	20	U
E	EEE	20	20	U
F	FFF	90	90	V
F	FFF	90	90	W
F	FFF	90	90	Y
H	HHH	90	90	V
H	HHH	90	90	W
H	HHH	90	90	Y

Como podemos observar, la comparación se efectuó por las columnas C3 y CA, que son donde se encontraron valores iguales, el resultado muestra una tupla por cada coincidencia encontrada.

Cuando las consultas se anidan se conoce como **Subquerys** o subconsultas. Este tipo de consulta obtiene resultados parciales reduciendo el espacio requerido para realizar una consulta.

Nota: Todas las consultas que se resuelven con subquerys pueden resolverse con Join de Querys, pero no todas las consultas hechas con Join de Querys pueden resolverse utilizando Subquerys.

Para ejemplificar lo anterior consideremos siguiente

Ejercicio: Tenemos la siguientes tablas de una escuela

ALUMNO - cursa - MATERIA, que tienen los siguientes atributos:

NControl	NControl	Clave
NombreA	Clave	NombreM
Especialidad	Calif	Creditos
Dirección		

Representando en tablas a los atributos quedarían de la siguiente forma:

Tabla alumno:

NControl	NombreA	Especialidad	Dirección

Tabla cursa:

NControl	Clave	Calif

Tabla materia:

Clave	NombreM	Creditos

Ejercicios:

- Obtener los nombres de los alumnos cuyas materias que cursan tienen créditos igual a ocho.

```
SELECT NombreA
FROM Alumno
WHERE NControl IN (SELECT Ncontrol
                   FROM Cursa
                   WHERE Clave IN (SELECT Clave
                                   FROM Materia
                                   WHERE Creditos=8));
```

- Obtener el nombre de la materia que cursa el alumno con número de control 97310211 con créditos igual a ocho.

```
SELECT NombreM
FROM Materia
WHERE creditos=8 and clave IN (SELECT clave
                               FROM cursa
                               WHERE   NControl=97310211;
```

- Obtener el número de control del alumno que tenga alguna calificación igual a 100

```
SELECT  DISTINCT NControl
FROM Cursa
WHERE Calif=100;
```

- Obtener el nombre de las materias que cursa el alumno Salvador Chávez.

```
SELECT NombreM
FROM Materia
WHERE Clave IN(SELECT DISTINCT (Clave)
               FROM Cursa
               WHERE NControl IN (SELECT NControl)
                                  FROM Alumno
                                  WHERE NombreA='Salvador
                                   Chávez'));
```

5.- FUNCIONES DE AGREGACIÓN

Existen funciones que permiten la agilización de consultas similares a una hoja de cálculo, ya que trabajan en base a renglones y columnas.

COUNT ( ): Cuenta el número de tuplas en la columna establecida
MIN ( ): Localiza el valor mínimo de la columna establecida
MAX ( ): Localiza el valor máximo de la columna establecida.
AVG ( ): Obtiene el promedio de valores de la columna establecida
SUM ( ): Obtiene el valor total que implican los valores obtenidos en la columna establecida.

Ejercicios:

- Obtener el número de alumnos que existen en la carrera de Ingeniería en Sistemas Computacionales.

```
SELECT Count (*)
FROM Alumno
WHERE especialidad='ISC';
```

- Obtener la máximo calificación que ha obtenido J.M. Cadena.

```
SELECT Max(Calif)
FROM Cursa
WHERE NControl IN (SELECT NControl
                   FROM Alumno
                   WHERE NombreA= 'J.M. Cadena ');
```

- Obtener el promedio de calificaciones de Salvador Chávez.

```
SELECT Avg (Calif)
FROM Cursa
WHERE NCotrol IN (SELECT NControl
                  FROM Alumno
                  WHERE NombreA='Salvador Chávez');
```

- Obtener la suma total de las calificaciones obtenidas por Daniel Colín.

```
SELECT Sum (Calif)
FROM Cursa
WHERE NControl IN (SELECT NControl
                   FROM Alumno
                   WHERE NombreA='Daniel Colín');
```

Hasta aquí hemos visto el manejo sencillo de realizar consultas con SQL, hay que destacar que en la realización de consultas anidadas se tiene que poner cuidando a la prioridad de los operadores, teniendo cuidado también al momento de agrupar los paréntesis que involucran las condiciones con los operadores.

6.- MODIFICACIÓN DE LA BASE DE DATOS

Como se mencionó al inicio de este apartado del SQL, esté cuenta con módulos DDL, para la definición de datos que nos permite crear o modificar la estructura de las tablas.

Las instrucciones para realizar estas operaciones son:

CREATE TABLE: Nos permite crear una tabla de datos vacía.
INSERT: Permite almacenar registros en una tabla creada.
UPDATE: Permite modificar datos de registros almacenados en la tabla.
DELETE: Borra un registro entero o grupo de registros de una tabla.
CREATE INDEX: Crea un índice que nos puede auxiliar para las consultas.
DROP TABLE: Permite borrar una tabla.
DROP INDEX: Borra el índice indicado.

Para ejemplificar las instrucciones anteriores consideremos el ejemplo

ALUMNO - cursa - MATERIA, que tienen los siguientes atributos:

ALUMNO	cursa	MATERIA
NControl	NControl	Clave
NombreA	Clave	NombreM
Especialidad	Calif	Creditos
Dirección		

6.1.- Estructura de la sentencia CREATE TABLE.

CREATE TABLE <Nombre de la tabla>
(
Atributo 1: tipo de dato longitud **,**
Atributo 2: tipo de dato longitud **,**
Atributo 3: tipo de dato longitud **,**
:
:
Atributo n: tipo de dato longitud **,**

PRIMARY KEY (Opcional)) **;**

Los campos pueden definirse como NOT NULL de manera opcional excepto en la llave primaria para lo cual es obligatorio. Además al definir la llave primaria se genera automáticamente un índice con respecto al campo llave; para definir la llave la denotamos dentro de los paréntesis de PRIMARY KEY.

Ejercicio:

Crear la tabla alumno con los atributos antes descritos, tomando como llave el número de control.

```
CREATE TABLE Alumno
( NControl char(8) NOT NULL,
NombreA char(20),
Especialidad char(3),
Dirección char(30),
PRIMARY KEY (NControl) );
```

Tabla Alumno:

NControl	NombreA	Especialidad	Dirección

Puede existir más de una llave primaria, esto es si se requiere, se crearán tantos índices como llaves primarias se establezcan.

Pueden existir tantos campos Not Null (No nulos) como se requieran; En si estructurar la creación de una tabla es siempre parecida al ejemplo anterior.

6.2 Estructura de la sentencia INSERT

```
INSERT
   INTO  Nombre de la tabla a la que se le va a insertar el registro
   VALUES  (Conjunto de valores del registro);
```

Ejercicio: Insertar en la tabla Alumno, antes creada los datos del alumno Daniel colín, con numero de control 95310518 de la especialidad de Ingeniería civil, con domicilio Abasolo Norte #45.

```
INSERT
INTO Alumno
VALUES("95310518","Daniel Colín","IC","Abasolo Norte #45") ;
```

Nótese que la inserción de los datos se realiza conforme la estructura que se implanto en la tabla, es decir en el orden en que se creo dicha tabla. En caso de querer omitir un dato que no sean no nulos solamente se ponen las comillas indicando el vacío de la cadena.

6.3.- Estructura de la Sentencia CREATE INDEX

CREATE INDEX Nombre que se le asignara al índice.
ON Nombre de la taba a la cual se le creara el índice (Campo(s) por el cual se creara el índice);

Ejercicio: Crear un índice de la tabla Alumno por el campo Especialidad.

```
CREATE INDEX Indice1
ON Alumno(Especialidad);
```

Este índice contendrá a todos los alumnos ordenados por el campo especialidad.

```
CREATE INDEX UNIQUE INDEX Indice2
ON Alumno (Especialidad);
```

En la creación de este índice utilizamos la sentencia UNIQUE, es un indicador para permitir que se cree un índice único por especialidad, esta sentencia siempre se coloca antes de CREATE INDEX. En este ejemplo se creara un índice que contenga un alumno por especialidad existente.

6.4.- Estructura de la sentencia UPDATE

UPDATE Nombre de la tabla en donde se modificaran los datos.
SET Valores
WHERE (Condición);

Ejercicio: Modificar el número de control del registro de Daniel Colín de la Tabla alumno por el número 96310518.

```
UPDATE Alumno
SET NControl  '96310518'
WHERE NombreA='Daniel Colín';
```

6.5.- Estructura de la sentencia DROP TABLE

DROP TABLE Nombre de la tabla a borrar;

Ejercicio: Borrar la tabla Alumno creada anteriormente.

DROP TABLE Alumno;

6.6.- Estructura de la sentencia DROP INDEX

DROP INDEX Nombre del índice a borrar;

Ejercicio: Borrar el índice Indice1 creado anteriormente.

DROP INDEX Indice1;

6.7.- Estructura de la sentencia DELETE

DELETE
FROM Nombre de la tabla
WHERE Condición;

Ejercicio:- Borrar el registro cuyo número de control es 95310386.

DELETE
FROM Alumno
WHERE Control='95310386';

- Borrar todos los registros de la tabla alumno.

DELETE
FROM Alumno;

En el primer ejemplo, se borrara todo el registro (todos los datos), del alumno con número de control = 95310386.

En el segundo ejemplo se borraran todos los registros de la tabla alumno, pero sin borrar la estructura de la tabla, ya que la orden **Delete** solo borra registros, la sentencia **Drop Table** es la que borra toda la estructura de la tabla junto con los registros de la misma.

7.- VISTAS.

Una vista se define en SQL usando la orden CREATE VIEW. Para definir una vista debemos dar a la vista un nombre y declarar la consulta que calcula la vista. Una vez que establecemos una vista, podemos ejecutar una sentencia SELECT que referencie a esa vista. El sistema asociará la vista SQL con una tabla base y extraerá y visualizará, entonces, los datos de la tabla base.

Esto significa que una vista no contiene datos duplicados de una tabla base. No tiene absolutamente ningún dato, puesto que no es una tabla real, todo el proceso

se realiza con los datos almacenados en la tabla base. Es decir se percibe como una tabla virtual.

Las órdenes que se utilizan para la manipulación de vistas son:

CREATE VIEW: Crea una tabla virtual.
DROP VIEW: Elimina una vista creada anteriormente.

7.1 Estructura de la sentencia CREATE VIEW.

CREATE VIEW Nombre de la vista **AS** (Expresión de consulta);

Para nuestros ejemplos consideremos de nuevo la tabla llamada CURSO, que contiene los siguientes campos:

Tabla 7.1.- Tabla Curso

Nombre del campo	Descripción
NumC	Número del curso, único para identificar cada curso
NombreC	Nombre del curso, también es único
DescC	Descripción del curso
Creditos	Créditos, número de estos que gana al estudiante al cursarlo
Costo	Costo del curso.
Depto	Departamento académico que ofrece el curso.

Contenido de la tabla Curso

Que contiene los siguientes datos:

NumC	NombreC	DescC	Creditos	Costo	Depto
A01	Liderazgo	Para público General	10	100.00	Admón.
S01	Introducción a la inteligencia artificial	Para ISC y LI	10	90.00	Sistemas.
C01	Construcción de torres	Para IC y Arquitectura	8	0.00	Ciencias

B01	Situación actual y perspectivas de la alimentación y la nutrición	Para IB	8	80.00	Bioquímica
E01	Historia presente y futuro de la energía solar	IE e II	10	100.00	Electromecánica.
S02	Tecnología OLAP	Para ISC y LI	8	100.00	Sistemas
C02	Tecnología del concreto y de las Estructuras	Para IC	10	100.00	Ciencias
B02	Metabolismo de lípidos en el camarón	Para IB	10	0.00	Bioquímica
E02	Los sistemas eléctricos de potencia	Para IE	10	100.00	Electromecánica
S03	Estructura de datos	Para ISC y LI	8	0.00	Sistemas
A01	Diseño bioclimático	Para Arquitectura	10	0.00	Arquitectura
C03	Matemáticas discretas	General	8	0.00	Ciencias
S04	Circuitos digitales	Para ISC	10	0.00	Sistemas
S05	Arquitectura de Computadoras	Para ISC	10	50.00	Sistemas
I01	Base de Datos Relacionales	Para ISC y LI	10	150.00	Informática

Ejercicio:

* Crear una vista (tabla virtual), denominada CursosS, que contenga las filas solo correspondientes a cursos ofrecidos por el departamento Sistemas. La vista deberá contener todas las columnas de la tabla CURSO, con la excepción de la columna Depto, la secuencia, de izquierda a derecha de las columnas, deberá ser: NombreC, NumC, Creditos, Costo Y DescC.

```
CREATE VIEW CursosS AS
  SELECT NombreC,NumC,Creditos,Costo,DescC
  FROM CURSO
  WHERE DescC='Sistemas';
```

Observemos que después del nombre de la vista ponemos la sentencia AS, esto para definir la estructura de la vista, la estructura en si de la vista esta formada por la consulta anteriormente vista utilizando la orden SELECT.

* Crear una vista denominada CursosCaros, correspondientes a las filas de la tabla CURSO, en donde la tarifa exceda de $150, las columnas de la vista deberán tener los nombres ClaveCurso, NombreCurso y CostoCaro.

```
CREATE VIEW CursosSCaros(ClaveCurso,NombreCurso,CostoCaro) As
   SELECT NumC,NombreC, Costo
   FROM Curso
   WHERE Costo > 150;
```

Observamos que después del nombre de la vista CursosCaros ponemos los nombres que se nos pidieron tuvieran los campos de la vista(ClaveCurso,...), después se realiza la consulta correspondiente para generar el resultado deseado.

Visualizar las vistas

Creamos una tabla virtual que contiene los datos de las consultas que deseamos, ahora nos falta visualizar estos datos, para ello utilizamos la sentencia SELECT y realizamos la consulta:

```
   SELECT *
   FROM CursosCaros;
```

De esta consulta podemos observar que mostramos todos los campos que la vista contiene, aunque podemos visualizar solo alguno de ellos, también observamos que sustituimos el nombre de la vista por el de la tabla junto a la sentencia FROM, esto es por que una vista es una tabla virtual, pero guarda los datos como cualquier tabla normal.

7.2 Estructura de la sentencia DROP VIEW.

Eliminar una vista

Como si fuera una tabla normal, las vistas también pueden borrarse, para ello utilizamos la sentencia DROP VIEW.

Estructura de la sentencia DROP VIEW.

```
      DROP VIEW Nombre de la vista a borrar;
```

Ejercicio: Borrar la vista CursosCaros creada anteriormente.

```
   DROP VIEW CursosCaros;
```

BIBLIOGRAFIA:

Bibliografía básica para el desarrollo de la Unidad:

- Silbertschatz, A., Korth, Sudarshan S. (2002) *Fundamentos de bases de datos*. México. Mc Graw Hill, (Cuarta edición).
- Piatini Mario, Castañon Adoración de Miguel. (1999). *Fundamentos y modelos de bases de datos*. (Segunda edición). México. Alfaomega ra-ma.
- R. Rebeca. (2000). *Diseño de bases de datos relacionales con Access y SQL server*. Mc. Graw Hill.

Bibliografía complementaria para el desarrollo de la Unidad:

- Tutorial de SQL. (2012). SQL básico. Fecha de consulta: Abril de 2013. Link http://sql.11sql.com/
- Cazares, Claudio. Tutorial de SQL. Fecha de consulta: Agosto 2013. Link: http://www.unalmed.edu.co/~mstabare/Sql.pdf