



ESCUELA DE CIENCIAS BÁSICAS E INGENIERÍA

Ingeniería de Sistemas

ASIGNATURA: Bases de Datos I

CORPORACIÓN UNIVERSITARIA REMINGTON
DIRECCIÓN PEDAGÓGICA

Este material es propiedad de la Corporación Universitaria Remington (CUR), para los estudiantes de la CUR en todo el país.

2011

CRÉDITOS



El módulo de estudio de la asignatura Bases de Datos I del Programa Ingeniería de Sistemas es propiedad de la Corporación Universitaria Remington. Las imágenes fueron tomadas de diferentes fuentes que se relacionan en los derechos de autor y las citas en la bibliografía. El contenido del módulo está protegido por las leyes de derechos de autor que rigen al país.

Este material tiene fines educativos y no puede usarse con propósitos económicos o comerciales.

AUTOR

Nelson Jovanny Isaza Morales

Técnico en sistemas con énfasis en mantenimiento Corporación universitaria de Girardota.

Ingeniero de sistemas Corporación Universitaria Remington.

Diplomado en Pedagogía Universitaria Corporación Universitaria Remington

nelsonjim04@hotmail.com

mediaciones.integrador03@remington.edu.co

Nota: el autor certificó (de manera verbal o escrita) No haber incurrido en fraude científico, plagio o vicios de autoría; en caso contrario eximió de toda responsabilidad a la Corporación Universitaria Remington, y se declaró como el único responsable.

RESPONSABLES

Escuela de Ciencias Básicas e Ingeniería

Director Dr. Mauricio Sepúlveda

Director Pedagógico

Octavio Toro Chica

dirpedagogica.director@remington.edu.co

Coordinadora de Medios y Mediaciones

Angélica Ricaurte Avendaño

mediaciones.coordinador01@remington.edu.co

GRUPO DE APOYO

Personal de la Unidad de Medios y Mediaciones

EDICIÓN Y MONTAJE

Primera versión. Febrero de 2011.

Derechos Reservados



Esta obra es publicada bajo la licencia Creative Commons. Reconocimiento-No Comercial-Compartir Igual 2.5 Colombia.

TABLA DE CONTENIDO

1.	MAPA DE LA ASIGNATURA	7
2.	INTRODUCCIÓN Y REPASO A LAS BASES DE DATOS	9
2.1.	Tipos de Base de Datos	10
2.2.	Modelos de Bases de Datos	12
2.3.	Introducción a Sistemas de Gestión de Bases de Datos (DBMS)	16
2.4.	Motores de Bases de Datos.....	17
2.5.	Repaso modelo Relacional Aplicaciones (Normalización).....	19
3.	INSTRUCCIONES PARA LA CREACIÓN DE TABLAS Y ESTRUCTURAS DE DATOS.....	21
3.1.	Creación de tablas y estructuras.....	21
3.2.	Instrucciones de manipulación de datos.....	38
3.3.	Instrucción para la recuperación de datos.....	49
4.	SISTEMAS DE GESTIÓN DE BASES DE DATOS (DBMS) MYSQL.....	61
4.1.	Funciones en cascada (eliminación y actualización)	61
4.2.	Relaciones	65
5.	PROCEDIMIENTOS ALMACENADOS Y TRIGGERS.....	74
5.1.	Procedimientos almacenados	74
5.2.	Disparadores o Triggers.....	78
6.	PISTAS DE APRENDIZAJE	81
7.	GLOSARIO	83
8.	FUENTES.....	84
8.1.1.	Fuentes Bibliográficas	84
8.2.	Fuentes digitales o electrónicas	85

1. MAPA DE LA ASIGNATURA

BASES DE DATOS I

PROPÓSITO GENERAL DEL MÓDULO

Entregar al estudiante los conocimientos necesarios para que desarrolle y administre de forma íntegra y eficiente los sistemas de bases de datos, utilizando las diferentes herramientas propuestas en este módulo, siendo consciente de la importancia que estas tienen en la vida diaria para las personas y las diferentes organizaciones.

OBJETIVO GENERAL

Proporcionar a los estudiantes conocimientos y estrategias que permitan el desarrollo de bases de datos bajo la reglamentación de normalizaciones formales, acatando todos los parámetros en que se basa la construcción de Software de alta calidad, lo cual les permitirá a los usuarios finales una visión abstracta de los datos que garantizan la integridad de la información.

OBJETIVOS ESPECÍFICOS

- ◆ Conocer la historia de las bases de datos, los tipos de bases de datos y su aplicabilidad en el campo de la Ingeniería de Sistemas.
- ◆ Entender los conceptos y la terminología implementada en la construcción de base de datos
- ◆ Operar el sistema de funciones en cascada, reconociendo la importancia que esta tiene y el nivel de seguridad y riesgo que representa su manipulación, sistemática.
- ◆ Conocer el manejo y funcionamiento de los procedimientos almacenados, reconociendo su importancia en las bases de datos, resaltando la agilidad y la rapidez que representa para el manejo de las diferentes operaciones, tanto de consulta, como de actualización e inserción, aplicando en los diferentes ejercicios y problemas de bases de datos. los Triggers. reconociendo

Unidades

UNIDAD 1	UNIDAD 2	UNIDAD 3	UNIDAD 4
Introducción y repaso a las bases de datos Analizar la importancia de las bases de datos y los diferentes diagramas que son necesarios para el desarrollo y distribución de las tablas.	Instrucciones para la Creación de tablas y estructuras de datos Logro de un alto desempeño en el manejo de las tablas y la información que estas contienen.	Sistemas de Gestión de Bases de Datos (DBMS) – MYSQL Capacidad para comprender el funcionamiento de las bases de datos y lo que con ellas puede hacerse.	Procedimientos almacenados y triggers Habilidad para crear las diferentes consultas que pueden ser utilizadas para cada procedimiento almacenado y cada trigger según la necesidad.

2. INTRODUCCIÓN Y REPASO A LAS BASES DE DATOS

OBJETIVO GENERAL

Conocer la historia de las bases de datos, los tipos de bases de datos y su aplicabilidad en el campo de la Ingeniería de Sistemas.

OBJETIVOS ESPECÍFICOS

- ◆ Aprender la diferencia que existe entre los tipos de bases de datos, teniendo en cuenta su diferencia dependiendo del contexto, el cual varía Según la variabilidad de los datos almacenados y según el contenido.
- ◆ Conocer la diferencia que existe entre los diferentes modelos de bases de datos, permitiendo así que el estudiante reconozca sus ventajas y aprenda a utilizarlas dependiendo de la necesidad que se le presenta en el medio.
- ◆ Retomar los conocimientos básicos para que el desarrollo en general de la asignatura sea más ágil, esté enriquecido y claro para el que comienza su formación en el campo de la gestión de bases de datos.
- ◆ Diferenciar los motores de bases de datos más utilizados en la actualidad por los administradores y gestores de bases de datos, observando las diferentes ventajas que estos representan para el manejo en general de las bases de datos.
- ◆ Recordar el manejo, la aplicabilidad y la necesidad que representa la el modelo relacional para el manejo ágil y rápido de las bases de datos a la hora de crearlas y montarlas en el programa gestor de bases de datos.

Prueba Inicial

Bajo su experiencia y en sus propias palabras defina los siguientes términos.

- ◆ Base de datos
- ◆ Herencia
- ◆ Polimorfismo
- ◆ Encapsulamiento

- ◆ MYSQL
- ◆ Clave Primaria
- ◆ Clave Foránea
- ◆ Integridad referencial
- ◆ Metadatos
- ◆ Datos
- ◆ Información
- ◆ Clase
- ◆ Relación de tablas

Que es

http://www.youtube.com/watch?v=7LFvH3i_PCk

Historia

<http://www.youtube.com/watch?v=jiX5y9G8RAI>

2.1. Tipos de Base de Datos

Las bases de datos se clasifican de varias formas, esto depende del contexto que se esté manejando, ya sea por su utilidad o por la necesidad que tenga aquella persona que las necesite.

Según la variabilidad de los datos almacenados

Bases de datos estáticas

Las Bases de datos estáticas son aquellas que, únicamente, pueden ser leídas; por lo regular se utilizan para almacenar información histórica, que posteriormente será usada para tomar decisiones, basadas en dichos datos históricos guardados en estas bases de datos estáticas, tenemos, por ejemplo, la información del año fiscal o económico de una empresa que se puede tener en cuenta para tomar una decisión en días futuros, también se puede tomar como ejemplo, el recorrido histórico de un país la cual es vista por el usuario final, pero no puede ser modificada por este. Concluimos entonces, que una base de datos estática es aquella que solo nos sirve para consultar información.

◆ Bases de datos dinámicas

Las Bases de datos Dinámicas son aquellas en las cuales la información, se está modificando constantemente con procesos de actualización, borrado o inserción, sumando a esto las diferentes operaciones de consulta, aplicadas en las bases de datos estáticas. Ejemplo de estas bases de

datos son las de los bancos, los supermercados y las universidades, en donde la información está cambiando constantemente.

Según el contenido

◆ Bases de datos bibliográficas

Las bases de datos Bibliográficas, son aquellas que poseen información sobre un autor, título, fecha de publicación, edición y editorial de una determinada publicación, incluso puede ser que contengan un pequeño resumen sobre la publicación, pero nunca el texto completo, ya que esto hace parte de otro tipo de base de datos, llamado bases de datos de texto completo, básicamente están compuestas por cifras y números, ejemplo de ellos es una base de datos de una biblioteca o de un laboratorio.

◆ Bases de datos de texto completo

Las bases de datos de texto completo, son bases de datos que almacenan todo el contenido de libros, revistas o colección de videos de un tema específico entre otros; son muy utilizadas en las diferentes revistas ya que estas organizaciones guardan los contenidos de todas las revistas publicadas, así, después de muchos años se puede consultar el contenido completo de la revista de hace 3 o 10 años dependiendo de la fecha en la cual se comenzó a llenar la base de datos, este tipo de bases de datos maneja un sistema de índices llamado **Tesaurus**, el cual permite una mayor integración de los datos y agiliza la búsqueda en el sistema en gestión.

◆ Directorios

Los directorios son bases de datos con información específica de un lugar, persona o empresa entre otros, el ejemplo más claro de estos es la guía telefónica, ya sea en formato electrónico o impresa, aunque impresa sería base de datos física.

◆ Bases de datos o "bibliotecas" de información Biológica

Este tipo de bases de datos almacenan información, que es muy exclusiva del campo científico, ya sea química, medicina, Biología, entre otros, estas a su vez se reclasifican en diferentes grupos como:

- ◆ Bases de datos clínicas.
- ◆ Bases de datos del genoma humano.
- ◆ Bases de datos Bibliográficas orientadas a la rama de la investigación científica.
- ◆ Bases de datos que están dedicadas al almacenamiento de proteínas o nucleótidos.

- ◆ Bases de datos de estructura, comprende los registros de datos experimentales sobre estructuras 3D de diferentes simuladores Biotecnológicos.
- ◆ Bases de datos de rutas metabólicas entre otras.

EJERCICIO DE AUTO EVALUACIÓN TIPOS DE BASE DE DATOS

Establezca la diferencia que hay entre las bases de datos dinámicas y las bases de datos estáticas.

Hable y resalte la importancia que tienen 3 de los diferentes tipos de bases de datos vistas previamente y para que se utilizan en el medio.

Para más información se sugiere mirar el siguiente video

<http://www.youtube.com/watch?v=uLP-vgYIXnI&feature=related>

2.2. Modelos de Bases de Datos

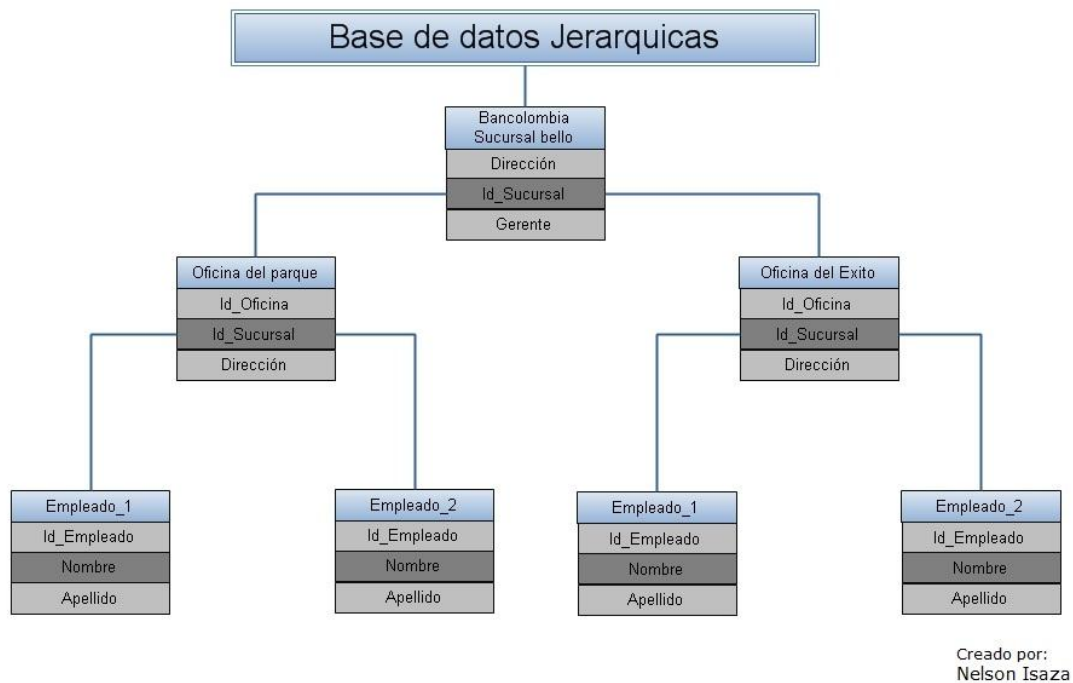
Al igual que existen varios tipos de bases de datos, también se han propuesto muchos modelos, existen unos más significativos que otros, hablaremos de los más significativos, que a su vez son los más utilizados en el medio.

◆ Base de datos Jerárquicas

En los modelos jerárquicos, los datos se organizan en estructura de árbol, en esta estructura los datos se organizan construyendo un eslabón de forma ascendente en cada registro, esto se hace con el fin de anidar la información; también se utiliza un campo de clase para guardar los registros con un orden particular en las listas de cada nivel.

Aunque la estructura de árbol o jerárquica es muy eficiente, para describir muchas relaciones, también es poco eficiente para presentar de manera eficaz la redundancia de datos.

1. En la relación Padre-hijo: El hijo sólo puede tener un padre pero un padre puede tener múltiples hijos. Los padres e hijos son atados juntos por eslabones "indicadores" llamados. Un padre tendrá una lista de indicadores de cada uno de sus hijos. (Wikipedia - http://es.wikipedia.org/wiki/Modelo_de_base_de_datos, rescatado el 18 de Julio de 2011)



◆ Base de datos de red

Las bases de datos de red, difieren de las bases de datos jerárquicas, principalmente, en el concepto de nodo, en este caso se permite que un mismo nodo tenga varios padres, lo que era prohibido en el modelo de bases de datos jerárquico.

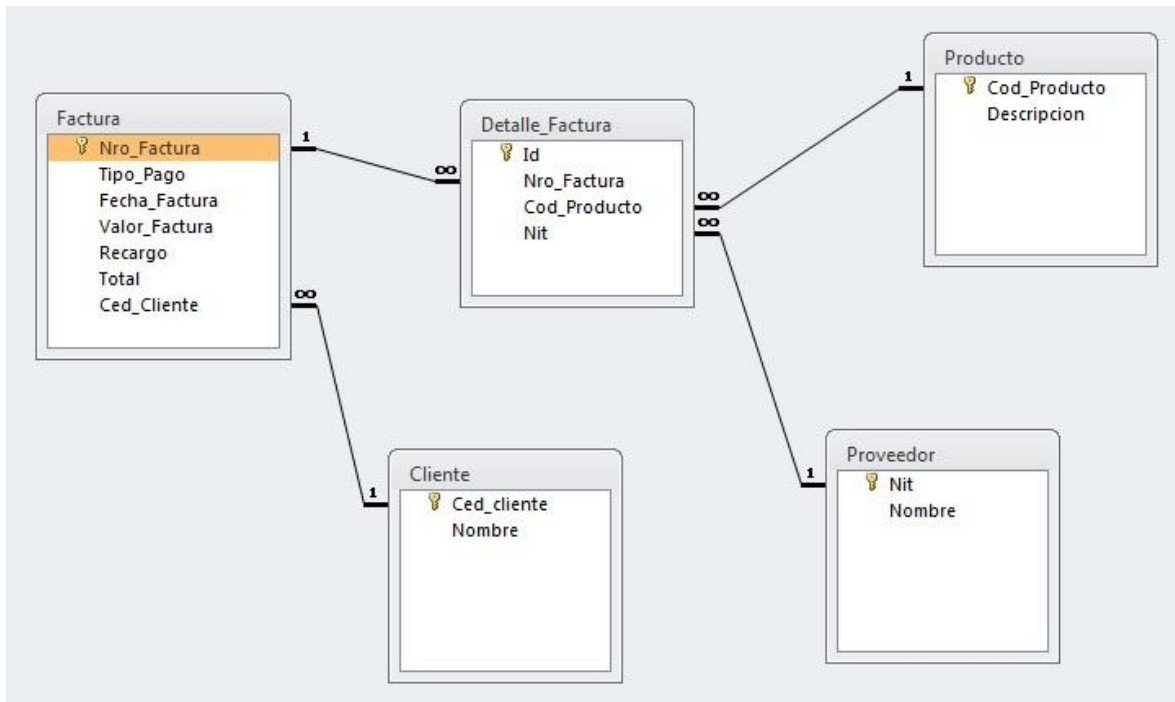
Este modelo de bases de datos mejoró mucho el manejo de información, ya que era mucho más eficiente en el manejo de redundancia. Este modelo es utilizado con mucha frecuencia por los programadores y más bien poco por los usuarios finales, dada su agilidad de en el manejo de información en Red.

◆ Base de datos Relacional

El modelo de bases de datos relacional es, actualmente, el más utilizado para modelar lo diferentes problemas actuales y llevar de forma más eficiente la administración de datos.

Este modelo fue desarrollado por primera vez en los laboratorios IBM, en los años 70, por el señor Edgar Frank Codd, en poco tiempo esta se convirtió en el nuevo paradigma de modelo de bases de datos y como su nombre lo dice está basada en relaciones.

Una de las grandes ventajas, es que en este modelo no importa el lugar y la forma en que se almacenen los datos, siendo así mucho más fácil de entender para un usuario esporádico. La información se almacena o recupera mediante consultas que hacen mucho más fácil y rápido el manejo de la información contenida en las bases de datos.



La herramienta más común en el uso de estas bases de datos relacionales es MYSQL (Structured Query Language o Lenguaje Estructurado de Consultas).

◆ Normalización de una base de datos

Durante el diseño y creación de la base de datos relacional, se pasa por un proceso de Normalización, este proceso consiste en aplicar una serie de reglas a las relaciones que se obtienen, al pasar del modelo de entidad de relación al modelo relacional.

Este proceso siempre debe hacerse para:

- ◆ Evitar la redundancia de datos
- ◆ Proteger la integridad de los datos
- ◆ Evitar problemas de actualización de los datos en tablas

Recuerda que

Para considerar una tabla a una relación, hay que tener en cuenta que:

Cada tabla debe tener un nombre único

No puede haber dos filas iguales, pues no están permitidos los duplicados

Todos los datos de una columna deben ser del mismo tipo

◆ **Base de datos Multidimensionales**

Las bases de datos multidimensionales han sido desarrolladas con el fin de obtener aplicaciones muy concretas, son muy parecidas a las bases de datos relacionales, la diferencia entre ambas se da, principalmente a nivel conceptual, una característica de estas es que los campos o tablas de este tipo de bases de datos pueden ser de dos clases o a su vez representar dimensiones de la tabla, también pueden estar representando métricas que se preparan para ser estudiadas.

◆ **Base de datos Orientadas a objetos**

El modelo de bases de datos orientadas a objetos, tiene como propósito almacenar los objetos completos (estado y comportamiento), estas bases de datos integran los conceptos más importantes del paradigma de objetos, como son herencia, polimorfismo, encapsulación.

En este modelo, los usuarios tienen la posibilidad de definir operaciones sobre los datos como parte de la definición de la misma base de datos.

EJERCICIO DE AUTOEVALUACIÓN- MODELOS DE BASES DE DATOS

- ◆ ¿Porque debe hacerse al normalización de datos?
- ◆ ¿Cuál es el Inconveniente de las bases de datos Jerárquicas?
- ◆ Mencione 3 ventajas de la base de datos relacional
- ◆ Describa con sus propias palabras la diferencia entre la base de datos orientada a objetos y multidimensional

2.3. Introducción a Sistemas de Gestión de Bases de Datos (DBMS)

El sistema de gestión de base de datos o Database management system (DBMS), es un grupo de programas que sirven para administrar, definir y construir una base de datos.

Cuando hablamos de **definir** una base de datos, estamos hablando de especificar los tipos de datos, las estructuras y las restricciones que aquellos datos que se van almacenar deben poseer para poder asegurar la integridad de la información.

De igual forma el proceso de **construcción** de una base de datos, consiste en almacenar los datos en algún medio de almacenamiento.

Por otro lado en la **administración** de la base de datos se tienen en cuenta algunos aspectos como son:

- ◆ Funciones de consulta
- ◆ Funciones de actualización
- ◆ Funciones de eliminación entre otros.

En los sistemas de gestión de base de datos debe existir un control de concurrencia que consiste en dejar ingresar varios usuarios al mismo tiempo, ya sea para trabajar en el misma base de datos o para hacer operaciones de consulta, además de esto también debe cumplir con las normas de redundancia e integridad de la información.

Algunas de las características más importantes que debe tener el sistema de gestión de base de datos es la de crear copias de seguridad y de recuperación de datos, restricción de acceso, múltiples interfaces de usuario y representación de relaciones complejas entre usuarios.

Los sistemas de gestión de base de datos se clasifican en base a los modelos de datos más habituales como lo son:

- ◆ **El modelo de base de datos relacional**
- ◆ **El modelo de base de datos orientado a objetos**
- ◆ Objeto - Relacional o relacional extendido
- ◆ Jerárquico
- ◆ Centralizado
- ◆ Distribuido

EJERCICIO DE AUTOEVALUACIÓN INTRODUCCIÓN A SISTEMAS DE GESTIÓN DE BASES DE DATOS (DBMS)

Con sus propias palabras explique en que consiste la gestión de bases de datos (DBMS)
Describa la cual es importancia de la gestión de las bases de datos.

2.4. Motores de Bases de Datos

Para el manejo y gestión de bases de datos existen varios motores que se usan dependiendo el tipo y el tamaño, además de la integridad que se exige, pues hay algunas bases de datos que requieren mayor seguridad que otras todo esto depende para que están siendo usadas.

Motores de bases de datos existen muchos, los más utilizados en el sistema de gestión de base de datos MySql son:

- ◆ MyISAM
- ◆ MERGE
- ◆ MEMORY (HEAP)
- ◆ BDB (BerkeleyDB)
- ◆ EXAMPLE
- ◆ FEDERATED
- ◆ CSV
- ◆ InnoDB
- ◆ Los motores de almacenamiento más utilizados en el medio son MyISAM e InnoDB, por esta razón se profundizará más sobre ellos.
- ◆ MyISAM: Este es el motor de almacenamiento por defecto, que se encuentra en mysql, este se basa en el código ISAM, pero tiene muchas extensiones útiles.

Cada tabla MyISAM se almacena en disco en tres ficheros. Los ficheros tienen nombres que comienzan con el nombre de tabla y tienen una extensión para indicar el tipo de fichero. Un fichero .frm almacena la definición de tabla. El fichero de datos tiene una extensión .MYD (MYData). El fichero índice tiene una extensión .MYI (MYIndex).

Para especificar explícitamente que quiere una tabla MyISAM, indíquelo con una opción ENGINE:

```
CREATE TABLE t (i INT) ENGINE = MYISAM;
```

Normalmente para hacer uso del motor MyISAM, no es necesario utilizar la instrucción “Engine=MyISAM”, ya que en el momento de crear las tablas, ellas automáticamente se están haciendo por MyISAM, en pocas palabras este es el motor por defecto (hasta la versión 5.5).

- ◆ **InnoDB:** InnoDB dota a MySQL de un motor de almacenamiento transaccional (conforme a ACID) con capacidades de commit (confirmación), rollback (cancelación) y recuperación de fallas. InnoDB realiza bloqueos a nivel de fila y también proporciona funciones de lectura consistente sin bloqueo al estilo Oracle en sentencias SELECT. Estas características incrementan el rendimiento y la capacidad de gestionar múltiples usuarios simultáneos. No se necesita un bloqueo escalado en InnoDB porque los bloqueos a nivel de fila ocupan muy poco espacio. InnoDB también soporta restricciones de clave foránea (FOREIGN KEY). En consultas SQL, aún dentro de la misma consulta, pueden incluirse libremente tablas del tipo InnoDB con tablas de otros tipos. (Tomado de MySQL 5.0 Manual <http://dev.mysql.com/doc/refman/5.0/es/innodb-overview.html> - rescatado el 19 de Julio de 2011)

Este motor ofrece mayor integridad en los datos, pues, nos permite trabajar con las funciones en cascada, ofreciendo de esta forma una mayor integridad de la información y haciendo el trabajo de las bases de datos más eficiente ya que evita los datos sin raíz en tablas relacionales, permitiendo así que la integración de datos o eliminación se haga en cascada (motor por defecto para la versión 5.5 en adelante)

EJERCICIO DE AUTOEVALUACIÓN - MOTORES DE BASES DE DATOS

- ◆ Realice un paralelo entre los dos motores de mysql más utilizados (MyISAM e InnoDB) donde se hable de las ventajas y desventajas.
- ◆ Consulte dos de los 6 motores de mysql que no han sido definidos y haga un paralelo entre ambos donde se hable de las ventajas y desventajas.

2.5. Repaso modelo Relacional Aplicaciones (Normalización)

Como se habló antes, el modelo relacional es uno de los más utilizados en la vida actual para llevar a cabo el proceso de modelado de problemas reales y la administración de datos dinámicos.

En este modelo todos los datos son almacenados en relaciones, donde cada relación es un conjunto de datos, donde no importa el orden de almacenamiento.

De manera simple, una relación representa una tabla que no es más que un conjunto de filas, cada fila es un conjunto de campos y cada campo representa un valor que interpretado describe el mundo real. Cada fila también se puede denominar tupla o registro y a cada columna también se le puede llamar campo o atributo. ((S.A). (S.F): Wikipedia -

http://es.wikipedia.org/wiki/Modelo_relacional Rescatado el día 21 de julio de 2011)

Para llevar a cabo la función o creación de bases de datos, bajo consola de mysql, es necesario crear un esquema, esto cuando la base de datos se compone de muchas tablas, este esquemas no es más que un diagrama relación.

EJERCICIO DE AUTOEVALUACIÓN – REPASO MODELO RELACIONAL

Una tienda de videojuegos desea realizar una base de datos que le permita almacenar:

- ◆ La información básica de sus clientes. La información básica de las referencias personales de cada cliente, tenga en cuenta que un mismo cliente puede tener varias referencias personales. Requiere llevar un control de sus películas con el título, género, Tipo de consola. Tenga en presente que la existencia de cada película depende del formato en que se encuentra (tipo de consola), recuerde que una película puede estar en varios formatos (para varias consolas). Los días de préstamo, el valor del préstamo, y la multa a cobrar por cada día de retraso en la entrega de una película depende también del formato en que se encuentre la película, teniendo mayor precio aquellas que están en formato para consolas más nuevas.
- ◆ Se requiere también llevar un control de los préstamos de películas con la fecha de préstamo, fecha de vencimiento, que película se prestó, en que formato y el cliente al que se le prestó.
- ◆ Por otro lado también se requiere llevar un control de las devoluciones de las películas para saber el total de multa a cobrar si hubo retraso en la entrega.

También se requiere de una tabla que indique cuando se adquirió al película por parte de la tienda para saber cuándo es tiempo de reponerla (actualizarla).

Realizar:

- ◆ Identificación de tablas referenciales.
- ◆ Modelo relacional.
- ◆ Diagrama relacional.

Para tener una mejor comprensión del tema se sugiere ver los siguientes videos:

<http://www.youtube.com/watch?v=IPKl19SbiYQ>

<http://www.youtube.com/watch?v=Ep-o3iD8ns0>

3. INSTRUCCIONES PARA LA CREACIÓN DE TABLAS Y ESTRUCTURAS DE DATOS

OBJETIVO GENERAL

Entender los conceptos y la terminología implementada en la construcción de base de datos.

OBJETIVOS ESPECÍFICOS

- ◆ Aprender la construcción de una base de datos y las tablas de las cuales se compone cada base de datos, aplicando los diferentes atributos, que permiten un manejo más eficiente de los datos.
- ◆ Conocer el funcionamiento y manejo de cada uno de los operadores y de las instrucciones necesarias para la correcta manipulación de la información contenida en las bases de datos.
- ◆ Asimilar comprendiendo y manejando las diferentes funciones e instrucciones adicionales que permiten el manejo y consulta de registros en las diferentes tablas, con más profundidad y resultados más exactos.

Prueba inicial

Haga un escrito sobre la importancia de las bases de datos en la vida diaria, en el campo empresarial y laboral.

3.1. Creación de tablas y estructuras

Tener en cuenta

Para poder trabajar con mysql, se debe tener instalado el servidor local AppServ o WampServer, existen más servidores pero estos son los más sencillos de utilizar, este servidor integra el paquete de mysql al momento de instalarlo entre otras herramientas que con el tiempo utilizaremos.

Aunque al instalar el servidor local (AppServ, WampServer, Xamp, Zend), este instala una aplicación llamada Php MyAdmin, que sirve para trabajar las bases de datos de forma gráfica, en esta materia se trabajará bajo consola de código.

Ingresando a Mysql

Por dicha razón para comenzar, después de tener instalado el servidor local, se hace clic en inicio, luego clic en ejecutar donde digitaremos cmd, para abrir la consola de mando D.O.S.

Por otro lado si estamos trabajando en win7 o vista hacemos clic en inicio y en buscar digitamos cmd, para abrir la consola de mando D.O.S

- ◆ Después de estar abierta la consola de mando D.O.S., si tenemos instalado el AppServ, simplemente digitamos: **mysql –uroot –padmin**, si es la contraseña digitada por el usuario y se presiona enter e inmediatamente ingresaremos en la consola de mando de mysql.
- ◆ Por otro lado si estamos utilizando WampServer, primero nos aseguramos que el servidor esté activado y funcionando correctamente, luego digitamos en la consola de mando D.O.S la siguiente instrucción: **cd..** y enter, luego una vez más **cd..** y enter para así quedar en la raíz del disco duro.
- ◆ **C:\>cd wamp\bin\mysql\mysql5.1.30\bin**
- ◆ Luego digitamos: **mysql –uroot y enter** después de esto, estaremos en la consola de mando de mysql.

Instrucción para la creación y visualización de la base de datos

Después de ingresar a la consola y de tener el modelo de entidad de relación desarrollado, de forma gráfica, pasamos al montaje y creación de la estructura en mysql.

Primero que todo se debe crear la base de datos, en este caso se creará la base de datos, se llamará: **banco**

Para ello se digita:

```
mysql> create database banco;
```

Para saber que bases de datos hay en el sistema, se digita:

```
mysql> show database;
```

Para poder utilizar la base de datos creada antes que nada hay que activarla o decirle al sistema en que base de datos vamos a trabajar, por esta razón se digita la siguiente instrucción.

```
mysql> use banco
```

En esta instrucción no es necesario el punto y coma (;).

Tipos de datos básicos

Los tipos de datos más utilizados en mysql son:

- ◆ Datos tipo numérico.
- ◆ Datos tipo fecha.
- ◆ Datos tipo cadena.

◆ **Datos tipo numérico**

El tipo de datos numérico, se divide en dos grupos, los que están en coma flotante o decimales y los que son números enteros en los cuales se incluyen los binarios.

TinyInt: Número entero con o sin signo, con signo va desde -128 hasta 127. Pero si no se les pone signo entonces va desde 0 hasta 255.

Bit ó Bool: Este es válido para números enteros ya sea 0 o 1, es el tipo binario.

SmallInt: Número entero con o sin signo, con signo va desde -32768 hasta 32767. Pero si no se les pone signo entonces va desde 0 hasta 65535.

MediumInt: Número entero con o sin signo, con signo va desde:

- ◆ 8388608 hasta 8388607. Pero si no se les pone signo entonces va desde 0 hasta 16777215.

Integer, Int: Número entero con o sin signo, con signo va desde:

- ◆ 2147483648 hasta 2147483647. Pero si no se les pone signo entonces va desde 0 hasta 4294967295.

BigInt: Número entero con o sin signo, con signo va desde:

- ◆ 9223372036854775808 hasta 9223372036854775807. Pero si no se les pone signo entonces va desde 0 hasta 18446744073709551615.

Float: Este es un número con coma flotante de precisión simple, sus valores validos con signo van desde: -3.402823466E+38 hasta -1.175494351E-38 o sin signo van desde 1.175494351E-38 hasta 3.402823466E+38.

xReal, Double: Este es un número con coma flotante de precisión doble, sus valores validos con signo van desde:

- ◆ 1.7976931348623157E+308 a -2.2250738585072014E-308 o sin signo van desde 2.2250738585072014E-308 a 1.7976931348623157E+308.

Decimal, Dec, Numeric: Este es un número en coma flotante desempaquetado. Y dicho número se almacena como una cadena.

Tipo de Campo	Tamaño de Almacenamiento
TINYINT	1 byte
SMALLINT	2 bytes
MEDIUMINT	3 bytes
INT	4 bytes
INTEGER	4 bytes
BIGINT	8 bytes
FLOAT(X)	4 u 8 bytes
FLOAT	4 bytes
DOUBLE	8 bytes
DOUBLE PRECISION	8 bytes
REAL	8 bytes
DECIMAL(M,D)	M+2 bytes sí D > 0, M+1 bytes sí D = 0
NUMERIC(M,D)	M+2 bytes if D > 0, M+1 bytes if D = 0

(Tomado de: <http://www.desarrolloweb.com/articulos/1054.php>)

◆ Datos tipo fecha

En mysql el momento de almacenar fechas no es algo muy estricto, por esto, se encarga de revisar que cada formato esté comprendido entre el rango que debe verificar que el mes esté entre 1 y 12 y que el día esté entre 1 y 31, la hora entre 0 y 23 entre otros.

Date: Este es el formato tipo fecha básico. Su rango de valores está comprendido entre 1 de enero de 1001, hasta el 31 de diciembre de 9999. Y su formato de almacenamiento es de año – mes -

día, pero en el momento de llevar a cabo consultas, se puede configurar para que me arroje los valores en un orden diferentes, estos órdenes varían mucho dependiendo del lugar y la necesidad del usuario, este puede ser entonces día/mes/año.

DateTime: Formato de combinación de fecha y hora y su rango de valores va desde el 1 de enero de 1001 con la hora 0, minuto 0, segundo 0 hasta el día 31 de diciembre de 9999 con las 23 horas, 59 minutos, 59 segundos. Su formato de almacenamiento es: año/mes/día hora: minuto: segundo, yy/mm/dd: hh-mm-ss.

TimeStamp: esta es una combinación de fecha y hora y su rango está comprendido desde: el 1 de enero de 1970 hasta el año 2037. Y su formato de almacenamiento depende del tamaño del campo:

Tamaño	Formato
14	AñoMesDiaHoraMinutoSegundo aaaammddhhmmss
12	AñoMesDiaHoraMinutoSegundo aammddhhmmss
8	ñoMesDia aaaammdd
6	AñoMesDia aammdd
4	AñoMes aamm
2	Año aa

(Tomado de: <http://www.desarrolloweb.com/articulos/1054.php>)

Time: Este sirve para almacenar una hora, con sus respectivos minutos y segundos y su rango de hora va desde: -838 horas, 59 minutos y 59 segundos, su formato de almacenamiento es 'HH:MM:SS'

Year: Este tipo de formato sirve para almacenar un año, y el rango de valores que maneja va desde el año 1901 hasta el año 2155. El campo puede tener tamaño 2 o tamaño 4, todo depende de la forma como queramos guardar los datos ya sea de 4 dígitos o de 2.

Tipo de Campo	Tamaño de Almacenamiento
DATE	3 bytes
DATETIME	8 bytes
TIMESTAMP	4 bytes
TIME	3 bytes
YEAR	1 byte

(Tomado de: <http://www.desarrolloweb.com/articulos/1054.php>)

◆ Datos tipo cadena

Los datos tipo cadena son aquellos de almacenamiento de texto, como el caso de el nombre de una persona, un texto de un libro entre otros. Por estas razones los datos que tienen estos campos de tipo texto, aunque contengan números, no se deben usar para hacer operaciones aritméticas, para ello, existen los de tipo numérico o en algunos casos para suma o resta de fechas podría usarse los de tipo fecha.

Char(n): Este tipo almacena una cadena caracteres de longitud fija, esta longitud se le programa o indica en el momento en el cual se está creando la tabla, esta cadena puede contener desde 0 a 255 caracteres.

VarChar(n): Este tipo almacena una cadena caracteres de longitud Variable, esta longitud no es necesario programarla, ya que esta utilizará los caracteres que necesite hasta llegar a su tope máximo, que en este caso es 255.

Los tipos de cadena vistos anteriormente (**Char** y **VarChar**) son los más comunes, estos serán los más utilizados durante el desarrollo de la materia aunque de igual forma cada tipo de dato depende del uso que vaya a tener, por eso existen otros tipos, ya que estos otros pueden admitir mayor cantidad de caracteres.

Estos otros tipos de caracteres son los BLOB (Binary Large Object) y los Text. Su diferencia radica en el momento de realizar comparaciones y ordenamientos, por ejemplo el tipo Text, se ordena sin tener en cuenta las minúsculas y las Mayúsculas, pero es tipo BLOB se ordena teniendo en cuenta éstas.

Los tipos blob son más comúnmente utilizados para almacenar datos binarios, un ejemplo de ello son los ficheros.

TinyText y TinyBlob: Se cataloga como una columna con un tamaño máximo de 255 caracteres.

Blob y Text: Este tipo está especializado en tipo texto y tiene una capacidad máxima de 65535 caracteres.

MediumBlob y MediumText: Este tipo al igual que el anterior, tiene está especializado para textos largos, y admite hasta 16.777.215 caracteres.

LongBlob y LongText: Al igual que los anteriores, este tipo se especializa en almacenamiento de caracteres que a su vez conforman extensos textos, tanto es así que puede almacenar 4.294.967.295 caracteres. Pero antes que nada hay que tener en cuenta que por protocolos de comunicación los paquetes deben tener un máximo de 16 Mb.

Enum: Este es un campo que puede tener un único valor de una lista específica, este tipo de campo acepta un máximo de 65535 valores diferentes.

Set: Este es un campo que puede contener ninguno, uno o varios valores de una lista y esta lista puede tener un máximo de 64 valores.

Tipo de campo	Tamaño de Almacenamiento
CHAR(n)	n bytes
VARCHAR(n)	n +1 bytes
TINYBLOB, TINYTEXT	Longitud+1 bytes
BLOB, TEXT	Longitud +2 bytes
MEDIUMBLOB, MEDIUMTEXT	Longitud +3 bytes
LOBLOB, LONGTEXT	Longitud +4 bytes
ENUM('value1','value2',...)	1 ó dos bytes dependiendo del número de valores
SET('value1','value2',...)	1, 2, 3, 4 ó 8 bytes, dependiendo del número de valores

(Tomado de: <http://www.desarrolloweb.com/articulos/1054.php>)

Diferencia entre los tipos **Char** y **VarChar**

Valor	CHAR(4)	Almacenamiento	VARCHAR(4)	Almacenamiento
"	"	4 bytes	"	1 byte
'ab'	'ab '	4 bytes	'ab'	3 bytes
'abcd'	'abcd'	4 bytes	'abcd'	
'abcdefgh'	'abcd'	4 bytes	'abcd'	5 bytes

(Tomado de: <http://www.desarrolloweb.com/articulos/1054.php>)

◆ Identificación de Motores MyISAM e InnoDB

mysql tiene unos 12 motores dentro de los cuales los mas comunes son estos:

MyISAM

MERGE

InnoDB

Memory

Archive

Federated

NDB

En donde MySAM e InnoDB, son los mas utilizados por los programadores, esto depende del nivel de integridad que la empresa exija y la agilidad en los datos.

A partir de la version 5.5 el motor por defecto es innodb que es creado por oracle,

Las diferencias que hay entre ambos motores es que el InnoDB, es especialmente utilizado para, para mejorar la integridad de las tablas y es el más especializado en las relaciones en cascada, haciendo u obligando al usuario a que cree las tablas en un orden y eliminarlas en el orden inverso a su creación, por el contrario el MySAM, está adaptado a trabajar en cualquier ambiente, permitiendo que el usuario cree las tablas como mejor le parezca, pero permitiendo mucha más información innecesaria principalmente en las tablas relacionales.

Creación de Tablas

La creación de tablas es una de las labores primordiales en el manejo de bases de datos, pues éstas, por lo general, se conforman de muchas tablas, sin tablas, las bases de datos, no son más que un nombre con extensión SQL, las tablas son las que almacenan los datos y donde están organizados los datos de la forma que el usuario los necesita.

Hasta este momento la creación de la base de datos ha sido una tarea sencilla.

En este momento nuestra base de datos, creada anteriormente llamada banco, es una base de datos vacía, por lo que es casi inservible, para ello vamos a crear una tabla, en la cual podamos ingresar datos.

Primero que todo para crear una tabla dentro de una base de datos, lo primero que debemos revisar es que la base de datos este en uso (este paso lo observamos previamente antes de hablar de los tipos de datos), para ello, ingresamos la siguiente instrucción:

```
mysql> use banco;
```

Con esta instrucción le estamos diciendo al sistema que queremos utilizar la base de datos **banco**.

Luego, después de haber seleccionado la base de datos que deseamos utilizar, podemos verificar si esta contiene tablas, para ello digitamos la siguiente información.

```
mysql> show tables;
```

Y muy seguramente el sistema arrojará la siguiente respuesta:

Empty set (0.00 sec)

Esto quiere decir que en la base de datos no se ha creado ninguna tabla.

Después de haber revisado la base de datos y que se ha corroborado que no tiene ninguna tabla, se procede a crear una:

Tener en cuenta

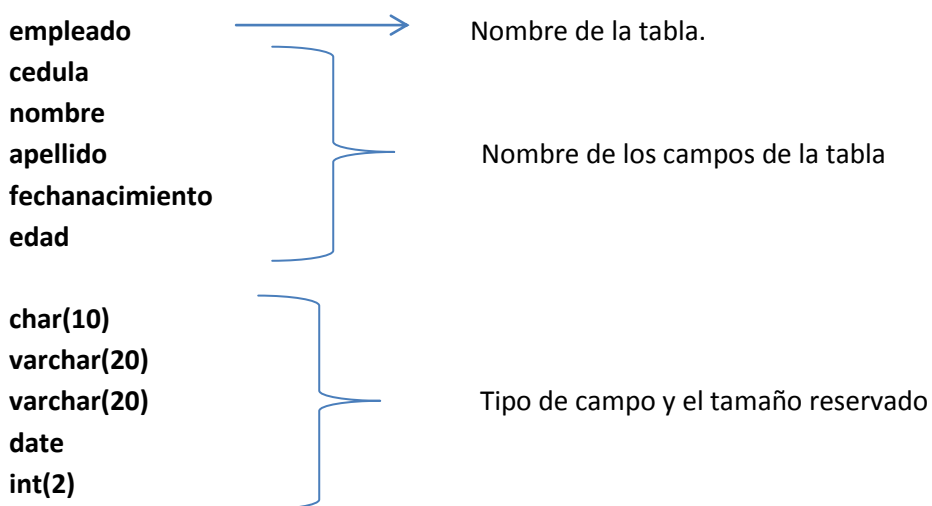
Es muy conveniente crear un modelo de entidad de relación para poder tener una clara expectativa o plan de cómo va a ser la estructura de nuestra base de datos. Especialmente cuando esta tiene muchas tablas que se relacionan y tablas que se comportan como relaciones.

Recordar que

En la creación de tablas no es correcto nombrarlas de forma plural ejemplo: Nombre de la tabla empleado o empleados, la forma correcta es empleado.

En este momento se creará una tabla para almacenar los nombres de los trabajadores del banco, para ello se debe ingresar la siguiente instrucción:

```
mysql> create table empleado (cedula char(10) not null, nombre varchar(20), apellido varchar(20), fechanacimiento date, edad int(2));
```



Después de haber creado la tabla, para visualizarla con detalles técnicos y mirar cómo fue creada, se debe digitar la siguiente instrucción:

```
mysql> describe empleado;
```

◆ Definición de Clave Primaria, Foránea e Índice

Clave Primaria (primary key PK):

La clave primaria es un campo o una combinación de dígitos de diferentes campos, que tiene como propósito identificar a una fila de una tabla, haciéndola así única

En cada tabla se debe definir una clave primaria, la cual garantizará que no se ingresarán, en este caso, dos personas con la misma cedula, por esta razón la clave primaria siempre debe ser un código o la cédula, ya que puede ser que hayan dos personas con el mismo nombre pero seguramente tienen cédulas diferentes.

Las tablas con clave primaria se crean de la siguiente forma:

```
mysql> create table empleado (cedula char(10) not null primary key, nombre varchar(20), apellido varchar(20), fechanacimiento date, edad int(2));
```

El campo que será clave primaria no debe tener datos tipo **null**, por lo tanto debe ir **not null** en la instrucción de la tabla al definir la clave primaria, ya sea en el momento de modificarla para agregarle la clave primaria o en el momento de crear la tabla.

La Clave Foránea (foreign key FK):

La clave Foránea, es una limitación referencial entre dos tablas, identifica una columna o un grupo de columnas en una tabla, que hace referencia a un grupo de columnas o una columna en otra tabla. Las columnas en la tabla referendo, deben ser la clave primaria u otra clave candidata en la tabla referenciada.

Una tabla puede tener una clave foránea, que haga referencia a un campo en una tabla relacional o tabla primaria, todo dependiendo de la necesidad.

Índice (o KEY, o INDEX):

Se considera índice a un grupo de datos que mysql asocia con una o varias columnas de la tabla, es en este grupo en el cual aparece la relación entre el contenido y el número de filas donde se encuentra ubicado.

Los índices, usualmente son utilizados para agilizar las consultas a las tablas, esto evita que se tenga que hacer el recorrido por todos los datos disponibles, agilizando con esto la búsqueda, igual que se hace en un libro.

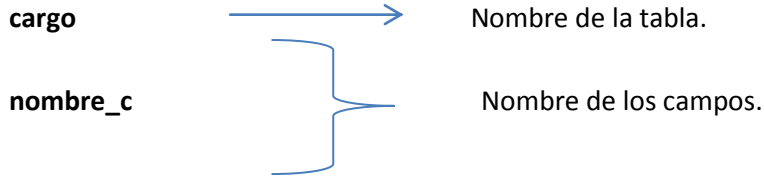
◆ **Modificación, Eliminación, Adición y Cambios en la Estructura de las tablas**

◆ **Modificación:**

Modificar una tabla en una base de datos es algo muy común, para todo programador y administrador de base de datos es algo necesario y cotidiano, ya sea eliminando, asignando claves primarias o foráneas, cambiando el nombre a algún campo o agregando o quitando atributos a los campos, cambiando su tipo de datos entre otros.

Ejemplo:

Para el ejemplo tomaremos la base de datos creada anteriormente llamada banco y agregaremos una tabla llamada cargo



Después de estar seguros que la base de datos está activa, se procede a **crear la tabla** utilizando la siguiente instrucción:

```
mysql> create table banco(nombre varchar(30));
```

Para visualizar todas las tablas se digita la siguiente instrucción:

```
mysql> show tables;
```

Para adicionar un campo al final de la tabla:

```
mysql> alter table banco add cantidad_suc int(3) not null;
```

Cambiar el nombre de la tabla:

```
mysql> alter table banco rename to cargo;
```

Recordar que

Al cambiar el nombre de una tabla, el contenido en ella no cambia. Por otro lado puede verse afectados los procedimientos almacenados, ya que si estos hacen referencia a una tabla con otro nombre entonces podrían arrojar error al no encontrarla con el nombre que originalmente fueron creados para hacer referencia.

Adicionar un campo al comienzo de la tabla:

```
mysql> alter table cargo add codigo_c varchar(10) not null first;
```

Adicionar un campo en cualquier lugar de la tabla:

Este campo se agregará en la tabla empleado.

```
mysql> alter table empleado add fecha_ingreso date not null after fechanacimiento;
```

Eliminar un campo:

```
mysql> alter table empleado drop fechanacimiento;
```

Ahora se modificará en la tabla **cargo** el campo **codigo_c** agregándole la clave primaria y agregando mayor capacidad al tipo datos (varchar (15)).

Modificar campo (tipos de datos y agregar clave primaria):

```
mysql> alter table cargo modify codigo_c varchar(15) not null primary key;
```

Eliminar una clave primaria:

```
mysql> alter table cargo drop primary key;
```

Adicionar clave primaria

```
mysql> alter table cargo modify codigo_c varchar(10) not null primary key;
```

Ahora se procederá a eliminar una tabla.

Eliminar una tabla:

```
mysql> drop table empleado;
```

Agregar registros a una table:

```
mysql> insert into cargo(codigo_c, nombre, cantidad_suc) values ('1001','contador','1');
```

```
mysql> insert into cargo(codigo_c, nombre, cantidad_suc) values ('1002','cajeros','7');
```

Mostrar registros de una tabla:

```
mysql> select * from cargo;
```

Recordar que

A los campos tipo numérico no es necesario hacerles la reserva de memoria como el caso del tipo texto.

Nombre de la tabla int(5) primary key... = nombre de la tabla int primary key ...

◆ **Instrucción unsigned**

Se han presentado algunos atributos extra, como son las claves primarias, los índices y claves foráneas. A los campos de tipo entero se le pueden agregar otros atributos en este caso se verá el uso de los campos unsigned o campos sin signo, este tipo de campo solo

permite valores positivos, muy adecuado para registros de edad, talla entre otros, pues una persona no puede tener -20 años, eso no tiene lógica.

Este tipo de campo también se puede aplicar a valores numéricos flotantes o de tipo decimal. Por lo visto anteriormente en los tipos de datos numéricos, es sabido que cuando se aplica el atributo unsigned o sin signo (valores positivos únicamente), el límite del tipo de datos se duplica.

Aunque por otro lado los valores de coma flotante, también lo aceptan pero estos no duplican su capacidad, estos la mantienen.

Para crear una tabla con atributo unsigned, esto se hace de igual forma que cuando aplicamos el atributo de clave primaria ejemplo:

Crearemos una tabla llamada libros:

Con los siguientes campos:

- ◆ Código (int, unsigned, primary key, not null)
- ◆ Título (varchar(20), not null)
- ◆ Autor (char(40))
- ◆ Editorial (varchar(15))
- ◆ Precio (float, unsigned)
- ◆ Cantidad (int, unsigned)

```
mysql> create table libro(codigo int unsigned not null primary key, titulo varchar(20) not null, autor char(40), editorial varchar(15), precio float unsigned, cantidad int unsigned);
```

Entonces quedaría así:

```
mysql> describe libro;
```

Field	Type	Null	Key	Default	Extra
codigo	int(10) unsigned	NO	PRI	NULL	
titulo	varchar(20)	NO		NULL	
autor	char(40)	YES		NULL	
editorial	varchar(15)	YES		NULL	
precio	float unsigned	YES		NULL	
cantidad	int(10) unsigned	YES		NULL	

6 rows in set (0.00 sec)

◆ Campo con auto_increment

El campo auto_increment (autoincremento), es uno de los más utilizados especialmente a la hora de facturar, para contar el numero facturas que se generan en “X” tienda, en donde nunca se encontrará una factura igual a otra, gracias al autoincremento.

El auto incremento es un atributo extra que puede tener un campo, este campo también debe ser clave primaria o estar indexado, además debe ser de tipo **int(entero)**, este generará valores únicos para cada registro que se inserta.

Para crear una tabla con atributo `auto_increment`, esto se hace de igual forma que cuando aplicamos el atributo de clave primaria ejemplo:

Crearemos una tabla llamada factura:

Con los siguientes campos:

- ◆ Codigo (int, auto_increment, primary key, not null)
- ◆ Producto (varchar(20), not null)
- ◆ Cantidad (int, not null)
- ◆ Referencia (char(10), not null)
- ◆ Valor (float, unsigned, not null)

```
mysql> create table factura(codigo int not null primary key auto_increment, product varchar(50) not null, cantidad int not null, referencia char(10) not null, valor float unsigned not null);
```

Al describirla, quedará así:

```
mysql> describe factura;
```

Field	Type	Null	Key	Default	Extra
codigo	int(11)	NO	PRI	NULL	auto_increment
product	varchar(50)	NO		NULL	
cantidad	int(11)	NO		NULL	
referencia	char(10)	NO		NULL	
valor	float unsigned	NO		NULL	

5 rows in set (0.02 sec)

De igual forma el auto increment puede iniciar desde un valor específico digamos desde 1000 ejemplo:

```
mysql> create table factura(codigo int not null primary key auto_increment, product varchar(50) not null, cantidad int not null, referencia char(10) not null, valor float unsigned not null) auto_increment = 1000;
```

En este caso los registros insertados en la tabla, se iniciaran en 1001, 1002...

◆ Instrucción Zerofill

El tributo Zerofill, usualmente es utilizado para algunas referencias de artículos, esto indica que la cuanta de registros ingresados en una tabla se puede comenzar en “0001”, en caso de no utilizar este comando, el conteo “0001” se convertirá automáticamente en “1”, en pocas palabras le quita los ceros a la izquierda.

El atributo Zerofill, rellena con ceros (0), los espacios disponibles a la izquierda.

Por ejemplo: creamos la tabla “libros” definiendo los campos **código** y **cantidad** con el tributo **Zerofill**:

```
mysql> create table libro(codigo int(4) unsigned not null primary key Zerofill, titulo varchar(20) not null, autor char(40), editorial varchar(15), precio float unsigned, cantidad smallint unsigned Zerofill);
```

Al describir la table, esta quedará así:

```
mysql> describe libro;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| codigo | int(4) unsigned zerofill | NO | PRI | NULL | auto_increment |
| titulo | varchar(20) | NO | | NULL | |
| autor | char(40) | YES | | NULL | |
| editorial | varchar(15) | YES | | NULL | |
| precio | float unsigned | YES | | NULL | |
| cantidad | smallint(5) unsigned zerofill | YES | | NULL | |
+-----+-----+-----+-----+-----+-----+
6 rows in set (0.00 sec)
```

EJERCICIO – DE AUTOEVALUACIÓN CREACIÓN DE TABLAS Y ESTRUCTURAS

Crear una base de datos llamada **empresa**

a. Crear una tabla llamada trabajadores con los campos:

- ◆ código (int)
- ◆ nombre (char)
- ◆ dirección (char)
- ◆ edad (date)

b. Crear una tabla llamada cargo con los campos:

- ◆ nombre (char)

c. Crear una tabla llamada nomina con los campos:

- ◆ s_basico (int)
- ◆ deducciones (varchar)
- ◆ s_neto(int)

Realizar:

- ◆ Agregar a la tabla cargo un campo de código del cargo.
- ◆ Renombrar la tabla trabajador por empleado.
- ◆ Adicione a la tabla empleado el campo número de hijos antes del campo edad.
- ◆ Modifique el tipo de campo de la tabla empleado de la siguiente forma:
 - ◆ Código (varchar (10))
 - ◆ Nombre (varchar (20))
- ◆ Adicione el campo cod_nomina al principio de la tabla nomina asignándole clave primaria.
- ◆ modifique todos los campos de la tabla nomina, signadoles tipo de campo **float** excepto por el de código que debe ser **varchar**
- ◆ eliminar el campo salario básico y deducciones de la tabla nómina.
- ◆ Eliminar la tabla nómina.
- ◆ Visualizar la estructura de las tablas
- ◆ Ingresar 5 registros a la tabla empleado y visualizarlos.

3.2. Instrucciones de manipulación de datos

Las instrucciones de manipulación de datos, como su nombre lo dice, permiten llevar a cabo tareas de manipulación de registros como: eliminar datos, insertar, actualizar registros, importar información entre otras.

◆ Operadores relacionales, lógicos, aritméticos y especiales

Los operadores más utilizados en mysql y manejo de datos, se encuentran los siguientes:

- ◆ Operadores Relacionales.
- ◆ Operadores Lógicos.
- ◆ Operadores Aritméticos.
- ◆ Operadores especiales.

◆ Operadores Relacionales

Los operadores relacionales lo que hacen es vincular un valor con un campo, para que así mysql pueda comparar cada registro (el campo especificado) con el valor dado.

Estos son los operadores relacionales:

=	Igual	Is null / is not null (si un valor es nulo o no).
<>	distinto	
>	Mayor que	between: “entre” si agregamos not antes de between , el valor se invierte.
<	Menor que	
>=	Mayor o igual que	In: Permite averiguar si el valor de un campo dado, está incluido en la lista de valores especificado.
<=	Menor o igual que	

◆ Operadores Lógicos

Los operadores lógicos son los siguientes:

and	Significa y
or	Significa y/o
xor	Significa o
not	Significa no (invierte el resultado)
()	parentesis

Los operadores Lógicos se usan para combinar condiciones.

Ejemplo:

Tomando la tabla de datos llamada libro, construida anterior mente y asumiendo que tiene mucha información.

Queremos mirar los libros cuyo autor sea “Borges” y/o cuya editorial sea “Planeta”:

mysql> **select** * from libro **where** autor='Borges' or editorial= 'Planeta';

Operadores Aritméticos

Select: este comando recupera los registros de una tabla, detallando los nombres de los campos, separados por comas, e indicamos que selecciones todo los campos de la tabla que nombramos.

Where: cláusula que es opcional, con ella se pueden especificar condiciones para la consulta select.

Los operadores aritméticos son los más básicos y utilizados en el mundo, estos se aprenden desde los primeros grados de estudio y son los siguientes:

+	Suma o adiciona
-	Resta o disminuye
*	Multiplica
/	Divide

Operadores especiales

Estos son operadores especiales exclusivos para el manejo de bases de datos.

Like	También es considerado un comando y se puede combinar con % para mayor precisión.
in	Es considerado un comando permite búsqueda de datos dispersos.
between	Se utiliza para buscar datos entre rangos.
having	Se utiliza para agrupar las búsquedas hechas con algunos de los anteriores como between

◆ Instrucción Insert

Este comando es utilizado para insertar información en una tabla, puede usarse para insertar un solo registro o insertar múltiples registros.

En algunos casos el comando **insert** suele combinarse con el comando **select**, esta combinación es usada para insertar registros de otras tablas.

Otras formas de trabajar con el comando insert es: **insert values** e **insert set**, estas formas se utiliza para insertar registros basados en valores explícitamente especificados.

Ejemplo:

```
mysql> insert into libro (titulo,autor,editorial,precio,cantidad) values ('el siglo de las luces','alejo carpentier','edilux','5000','5');
```

En este caso, no se nombró el campo código ya que este está como autoincremento, por dicha razón no se nombra ni se inserta, el sistema lo hace automáticamente.

En caso que el campo código no estuviera como auto incremento, este debe nombrarse e insertarse de la siguiente manera:

```
mysql> insert into libro (código,titulo,autor,editorial,precio,cantidad) values ('0001','el siglo de las luces','alejo carpentier','edilux','5000','5');
```

No olvide que

Al realizar las diferentes instrucciones se debe tener muy en cuenta las comas (,), el punto y coma al final (;), las comillas sencillas ('_ '), a la hora de insertar datos o crear alias, y no olvide poner mucho cuidado en la hora de escribir la sintaxis, para evitar errores.

```
mysql> select * from libro;
```

codigo	titulo	autor	editorial	precio	cantidad
0001	el siglo de las luce	alejo carpentier	edilux	5000	00005
0002	la iliada	homero	gredos	5000	00010
0003	la odisea	homero	gredos	8000	00003

```
3 rows in set (0.00 sec)
```

El comando **insert** también se puede combinar con la instrucción **delayed** esta combinación permite que el cliente obtenga un ok del servidor, el cual esta almacenando la información, en archivos temporales mientras la base de datos está en uso, posteriormente cuando la base de datos este libre, la información se almacenará en el lugar correspondiente.

Otro de los beneficios es que las inserciones de muchos clientes se manejan al mismo tiempo, las cuales son escritas en un mismo bloque. Esto que agiliza más el proceso de inserción.

◆ Importar información

La importación de información es una de las formas más rápidas de insertar información en una tabla.

La importación de información llena todos los campos indicados, de una fuente de datos tabulada correctamente en un archivo de texto creado en bloc de notas.

Tabla que se utilizará

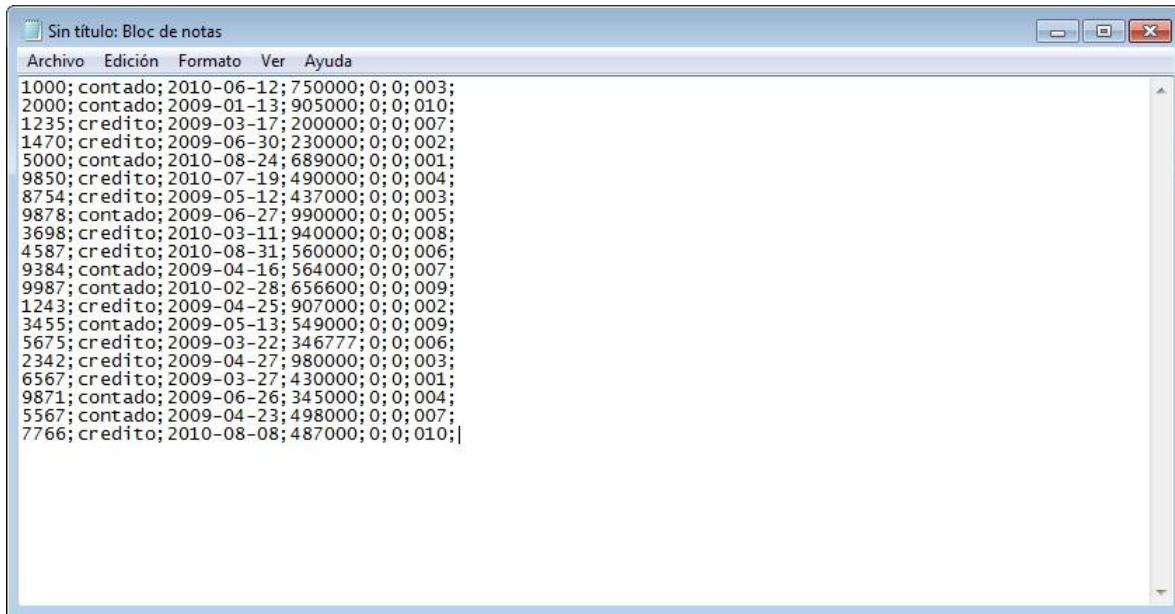
Field	Type	Null	Key	Default	Extra
nro_factura	varchar(10)	NO	PRI	NULL	
tipopago	char(7)	NO		NULL	
fechafactura	date	YES		NULL	
valorfactura	int(11)	NO		NULL	
recargo	int(11)	NO		NULL	
total	int(11)	NO		NULL	
ced_cliente	varchar(10)	NO		NULL	
7 rows in set (0.00 sec)					

La información debe estar tabulada por columnas de la siguiente manera:

```
mysql> load data local infile 'c:/bloc1.txt' into table factura fields terminated by ';';
```

'c:/bloc1.txt': Ubicación del archivo de texto que tiene la información que está disponible para exportar, el nombre del archivo en este caso es **bloc1**.

La información debe estar tabulada correctamente como se muestra en la siguiente imagen:



terminated by ';': Esta parte de la instrucción indica que se tomaran los valores que están entre el punto y coma(';'), aunque se puede utilizar otro tipo de distintivo o separador.

Nota: en caso que la tabla que va a ser llenada con la información del bloc de notas tenga una columna de auto incremento, en el bloc de notas se debe reservar esta columna dejando un espacio en blanco entre los caracteres que dividirán las columnas (;).

◆ Instrucción Select

Como se mencionó antes el comando select tiene como función recibir los registros de una o varias tabla.

Esta instrucción también puede ser utilizada para recuperar registros computados sin referencia a alguna tabla, ejemplo:

mysql> **select** 25 * 2;

```
mysql> select 25 * 2;  
+-----+  
| 25 * 2 |  
+-----+  
|      50 |  
+-----+  
1 row in set (0.00 sec)
```

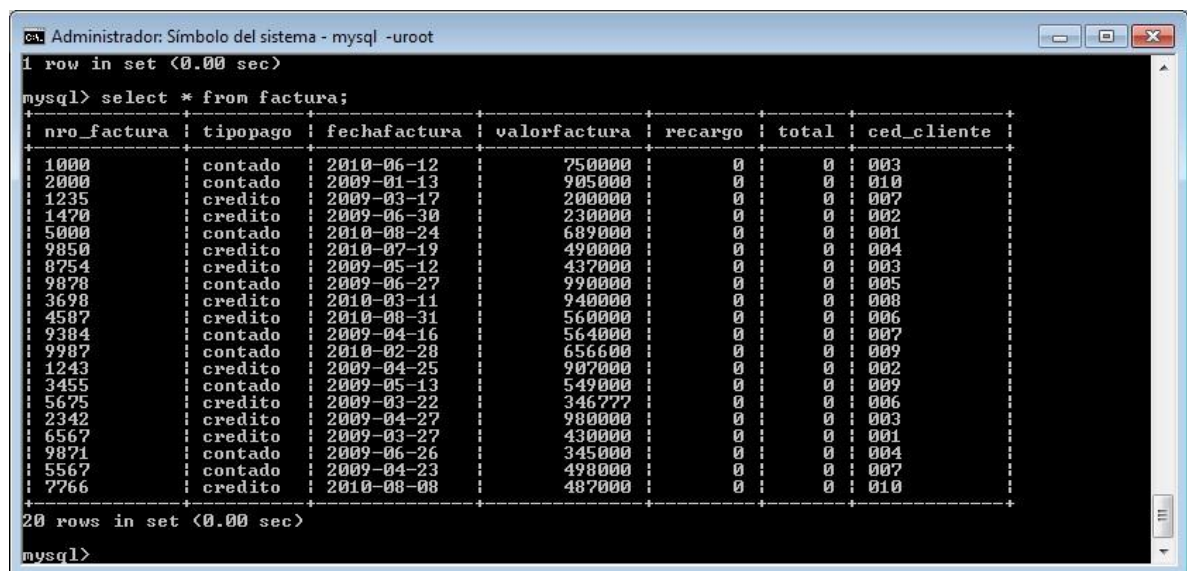
```
mysql> select (25*2)-(45+1);
+-----+
| (25*2)-(45+1) |
+-----+
| 4 |
+-----+
1 row in set (0.00 sec)
```

Pero la instrucción **select**, sirve para mucho más que hacer operaciones matemáticas, y puede ser combinada con cláusulas como **from**, esta es la más usada para consultar datos de tablas existentes, por ejemplo:

Tomando como referencia la tabla hecha creada anteriormente llamada factura:

```
mysql> select * from factura;
```

Función que sirve para consultar toda la información contenida en la tabla factura.



The screenshot shows a MySQL command window titled "Administrador: Símbolo del sistema - mysql -uroot". The output of the query "select * from factura;" is displayed in a table format. The table has 7 columns: nro_factura, tipopago, fechafactura, valorfactura, recargo, total, and ced_cliente. There are 20 rows of data. The output is as follows:

nro_factura	tipopago	fechafactura	valorfactura	recargo	total	ced_cliente
1000	contado	2010-06-12	750000	0	0	003
2000	contado	2009-01-13	905000	0	0	010
1235	credito	2009-03-17	200000	0	0	007
1470	credito	2009-06-30	230000	0	0	002
5000	contado	2010-08-24	689000	0	0	001
9850	credito	2010-07-19	490000	0	0	004
8754	credito	2009-05-12	437000	0	0	003
9878	contado	2009-06-27	990000	0	0	005
3698	credito	2010-03-11	940000	0	0	008
4587	credito	2010-08-31	560000	0	0	006
9384	contado	2009-04-16	564000	0	0	007
9987	contado	2010-02-28	656000	0	0	009
1243	credito	2009-04-25	907000	0	0	002
3455	contado	2009-05-13	549000	0	0	009
5675	credito	2009-03-22	346777	0	0	006
2342	credito	2009-04-27	980000	0	0	003
6567	credito	2009-03-27	430000	0	0	001
9871	contado	2009-06-26	345000	0	0	004
5567	contado	2009-04-23	498000	0	0	007
7766	credito	2010-08-08	487000	0	0	010

20 rows in set (0.00 sec)

También puede ser utilizada para hacer consultas más específicas, como buscar solo el valor de una columna.

```
mysql> select fechafactura from factura;
```

Instrucción que solo mostrará la columna fechafactura de la tabla factura.


```
mysql> select fechafactura from factura;
+-----+
| fechafactura |
+-----+
| 2010-06-12   |
| 2009-01-13   |
| 2009-03-17   |
| 2009-06-30   |
| 2010-08-24   |
| 2010-07-19   |
| 2009-05-12   |
| 2009-06-27   |
| 2010-03-11   |
| 2010-08-31   |
| 2009-04-16   |
| 2010-02-28   |
| 2009-04-25   |
| 2009-05-13   |
| 2009-03-22   |
| 2009-04-27   |
| 2009-03-27   |
| 2009-06-26   |
| 2009-04-23   |
| 2010-08-08   |
+-----+
20 rows in set (0.00 sec)
```

De igual forma se pueden visualizar dos o tres... las que el usuario desee.

```
mysql> select fechafactura, nro_factura from factura;
```

```
mysql> select fechafactura, nro_factura from factura;
+-----+-----+
| fechafactura | nro_factura |
+-----+-----+
| 2010-06-12   | 1000       |
| 2009-01-13   | 2000       |
| 2009-03-17   | 1235       |
| 2009-06-30   | 1470       |
| 2010-08-24   | 5000       |
| 2010-07-19   | 9850       |
| 2009-05-12   | 8754       |
| 2009-06-27   | 9878       |
| 2010-03-11   | 3698       |
| 2010-08-31   | 4587       |
| 2009-04-16   | 9384       |
| 2010-02-28   | 9987       |
| 2009-04-25   | 1243       |
| 2009-05-13   | 3455       |
| 2009-03-22   | 5675       |
| 2009-04-27   | 2342       |
| 2009-03-27   | 6567       |
| 2009-06-26   | 9871       |
| 2009-04-23   | 5567       |
| 2010-08-08   | 7766       |
+-----+-----+
20 rows in set (0.00 sec)
```

select es uno de los comandos más utilizado en el mundo de las bases de datos especialmente en mysql , más adelante se podrán observar otras combinaciones y aplicaciones un poco más complejas, muy necesarias a la hora de manejar las bases de datos.

◆ Exportar información

La exportación de la información es una forma rápida de pasar la información que tenemos en una tabla de mysql, a otro tipo de formato, desde texto (bloc de notas) hasta Excel u otro formato que necesite usuario.

Ejemplo de exportación en formato txt (bloc de notas)

```
mysql> select * into outfile 'c:/data.txt' fields terminated by '|' from factura;  
Query OK, 20 rows affected (0.02 sec)
```

```
mysql> select * from factura into outfile 'c:/data1.txt' fields terminated by '|';  
Query OK, 20 rows affected (0.00 sec)
```

Ejemplo de exportación en formato xls (Excel)

```
mysql> select * from factura into outfile 'c:/data1.xls';  
Query OK, 20 rows affected (0.00 sec)
```

◆ Instrucción Update

El comando **update** es utilizado en mysql para llevar a cabo la actualización de columnas en registros de tablas existentes con nuevos valores. Este comando se combina con la cláusula **set** la cual indica que columna modificar y sus respectivos valores, incluso en una gran consulta que implique muchas exigencias, el comando **update** puede combinarse con el comando **select**, y con cláusulas como: set, where, having, inner join entre otras.

Ejemplo:

Tomando como ejemplo la tabla que ha sido creada anteriormente llamada factura, se procederá a actualizar la columna recargo.

```
mysql> update factura set recargo='5000';
```

```
mysql> select * from factura;
```

nro_factura	tipopago	fechafactura	valorfactura	recargo	total	ced_cliente
1000	contado	2010-06-12	750000	5000	0	003
2000	contado	2009-01-13	905000	5000	0	010
1235	credito	2009-03-17	200000	5000	0	007
1470	credito	2009-06-30	230000	5000	0	002
5000	contado	2010-08-24	689000	5000	0	001
9850	credito	2010-07-19	490000	5000	0	004
8754	credito	2009-05-12	437000	5000	0	003
9878	contado	2009-06-27	990000	5000	0	005
3698	credito	2010-03-11	940000	5000	0	008
4587	credito	2010-08-31	560000	5000	0	006
9384	contado	2009-04-16	564000	5000	0	007
9987	contado	2010-02-28	656600	5000	0	009
1243	credito	2009-04-25	907000	5000	0	002
3455	contado	2009-05-13	549000	5000	0	009
5675	credito	2009-03-22	346777	5000	0	006
2342	credito	2009-04-27	980000	5000	0	003
6567	credito	2009-03-27	430000	5000	0	001
9871	contado	2009-06-26	345000	5000	0	004
5567	contado	2009-04-23	498000	5000	0	007
7766	credito	2010-08-08	487000	5000	0	010

20 rows in set (0.00 sec)

En este ejemplo se puede observar que al campo recargo se le está llevando el valor 5000, en otro caso este valor de 5000, se puede reemplazar con una operación matemática, haciendo referencia a otras columnas, un ejemplo de esto es el siguiente:

```
mysql> update factura set total=recargo+valorfactura;
```

```
mysql> update factura set total=recargo+valorfactura;
Query OK, 20 rows affected (0.00 sec)
Rows matched: 20  Changed: 20  Warnings: 0

mysql> select * from factura;
```

nro_factura	tipopago	fechafactura	valorfactura	recargo	total	ced_cliente
1000	contado	2010-06-12	750000	5000	755000	003
2000	contado	2009-01-13	905000	5000	910000	010
1235	credito	2009-03-17	200000	5000	205000	007
1470	credito	2009-06-30	230000	5000	235000	002
5000	contado	2010-08-24	689000	5000	694000	001
9850	credito	2010-07-19	490000	5000	495000	004
8754	credito	2009-05-12	437000	5000	442000	003
9878	contado	2009-06-27	990000	5000	995000	005
3698	credito	2010-03-11	940000	5000	945000	008
4587	credito	2010-08-31	560000	5000	565000	006
9384	contado	2009-04-16	564000	5000	569000	007
9987	contado	2010-02-28	656600	5000	661600	009
1243	credito	2009-04-25	907000	5000	912000	002
3455	contado	2009-05-13	549000	5000	554000	009
5675	credito	2009-03-22	346777	5000	351777	006
2342	credito	2009-04-27	980000	5000	985000	003
6567	credito	2009-03-27	430000	5000	435000	001
9871	contado	2009-06-26	345000	5000	350000	004
5567	contado	2009-04-23	498000	5000	503000	007
7766	credito	2010-08-08	487000	5000	492000	010

20 rows in set (0.00 sec)

◆ Instrucción Delete

El comando **delete**, es utilizado para llevar a cabo operaciones de borrado de registros, este comando requiere mucho cuidado y estar seguros de la función que se pretenderealizar, ya que

mysql no pregunta si está seguro de la acción, este simplemente borra lo que se le ordene y en el peor de los casos podría borrar toda la información.

El comando **delete** debe ir acompañado de la cláusula **where**, cuando el objetivo es borrar un registro específico, de no hacer uso de la cláusula **where**, **el comando borra todos los registros**, perdiéndose así la información completa de la tabla.

Ejemplo de cómo usar el comando delete:

```
mysql> delete from factura where nro_factura='1000';
```

◆ Instrucción Truncate

El comando truncate, tiene una función muy similar a la del comando delete, solo que el comando truncate es exclusivamente para vaciar una tabla, y no para borrar por registros específicos.

Su forma de uso es la siguiente:

```
mysql> truncate libro;
```

En donde libro es el nombre de la tabla.

◆ Instrucción Limit

Más que un comando, o una instrucción, **Limit** es una cláusula que se combina con el comando **select**, la cláusula **limit** tiene como función restringir el número de registros retornados por el comando **select**. La cláusula **limit**, tiene uno o dos argumentos numéricos, los cuales deben ser enteros positivos.

El primer argumento especifica el desplazamiento del primer registro a retornar el desplazamiento del primer registro es 0 (no 1).

Ejemplo:

```
mysql> select * from factura limit 2,5;
```

En esta instrucción se está diciendo al sistema que a partir del Segundo (2) registro, me muestre los cinco (5) registros siguientes.

En caso de hacerse de la siguiente forma:

mysql> select * from factura **limit 7**;

Se está diciendo al sistema que me cuente los 6 primeros registros.

◆ Backup y Restauración de información

La operación de crear backups, es una de las más importantes a la hora de trabajar con las bases de datos, ya que esta nos permite recuperar la información perdida después de un error en el manejo de estas o incluso de un daño en el disco duro, por esta razón se recomienda hacerlas en un lugar diferente al disco duro en el cual se está guardando por defecto.

Para hacer backups de una base de datos completa se debe seguir la siguiente instrucción:

1. Salir de mysql
2. C:**mysqldump -uroot -padmin --routines banco > f:\banco.sql**

- ◆ **B:** copia la estructura de la base de datos
- ◆ **uroot:** nombre de usuario del servidor local, que contiene el motor mysql.
- ◆ **padmin:** contraseña del servidor local, que contiene el motor mysql
- ◆ **routines:** procedimiento que permite guardar los procedimientos almacenado, triggers entre otros.

Banco: nombre de la base de datos.

f:\banco.sql: dirección en la cual se guardará la base de datos y su respectivo nombre.

Nota: en caso de usar wampserver, se omite la parte de **-padmin**, el resto es igual

C:\wamp\bin\mysql\mysql5.1.30\bin>**mysqldump -uroot --routines banco > k:\banco.sql**

La ruta es más larga ya que al trabajar con wampserver, se debe hacer esa ruta para abrir la consola de mysql, pero las instrucciones en general para hacer copias es igual.

Recuerde que

En el momento de hacer copias de seguridad no es necesario poner punto y coma (;) al final de la instrucción.

Al hacer copias de seguridad no es necesario usar la parte **--routines**, esta solo es necesaria a la hora de respaldar procedimientos almacenados y triggers.

Restauracion

Para llevar a cabo la operación de restauración de la base de datos, se digita la siguiente instrucción estando ubicados en la consola de mysql

```
mysql> source c:/banco.sql;
```

En este caso se asume que la base de datos **banco**, se encuentra en la raíz del disco **c**

3.3. Instrucción para la recuperación de datos

En este tema se verán algunas instrucciones (las más necesarias) para llevar a cabo la diferente recuperación de datos.

◆ Funciones (count, max, min, sum, avg, etc)

Función Count

La función **count**, es muy utilizada para llevar a cabo operaciones de conteo, esta función se utiliza con el comando **select** y a su vez puede combinarse con la función **where** ejemplo de su uso es:

```
mysql> select count(*) from factura;
```

Esta instrucción cuenta todos los registros que hay en la tabla factura

Para contar registros de una columna en especial se hace de la siguiente forma:

```
mysql> select count(valorfactura) from factura;
```

Entonces para llevar a cabo operaciones de conteo de registros con una característica en especial se combina con **where** y se hace de la siguiente forma:

```
mysql> select count(*) from factura where tipopago='credito';
```

De esta forma se pueden hacer consultas de conteo un poco más rápidas, que llevando a cabo la operación de forma manual.

Función Min y Max

Estas funciones son utilizadas para llevar a cabo operaciones de consulta orientadas especialmente a valores numéricos ya que esta permite visualizar el valor máximo que contiene

una columna o el registro más alto en el caso de la cláusula **Max**, por el contrario la cláusula **Min**, permite visualizar el valor más bajo.

Ejemplo:

Utilizando la cláusula **min**

```
mysql> select min(nro_factura) from factura;
```

Utilizando la cláusula **min** combinándola con la función **where**

```
mysql> select min(nro_factura) from factura where tipopago='contado';
```

En cuyo caso nos mostrará el número de factura más bajito que el tipo de pago sea de contado.

Utilizando la cláusula **max**

```
mysql> select max(nro_factura) from factura;
```

Utilizando la cláusula **max** combinándola con la función **where**

```
mysql> select max(nro_factura) from factura where tipopago='credito';
```

En este caso el sistema arrojará el número de factura más alto que cumple con la condición de haber pagado a crédito.

◆ Función Sum

La función **sum**, es utilizada para hacer operaciones de suma, muy parecida a la que se usa en Excel.

Esta función al igual que las que hemos visto anteriormente, puede combinarse con **where**, para hacer consultas específicas en este caso suma.

Ejemplo:

```
mysql> select sum(recargo) from factura;
```

En este caso se está ordenando al sistema que haga la función de sumar toda la columna recargo de la tabla factura.

Un ejemplo muy común puede ser sumar el valor de las facturas que han sido vendidas a crédito, en este caso sería lo que le deben al almacén.

Para ello la función **max** debe combinarse con la cláusula **where**
mysql> select **sum(total)** from factura **where** tipopago='credito';

Función Avg

La función avg, es utilizada para sacar el promedio de una columna, y esta puede combinarse con varias cláusulas entre ellas where; como hemos visto anteriormente esta última puede combinarse con todas.

```
mysql> select avg(total) from factura;
```

En este caso el sistema arrojará el promedio de la columna total.

```
mysql> select avg(total) from factura where tipopago='contado';
```

En esta última instrucción se le está ordenando al sistema que calcule el promedio de la columna total que cumple con la condición de haber pagado de contado.

Función STD

La función **std** es muy utilizada para operaciones estadísticas, ya que esta calcula la varianza de una columna específica y también puede combinarse con la cláusula **where**.

Ejemplo:

```
mysql> select std(total) from factura;
```

```
mysql > select std(total) from factura where tipopago = 'credito';
```

Función Variance

La function variance también es una de las más utilizadas en operaciones de estadística, ya que sirve para calcular la varianza de una columna y como las demás también tiene la posibilidad de ser combinada con la cláusula where.

Ejemplo:

```
mysql> select variance(total) from factura;
```

```
mysql> select variance (total) from factura where tipopago='contado';
```

◆ Operador Like y not like

El operador **Like** y **not Like** es utilizado en casi todo lo que tiene que ver con búsqueda de texto o cadenas de caracteres, este se complementa con el símbolo de porcentaje (%) este operador debe estar combinado con la cláusula **where** para poder hacerse efectiva y funcionar de manera correcta.

Valor% → Comience por ciertas letras de la cadena.
%Valor → Termine por ciertas letras de la cadena.
%Valor% → Que tenga las letras en cualquier lado.
Después de crear una tabla nueva llamada **producto** en la base de datos existente.

```
mysql> select * from producto;
```

cod_producto	nombre
200	nevera
250	estufa
350	lavadora
500	tv
550	pc
400	licuadora
600	reloj
700	telefono
750	mouse
650	teclado
800	reloj
900	lampara

```
12 rows in set (0.00 sec)
```

Procedemos a utilizar el operador **Like** y **not Like**,
mysql> select * from producto **where** nombre like '%a';

```
mysql> select * from producto where nombre like '%a';
```

cod_producto	nombre
200	nevera
250	estufa
350	lavadora
400	licuadora
900	lampara

```
5 rows in set (0.00 sec)
```

En el ejemplo anterior se ordena al sistema que muestre en pantalla los registros que en su campo nombre estén terminados en la letra 'a'

Ahora se pedirá al sistema que muestre en pantalla los registros que comiencen en 't'
mysql> select * from producto **where** nombre like 't%';

```
mysql> select * from producto where nombre like 't%';
```

cod_producto	nombre
500	tv
700	telefono
650	teclado

```
3 rows in set (0.00 sec)
```

Ahora se procederá a buscar un registro en el cual su nombre contenga la letra 'n' en cualquier parte.

```
mysql> select * from producto where nombre like '%n%';
```

```
mysql> select * from producto where nombre like '%n%';
+-----+-----+
| cod_producto | nombre |
+-----+-----+
| 200          | nevera |
| 700          | telefono |
+-----+-----+
2 rows in set (0.00 sec)
```

De igual forma se utiliza el operador **not like**, con la diferencia que este actúa de forma contraria.

```
mysql> select * from producto where nombre not like '%n%';
```

Con esta instrucción, el sistema mostrará en pantalla todos los registros que no tengan en alguna parte de su nombre la letra 'n'

```
mysql> select * from producto where nombre not like '%n%';
+-----+-----+
| cod_producto | nombre |
+-----+-----+
| 250          | estufa |
| 350          | lavadora |
| 500          | tv |
| 550          | pc |
| 400          | licuadora |
| 600          | reloj |
| 750          | mouse |
| 650          | teclado |
| 800          | reloj |
| 900          | lampara |
+-----+-----+
10 rows in set (0.00 sec)
```

◆ Instrucción between – in

Between es una instrucción usada en consulta, con el fin de buscar entre rangos. Esta puede reemplazar en algunos aspectos el utilizar el mayor que (\geq) y menor que (\leq), esta instrucción se combina con where para llevar a cabo la consulta.

```
mysql> select * from producto where cod_producto between 350 and 600;
```

Esta consulta arrojará como resultado los registros de los productos que su código, esté entre 350 y 600.

```
mysql> select * from producto where cod_producto between 350 and 600;
```

cod_producto	nombre
350	lavadora
500	tv
550	pc
400	licuadora
600	reloj

```
5 rows in set (0.02 sec)
```

In, esta instrucción sirve para llevar a cabo búsquedas con mucha más precisión, también se combina con la cláusula **where**.

In permite especificar valores y criterios dentro de la cláusula **where**, es utilizado básicamente para buscar registros que cumplan con más de un criterio, el operador **in** puede ser negado, para llevar a cabo la búsqueda de valores que por el contrario no cumplan con las especificaciones o criterios dados, para negar se utiliza **not in**.

```
mysql> select * from producto where cod_producto in (500,800);
```

```
mysql> select * from producto where cod_producto in (500,800);
```

cod_producto	nombre
500	tv
800	reloj

```
2 rows in set (0.01 sec)
```

En el ejemplo anterior, si se cambia el **in** por **not in**, el sistema mostrará en pantalla todos los productos excepto los que su código es **500** y **800**.

```
mysql> select * from producto where cod_producto in (500,800);
```

```
mysql> select * from producto where cod_producto not in (500,800);
```

cod_producto	nombre
200	nevera
250	estufa
350	lavadora
550	pc
400	licuadora
600	reloj
700	telefono
750	mouse
650	teclado
900	lampara

```
10 rows in set (0.00 sec)
```

◆ Instrucción Distinct

La instrucción **distinct**, tiene como función mostrar los registros que no están repetidos, filtrando el contenido de la tabla para sí mostrar únicamente los valores distintos.

La instrucción **distinct**, debe ir combinada con el comando **select**.

Ejemplo:

Primero se ingresaran 3 nuevos registros, con nombres que ya existan en la tabla como: nevera, lavadora, teclado.

```
mysql> insert into product (cod_producto, nombre) values (210, 'nevera');  
mysql> insert into product (cod_producto, nombre) values (310, 'lavadora');  
mysql> insert into product (cod_producto, nombre) values (410, 'teclado');
```

```
mysql> select * from producto;  
+-----+-----+  
| cod_producto | nombre |  
+-----+-----+  
| 200          | nevera |  
| 250          | estufa |  
| 350          | lavadora |  
| 500          | tv     |  
| 550          | pc     |  
| 400          | licuadora |  
| 600          | reloj |  
| 700          | telefono |  
| 750          | mouse |  
| 650          | teclado |  
| 800          | reloj |  
| 900          | lampara |  
| 210          | nevera |  
| 310          | lavadora |  
| 410          | teclado |  
+-----+-----+  
15 rows in set (0.00 sec)
```

La Instrucción **distinct**, se usa de la siguiente forma:

```
mysql> select distinct nombre from product;
```

```
mysql> select distinct nombre from producto;  
+-----+  
| nombre |  
+-----+  
| nevera |  
| estufa |  
| lavadora |  
| tv     |  
| pc     |  
| licuadora |  
| reloj |  
| telefono |  
| mouse |  
| teclado |  
| lampara |  
+-----+  
11 rows in set (0.00 sec)
```

◆ Alias para las columnas y las tablas

Los alias son muy convenientes especialmente para dar un nombre efímero a una columna que se forma después de hacer una consulta o incluso para una tabla.

Para usarlo en una tabla se usa la siguiente instrucción:

mysql> select * from producto as **mi_alias**;

```
mysql> select * from producto as mi_alias;
+-----+-----+
| cod_producto | nombre |
+-----+-----+
| 200          | nevera |
| 250          | estufa |
| 350          | lavadora |
| 500          | tv     |
| 550          | pc     |
| 400          | licuadora |
| 600          | reloj  |
| 700          | telefono |
| 750          | mouse  |
| 650          | teclado |
| 800          | reloj  |
| 900          | lampara |
| 210          | nevera |
| 310          | lavadora |
| 410          | teclado |
+-----+-----+
15 rows in set (0.00 sec)
```

Recordemos que el alias no le quita ni le cambia el nombre permanente a la tabla, esto es temporal.

Para usar el alias en una columna o una consulta, se hace de la siguiente forma:

mysql> select nombre '**mi_alias**' from producto;

En este ejemplo se está dando un nombre efímero o alias a la columna nombre de la tabla producto, esta acción a su vez es una operación de consulta.

mysql> select nombre '**mialias**' from producto where cod_producto between 200 and 600;

No olvide que

Cuando se refiere a una columna o una consulta, es necesario que el alias esté entre comillas sencillas y nunca comillas dobles.

◆ Group by

El comando **group by** este nos permite agrupar registros de que tienen ciertas características y aunque se parece a un where con condiciones, este es muy diferente, **group by** debe combinarse con el comando **select**.

Para llevar a cabo el ejemplo del uso de group by, se construirá otra tabla llamada visitantes.

```
mysql> create table visitantes(nombre varchar(30)not null, edad int not null, ciudad varchar(20)not
```

```
null);
```

```
Query OK, 0 rows affected (0.07 sec)
```

Luego se procede a insertar información, esta será importada para que sea más rápido de ingresar

```
mysql> load data local infile 'c:/datos.txt' into table visitantes fields terminated by '|';
```

```
mysql> select * from visitantes;
```

nombre	edad	ciudad
susana molina	20	medellin
andrea cardona	25	cali
elizabeth quintero	19	bogota
diana lopez	18	medellin
catalina gaviria	15	san andres
pablo yepes	30	san andres
nicolas aranzazu	28	medellin
emilio ruiz	23	bogota
alejandra botero	34	cali
sebastian ocampo	40	bogota
monica benjumea	36	medellin
edilberto monsalve	27	cali
lina moreno	28	san andres
catalina cano	24	leticia

```
14 rows in set (0.03 sec)
```

Como ejemplo se hará el ejercicio de saber la cantidad de visitantes que tenemos de cada ciudad, esto se podría hacer utilizando la cláusula **where** pero se tendría que hacer una consulta por cada ciudad, si se utiliza el **group by** entonces todo ese proceso puede hacerse con una sola consulta.

```
mysql> select ciudad, count(*) from visitantes group by ciudad;
```

```
mysql> select ciudad, count(*) from visitantes group by ciudad;
```

ciudad	count(*)
bogota	3
cali	3
leticia	1
medellin	4
san andres	3

```
5 rows in set (0.01 sec)
```

◆ Order By

La cláusula **order by** se combina con el comando select, para poder llevar a cabo el ordenamiento de una consulta, ya sea ascendente o descendente haciendo referencia a una columna específica.

En este caso se procederá a ordenar la tabla visitantes tomando como referencia la columna nombre.

```
mysql> select * from visitantes order by nombre;
```

nombre	edad	ciudad
alejandra botero	34	cali
andrea cardona	25	cali
catalina cano	24	leticia
catalina gaviria	15	san andres
diana lopez	18	medellin
edilberto monsalve	27	cali
elizabeth quintero	19	bogota
emilio ruiz	23	bogota
lina moreno	28	san andres
monica benjumea	36	medellin
nicolas aranzazu	28	medellin
pablo yepes	30	san andres
sebastian ocampo	40	bogota
susana molina	20	medellin

```
14 rows in set (0.00 sec)
```

```
mysql> select * from visitantes order by nombre;
```

Si se desea ordenar en forma ascendente se escribe la sigla **asc**, después de la columna que se está tomando como referencia.

```
mysql> select * from visitantes order by nombre asc;
```

Por el contrario si se desea ordenar de forma descendente, se debe digitar la sigla **desc**, después de la columna que se está tomando como referencia.

```
mysql> select * from visitantes order by nombre desc;
```

Una tabla puede ordenarse por varias de sus columnas, es decir que una se ordene de forma ascendente y la otra de forma descendente.

```
mysql> select * from visitante order by ciudad asc , nombre desc;
```

En este caso si encuentra varias personas que visitaron la misma ciudad, se ubicaran de forma descendente, y por el contrario las ciudades se ubicarán de forma ascendente.

```
mysql> select * from visitantes order by ciudad asc, nombre desc;
+-----+-----+-----+
| nombre      | edad | ciudad |
+-----+-----+-----+
| sebastian ocampo | 40   | bogota |
| emilio ruiz      | 23   | bogota |
| elizabeth quintero | 19   | bogota |
| edilberto monsalve | 27   | cali   |
| andrea cardona   | 25   | cali   |
| alejandra botero  | 34   | cali   |
| catalina cano     | 24   | leticia |
| susana molina     | 20   | medellin |
| nicolas aranzazu  | 28   | medellin |
| monica benjumea   | 36   | medellin |
| diana lopez       | 18   | medellin |
| pablo yepes       | 30   | san andres |
| lina moreno       | 28   | san andres |
| catalina gaviria  | 15   | san andres |
+-----+-----+-----+
14 rows in set (0.00 sec)
```

◆ Having

La cláusula **Having** permite seleccionar un grupo de registros individuales al igual que la cláusula **where**, esta función es complementaria a la función **group by** la instrucción **having**, siempre se ubica al final de la sentencia.

Su característica principal están que permite que se haga comparación con funciones sum, min, max, count y avg

Ejemplo:

Si se tiene una agrupación como la anterior, por ciudades

```
mysql> select ciudad, count(*) from visitantes group by ciudad;
```

Y queremos que muestre solo las que tienen más de 3 personas; esto no puede hacerse con una comparación tradicional, en este caso se debe hacer uso de la cláusula **having** de la siguiente forma:

```
mysql> select ciudad, count(*) from visitantes group by ciudad having count(*)>=4;
```

```
mysql> select ciudad, count(*) from visitantes group by ciudad having count(*)>=4;
+-----+-----+
| ciudad | count(*) |
+-----+-----+
| medellin | 4 |
+-----+-----+
1 row in set (0.00 sec)
```

En caso contrario si se buscan las ciudades tienen menos de 3 visitantes:

mysql> select ciudad, count(*) from visitantes **group by** ciudad **having** count(*)<3;

```
mysql> select ciudad, count(*) from visitantes group by ciudad having count(*)<3;
+-----+-----+
| ciudad | count(*) |
+-----+-----+
| leticia |         1 |
+-----+-----+
1 row in set (0.00 sec)
```

◆ Vistas

Las vistas son registros de consultas que se hacen con determinado propósito.

Las vistas comparten el mismo espacio que las tablas, por esta razón no debe haber tablas con el mismo nombre que las vistas y viceversa.

Para crear una vista se utiliza el comando **create**; para el ejemplo tendremos en cuenta la tabla visitantes.

mysql> **create view vista1 as** select ciudad, count(*) from visitantes group by ciudad having count(*) between 2 and 3;

vista1 es el nombre de la vista y de **select** en adelante esta la consulta.

Para invocar la vista, solo es necesario digitar:

mysql> select * from vista1;

Para llevar a cabo la modificación de las vistas creadas es necesario utilizar la sentencia **alter view**, pero además de modificar las Vistas, estas también pueden ser eliminadas

Este proceso se hace de la forma más sencilla:

mysql> **drop view** vista1;

Si se desea saber cuál fue la forma como está creada la vista estructuralmente, se hace una consulta de la siguiente manera:

mysql> **show create view** vista1;

Ejercicio de Autoevaluación

Revisar bloc de notas 2 haciendo clic aquí

4. SISTEMAS DE GESTIÓN DE BASES DE DATOS (DBMS) MYSQL

OBJETIVO GENERAL

Operar el sistema de funciones en cascada, reconociendo la importancia que esta tiene y el nivel de seguridad y riesgo que representa su manipulación sistemática.

OBJETIVOS ESPECÍFICOS

- ◆ Conocer el uso de los procedimientos en cascada comprendiendo las ventajas y desventajas que estos representan para el manejo de registros en las tablas de una base de datos específica.
- ◆ Aprender el funcionamiento y uso de las diferentes herramientas que se utilizan para la relación de tablas en las bases de datos, reconociendo la enorme ventaja que esto representa para el usuario y la necesidad que se presenta en la actualidad.

prueba inicial

Elabore un escrito en cual diga para usted en que consiste la gestión de bases de datos, y la importancia.

Consulte las principales instrucciones para llevar a cabo las relaciones entre tablas.

4.1. Funciones en cascada (eliminación y actualización)

Las funciones en cascada, permiten llevar a cabo tareas de actualización de varias tablas que dependen de otra, para ello en el momento de crear las tablas, es necesario especificar de forma puntual, que su función está definida en cascada, para que de tal forma se puedan actualizar los datos o eliminarlos.

Las principales funciones que se trabajan en cascada son Eliminar y Actualizar

Las funciones en cascada requieren trabajar con el motor **InnoDB**, el cual proporciona mayor integridad en los datos.

Para trabajar las funciones en cascada, algo llamado integridad referencial y también deben ser utilizadas las claves foráneas que permiten la relación en los datos.

Para ello se creará una base de datos nueva que se llamará inscripción

Se construirán dos tablas llamadas

Carrera y alumno

Bd: inscripción

Tabla: Carrera

Codcarrera: char(10) not null primary key

Nombre: char(30) not null

Tabla: alumno

Carnet: char(10) not null primary key

Nombre: char(40) not null

CodCarrera: char(10) not null **foreign key**

Estas tablas se deben crear utilizando el motor **innodb**

Las tablas que llevan integridad referencial en **cascada** y que utilizan el motor **InnoDB**, se construyen de la siguiente forma:

```
mysql> create database inscripción;
```

Después de crear la base de datos se procede a activarla.

```
mysql> use inscripción;
```

El paso a seguir es crear las tablas:

```
mysql> create table carrera(codcarrera char(10) not null primary key, nombre char(30) not null)  
engine=innodb;
```

```
mysql> create table alumno(carnet char(10) not null primary key, nombre char(40) not null,  
codcarrera char(10) not null, foreign key(codcarrera) references carrera(codcarrera) on delete  
cascade on update cascade) engine=innodb;
```

En la creación de la tabla alumno, como puede verse, la clave foránea(foreign key) es la columna codcarrera esta está haciendo referencia(references) a la columna codcarrera de la tabla carrera, y

la parte que está en rojo, indica que los registros de esta tabla se actualizarán(**update**) y borrarán(**delete**) en cascada(**on cascade**)

Se procede a ingresar la información, quedando las tablas de la siguiente forma:

```
mysql> select * from alumno;
+-----+-----+-----+
| carnet | nombre | codcarrera |
+-----+-----+-----+
| 100    | lizet  | 10         |
| 200    | pablo  | 10         |
| 300    | lina   | 20         |
| 400    | luz    | 20         |
| 500    | nicolas| 30         |
| 600    | alejandra| 30        |
| 700    | andrea | 02         |
| 800    | emilio | 02         |
+-----+-----+-----+
8 rows in set (0.00 sec)

mysql> select * from carrera;
+-----+-----+
| codcarrera | nombre |
+-----+-----+
| 02         | veterinaria |
| 10         | sistemas   |
| 20         | medicina   |
| 30         | turismo    |
+-----+-----+
4 rows in set (0.00 sec)
```

◆ Función Actualizar (Update)

Como su nombre lo dice la tarea de esta función es actualizar los registros en cascada a partir de los resultados ingresados en la tabla padre en este caso es la tabla carrera.

```
mysql> update carrera set codcarrera='03' where codcarrera='30';
```

En esta instrucción se está cambiando el código de la carrera turismo (**30**) en la tabla padre (carrera), por el valor (**03**).

```
mysql> update carrera set codcarrera='03' where codcarrera='30';
Query OK, 1 row affected (0.04 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql> select * from alumno;
+-----+-----+-----+
| carnet | nombre | codcarrera |
+-----+-----+-----+
| 100    | lizet  | 10         |
| 200    | pablo  | 10         |
| 300    | lina   | 20         |
| 400    | luz    | 20         |
| 500    | nicolas| 03         |
| 600    | alejandra| 03        |
| 700    | andrea | 02         |
| 800    | emilio | 02         |
+-----+-----+-----+
8 rows in set (0.00 sec)

mysql> select * from carrera;
+-----+-----+
| codcarrera | nombre |
+-----+-----+
| 02         | veterinaria |
| 03         | turismo    |
| 10         | sistemas   |
| 20         | medicina   |
+-----+-----+
4 rows in set (0.00 sec)
```

◆ Función Eliminar (Delete)

La función delete en cascada le permite al usuario llevar a cabo tareas de borrado de registros en las tablas hijas si en la tablas padres se elimina la información principal, que alimenta las registros de las tablas hijas.

```
mysql> select * from alumno;
+-----+-----+-----+
| carnet | nombre | codcarrera |
+-----+-----+-----+
| 100    | lizet  | 10         |
| 200    | pablo  | 10         |
| 300    | lina   | 20         |
| 400    | luz    | 20         |
| 500    | nicolas| 03         |
| 600    | alejandra| 03        |
| 700    | andrea | 02         |
| 800    | emilio | 02         |
+-----+-----+-----+
8 rows in set (0.06 sec)

mysql> select * from carrera;
+-----+-----+
| codcarrera | nombre |
+-----+-----+
| 02         | veterinaria |
| 03         | turismo    |
| 10         | sistemas   |
| 20         | medicina   |
+-----+-----+
4 rows in set (0.03 sec)
```

Retomando el ejemplo anterior, se procederá a realizar el borrado en cascada, se eliminará la carrera identificada con el código '20', en este caso es medicina, y al eliminar la carrera medicina, automáticamente serán eliminados los alumnos que cursen esta carrera.

mysql> **delete** from carrera where **codcarrera**='20';

```
mysql> select * from carrera;
+-----+-----+
| codcarrera | nombre      |
+-----+-----+
| 02         | veterinaria |
| 03         | turismo     |
| 10         | sistemas    |
+-----+-----+
3 rows in set (0.00 sec)

mysql> select * from alumno;
+-----+-----+-----+
| carnet | nombre      | codcarrera |
+-----+-----+-----+
| 100    | lizet       | 10         |
| 200    | pablo       | 10         |
| 500    | nicolas     | 03         |
| 600    | alejandra   | 03         |
| 700    | andrea      | 02         |
| 800    | emilio      | 02         |
+-----+-----+-----+
6 rows in set (0.00 sec)
```

Recuerde que

Utilizar las tablas en cascada es opcional, todo ello depende para la empresa en la cual se está trabajando, las características y especificaciones que la misma empresa requiere.

4.2. Relaciones

Primero que todo para poder llevar a cabo las relaciones que puedan ser utilizadas con foreign key (clave foránea), se debe utilizar el motor InnoDB.

La relación está definida como una asociación establecida entre campos comunes, existen tres tipos de relaciones:

De uno a uno: “relaciona un único registro de la tabla principal con uno sólo de la tabla relacionada”.

De uno a varios: “un único registro de la tabla principal se puede relacionar con varios de la tabla relacionada”.

De vario a varios: “un registro de la tabla principal se relaciona con varios de la tabla relacionada y, además, un registro de la tabla relacionada se relaciona con varios de la tabla principal”.

(Tomado de:

<http://www.adrformacion.com/curso/aplicacionesaccesxp/leccion2/RelacionesTablas.htm>)

Para la relación entre tablas, es aconsejable nombrar de la misma forma los campos que se van a relacionar.

Para llevar a cabo la relación de las tablas, estas deben crearse antes con integridad referencial utilizando claves foráneas (foreign key)

left join

Un left join se usa para hacer coincidir registros en una tabla (izquierda) con otra (derecha).

En el caso en que un valor de la primera tabla, no encuentre coincidencia alguna en la segunda tabla (derecha), el sistema genera una fila adicional con todos los campos listos a null

(Tomado de: <http://www.mysqlya.com.ar/temarios/descripcion.php?cod=58&punto=64>)

La sintaxis es la siguiente:

Tomando el ejemplo de la base de datos creada anteriormente llamada **inscripción**, y sus tablas **alumno** y **carrera** se digita lo siguiente:

```
mysql> select * from alumno left join carrera on carrera.codcarrera=alumno.codcarrera;
```

Entonces el sistema arrojará el siguiente resultado:

```
mysql> select * from alumno left join carrera on carrera.codcarrera=alumno.codcarrera;
+-----+-----+-----+-----+-----+
| carnet | nombre | codcarrera | codcarrera | nombre |
+-----+-----+-----+-----+-----+
| 100    | lizet  | 10         | 10         | sistemas |
| 200    | pablo  | 10         | 10         | sistemas |
| 500    | nicolas | 03         | 03         | turismo  |
| 600    | alejandra | 03         | 03         | turismo  |
| 700    | andrea  | 02         | 02         | veterinaria |
| 800    | emilio  | 02         | 02         | veterinaria |
+-----+-----+-----+-----+-----+
6 rows in set (0.00 sec)
```

La cláusula **where** también puede combinarse con el **left join** ello depende de la necesidad del usuario.

◆ inner Join

El **Inner join**, igual que los demás **join**, sirven para hacer relaciones en tablas a la hora de hacer consultas.

Con el **inner join**, todos los registros que no coinciden en las tablas que se van relacionar, son descartados, por esta razón el sistema simplemente arroja los registros que coinciden, entregando una consulta más exacta.

Ejemplo:

Se creará una base de datos llamada arrendamientos

Tabla:

inquilino
cedinq
nombre

```
mysql> create table inquilino(cedinq varchar(10) not null primary key, nombre char(40) not null);
```

Tabla: inmueble

codpro
dirección
valorarriendo
cedpro

```
mysql> create table inmueble(codpro varchar(10) not null, direccion char(50) not null, valorarriendo int not null, cedpro varchar(10) not null);
```

Tabla: contrato

codigo (auto_increment)
cedinq
fechacontrato
codpro

```
mysql> create table contrato(codigo int not null auto_increment primary key, ceding varchar(10) not null, fechacontrato date not null, codpro varchar(10) not null);
```

Tabla: propietario

cedprop

nombre

```
mysql> create table propietario(cedpro varchar(10) not null primary key, nombre char(40) not null);
```

Luego se procede a ingresar la información, quedando las tablas de la siguiente forma:

```
mysql> select * from inquilino;
+-----+-----+
| ceding | nombre                |
+-----+-----+
| 123444 | sandra betancur       |
| 234555 | maria fernanda vallejo |
| 345666 | liliana valencia      |
| 456777 | eliana gonzalez       |
| 567888 | isabel garcia         |
| 678999 | ricardo bermudez      |
| 789000 | andres gomez          |
| 890111 | alberto perez         |
+-----+-----+
8 rows in set (0.00 sec)

mysql> select * from propietario;
+-----+-----+
| cedprop | nombre                |
+-----+-----+
| 111123 | gloria jimenez        |
| 222234 | rodrigo mena          |
| 333345 | guillermo rojas       |
| 444456 | lina vallejo          |
| 555567 | maria lopez           |
+-----+-----+
5 rows in set (0.00 sec)

mysql> select * from inmueble;
+-----+-----+-----+-----+
| codpro | direccion              | valorararriendo | cedprop |
+-----+-----+-----+-----+
| 100    | laureles               | 750000          | 444456  |
| 150    | poblado                | 1120000         | 444456  |
| 200    | centro                 | 420000          | 111123  |
| 250    | bello                  | 350000          | 111123  |
| 300    | buenos aires           | 290000          | 222234  |
| 350    | floresta               | 425000          | 555567  |
| 400    | los colores            | 840000          | 555567  |
| 450    | laureles               | 680000          | 333345  |
| 500    | la milagrosa           | 320000          | 222234  |
| 550    | envigado               | 556000          | 555567  |
| 600    | sabaneta               | 500000          | 555567  |
| 650    | itag                   | 450000          | 444456  |
| 700    | envigado               | 1500000         | 111123  |
| 750    | poblado                | 935000          | 222234  |
+-----+-----+-----+-----+
14 rows in set (0.00 sec)

mysql> select * from contrato;
+-----+-----+-----+-----+-----+
| codigo | ceding | fechacontrato | codpro | tiempo |
+-----+-----+-----+-----+-----+
| 1      | 890111 | 1999-04-25    | 700    | 11     |
| 2      | 123444 | 2003-11-25    | 300    | 7      |
| 3      | 234555 | 2006-10-23    | 150    | 4      |
| 4      | 345666 | 1995-01-20    | 250    | 15     |
| 5      | 567888 | 2004-10-10    | 400    | 6      |
| 6      | 456777 | 2000-05-30    | 650    | 10     |
| 7      | 456777 | 2003-07-14    | 100    | 7      |
| 8      | 678999 | 2000-06-28    | 200    | 10     |
| 9      | 890111 | 1997-02-11    | 500    | 13     |
| 12     | 567888 | 2005-08-29    | 450    | 5      |
| 13     | 123444 | 2007-01-01    | 550    | 3      |
| 14     | 890111 | 2000-01-31    | 750    | 10     |
+-----+-----+-----+-----+-----+
12 rows in set (0.00 sec)
```

Luego se procederá hacer una consulta en la cual se necesita saber cuántos inquilinos llevan más de 10 años.

```
mysql> select count(inquilino.nombre) from inquilino inner join contrato on
contrato.ceding=inquilino.ceding where year(fechacontrato)<2000;
```

```
mysql> select count(inquilino.nombre) from inquilino inner join contrato
-> on contrato.ceding=inquilino.ceding where year(fechacontrato)<2000;
+-----+
| count(inquilino.nombre) |
+-----+
| 3 |
+-----+
1 row in set (0.00 sec)
```

Como puede verse, para poder realizar esta consulta, fue necesario utilizar el **inner join**, ya que este permite relacionar las tablas que se necesitaban para llevar a cabo la consulta y, además de eso, entrega la información de forma más precisa que el **left join**.

Recuerde que

Tanto el inner join como el left join, el right join entre otros join, tienen como función permitir consultas mediante tablas relacionadas, pero todos arrojan resultados dependiendo de la necesidad del usuario, en pocas palabras: todos son para casos especiales.

◆ Right Join

El comando **right join** tiene una función similar a la de **left join** con la única novedad que la búsqueda de coincidencias o de registros similares, la hace de forma contraria(inversa), en este caso busca valores coincidencia de valores desde la tabla de la derecha a la tabla de la izquierda.

Para establecer más claramente la diferencia entre **left join** y **right join**, se utilizará el mismo ejemplo que ha sido usado para **left join**:

```
mysql> select * from alumno right join carrera on carrera.codcarrera=alumno.codcarrera;
```

```
mysql> select * from alumno right join carrera on carrera.codcarrera=alumno.codcarrera;
+-----+-----+-----+-----+-----+
| carnet | nombre | codcarrera | codcarrera | nombre |
+-----+-----+-----+-----+-----+
| 700 | andrea | 02 | 02 | veterinaria |
| 800 | emilio | 02 | 02 | veterinaria |
| 500 | nicolas | 03 | 03 | turismo |
| 600 | alejandra | 03 | 03 | turismo |
| 100 | lizet | 10 | 10 | sistemas |
| 200 | pablo | 10 | 10 | sistemas |
+-----+-----+-----+-----+-----+
6 rows in set (0.06 sec)
```

Como se puede observar, la función es la misma solo que el sistema busca de manera diferente entre los datos.

◆ Natural Join

El natural join es utilizado para cuando los campos que cumplen la función de enlazar las tablas, tienen el mismo nombre.

Se utiliza de la siguiente forma:

```
mysql> select * from inquilino natural join contrato;
```

```
mysql> select * from inquilino natural join contrato;
```

cedinq	nombre	codigo	fechacontrato	codpro	tiempo
890111	alberto perez	1	1999-04-25	700	11
123444	sandra betancur	2	2003-11-25	300	7
234555	maria fernanda vallejo	3	2006-10-23	150	4
345666	liliana valencia	4	1995-01-20	250	15
567888	isabel garcia	5	2004-10-10	400	6
456777	eliana gonzalez	6	2000-05-30	650	10
456777	eliana gonzalez	7	2003-07-14	100	7
678999	ricardo bermudez	8	2000-06-28	200	10
890111	alberto perez	9	1997-02-11	500	13
567888	isabel garcia	12	2005-08-29	450	5
123444	sandra betancur	13	2007-01-01	550	3
890111	alberto perez	14	2000-01-31	750	10

12 rows in set (0.00 sec)

Como se puede observar el **natural join** hace una relación natural de las tablas, en este caso el campo de la coincidencia es **cedinq**.

◆ Subconsultas

Las **subconsultas** en general consisten en hacer una consulta dentro de otra consulta para mostrar en pantalla una cantidad de información que no puede hacerse en una sola consulta.

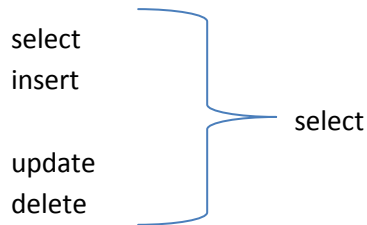
La sub consulta como tal no tiene sintaxis, esta puede tener los select que se deseen o se necesiten, en el lugar donde sean necesarias.

Se puede:

```
mysql> select (select) from tablas where (select);
```

nota: se utiliza **join** solo cuando en el **from** van 2 tablas y se establece condición entre la tabla principal y las subconsultas.

Las subconsultas no son utilizadas únicamente para visualizar información también son utilizadas para actualizar, borrar e insertar.



Para el ejemplo de las subconsultas se construirá una nueva base de datos

Base de datos: Sub_consultas

Tabla detallefactura

id
nrofactura
codigo
cantidad
valor
total

Tabla producto

codigo
articulo
valorunitario
cantidad
valorventa
existencia

Luego se inserta la información, quedando las tablas de la siguiente forma:

```
mysql> select * from detallefactura;
```

id	nrofactura	codigo	cantidad	valor	total
1	1001	10	1	0	0
2	1001	40	2	0	0
3	1001	70	1	0	0
4	2002	60	2	0	0
5	3003	20	4	0	0
6	3003	80	1	0	0
7	4004	10	2	0	0
8	4004	20	1	0	0
9	4004	30	1	0	0
10	4004	60	1	0	0
11	4004	70	1	0	0
12	4004	80	2	0	0
13	5005	10	3	0	0
14	6006	20	1	0	0
15	6006	80	2	0	0
16	6006	30	2	0	0
17	6006	90	2	0	0
18	6006	50	1	0	0
19	6006	40	2	0	0
20	7007	10	1	0	0
21	7007	90	2	0	0
22	8008	60	2	0	0
23	8008	40	2	0	0
24	8008	50	1	0	0
25	9009	50	2	0	0
26	9009	80	1	0	0
27	1101	30	1	0	0
28	2202	30	2	0	0
29	1101	60	1	0	0
30	3303	60	2	0	0
31	4404	90	3	0	0
32	5505	80	4	0	0
33	6606	70	2	0	0
34	7707	10	1	0	0
35	8808	40	2	0	0
36	9909	20	3	0	0
37	9909	30	4	0	0
38	6606	50	5	0	0
39	7707	40	1	0	0
40	1101	30	1	0	0

40 rows in set (0.00 sec)

```
mysql> select * from producto;
```

codigo	articulo	valorunitario	cantidad	valorventa	existencia
10	tv	1250000	22	0	14
20	auriculares	75000	27	0	18
30	mp3	150000	24	0	13
40	mouse	25000	33	0	24
50	taclado	130000	45	0	36
60	disco duro	203000	17	0	9
70	unidad dvd	250000	19	0	15
80	usb	80000	28	0	18
90	lapiz optico	134000	17	0	10

Ahora procederemos a hacer una operación de subconsulta, en la cual actualizaremos una de las tablas:

```
mysql> update detallefactura set valor=(select valorunitario + (valorunitario * 0.23) from producto
where detallefactura.codigo=producto.codigo);
```

En esta instrucción se actualizó el campo valor de la tabla detallefactura tomando como referencia valores de la tabla producto.

```
mysql> select * from detallefactura;
```

id	nrofactura	codigo	cantidad	valor	total
1	1001	10	1	1537500	0
2	1001	40	2	30750	0
3	1001	70	1	307500	0
4	2002	60	2	249690	0
5	3003	20	4	92250	0
6	3003	80	1	98400	0
7	4004	10	2	1537500	0
8	4004	20	1	92250	0
9	4004	30	1	184500	0
10	4004	60	1	249690	0
11	4004	70	1	307500	0
12	4004	80	2	98400	0
13	5005	10	3	1537500	0
14	6006	20	1	92250	0
15	6006	80	2	98400	0
16	6006	30	2	184500	0
17	6006	90	2	164820	0
18	6006	50	1	159900	0
19	6006	40	2	30750	0
20	7007	10	1	1537500	0
21	7007	90	2	164820	0
22	8008	60	2	249690	0
23	8008	40	2	30750	0
24	8008	50	1	159900	0
25	9009	50	2	159900	0
26	9009	80	1	98400	0
27	1101	30	1	184500	0
28	2202	30	2	184500	0
29	1101	60	1	249690	0
30	3303	60	2	249690	0
31	4404	90	3	164820	0
32	5505	80	4	98400	0
33	6606	70	2	307500	0
34	7707	10	1	1537500	0
35	8808	40	2	30750	0
36	9909	20	3	92250	0
37	9909	30	4	184500	0
38	6606	50	5	159900	0
39	7707	40	1	30750	0
40	1101	30	1	184500	0

40 rows in set (0.00 sec)

Ejercicio de autoevaluación

Revisar el bloc denotas 3 haciendo clic aquí

5. PROCEDIMIENTOS ALMACENADOS Y TRIGGERS

OBJETIVO GENERAL

Conocer el manejo y funcionamiento de los procedimientos almacenados, reconociendo su importancia en las bases de datos, resaltando la agilidad y la rapidez que representa para el manejo de las diferentes operaciones, tanto de consulta, como de actualización e inserción, aplicando en los diferentes ejercicios y problemas de bases de datos, los Triggers, reconociendo la importancia, los riesgos y la eficiencia que tienen estos, para el buen desarrollo y manejo de las bases de datos permitiendo así un desempeño más rápido en las diferentes operaciones, ya sea de: consulta, actualización, eliminación, entre otras.

OBJETIVOS ESPECÍFICOS

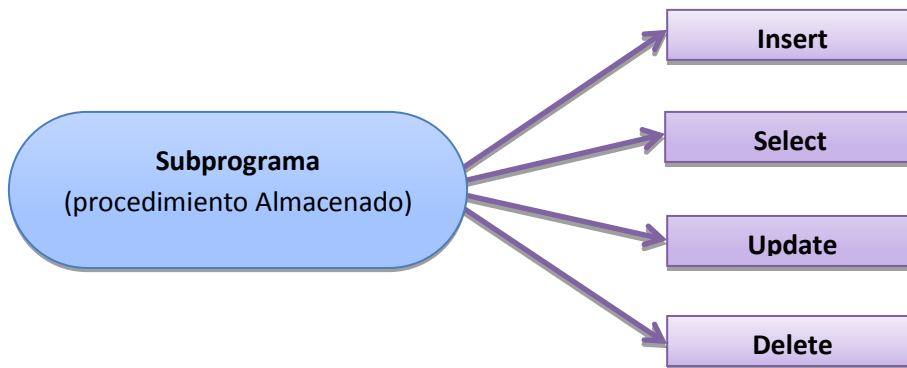
- ◆ Conocer mediante el estudio y la aplicación de los procedimientos almacenados, la importancia y ventaja que estos representan para el manejo de información en la base de datos.
- ◆ Aprender la construcción, el uso y la importancia que tienen los trigger para el manejo de los registros de una base de datos reconociendo la enorme agilidad que este representa en el momento de la manipulación de datos.

Prueba Inicial

Favor revisar el archivo de bloc de notas llamado matricula haciendo clic aquí

5.1. Procedimientos almacenados

Los procedimientos almacenados son subprogramas, que permiten hacer ciertas labores especificadas por el usuario o programador, en estos pueden ir sentencias que consten de un select, update, delete, insert, esto como ha sido nombrado anteriormente depende de la necesidad del usuario, de igual forma pueden ir subconsultas, las cuales constan de uno o varios select anidados.



Los procedimientos almacenados tienen muchas **ventajas** algunas de estas son:

- ◆ Evita estar repitiendo instrucciones, gracias a que estas quedan grabadas, por lo que el usuario simplemente tiene que invocar al procedimiento almacenado, librándose a sí de estar escribiendo una consulta o instrucción muy larga.
- ◆ El procedimiento almacenado brinda mucha seguridad gracias a que evitará que el usuario se equivoque en la constante escritura de instrucciones, las cuales podrían arrojarle un resultado erróneo.
- ◆ Son muy eficientes para usuarios con poco conocimiento de SQL.
- ◆ Permite una gran facilidad de manejo, pues simplemente hay que invocar el procedimiento almacenado.

Condiciones

Generales: no tienen parámetros.

Específicos: Tienen parámetro o parámetros.

La instrucción delimiter anula temporalmente el “;”, para que la instrucción no termine antes de tiempo.

Ejemplo:

```
mysql> delimiter //
```

```
mysql> create procedure listar_producto()
```

```
-> begin
```

```
-> select * from producto order by articulo;
```



```
-> end  
-> //  
mysql> delimiter ;
```

(Entre la palabra delimiter y el punto y coma (;), debe haber un espacio, de igual forma al principio entre la palabra delimiter y los dos signos de slash //).

Para visualizar otro ejemplo hacer clic en el siguiente enlace:

<http://www.youtube.com/watch?v=1mXLGeCV3do>

```
mysql> delimiter //  
mysql> create procedure listar_producto()  
-> begin  
-> select * from producto order by articulo;  
-> end  
-> //  
Query OK, 0 rows affected (0.05 sec)  
mysql> delimiter ;
```

```
mysql> show procedure status;
```

Esta sentencia se utiliza cuando se desea saber que procedimientos almacenados hay en la base de datos.

```
mysql> show procedure status;
```

Db	Name	Type	Definer	Modified	Created	Security_type	Comment	character_s
alquiler	alquiler_peliculas	PROCEDURE	root@localhost	2011-05-25 11:13:46	2011-05-25 11:13:46	DEFINER		latin1
alquiler	clientes	PROCEDURE	root@localhost	2011-05-25 11:08:02	2011-05-25 11:08:02	DEFINER		latin1
alquiler	consulta_cliente	PROCEDURE	root@localhost	2011-05-25 11:44:32	2011-05-25 11:44:32	DEFINER		latin1
alquiler	consulta_pelicula	PROCEDURE	root@localhost	2011-05-25 14:56:29	2011-05-25 14:56:29	DEFINER		latin1
alquiler	fecha_devolucion	PROCEDURE	root@localhost	2011-05-24 13:52:21	2011-05-24 13:52:21	DEFINER		latin1
alquiler	ingreso_alquiler	PROCEDURE	root@localhost	2011-05-24 07:53:40	2011-05-24 07:53:40	DEFINER		latin1
alquiler	valor_pagado	PROCEDURE	root@localhost	2011-05-25 10:49:50	2011-05-25 10:49:50	DEFINER		latin1
apertura	insertar	PROCEDURE	root@localhost	2011-04-26 15:10:42	2011-04-26 15:10:42	DEFINER		utf8
parcialii	listar_editorial	PROCEDURE	root@localhost	2011-04-28 16:55:42	2011-04-28 16:55:42	DEFINER		utf8
parcialii	listar_pais	PROCEDURE	root@localhost	2011-04-28 16:55:42	2011-04-28 16:55:42	DEFINER		utf8
sub_consultas	listar_producto	PROCEDURE	root@localhost	2011-08-08 10:27:20	2011-08-08 10:27:20	DEFINER		latin1

```
11 rows in set (0.03 sec)
```

Visualización de estructura de un procedimiento almacenado

```
mysql> show create procedure listar_producto;
```

Esta instrucción se utiliza con el fin de saber cómo fue creado el procedimiento almacenado, con esta instrucción podremos visualizar la instrucción de consulta que tiene almacenado el procedimiento.

```
mysql> show create procedure listar_producto;
+-----+-----+-----+-----+
| Procedure | sql_mode | Create Procedure | Database Collation |
+-----+-----+-----+-----+
| listar_producto | | CREATE DEFINER='root'@'localhost' PROCEDURE 'listar_producto'()
begin
select * from producto order by articulo;
end | latin1 | latin1_swedish_ci | utf8_general_ci |
+-----+-----+-----+-----+
1 row in set (0.03 sec)
```

Invocar el procedimiento almacenado que se ha creado:

```
mysql> call listar_producto;
```

De esta forma se llama un procedimiento almacenado que ha sido diseñado para mostrar información.

```
mysql> call listar_producto;
+-----+-----+-----+-----+-----+-----+
| codigo | articulo | valorunitario | cantidad | valorventa | existencia |
+-----+-----+-----+-----+-----+-----+
| 20 | auriculares | 75000 | 27 | 0 | 18 |
| 60 | disco duro | 203000 | 17 | 0 | 9 |
| 90 | lapiz optico | 134000 | 17 | 0 | 10 |
| 40 | mouse | 25000 | 33 | 0 | 24 |
| 30 | mp3 | 150000 | 24 | 0 | 13 |
| 50 | taclado | 130000 | 45 | 0 | 36 |
| 10 | tv | 1250000 | 22 | 0 | 14 |
| 70 | unidad dvd | 250000 | 19 | 0 | 15 |
| 80 | usb | 80000 | 28 | 0 | 18 |
+-----+-----+-----+-----+-----+-----+
9 rows in set (0.07 sec)

Query OK, 0 rows affected (0.11 sec)
```

Eliminar un procedimiento almacenado:

```
mysql> drop procedure listar_producto;
```

Esta es la instrucción utilizada para llevar a cabo la eliminación de un procedimiento almacenado.

5.2. Disparadores o Triggers

Son procesos o sub programas automáticos, esto quiere decir que ese proceso va ligado a una tabla o una acción, estos se crean de forma parecida a los procedimientos almacenados.

Su propósito es actualizar, eliminar o insertar datos, que permitan actualizar varios registros de forma automática, por ello son llamados también disparadores.

Particularidad:

A diferencia de los procedimientos almacenados y los demás procesos vistos anteriormente, el **trigger** no permite el **select**, ya que el **trigger** es un proceso automático, por esta razón solo admite o tiene el propósito de **insertar, actualizar y eliminar**

El **trigger** no recibe parámetros

El **trigger** es utilizado para actualizar cierta información de otras tablas, como gracias a la acción de insertar información en alguna tabla principal o también en el momento de actualizar algún tipo de información en una tabla o un registro en particular, el **trigger** se aplica actualizando todos los registros que dependen de esta nueva información ingresada.

Los triggers se construyen de la siguiente forma:

```
mysql> delimiter //  
mysql> create trigger actualizar after insert on detallefactura  
-> for each row  
-> begin  
-> update producto set existencia = (select sum(cantidad) from detallefactura where  
detallefactura.codigo=producto.codigo);  
-> end  
-> //  
mysql> delimiter ;
```

```
mysql> delimiter //  
mysql> create trigger actualizar after insert on detallefactura  
-> for each row  
-> begin  
-> update producto set existencia=(select sum(cantidad) from detallefactura where detallefactura.codigo=producto.codigo);  
-> end  
-> //  
Query OK, 0 rows affected (0.04 sec)  
mysql> delimiter ;
```

(Entre la palabra delimitar y el punto y coma (;), debe haber un espacio, de igual forma al principio entre la palabra delimitar y los dos signos slash //).

Para visualizar otro ejemplo, haga clic en el siguiente enlace
<http://www.youtube.com/watch?v=PucZENTyYLU>

Recuerde que

Por cada tabla solo pueden escribirse 3 triggers.

Ara **visualizar todos los triggers** que han sido utilizados en la base de datos que esta activada, ademas de una **información detallada** de cada trigger, se digita la siguiente instrucción:

```
mysql> select * from information_schema.triggers;
```

Este comando nos arroja información de cada **trigger** que compone la base de datos de una forma poco ordenada.

```
mysql> select * from information_schema.triggers\G;
```

```
mysql> select * from information_schema.triggers\G;
***** 1. row *****
      TRIGGER_CATALOG: NULL
      TRIGGER_SCHEMA: alquiler
      TRIGGER_NAME: actualiza_usuario
      EVENT_MANIPULATION: INSERT
      EVENT_OBJECT_CATALOG: NULL
      EVENT_OBJECT_SCHEMA: alquiler
      EVENT_OBJECT_TABLE: alquiler
      ACTION_ORDER: 0
      ACTION_CONDITION: NULL
      ACTION_STATEMENT: begin
update usuario set alquileres=(select count(*) from alquiler where usuario.cedula=alquiler.cedula);
update usuario set tipo=case when alquileres>15 then 'Preferencial' else 'Tradicional' end;
end
      ACTION_ORIENTATION: ROW
      ACTION_TIMING: AFTER
      ACTION_REFERENCE_OLD_TABLE: NULL
      ACTION_REFERENCE_NEW_TABLE: NULL
      ACTION_REFERENCE_OLD_ROW: OLD
      ACTION_REFERENCE_NEW_ROW: NEW
      CREATED: NULL
      SQL_MODE:
      DEFINER: root@localhost
      CHARACTER_SET_CLIENT: latin1
      COLLATION_CONNECTION: latin1_swedish_ci
      DATABASE_COLLATION: utf8_general_ci
***** 2. row *****
      TRIGGER_CATALOG: NULL
      TRIGGER_SCHEMA: matriculas
      TRIGGER_NAME: nuevo
      EVENT_MANIPULATION: INSERT
      EVENT_OBJECT_CATALOG: NULL
      EVENT_OBJECT_SCHEMA: matriculas
      EVENT_OBJECT_TABLE: estudiante
      ACTION_ORDER: 0
      ACTION_CONDITION: NULL
      ACTION_STATEMENT: begin
insert into matricula <id> values('');
end
      ACTION_ORIENTATION: ROW
      ACTION_TIMING: AFTER
      ACTION_REFERENCE_OLD_TABLE: NULL
      ACTION_REFERENCE_NEW_TABLE: NULL
      ACTION_REFERENCE_OLD_ROW: OLD
      ACTION_REFERENCE_NEW_ROW: NEW
      CREATED: NULL
      SQL_MODE: NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION
      DEFINER: root@localhost
      CHARACTER_SET_CLIENT: utf8
      COLLATION_CONNECTION: utf8_general_ci
      DATABASE_COLLATION: utf8_general_ci
2 rows in set (0.32 sec)
```

Si adicionamos el símbolo **backslash** y la letra **“G” mayúscula** a la instrucción anterior, se podrá observar la información de cada **trigger** en la base de datos con mucho detalle y de forma más ordenada.

Para llevar a cabo el uso del trigger se aplica la siguiente instrucción:

```
mysql> insert into valores('1010','10','3');
```

Para eliminar un trigger, es necesario utilizar la siguiente instrucción:

```
mysql> drop trigger actualizar;
```

Ejercicio de autoevaluación para los dos temas

Revisar los archivos de texto llamados “modelo” y “planteamiento”

6. PISTAS DE APRENDIZAJE

Recuerda que

Para considerar una tabla como relación, hay que tener en cuenta que:

- ◆ Cada tabla debe tener un nombre único
- ◆ No pueden haber dos filas iguales, pues no están permitidos los duplicados
- ◆ Todos los datos de una columna deben ser del mismo tipo

Tener en cuenta

Para poder trabajar con mysql, se debe tener instalado el servidor local **AppServ** o **WampServer**, existen más servidores pero estos son los más sencillos de utilizar, este servidor integra el paquete de mysql al momento de instalarlo entre otras herramientas que con el tiempo utilizaremos.

Tener en cuenta

- ◆ Es muy conveniente crear un modelo de entidad de relación para poder tener una clara expectativa o plan de cómo va a ser la estructura de nuestra base de datos. Especialmente cuando esta tiene muchas tablas que se relacionan y tablas que se comportan como relaciones.

Recordar que

En la creación de tablas no es correcto nombrarlas de forma plural ejemplo: Nombre de la tabla empleado o empleados, la forma correcta es empleado.

Recordar que

Al cambiar el nombre de una tabla, el contenido en ella no cambia. Por otro lado puede verse afectados los procedimientos almacenados, ya que si estos hacen referencia a una tabla con otro nombre entonces podrían arrojar error al no encontrarla con el nombre que originalmente fueron creados para hacer referencia.

Recordar que

A los campos tipo numérico no es necesario hacerles la reserva de memoria como el caso del tipo texto.

Nombre de la tabla int(5) primary key... = nombre de la tabla int primary key

No olvide que

Al realizar las diferentes instrucciones se debe tener muy en cuenta las comas (,), el punto y coma al final (;), las **comillas sencillas** ('_ '), a la hora de insertar datos o crear alias, y no olvide poner mucho cuidado en la hora de escribir la sintaxis, para evitar errores.

Recuerde que

En el momento de hacer copias de seguridad no es necesario poner punto y coma (;) al final de la instrucción.

No olvide que

Cuando se refiere a una columna o una consulta, es necesario que el alias esté entre comillas sencillas y nunca comillas dobles.

Recuerde que

Tanto el inner join como el left join, el right join entre otros join, tienen como función permitir consultas mediante tablas relacionadas, pero todos arrojan resultados dependiendo de la necesidad del usuario, en pocas palabras: todos son para casos especiales.

Recuerde que

Por cada tabla solo pueden escribirse 3 triggers.

No olvide que

Siempre portar una memoria usb en la cual tenga todos sus ejercicios

Recuerde que

Para uso posterior de algunas sentencias, es bueno copiar las instrucciones debajo de cada pregunta en los diferentes ejercicios.

Recuerde

Mantener copias de seguridad en la nube (internet) o en memorias usb, si es posible en ambos.

No olvide

Poner una pequeña portada o sus datos personales en el correo en el cual entrega su trabajo de base de datos, en el cual debe ir las preguntas y como se resolvieron, y además de ello el archivo de base de datos.

7. GLOSARIO

Base de datos: es un conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso. En este sentido, una biblioteca puede considerarse una base de datos compuesta en su mayoría por documentos y textos impresos en papel e indexados para su consulta. (Tomado de: Wikipedia - http://es.wikipedia.org/wiki/Base_de_datos)

Trigger: procedimiento que se ejecuta cuando se cumple una condición establecida al realizar una operación. (Tomado de: Wikipedia - http://es.wikipedia.org/wiki/Trigger_%28base_de_datos%29)

Procedimiento Almacenado: es un programa (o procedimiento) el cual es almacenado físicamente en una base de datos. (Tomado de: Wikipedia - http://es.wikipedia.org/wiki/Procedimiento_almacenado)

Mysql: es un sistema de gestión de bases de datos relacional, multihilo y multiusuario con más de seis millones de instalaciones. Se ofrece bajo la GNU GPL para cualquier uso compatible con esta licencia, pero para aquellas empresas que quieran incorporarlo en productos privativos deben comprar a la empresa una licencia específica que les permita este uso. (Tomado de: Wikipedia - <http://es.wikipedia.org/wiki/MySQL>)

SQL: 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 éstas. Una de sus características es el manejo del álgebra y el cálculo relacional permitiendo efectuar consultas con el fin de recuperar -de una forma sencilla- información de interés de una base de datos, así como también hacer cambios sobre ella. (Tomado de: Wikipedia - <http://es.wikipedia.org/wiki/SQL>)

POO: (Programación Orientada a Objetos) es un paradigma de programación que usa objetos y sus interacciones, para diseñar aplicaciones y programas informáticos. Está basado en varias técnicas, incluyendo herencia, abstracción, polimorfismo y encapsulamiento. (Tomado de: Wikipedia - http://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos)

8. FUENTES

8.1.1. Fuentes Bibliográficas

J. Date. (S.F): Introducción a los Sistemas de Bases de Datos.

Henry Korth. (S.F): Fundamentos de las Bases de Datos.

James Martín. (S.F): Organización de las Bases de Datos.

David M. Kroenke. (S.F): Procesamiento de las Bases de Datos.

Adoración de Miguel, Mario Piattini. (S.F): Fundamentos y Modelo de Bases de Datos.

Gary W. Hansen, James V. Hansen. (S.F): Diseño y Administración de Bases de Datos.

Andrés Bejarano, Piedad Cabanzo Dueñas (AUC). (S.F): Diseño de Bases de Datos Relacionales Avanzadas.

James R. Groff y Paul N. Weinberg. (S.F): Guía de SQL.

Groff/Weinberg. (S.F): Aplique SQL.

Amy Hillier (S.F): Arcgis 9.3

Referencia Oficial Mysql 5.1 y 6.0Beta

8.2. Fuentes digitales o electrónicas

Miguel Ángel Álvarez. (S.F): “Manual de Oracle” -
<http://www.desarrolloweb.com/manuales/tutorial-oracle.html>

Joaquin Gracia Murugarren (S.F): Mysql Manual - <http://www.webestilo.com/mysql/intro.phtml>

Oracle Corporation (2011): Manual de Mysql -
<http://dev.mysql.com/doc/refman/5.0/es/index.html>

Oracle Corp. (S.F): Manual de Mysql - <http://downloads.mysql.com/docs/refman-5.0-es.a4.pdf>

Wikipedia (S.F): Base de datos - http://es.wikipedia.org/wiki/Base_de_datos

Alegsa (S.F): <http://www.alegsa.com.ar/Dic/sghd.php>

(S.A), (S.F): MySQL 5.0 Manual <http://dev.mysql.com/doc/refman/5.0/es/innodb-overview.html> -
rescatado el 19 de Julio de 201.

Carlos Luis Cuenca. (febrero de 2003): “Tipos de datos de Mysql” -
<http://www.desarrolloweb.com/articulos/1054.php>

(S.A). (S.F): MySQL Ya desde CERO. - <http://www.mysqlya.com.ar/>