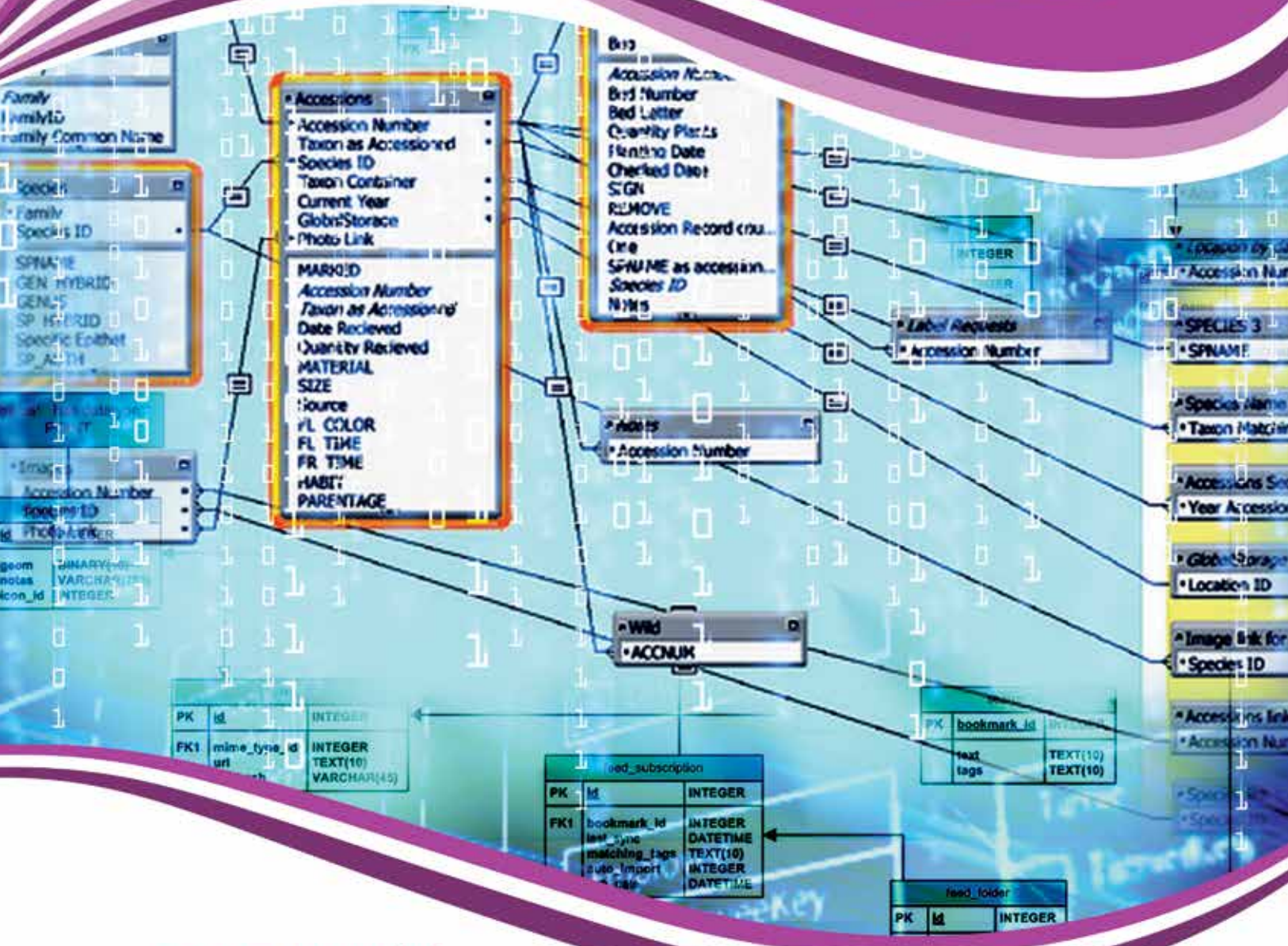
Objetos de Base de Datos



Universidad Tecnológica de Bolívar

CARTAGENA DE INDIAS

OBJETOS DE BASE DE DATOS
TABLA DE CONTENIDO

Presentación 5
Generalidades del Módulo 6

Sesión 0: Contextualización 14

Unidad 1: Sentencias de Definición de Datos (DDL) 22
Sesión 1: Definición de objetos en la Base de Datos 22
Sesión 2: Modificar la estructura de las tablas 31
Sesión 3: Practiquemos SQL 39

Unidad 2: Sentencias de Manipulación de Datos (DML) 48
Sesión 4: Formas de insertar datos en la Base de Datos 48
Sesión 5: Consultas con funciones de agrupamiento 58
Sesión 6: Consultas multitabla 67

Unidad 3: Trabajando con subconsultas 75
Sesión 7: Subconsultas anidadas 75
Sesión 8: Diferenciando registros 82
Sesión 9: Algunas funciones útiles 89

Unidad 4: Optimización de procesos en el Sistema Gestor de Base de Datos (SGBD) 96
Sesión 10: Optimización del rendimiento 96
Sesión 11: Optimización de sentencias de consulta 103
Sesión 12: Índices y otras tecnologías de Base de Datos 111

PRESENTACIÓN

El módulo Objetos de Base de Datos ha sido creado con el fin de dar a conocer las diferentes sentencias de SQL, que le permiten realizar un manejo de los objetos y datos de la base de datos de forma óptima.

En el Lenguaje de Consulta Estructurado (SQL), cada sentencia tiene una función específica por realizar, la integridad de los datos y la agilidad en la consultas de información en la Base de Datos, dependerá en gran parte de las sentencias seleccionadas y la forma como se utiliza dicha instrucción. Es por ello, que conocer y dominar las sentencias de SQL para la creación de objetos y manipulación de datos, se convierte en un factor importante dentro la gestión de Base de Datos.

En este módulo el estudiante afianzará sus conocimientos en la gestión de una Base de Datos desde su lenguaje SQL, igualmente aumentará sus habilidades para analizar las bases de datos y optimizar los procesos de consulta en la misma.

Este módulo está dividido en cuatro (4) unidades que le permitirá profundizar y dominar el lenguaje SQL en la manipulación de datos, utilizando de manera óptima los recursos físicos de la Base de Datos. Las unidades a trabajar son:

- Sentencias de definición de datos
- Sentencias de manipulación de datos
- Trabajando con subconsultas
- Optimización de procesos en el SGBD

La primera unidad le permitirá manejar las sentencias de definición de los objetos de la base de datos, así como modificar eficientemente la estructura de la base de datos. La segunda y tercera unidad, se enfocan en el ingreso de datos de diferentes maneras, permitiendo realizar cargas masivas y realizar consulta de los datos bajo varias opciones de sentencias existentes en SQL. En la última unidad, les permitirá verificar que las sentencias anteriormente creadas sean eficientes al manejar grandes volúmenes de información. En caso de no ser eficientes, sabrán optimizar los procesos e instrucciones planteadas.

Es importante resaltar que el módulo está diseñado para un proceso de autoformación y para aprovechar al máximo el espacio de interrelación con el docente en las teleclases, el estudiante deberá estudiar previamente las unidades y participar activamente en los ejercicios planteados en este espacio. Así mismo, es primordial el ingreso frecuente al aula virtual, allí se encuentra tanto la teleclase como el material complementario y las actividades que les permitirán afianzar los conocimientos.

Una adecuada administración del tiempo y una constante dedicación serán necesarias para la culminación satisfactoria de este módulo.

¡Bienvenidos!

GENERALIDADES DEL MÓDULO

FICHA TÉCNICA

Número de créditos: 4
Duración: 17 semanas
Total sesiones: 12

CONTENIDO PROGRAMÁTICO

UNIDADES	SESIONES	TEMAS
	0. Contextualización	• Gestores de Base de Datos • Fases del diseño de Base de Datos • SQL • Structured Query Language – SQL o Lenguaje de Definición de Datos o (DDL) o Lenguaje de Manipulación de o Datos (DML) o Lenguaje de Control de Datos o (DCL) • Cláusulas
Sentencias de Definición de Datos (DDL)	1. Definición de objetos en la Base de Datos	• Creando Base de Datos • Create Databases • Creando tablas • Create Table
	2. Modificar la estructura de tablas	• Modificando definición de las tablas o Alter talble o Operaciones con ADD o First /After o Add constraint o Change Vs Modify o Rename • Operaciones con DROP
	3. Practiquemos SQL	• Estudio de caso a trabajar • El modelo relacional • Codificando el modelo • Sentencias útiles de SQL o Use o Describe o Show

UNIDADES	SESIONES	TEMAS

Sentencias de manipulación de datos (DML)	4. Formas de insertar datos a la Base de Datos	• Insert
		  ○ Insert … values
		  ○ Insert … Set
		  ○ Insert … Select
		• Replace
		• Load data infile

	5. Consultas con funciones de agrupamiento	• Consultas básicas
		• Funciones de agrupamiento
		  ○ COUNT, SUM , AVG, MAX, MIN
		  ○ Group by - Having

	6. Consultas multitablas	• IJOIN • INNER JOIN • CROSS JOIN • LEFT JOIN • RIGHT JOIN

Trabajando con subconsultas	7. Subconsultas anidadas	• Subconsultas
		  ○ Subconsulta en la lista de selección
		  ○ Subconsulta en la cláusula FROM
		  ○ Subconsulta en la cláusula WHERE y HAVING
		• Otras cláusulas para subconsultas
		  ○ ANY, IN, EXISTS

	8. Diferenciando registros	○ UNION ○ ALL ○ DISTINCT ○ DISTINCROW
	9. Algunas funciones útiles	• Funciones numéricas • Funciones de cadena de caracteres • Funciones de manejo de fechas

UNIDADES	SESIONES	TEMAS
Optimización de procesos en el SGBD	10. Optimización del rendimiento	• Estrategias para la optimización del rendimiento ○ Optimización del servidor SQL ○ Optimización de la base de datos ○ Optimización de las consultas • Optimizar las sentecias DML ○ INSERT ○ UPDATE ○ DELETE
	11. Optimización de sentencias de consulta	• Velocidad de las consultas SELECT • Optimizando otras sentencias: ○ WHERE ○ ORDER BY ○ GROUP BY ○ LIMIT
	12. Índices y otras tecnologías de Bases de Datos	• Definición y tipos de índices • Creación de índice • Bases de datos NoSQL ○ ¿Qué son las bases de datos NoSQL? ○ Características • Gestores de bases de datos NoSQL

COMPETENCIA

Interpretar y aplicar las sentencias para el manejo de base de datos, adecuadas para la manipulación y organización de los mismos, siguiendo los criterios de sintaxis definidos en el lenguaje de consulta de base de datos y estándares de la optimización de consultas.

ELEMENTOS DE COMPETENCIA

- Desarrollar alternativas y seleccionar la más adecuada, al momento de plantear soluciones de manipulación de datos a solicitudes específicas.
- Manejar adecuadamente las sentencias de definición de datos que le permita definir y describir los objetos de la base de datos, su estructura, relaciones y restricciones.
- Aplicar optimización de consultas (SQL Tuning) que permita mejorar los tiempos de procesamiento y ejecución de sentencias en un DBMS.
- Conocer comandos básicos de bases de datos NoSQL que permitan comprender el funcionamiento de esta nueva tecnología.

TABLA DE SABERES

SESIÓN CONTEXTUALIZACIÓN		
HACER	**SABER**	**SER**
Reconocer los elementos de los modelos, software y lenguaje para la creación y manipulación de una Base de Datos	• Gestores de Base de Datos • Fases del diseño de base de datos • Structured Query Language - SQL	Reflexionar sobre el uso de las Bases de Datos y su utilidad dentro de la Sociedad de la Información

UNIDAD 1 SENTENCIAS DE DEFINICIÓN DE DATOS

SESIÓN 1 DEFINICIÓN DE OBJETOS EN LA BASE DE DATOS		
HACER	**SABER**	**SER**

Crear el modelo físico de la base de datos en el Sistema Gestor de Base de Datos (SGBD) seleccionado	• Creando BD
	  ○ Create Databases
	• Creando tablas
	  ○ Create Table
		Reconocer la importancia de la estructura de la Base de Datos en los Sistemas de Información

SESIÓN 2 MODIFICAR LA ESTRUCTURA DE UNA TABLA		
HACER	**SABER**	**SER**

Modificar la estructura de las tablas, incluso con datos dentro de ellas	• Modificando definición de las tablas
	  ○ Alter talble
	  ○ Operaciones con ADD
	  ○ First /After
	  ○ Add constraint
	  ○ Change Vs Modify
	  ○ Rename
	  ○ Operaciones con DROP
		Colaborar en la mejoras de diseño de las base de datos

SESIÓN 3 PRACTIQUEMOS SQL		
HACER	**SABER**	**SER**

Poner en práctica las sentencias DDL y DML en un caso particular, así como manejar algunas sentencias útiles para mostrar información importante de la Base de Datos	• Sentencias DDL y DML
	• Sentencias útiles de SQL
	  o Describe
	  o Use
	  o Show
		Valorar la importancia del buen diseño lógico de la base de datos para implementarlo en el SGBD

UNIDAD 2 SENTENCIAS DE MANIPULACIÓN DE DATOS

SESIÓN 4 FORMAS DE INSERTAR DATOS A LA BASE DE DATOS		
HACER	**SABER**	**SER**
Usar las diferentes formas de inserción de datos, incluso para carga masiva de datos	• Insert • Replace • Load data infile	Reconocer la importancia de la integridad de los datos de la BD

SESIÓN 5 CONSULTAS CON FUNCIONES DE AGRUPAMIENTO		
HACER	**SABER**	**SER**
Plantear consultas con agrupamiento de registros y con funciones que permiten contar registros, calcular sumas, promedios, obtener valores máximos y mínimos	• Consultas básicas • Group by – Having • Funciones de agrupamiento o AVG, COUNT, SUM , MAX, MIN	Valorar el uso apropiado de las sentencias de consulta de información.

SESIÓN 6 CONSULTAS MULTITABLA		
HACER	**SABER**	**SER**

Plantear consultas que permite combinar registros de dos o más tablas en una base de datos relacional	• Consultas con:
	  o JOIN
	  o INNER JOIN
	  o CROSS JOIN
	  o LEFT JOIN
	  o RIGHT JOIN
		Interiorizar el proceso de extracción de información desde diferentes tablas de la Base de Datos

UNIDAD 3 TRABAJANDO CON SUBCONSULTAS

SESIÓN 7 SUBCONSULTAS ANIDADAS		
HACER	**SABER**	**SER**
Plantear diversas sentencias de consultas frente a un requerimiento específico	• Subconsultas • Otras cláusulas para subconsultas ○ ANY ○ IN ○ SOME	Conocer diferentes formas de consultar la información con subconsultas

SESIÓN 8 DIRERENCIANDO REGISTROS		
HACER	**SABER**	**SER**
Eliminar los registros duplicados de una consulta	• Subconsultas con las siguientes sentencias: ○ UNION ○ ALL ○ DISTINCT ○ DISTINCTROW	Reconocer la importancia de la forma como se muestra la información en los informes

SESIÓN 9 ALGUNAS FUNCIONES ÚTILES		
HACER	**SABER**	**SER**
Aplicar funciones propias de los SGBD, para optimizar la generación de informes solicitados	• Funciones numéricas • Funciones de cadena de caracteres • Funciones de manejo de fechas	Ver ampliamente la funcionalidad de SQL para integrar grupos de desarrollo de software.

UNIDAD 4 OPTIMIZACIÓN DE PROCESOS EN EL SGBD

SESIÓN 10 OPTIMIZACIÓN DEL RENDIMIENTO		
HACER	**SABER**	**SER**
Conocer la optimización del rendimiento de las sentencias SQL e identificar estrategias para optimizarlas en caso de ser necesario	• Estrategias para la optimización del rendimiento • Optimizar las siguientes sentencias: ○ INSERT ○ UPDATE ○ DELETE	Analizar y reflexionar sobre el uso eficiente de las sentencias SQL en todos los proceso con las Bases de Datos

SESIÓN 11 OPTIMIZACIÓN DE SENTENCIAS DE CONSULTA		
HACER	**SABER**	**SER**
Optimizar las sentencias de consulta de información en la Base de Datos	• Velocidad de las consultas SELECT • Optimizando otras sentencias: ○ WHERE, ORDER BY, GROUP BY, LIMIT	Reconoce la importancia de optimizar los procesos de búsqueda de información

SESIÓN 12 DISEÑO Y CREACIÓN DE ÍNDICES		
HACER	**SABER**	**SER**
Crear y manipular índices en la Base de Datos, así como identificar nuevas tecnologías de bases de datos	• Definición de índice • Creación de índices • Manipulación de índices • Bases de datos NoSQL	Reconocer la importancia de los índices en el proceso de optimización de procesos en a BD y el uso de nuevas tecnologías como solución a graves problemas de optimización

CONTEXTUALIZACIÓN

SESIÓN

CONTEXTUALIZACIÓN

Usualmente, los diseñadores de bases de datos realizan la creación de una base de datos teniendo en cuenta las fases definidas para el diseño. Esto facilita la identificación de las necesidades del cliente y el diseño adecuado de la base de datos. En esta sesión recordaremos las etapas del diseño de una base de datos y el modelo que se realiza en cada una de estas.

¿QUÉ SABES SOBRE...?

1. Identifique los Gestores de Base de Datos más utilizados y clasifíquelos si tiene licencia libre o comercial.

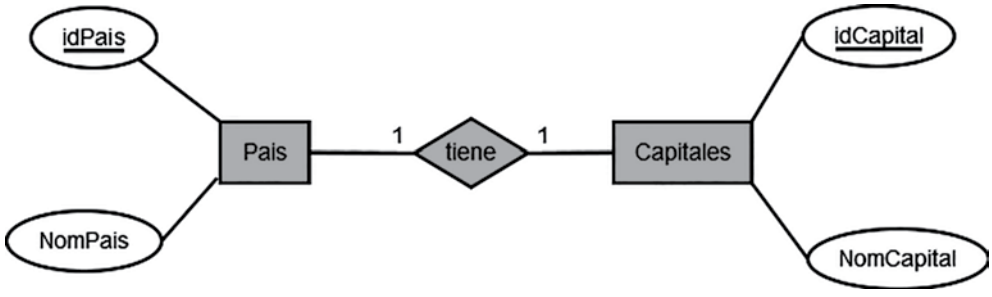
_____

_____

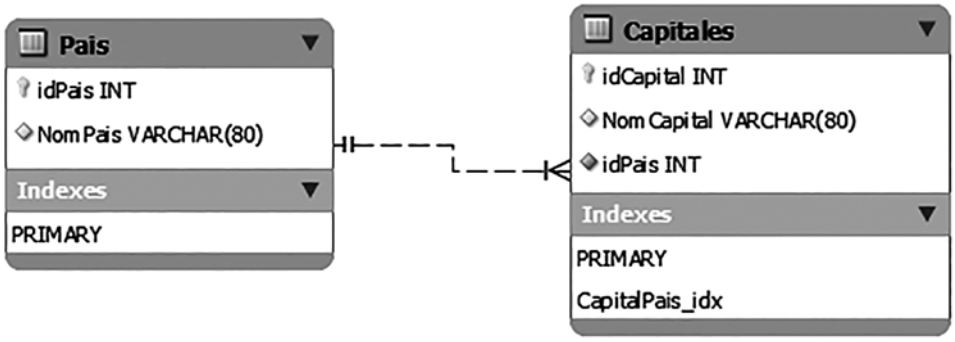
_____

_____

2. Identifique el nombre de los siguientes diagramas de bases de datos.



Tipo de diagrama: _____



Tipo de diagrama: _____

1. GESTORES DE BASE DE DATOS

Los Sistemas Gestores de Base de Datos (SGBD) o también llamados DBMS (Data Base Management System) son un con conjunto de aplicaciones que sirven para definir, construir y manipular una base de datos. Estos SGBD prestan los siguientes servicios:

- Creación y definición de la Base de Datos
- Manipulación de los datos
- Acceso controlado a los datos de la Base de Datos
- Mantener seguridad e integridad de los datos
- Acceso compartido a la base de datos
- Mecanismos de respaldo y recuperación

Estos paquetes de software son complejos y deben proporcionar una serie de servicios que permiten almacenar y exportar los datos de forma eficiente. Los componentes principales de los SGBD son:

- ***Lenguajes de los SGBD:*** Todos los SGBD, manejan el Lenguaje de Consulta Estructurado (SQL), el cual se divide en varios grupos para ofrecer las sentencias apropiadas a cada tipo de usuario: administradores, diseñadores, programadores y usuarios finales.

- ***El diccionario de datos:*** es el lugar donde se guarda información acerca de todos los datos que forman la BD. Este contiene todas las características lógicas de la BD, se incluye nombre, descripción, alias, definición de las tablas y atributos.

- ***Seguridad e integridad de datos:*** debe garantizar la protección de los datos, realizar copias de seguridad y restauración en caso de ser necesario.

- ***El Administrador de la Base de Datos (DBA):*** Su función es precisamente administrar la base de datos, por lo tanto es el usuario con una gran responsabilidad ya que posee el máximo nivel de privilegios para desempeñar correctamente su función.

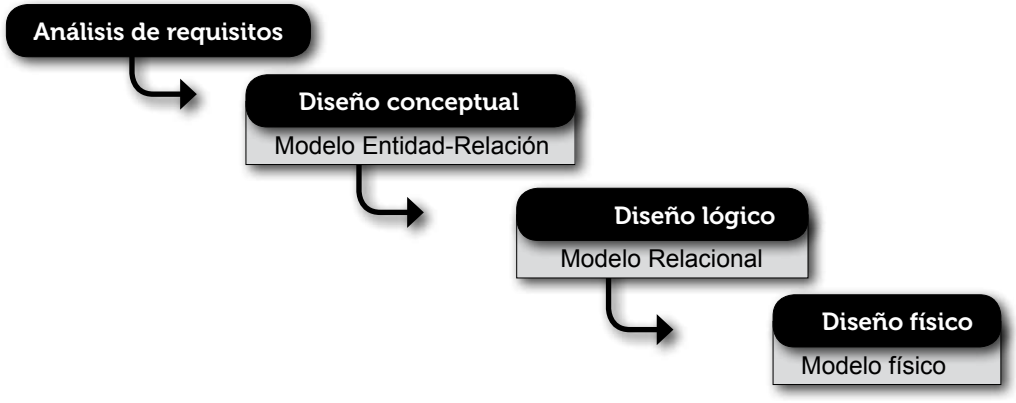
Entre los SGBD más conocidos y utilizados en el desarrollo de aplicaciones tenemos los siguientes:

Gráfico No. 1. Algunos SGDB.

2. FASES DEL DISEÑO DE BASE DE DATOS SQL

En el área del desarrollo de software, el desarrollo de Sistemas de Información se realiza bajo metodologías de desarrollo de software. Estas permiten estructurar, planificar y controlar el proceso de desarrollo de los aplicativos.

Para el desarrollo de Interfaces Gráficas, una determinada metodología no es necesariamente aplicable a todo tipo de proyecto. Cuando se habla de bases de datos, también se aplica una metodología que contiene cuatro (4) fases principales, tal como se muestra en la figura N° 1.



Esquema No. 1. Fases del diseño de base de datos.

La ***fase de análisis de requisitos***, implica un estudio del negocio o actividad del cliente. Esto permite definir con el usuario final, los requisitos exactos que deberá poseer la base de datos. Se pueden aplicar diferentes técnicas de para la determinación de requerimientos. Entre estos se encuentra: la observación, entrevistas y cuestionarios, revisión de documentos.

Durante la ***fase de diseño conceptual***, la información recopilada se transforma en un modelo conceptual. Esto es independientes del SGBD seleccionado y el lenguaje de programación a utilizar. Aquí se determinan las entidades, con su propio conjunto de atributos, y cómo se conectan las entidades. Estas relaciones proporcionan una representación gráfica de alto nivel, el cual es conocido como Modelo Entidad Relación (MER). En el gráfico N°2, se puede identificar un MER.

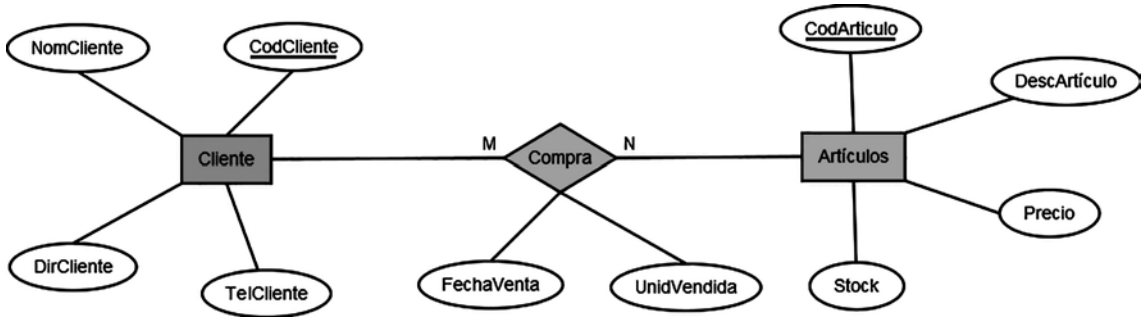


Gráfico No. 2. Modelo Entidad Relación.

El propósito de la ***fase de diseño lógico***, es validar las relaciones en la base de datos utilizando una técnica conocida como normalización. La normalización es el proceso de organización de datos en una base de datos, en donde se elimina la redundancia de datos y permite corregir los errores en el diseño. Este modelo es dependiente del SGBD que se vaya a utilizar. La entrada a la fase de diseño lógico es el Modelo Entidad Relación y la salida es el conjunto relaciones normalizadas, a este modelo se le conoce como Modelo Relacional. Ver gráfico N°3.

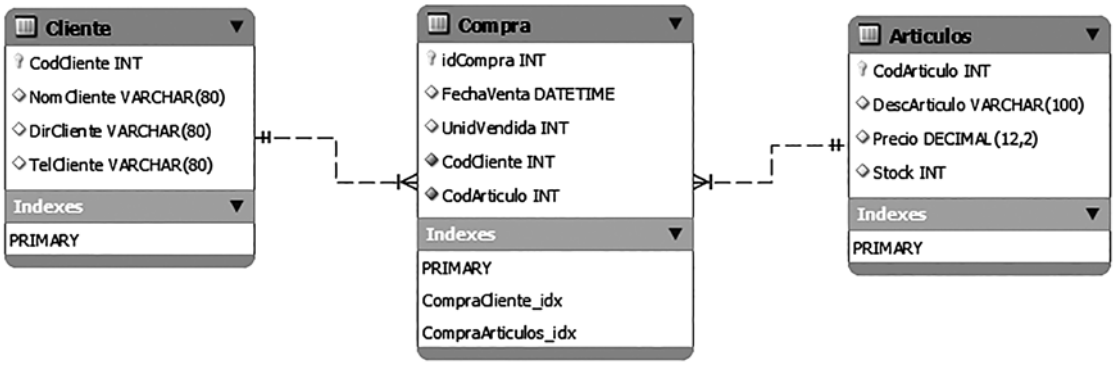


Gráfico No. 3. Modelo Relacional.

La ***fase de diseño físico*** consiste en especificar el SGBD relacional a utilizar, tal como MySQL, Oracle, SQL Server, entre otros, en el cual se implementará el diseño lógico base de datos. De igual manera, se define las restricciones de integridad, índices y políticas de seguridad del sistema.

3. STRUCTURED QUERY LANGUAGE - SQL

El Lenguaje de Consulta Estructurado (SQL), es un lenguaje de base de datos normalizado, utilizado por los diferentes SGBD para realizar determinadas operaciones sobre los datos o sobre la estructura de los mismos. Hay que tener en cuenta que cada SGBD tiene sus particularidades y lo hace diferente de otro motor.

En SQL existen bastantes sentencias que se pueden utilizar para realizar diversas tareas. Dependiendo de la tarea, estas sentencias se pueden clasificar en tres grupos principales: Lenguaje de Definición de Datos (DDL), Lenguaje de Manipulación de Datos (DML) y Lenguaje de Control de Datos (DCL), aunque existe otro grupo de sentencias que no está dentro del lenguaje SQL sino del PL/SQL (Lenguaje de Programación / Lenguaje de Consulta Estructurado). Además de estas sentencias o cláusulas, el lenguaje SQL está compuesto por cláusulas, operadores y funciones de agregado. Estos elementos se combinan en las instrucciones para crear, actualizar y manipular las bases de datos.

Lenguaje de Definición de Datos (DDL)

Las sentencias DDL se utilizan para crear y modificar la estructura de las tablas y otros objetos de la base de datos. Las sentencias que se encuentran en este grupo son las siguientes:

SENTENCIA	DESCRIPCIÓN
CREATE	Utilizado para crear nuevas tablas, campos e índices y todos los objetos de la base de datos.
ALTER	Se utiliza para para modificar las tablas agregando campos o cambiando la definición de los mismos.
DROP	Empleado para eliminar objetos de la base de datos como tablas, índices, procedimientos, entre otros.
RENAME	Con esta sentencia podemos cambiar el nombre de objetos de base de datos como tablas, sinónimos, vistas, entre otros.
COMMENT	Este comando permite comentar o documentar objetos de base de datos como una tabla o columna, así de esta manera se almacena la documentación de dicho objeto.

Lenguaje de Manipulación de Datos (DML)

En este grupo están las sentencias que requieren de confirmación para hacer definitivos los cambios hechos a través de este tipo de sentencias, es decir, no tienen commit implícito. Las sentencias que se encuentran en este grupo son:

SENTENCIA	DESCRIPCIÓN
SELECT	Se usa para recibir registros seleccionados desde una o más tablas
INSERT	Inserta nuevos registros en una tabla existente.
UPDATE	Actualiza columnas en registros de tabla existentes con nuevos valores
DELETE	Elimina los registros de la base de datos.
TRUNCATE	Es una sentencia que borra toda la información de una tabla, pero, que no genera información que pueda deshacerse con rollback.
REPLACE	Funciona exactamente como INSERT, excepto que si un valor de la tabla tiene el mismo valor que un nuevo registro para un índice PRIMARY KEY o UNIQUE, el antiguo registro se borra antes de insertar el nuevo.

Lenguaje de Control de Datos (DCL)

En este grupo están las sentencias que permiten configurar controles de seguridad sobre objetos de la base de datos. Estos a su vez se subdividen en las sentencias de Control de Acceso y Control de Transacciones. Las sentencias que se encuentran en este grupo son:

Control de Acceso:

SENTENCIA	DESCRIPCIÓN
GRANT	Concede privilegios de acceso a usuarios a la base de datos.
REVOKE	Elimina privilegios de acceso a usuarios a la base de datos.

Control de Transacciones:

SENTENCIA	DESCRIPCIÓN
COMMIT	Marca el final de una transacción correcta, implícita o explícita. Hace que todas las modificaciones efectuadas sobre los datos desde el inicio de la transacción sean parte permanente de la base de datos, libera los recursos mantenidos por la transacción.
ROLBACK	Deshace una transacción explícita o implícita hasta el inicio de la transacción o hasta un punto de retorno dentro de la transacción. Borra todas las modificaciones de datos realizadas desde el inicio de la transacción o hasta un punto de retorno. También libera los recursos que mantiene la transacción.

Cláusulas

Las cláusulas son las condiciones que modifican nuestras consultas y son utilizadas para definir los datos que desea seleccionar o manipular.

COMANDO	DESCRIPCIÓN
FROM	Utilizada para especificar la(s) tabla(s) de la cual(es) se van a seleccionar los registros.
WHERE	Utilizada para especificar las condiciones que deben reunir los registros que se van a seleccionar.
GROUP BY	Utilizada para separar los registros seleccionados en grupos específicos.
HAVING	Utilizada para expresar la condición que debe satisfacer cada grupo si fue utilizado el GROUP BY.
ORDER BY	Utilizada para ordenar los registros seleccionados de acuerdo con un orden específico.

ACTIVIDAD DE APRENDIZAJE

Propósito: Manejar adecuadamente las fases del diseño de una Base de Datos.

Descripción:

Ic Construir el Modelo Entidad Relación y Modelo Relacional que refleje toda la información necesaria para la gestión de libros manejados en una imprenta. Tenga en cuenta los siguientes requerimientos:

Aa

a. Se necesita almacenar información sobre los libros que imprime una imprenta.
b. Cada libro es identificado únicamente por su ISBN. El resto de información sobre un libro incluye la fecha del título y la de publicación.
c. Además de la información del libro, hay también información guardada sobre el editor del libro. Esto incluye un identificador único del editor, el nombre del editor, y la dirección. Un libro solo se puede publicar por un editor.
d. La información sobre los autores de un libro también se guarda. Esta información incluye el código, el nombre, y el número de identificación del autor.
e. Cualquier libro se puede escribir por uno o varios autores.
f. Cuando se imprime el libro, se envía a una imprenta. La información sobre la imprenta incluye el identificador único, el nombre y la dirección. Se escribe un contrato que indica que el número de libros que imprimirá la imprenta y el plazo de impresión.
g. Ocasionalmente, un mismo libro se puede contratar a varias imprentas si la cantidad requerida excede la producción de la imprenta.

Iv Construya los diagramas en las aplicaciones que se trabajan para el desarrollo del módulo. Participe en el foro de la sesión con la solución de la actividad.

Criterios de autoevaluación

- Realizar los modelos de base de datos para el caso de la imprenta. Tenga en cuenta que el Modelo Relacional debe estar normalizado.
- Participar activamente en el foro con la solución de la actividad y realiza aportes a los trabajos de sus compañeros.

WEBGRAFÍA

- Sentencias de definición de datos (Data Definition Statements). Documentación MySQL. Recuperado el 02 de octubre de 2014 de: http://goo.gl/1EqAnZ
- Sentencias de manipulación de datos (Data Manipulation Statements). Documentación MySQL. Recuperado el 02 de octubre de 2014 de: http://goo.gl/5psVVB

ANOTACIONES

UNIDAD 1

SENTENCIAS DE DEFINICIÓN DE DATOS

SESIÓN 1

DEFINICIÓN DE OBJETOS EN LA BASE DE DATOS

El lenguaje SQL es el lenguaje universal en los sistemas de base de datos. Este lenguaje nos permite realizar crear todos los elementos o componentes de la base de datos, así como crear consultas a nuestras bases de datos para mostrar, insertar, actualizar y borrar datos. En esta sesión trabajaremos la creación de la base de datos, tablas y sus relaciones.

¿QUÉ SABES SOBRE...?

1. Escriba el orden en que se deben crear las tablas de la siguiente base de datos.

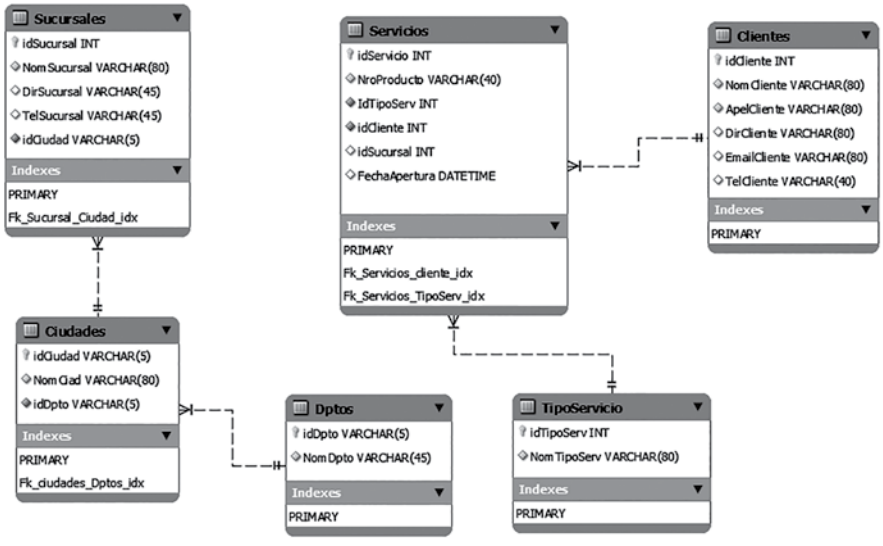


Gráfico No. 4. Modelo Relacional – Caso Banco.

2. Indique que tipos de relaciones entre entidades pueden existir en una base de datos. De un ejemplo de cada una de ellas.

1. CREANDO BASE DE DATOS

Para crea una base de datos con el nombre dado se utiliza la sentencia CREATE DATABASE. Para utilizar esta sentencia, se necesita que el usuario en el que está trabajando tenga el privilegio para CREAR la base de datos. CREATE SCHEMA otra sentencia que es sinónimo de CREATE DATABASE.

CREATE DATABASE

La sintaxis para la creación de la Base de Datos es la siguiente:

```
CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] db_name
    [create_specification]...

create_specification:
    [DEFAULT] CHARACTER SET [=] charset_name
  | [DEFAULT] COLLATE [=] collation_name
```

Figura No. 1. Sintaxis sentencia CREATE DATABASE.

Para explicar la implementación de la sintaxis de las sentencias SQL trabajaremos el caso del banco, donde su Modelo Relacional es el siguiente:

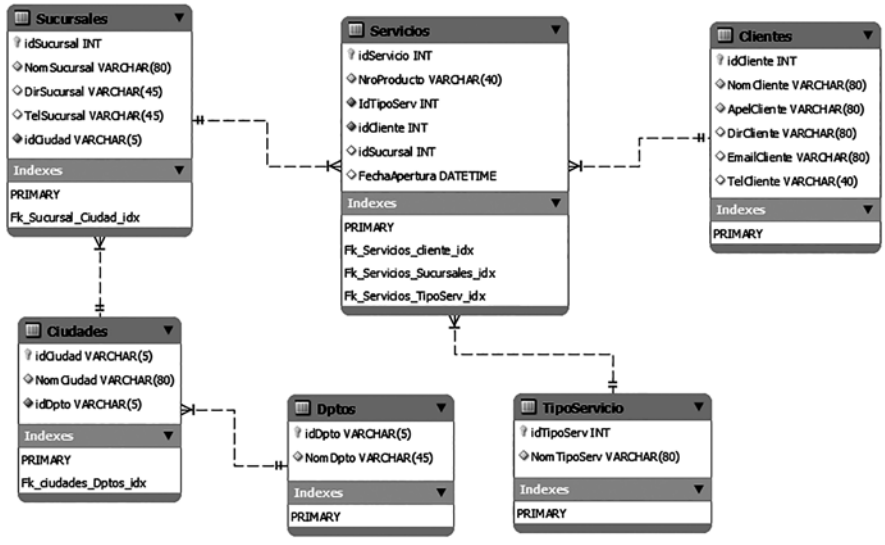


Gráfico No. 5. Modelo Relacional – Caso Banco.

```
1   CREATE DATABASE Banco DEFAULT CHARACTER SET latin1 COLLATE
    latin1_swedish_ci;
2   USE banco;
```

Figura No. 2. Sentencia CREATE DATABASE.

Expliquemos código que se encuentra en la figura No. 2:

Código de creación de la base de datos.

- **1:** Se crea la base de datos denominada Banco. CHARSET hace referencia a cómo MySQL guarda internamente el dato y COLLATION es una manera de decirle cómo debe comparar el texto y/o ordenarlo. Si se omiten, MySQL los definirá por defecto.

- **2:** Se selecciona la BD en la que se desea trabajar. Para nuestro caso es la BD *Banco*.

RECOMENDACIONES PARA NOMBRES DE LOS OBJETOS DE LA BD

Los nombres de bases de datos, tablas, índices, columnas y alias son identificadores. La sintaxis permitida para los identificadores se describe a continuación:

IDENTIFICADOR	LONGITUD MÁXIMA (bytes)	CARACTERES PERMITIDOS
Base de datos	64	Cualquier carácter permitido en un nombre de directorio, excepto '/', '\', o '.'
Tabla	64	Cualquier carácter permitido en un nombre de fichero, excepto '/', '\', o '.'
Columna	64	Todos los caracteres.
Índice	64	Todos los caracteres.
Alias	255	Todos los caracteres.

Adicionalmente, hay otras recomendaciones al momento de definir un identificador. Estos son:

- Ningún identificador puede contener un carácter ASCII 0 o un byte con un valor de 255.
- Los nombres de bases de datos, tablas y columnas no deberían terminar con caracteres de espacio.
- Se permite el uso de comillas en identificadores, aunque es mejor evitarlos.

2. CREANDO TABLAS

Las tablas son la estructura básica donde se almacena la información en la base de datos. Existen herramientas de base de datos que le permiten crear tablas sin ingresar código SQL, pero debido a que las tablas son los contenedores de toda la información, es importante conocer la sintaxis CREATE TABLE.

Las tablas se dividen en filas y columnas. Cada fila representa una parte de los datos y es conocida como *registro o tupla*, y cada columna es una parte que compone la información del registro. Entonces, por ejemplo, si tenemos una tabla para registrar la información del cliente, las columnas pueden ser Nombres, Apellidos, Dirección, Ciudad, Email, Celular, Fecha de Nacimiento, entre otros. Como resultado, cuando se crea una tabla, incluimos los títulos de columna y los tipos de datos para esta columna en particular.

CREATE TABLE

La sintaxis para la creación de la tabla es la siguiente:

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name (
  col_name type [NOT NULL | NULL] [DEFAULT default_value]
        [AUTO_INCREMENT] [UNIQUE [KEY] | [PRIMARY] KEY]
        [COMMENT 'string'],
  …,
 CONSTRAINT const_name PRIMARY KEY (col_name,…),
 [CONSTRAINT const_name FOREIGN KEY (col_name,…)
  REFERENCES tbl_name]
);
```

Figura No. 3. Sintaxis sentencia CREATE TABLE.

Nota:

para conocer al detalle de los tipos de datos que maneja MySQL y sus restricciones, te recomiendo consultar el siguiente enlace: ***http://goo.gl/y1yMbj.***

Las restricciones que puede tener una columna de la tabla son los siguientes:

RESTRICCIÓN	DESCRIPCIÓN
NOT NUL | NULL	NULL implica ausencia de información, no representa cero ni cadena vacía. Es decir, es la obligatoriedad de definir un dato para la columna al momento de insertar un registro. En forma predeterminada la columna puede ser NULL.

DEFAULT	La restricción DEFAULT se usa para insertar un valor predeterminado a una columna. El valor por defecto se añadirá a todos los nuevos registros, si no se especifica ningún otro valor.
AUTO_INCREMENT	Permite un número único que se genera cuando se inserta un nuevo registro en una tabla. Por defecto, el valor inicial para AUTO_INCREMENT es 1, y que se incrementará en 1 por cada nuevo registro, pero se puede iniciar la secuencia con otro valor. Solo puede existir un campo en la tabla con esta propiedad.
UNIQUE	La restricción UNIQUE identifica de forma exclusiva cada registro de una tabla de base de datos. Las restricciones de clave única y primaria tanto proporcionan una garantía de unicidad para una columna o conjunto de columnas. Una restricción PRIMARY KEY automáticamente tiene una restricción UNIQUE definida en él. Tenga en cuenta que una tabla puede tener muchas restricciones UNIQUE, pero sólo una restricción PRIMARY KEY por tabla.
PRIMARY KEY	La restricción PRIMARY KEY identifica de forma exclusiva cada registro de una tabla de base de datos. Las claves principales deben contener valores únicos. Una columna de clave principal no puede contener valores NULL. La mayoría de las tablas deben tener una clave principal y cada tabla sólo puede tener una clave primaria.
COMMENT	Un comentario para una columna puede especificarse con esta opción. El comentario se muestra con los comandos SHOW CREATE TABLE y SHOW FULL COLUMNS.

RECOMENDACIÓN: Para crear una base de datos relacional, debe asignar un orden de creación a sus tablas del diseño. Primero se crean las tablas que solo tienen llave primaria y luego se crean las que tienen llaves foráneas, ya que al momento de crear la FOREIGN KEY, debe existir la tabla con la cual se va a relacionar.

Ahora vamos a ver la implementación de la sintaxis de la sentencia CREATE TABLE. Recuerde que trabajaremos el Modelo Relacional del caso del banco que se encuentra en el gráfico No. 5.

Antes de empezar a codificar, se define el orden de creación de las tablas. El orden recomendado es el siguiente: Clientes, TipoServicio, Dptos, Ciudades, Sucursales y Servicios.

Ahora revisemos la sintaxis de creación de cada una de estas tablas.

```
1    CREATE TABLE IF NOT EXISTS clientes (
2      idCliente int(11) NOT NULL,
3      NomCliente varchar(80) NOT NULL,
4      ApelCliente varchar(80) NOT NULL,
5      DirCliente varchar(80) DEFAULT NULL,
6      EmailCliente varchar(80) DEFAULT NULL,
7      TelCliente varchar(40) DEFAULT NULL,
8      CONSTRAINT PK_Clientes PRIMARY KEY (idCliente)
9    ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Figura No. 4. Código de creación de la tabla CLIENTES.

Expliquemos código que se encuentra en la figura No. 4:

Código de creación de la tabla CIENTES. Esta es una entidad que solo tiene llave primaria.

- **1:** Sentencia para la creación de la tabla ***CLIENTES***. Se incluye ***IF NOT EXIST***, la cual permite verificar si existe la tabla, en caso de no existir la crea. Si existe, omite el código.

- **2 – 4:** Los atributos tiene la restricción ***NOT NULL***, lo cual implica que se debe incluir información obligatoria para estos campos.

- **5 – 7:** Los atributos tiene la restricción ***DEFAULT***, lo cual indica que se está definiendo un valor por defecto para los atributos cuando no se incluya información en un ***INSERT***. El valor por defecto es ***NULL***.

- **8:** Se define que el atributo ***idCliente*** será la llave primaria de la tabla.

- **9:** ***CHARSET*** hace referencia a cómo MySQL guarda internamente el dato y ***ENGINE*** define un motor de almacenamiento transaccional con capacidades realizar ***COMMIT*** (confirmación), ***ROLLBACK*** (cancelación) y recuperación de fallas.

Ahora vamos a revisar el código de la tabla SERVICIOS, del modelo del caso Banco. Esta entidad contiene varias llaves foráneas, lo que indica que se relaciona con varias entidades en el modelo.

```
1   CREATE TABLE IF NOT EXISTS servicios (
2     idServicio int(11) NOT NULL AUTO_INCREMENT,
3     NroProducto varchar(40) NOT NULL,
4     IdTipoServ int(11) NOT NULL,
5     idCliente int(11) NOT NULL,
6     idSucursal int(11) DEFAULT NULL,
7     FechaApertura datetime DEFAULT NULL,
8     CONSTRAINT PK_Servicios PRIMARY KEY (idServicio),
9     CONSTRAINT Fk_Servicios_cliente FOREIGN KEY (idCliente)
      REFERENCES clientes (idCliente),
10    CONSTRAINT Fk_Servicios_Sucursales FOREIGN KEY (idSucursal)
      REFERENCES sucursales (idSucursal),
11    CONSTRAINT Fk_Servicios_TipoServ FOREIGN KEY (IdTipoServ)
      REFERENCES tiposervicio (idTipoServ)
12  ) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;
```

Figura No. 5. Código de creación de la tabla SERVICIOS.

Expliquemos código que se encuentra en la figura No. 5:

- **1:** Sentencia para la creación de la tabla ***SERVICIOS***. Se incluye ***IF NOT EXIST***, la cual permite verificar si existe la tabla, en caso de no existir la crea. Si existe, omite el código.

- **2:** Se define el atributo ***idServicio*** con propiedad de ***AUTO_INCREMENT***, lo cual permite que se incremente automáticamente el valor del atributo cada vez que se inserte un registro.

- **2 – 5:** Los atributos tiene la restricción ***NOT NULL***, lo cual implica que se debe incluir información obligatoria para estos campos.

- **6 – 7:** Los atributos tiene la restricción ***DEFAULT***, lo cual indica que se está definiendo un valor por defecto para los atributos cuando no se incluya información en un ***INSERT***. El valor por defecto es ***NULL***.

- **8:** Se define que el atributo ***idServicio*** será la llave primaria de la tabla.

- **9:** Se define la llave foránea o relación con la tabla ***CLIENTES*** a través del atributo ***idCliente***.

- **10:** Se define la llave foránea o relación con la tabla ***SUCURSALES*** a través del atributo ***idSucursal***.

- **11:** Se define la llave foránea o relación con la tabla ***TIPOSERVICIO*** a través del atributo ***IdTipoServ***.

- **12:** ***CHARSET*** hace referencia a cómo MySQL guarda internamente el dato y ***ENGINE*** define un motor de almacenamiento transaccional con capacidades realizar ***COMMIT*** (confirmación), ***ROLLBACK*** (cancelación) y recuperación de fallas. También se define el valor inicial de 1 del atributo con propiedad **AUTO_ INCREMENT**.

ACTIVIDAD DE APRENDIZAJE

Propósito: Utilizar correctamente las sentencias SQL del grupo DDL y define correctamente las restricciones apropiadas y necesarias a cada atributo del modelo.

Descripción:

Ic Codifica en el Gestor de Base de Datos seleccionado, el siguiente Modelo Relacional para el caso del Banco, donde se almacena la información de los clientes y los servicios que tiene cada uno de ellos.

Aa

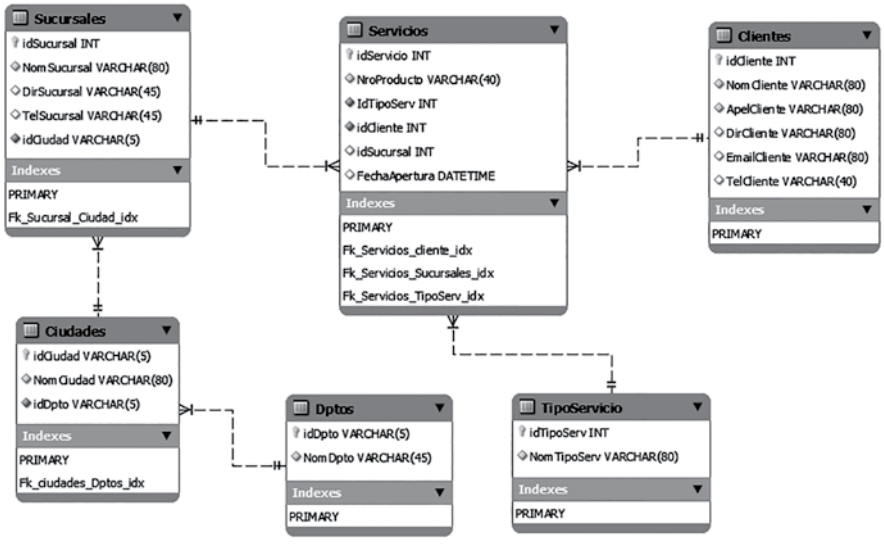


Gráfico No. 6. Modelo Relacional – Caso Banco.

Iv Construya el código donde crea toda la BD del banco y defina a cada atributo la(s) restricción(es) adecuada(s). Participe en el foro de la sesión con la solución de la actividad.

Criterios de autoevaluación:

- Utilizar las sentencias apropiadas para crear la Base de Datos y definir las restricciones apropiadas a cada atributo.
- Participar activamente en el foro con la solución de la actividad y realiza aportes a los trabajos de sus compañeros.

WEBGRAFÍA

- Manual de referencia MySQL. Sintaxis de CREATE DATABASE. Recuperado el 03 de octubre de 2014 de: *http://goo.gl/nlwOdM*
- Manual de referencia MySQL. Sintaxis de CRATE TABLE. Recuperado el 03 de octubre de 2014 de: *http://goo.gl/ydyg8C*

ANOTACIONES

UNIDAD 1

SENTENCIAS DE DEFINICIÓN DE DATOS

SESIÓN 2

MODIFICAR LA ESTRUCTURA DE UNA TABLA

El lenguaje SQL es el lenguaje universal en los sistemas de base de datos. Este lenguaje nos permite realizar crear todos los elementos o componentes de la base de datos, así como crear consultas a nuestras bases de datos para mostrar, insertar, actualizar y borrar datos. En esta sesión trabajaremos la creación de la base de datos, tablas y sus relaciones.

¿QUÉ SABES SOBRE...?

1 ¿Cuál es la sentencia para realizar modificaciones a la estructura de una tabla? Defina su sintaxis.

_____

_____

_____

2. Indique que tipos de relaciones entre entidades pueden existir en una base de datos. De un ejemplo de cada una de ellas.

_____

_____

_____

1. MODIFICANDO DEFINICIÓN DE LAS TABLAS

Para modificar la estructura de objetos ya creados en una base de datos, se utiliza la sentencia ALTER. Este comando pertenece a la categoría DDL (Data Definition Language). Entonces, para modificar la estructura de una tabla, se utiliza la sentencia

ALTER TABLE, el cual le permite añadir o borrar columnas, agregar o eliminar índices, cambiar el tipo de columnas existentes, o renombrar columnas o la misma tabla, así como su comentario.

ALTER TABLE

La sintaxis para la modificar la definición de una tabla es la siguiente:

```
ALTER [IGNORE] TABLE tbl_name
    alter_specification [, alter_specification] ...

alter_specification:
  ADD [COLUMN] column_definition [FIRST | AFTER col_name ]
  | ADD [COLUMN] (column_definition, …)
  | ADD [CONSTRAINT [symbol]]
        PRIMARY KEY (col_name, …)
  | ADD [CONSTRAINT [symbol]]
      FOREIGN KEY (col_name, …)
        [reference_definition]
  | ALTER [COLUMN] col_name {SET DEFAULT literal | DROP DEFAULT}
  | CHANGE [COLUMN] old_col_name column_definition
        [FIRST|AFTER col_name]
  | MODIFY [COLUMN] column_definition [FIRST | AFTER col_name]
  | DROP [COLUMN] col_name
  | DROP PRIMARY KEY
  | DROP INDEX index_name
  | DROP FOREIGN KEY fk_symbol
  | DISABLE KEYS
  | ENABLE KEYS
  | RENAME [TO] new_tbl_name
  | ORDER BY col_name
```

Figura No. 6. Sintaxis sentencia ALTER TABLE.

A continuación encontrarás ejemplos de los procesos más utilizados con la sentencia ALTER TABLE.

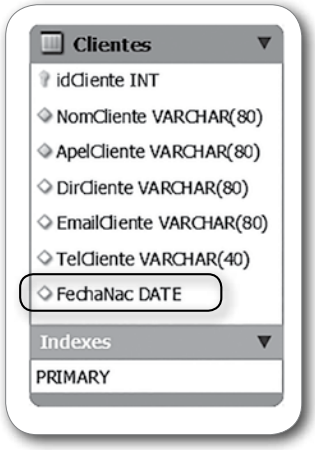
Operaciones con ADD

Permite agregar una columna en una tabla.

```
ALTER TABLE table_name
ADD column_name datatype;
```

Figura No. 7. Sintaxis para agregar una columna.

Se tiene el nuevo campo de la ***fecha de nacimiento*** en la tabla ***CLIENTES***.




```
ALTER TABLE Clientes
ADD FechaNac Date;
```

Figura No. 8. Agregar una columna a la tabla Clientes.

First / After

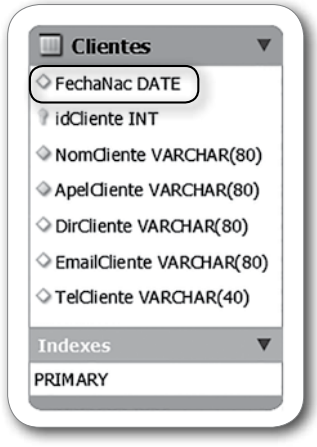
Permite posicionar un campo de la tabla después de otro campo o como el primer campo de la tabla.

```
ALTER TABLE Clientes
ADD FechaNac Date AFTER ApelCliente;
```

Figura No. 9. Agregar una columna a la tabla Clientes – AFTER.

```
ALTER TABLE Clientes
ADD COLUMN FechaNac Date FIRST;
```

Figura No. 10. Agregar una columna a la tabla Clientes – FIRST.



Add Constraint

Cuando se codifica y se olvida incluir la llave foránea de la tabla, se puede agregar sin necesidad de eliminar la tabla completa. Vea el siguiente ejemplo.

```
ALTER TABLE Ciudades
ADD CONSTRAINT FK_Ciudades_Dptos FOREIGN KEY (idDpto)
    REFERENCES Dptos(idDpto);
```

Figura No. 11. Agregar un CONTRAINT a la tabla Clientes.

Se observa que es la misma sintaxis de la definición, solo se agrega ADD. Lo mismo se puede realizar con la PRIMARY KEY. Solo se incluye la definición de la llave primaria.

CHANGE Vs. MODIFY

Cuando se necesita cambiar el nombre a una columna, puede usarse la cláusula CHANGE, esta tiene la siguiente estructura:

```
CHANGE old_col_name create_definition
```

Para hacerlo, hay que especificar los nombres antiguo y nuevo de la columna y el tipo de dato que la columna tiene actualmente. Veamos el siguiente ejemplo:

Se tiene el atributo ***Nomcudad de la tabla Ciudad*** y se desea cambiar su nombre a ***NomCiudad.***

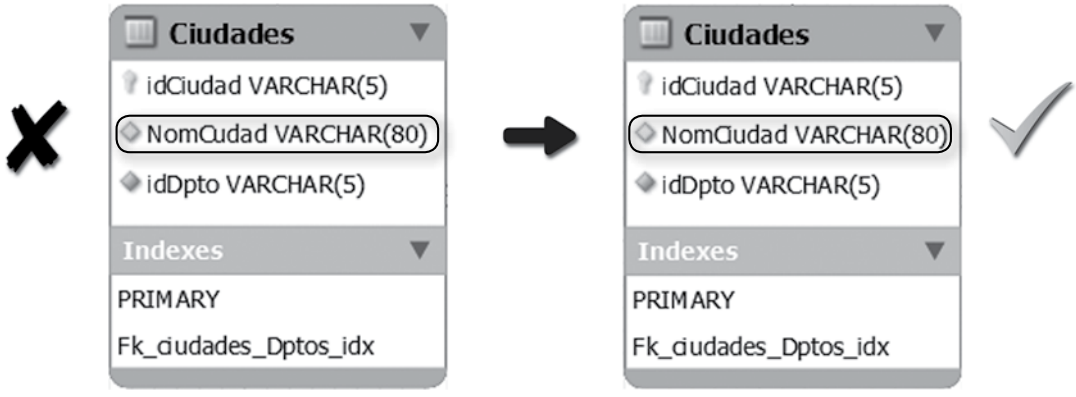


Gráfico No. 7. Tabla Ciudades. Error en el nombre del atributo.

```
ALTER TABLE Ciudades
CHANGE NomCudad Nomciudad VARCHAR(80);
```

Figura No. 12. Ejemplo de la instrucción CHANGE. Cambiar nombre del atributo.

Si se quiere cambiar el tipo de dato de una columna, pero no su nombre, la sintaxis de **CHANGE** es válida, pero sigue necesitando un nombre de columna antiguo y nuevo, aunque mantenga en mismo nombre. Por ejemplo:

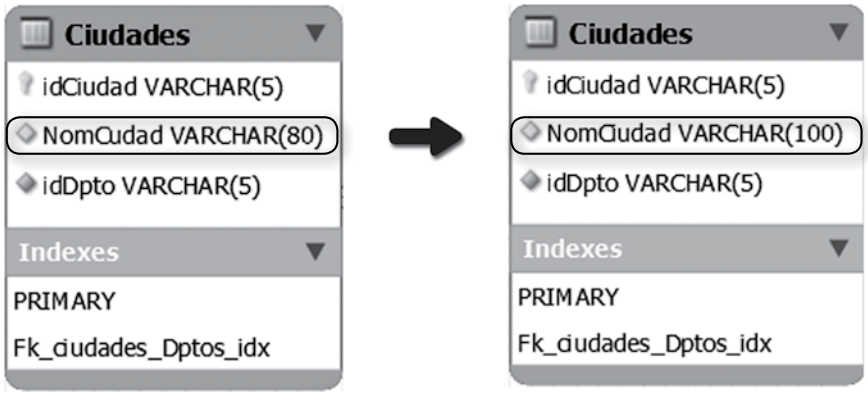


Gráfico No. 8. Tabla Ciudades. Cambio de tipo de dato.

```
ALTER TABLE Ciudades
CHANGE  Nomciudad Nomciudad VARCHAR(100);
```

Figura No. 13. Ejemplo de la instrucción CHANGE. Cambiar tipo de dato del atributo.

Si desea modificar solo el tipo de dato de la columna, puede usar también **MODIFY** sin necesidad de renombrarla.

```
ALTER TABLE Ciudades
MODIFY  Nomciudad VARCHAR(100)NOT NULL;
```

Figura No. 14. Ejemplo de la instrucción MODIFY. Cambiar tipo de dato del atributo.

Se aclara, que cuando se cambia un tipo de columna usando ***CHANGE*** o ***MODIFY***, MySQL intenta convertir datos al nuevo tipo lo mejor posible. También es posible utilizar ***FIRST*** y ***AFTER*** con estas dos cláusulas.

RENAME

Esta opción de ALTER TABLE, permite cambiar el nombre de la tabla actual.

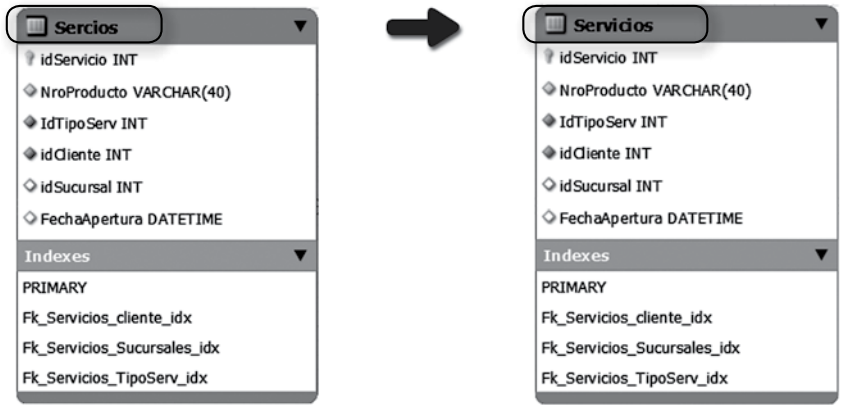


Gráfico No. 9. Tabla Servicios. Cambio de nombre de la tabla.

```
ALTER TABLE Sercios
RENAME Servicios;
```

Figura No. 15. Ejemplo de la instrucción RENAME.

Operaciones con DROP

Con ***DROP***, se puede eliminar cualquier elemento de la tabla en la que se encuentra trabajando. Puede eliminar columnas (atributos), llave primaria, llaves foráneas e index, solo se debe indicar el objeto a elemento y el nombre del mismo. A continuación podrás observar algunas aplicaciones de la opción ***DROP*** en la cláusula ***ALTER TABLE***.

```
ALTER TABLE Clientes
DROP FechaNac;
```

Figura No. 16. Ejemplo de la instrucción DROP. Eliminar un atributo.

```
ALTER TABLE Clientes
DROP PRIMARY KEY;
```

Figura No. 17. Ejemplo de la instrucción DROP. Eliminar una llave primaria.

```
ALTER TABLE Servicios
DROP FOREIGN KEY Fk_servicios_cliente;
```

Figura No. 18. Ejemplo de la instrucción DROP. Eliminar una llave foránea.

ACTIVIDAD DE APRENDIZAJE

Propósito: Utilizar correctamente las sentencias SQL para realizar modificaciones a la definición de cada tabla y la Base de datos en general.

Descripción:

Ic Codifica en el Gestor de Base de Datos seleccionado, el siguiente Modelo Relacional para el caso del Banco, donde se almacena la información de los clientes y los servicios que tiene cada uno de ellos

Aa

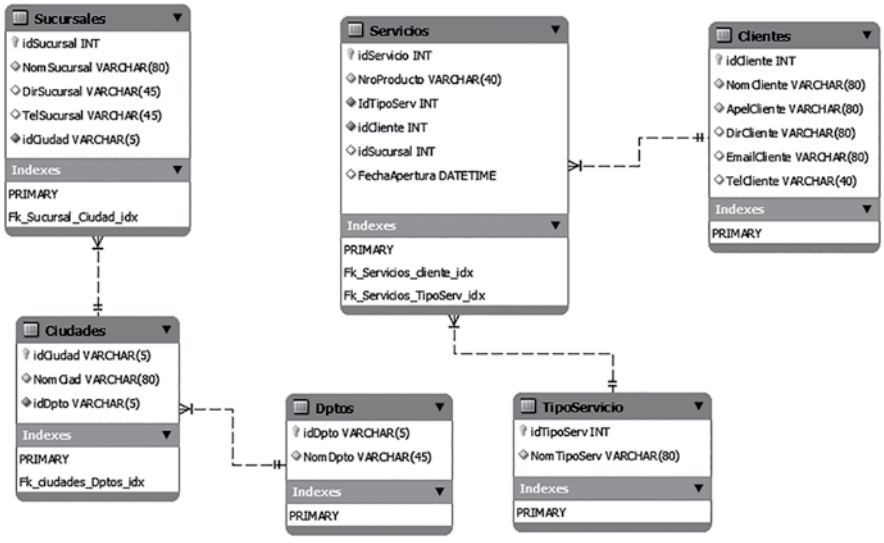


Gráfico No. 10. Modelo Relacional – Caso Banco

Iv Con el modelo del caso de Banco realice las siguientes acciones:

- Cree la base de datos tal como lo muestra el diseño.
- En la tabla cliente, agregue al atributo ***FechaNac*** (tipo ***DATE***) después del atributo ***ApelCliente***.
- En la tabla ***Servicios***, agregar la llave foránea denominada ***Fk_Servicios_ sucursales***.
- En la tabla ***Ciudades***, cambiar el nombre al atributo ***Nomciad*** por ***NomCiudad***.
- En la tabla ***TipoServicio***, cambiar el tipo de dato del atributo ***NomTipoServ***, de ***VARCHAR(80)*** por ***TEXT***.

Criterios de autoevaluación

- Utilizar las sentencias apropiadas para crear la Base de Datos y realizar las modificaciones a la definición con las sentencias apropiadas a cada solicitud.
- Participar activamente en el foro con la solución de la actividad y realiza aportes a los trabajos de sus compañeros.

WEBGRAFÍA

- Manual de referencia MySQL. Sintaxis de ALTER TABLE. Recuperado el 05 de octubre de 2014 de: *http://goo.gl/6eAedP*

ANOTACIONES

UNIDAD 1

SENTENCIAS DE DEFINICIÓN DE DATOS

SESIÓN 3

PRÁCTIQUEMOS SQL

SQL es un lenguaje completamente normalizado, el cual aplica a cualquier SGBD y se puede conectar con cualquier lenguaje de programación. Sin embargo, esto no quiere decir que las funciones de u SGBD siempre funcionan en otros. En esta sesión se desarrollará todo un caso estudio utilizando el SGBD de MySQL.

¿QUÉ SABES SOBRE...?

1. Describa para que sirve cada una de las siguientes sentencias SQL.

SENTENCIA	FUNCIÓN
CREATE SCHEMA	
DROP TABLE	
SELECT	
ALTER TABLE	

2. Plantee un ejemplo de la sentencia SQL para solucionar cada una de las siguientes situaciones.

SITUACIÓN	SOLUCIÓN
Cambiar solo el tipo de dato de una columna	

SITUACIÓN	SOLUCIÓN
Cambiar el nombre a una tabla	
Crear una base de datos	
Agregar una llave foránea a una tabla existente	
Crear una tabla	

1. ESTUDIO DE CASO A TRABAJAR

El caso a desarrollar en esta sesión, es sobre la gestión de una tienda de productos informáticos. Para este caso se plantea el Modelo Relacional y se codifica completamente. Se debe tener en cuenta los siguientes supuestos para el diseño:

a. De cada producto se desea guardar el código, descripción, precio, stock, categoría del producto.

b. De cada cliente se desea guardar su información personal, tales como: identificación, nombre, apellidos, dirección, email y teléfono.

c. Un cliente puede comprar varios productos en la tienda y un mismo producto puede ser comprado por diferentes clientes.

d. Cada vez que se compre un artículo quedará registrada la factura con la fecha en la que se ha comprado el artículo.

e. La tienda tiene varios proveedores que son los que suministran los productos.

f. Un mismo producto puede ser suministrado por varios proveedores.

g. De cada proveedor se desea guardar el NIT, razón social, nombre del representante, apellidos del representante, dirección, email, ciudad y teléfono de contacto.

El Modelo Relacional

Teniendo en cuenta los requerimientos planteados, el Modelo Relacional del caso se muestra en el siguiente gráfico.

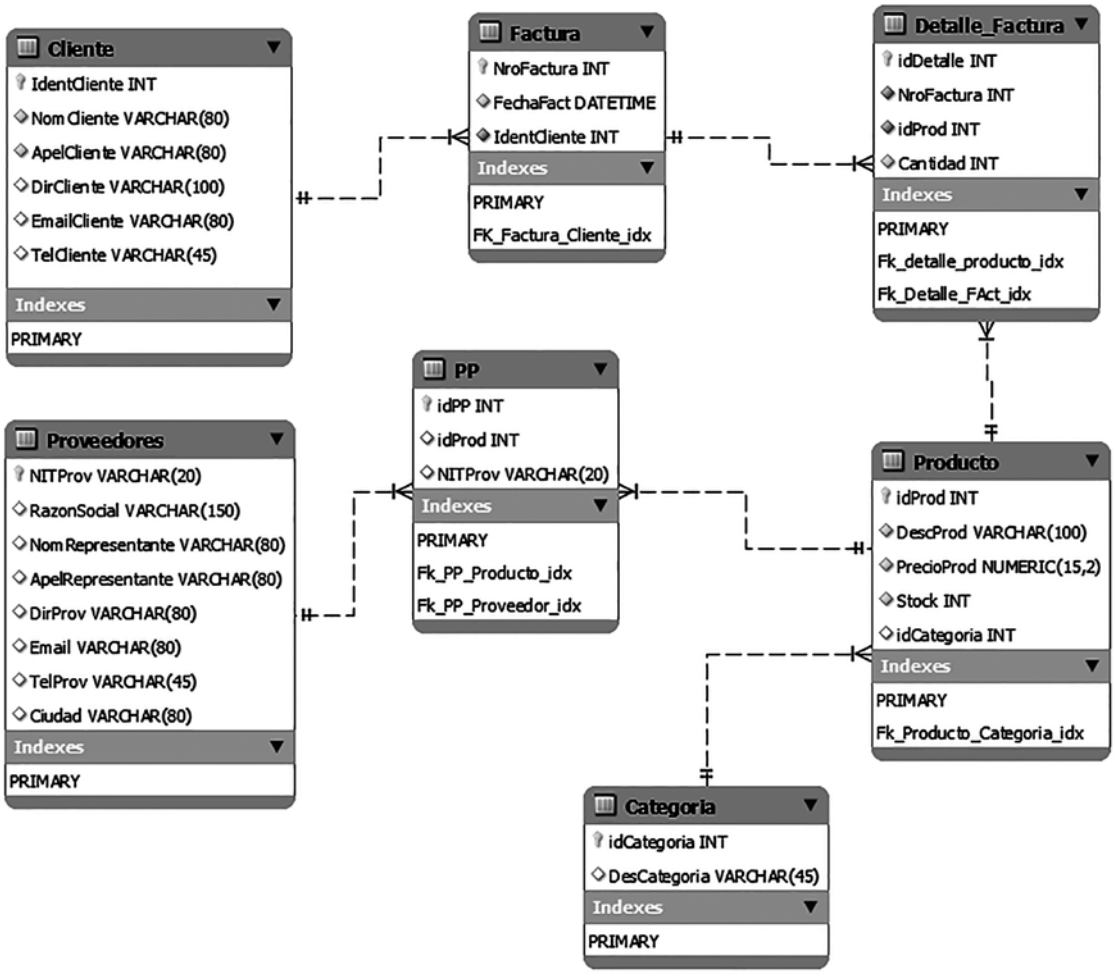


Gráfico No. 11. Modelo Relacional – Caso Tienda productos informáticos.

Codificando el modelo

Teniendo el modelo de la BD listo, ahora se procede a codificar. Cabe recordar que este proceso tiene un orden preciso para crear las tablas. Primero se debe crear las tablas que solo tienen llave primaria, y luego las que tienen, además, llave foránea. Esto es porque debe existir la entidad con la cual se relaciona.

En nuestro caso de la Tienda de productos informáticos, teniendo en cuenta las recomendaciones anteriores, el orden de creación es el siguiente:

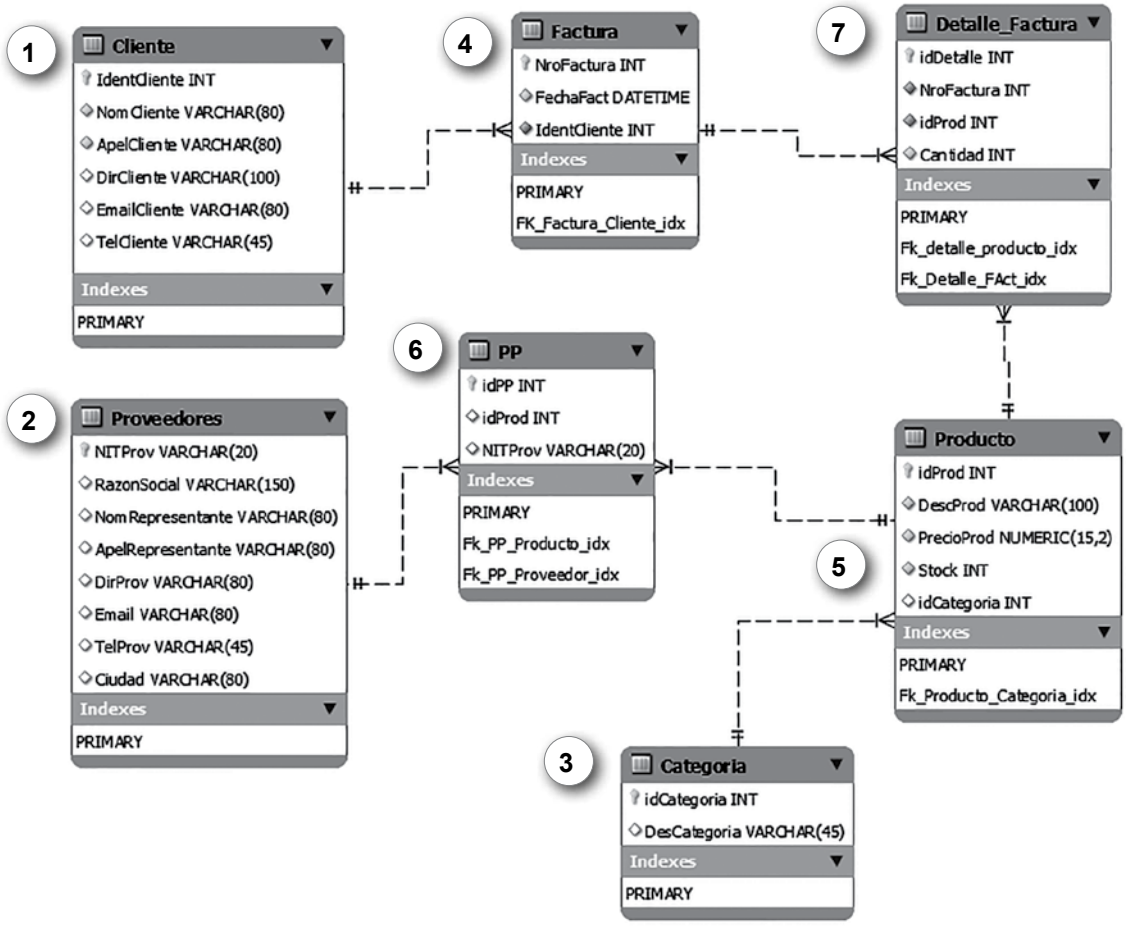


Gráfico No. 12. Modelo Relacional – Caso Tienda productos informáticos.

A continuación podrá encontrar la codificación del modelo.

```
CREATE SCHEMA Informatica DEFAULT CHARACTER SET utf8 COLLATE utf8_general_ci;
USE Informatica;
```

Figura No. 19. Creación de la base de datos.

```
CREATE TABLE Cliente (
  IdentCliente INT NOT NULL COMMENT 'Número de identificación del cliente',
  NomCliente VARCHAR(80) NOT NULL COMMENT 'Nombre del cliente',
  ApelCliente VARCHAR(80) NOT NULL COMMENT 'Apellido del cliente',
  DirCliente VARCHAR(100) NULL COMMENT 'Dirección del cliente',
  EmailCliente VARCHAR(80) NULL COMMENT 'Email del cliente',
  TelCliente VARCHAR(45) NULL COMMENT 'Teléfono de contacto del cliente.',
  CONSTRAINT Proveedores PRIMARY KEY (IdentCliente)
);
```

Figura No. 20. Creación de la tabla CLIENTE.

```
CREATE TABLE Proveedores (
  NITProv VARCHAR(20) NOT NULL COMMENT 'NIT del proveedor',
  RazonSocial VARCHAR(150) NULL COMMENT 'Razón social del proveedor',
  NomRepresentante VARCHAR(80) NULL COMMENT 'Nombre del representante',
  ApelRepresentante VARCHAR(80) NULL COMMENT 'Apellido del representante',
  DirProv VARCHAR(80) NULL COMMENT 'Dirección del proveedor',
  Email VARCHAR(80) NULL COMMENT 'Email del proveedor',
  TelProv VARCHAR(45) NULL COMMENT 'Teléfono de contacto del proveedor',
  Ciudad VARCHAR(80) NULL COMMENT 'Ciudad del proveedor',
  CONSTRAINT Proveedores PRIMARY KEY (NITProv)
);
```

Figura No. 21. Creación de la tabla PROVEEDORES.

```
CREATE TABLE Categoria (
  idCategoria INT NOT NULL AUTO_INCREMENT COMMENT 'Identificación de la
  categoría.',
  DesCategoria VARCHAR(45) NULL COMMENT 'Descripción de la categoría.',
  CONSTRAINT Categoria PRIMARY KEY (idCategoria)
);
```

Figura No. 22. Creación de la tabla CATEGORIA.

```
CREATE TABLE Factura (
  NroFactura INT NOT NULL AUTO_INCREMENT,
  FechaFact DATETIME NOT NULL,
  IdentCliente INT NOT NULL,
  CONSTRAINT Factura PRIMARY KEY (NroFactura),
  CONSTRAINT FK_Factura_Cliente FOREIGN KEY (IdentCliente)
    REFERENCES Cliente (IdentCliente)
);
```

Figura No. 23. Creación de la tabla FACTURA.

```
CREATE TABLE Producto (
  idProd INT NOT NULL AUTO_INCREMENT,
  DescProd VARCHAR(100) NOT NULL,
  PrecioProd DECIMAL(15,2) NOT NULL,
  Stock INT NOT NULL,
  idCategoria INT NULL,
  CONSTRAINT Producto PRIMARY KEY (idProd),
  CONSTRAINT Fk_Producto_Categoria FOREIGN KEY (idCategoria)
    REFERENCES Categoria (idCategoria)
   );
```

Figura No. 24. Creación de la tabla PRODUCTO.

```
CREATE TABLE PP (
  idPP INT NOT NULL AUTO_INCREMENT,
  idProd INT NULL,
  NITProv VARCHAR(20) NULL,
  CONSTRAINT Pk_PP PRIMARY KEY (idP),
  CONSTRAINT Fk_PP_Producto FOREIGN KEY (idProd)
    REFERENCES Producto (idProd),
  CONSTRAINT Fk_PP_Proveedor FOREIGN KEY (NITProv)
    REFERENCES Proveedores (NITProv)
   );
```

Figura No. 25. Creación de la tabla PP.

```
CREATE TABLE Detalle_Factura (
  idDetalle INT NOT NULL AUTO_INCREMENT,
  NroFactura INT NOT NULL,
  idProd INT NOT NULL,
  Cantidad INT NOT NULL,
  CONSTRAINT Pk_ Detalle_factura PRIMARY KEY (idDetalle),
  CONSTRAINT Fk_detalle_producto FOREIGN KEY (idProd)
    REFERENCES Producto (idProd),
  CONSTRAINT Fk_Detalle_Fact FOREIGN KEY (NroFactura)
    REFERENCES Factura (NroFactura)
);
```

Figura No. 26. Creación de la tabla DETALLE_FACTURA.

2. SENTENCIAS ÚTILES DE SQL

A continuación encontrarás sentencias de SQL que permiten realizar tareas frecuentes con mayor facilidad, y que en caso de estar trabajando solo en consola, te facilitará tareas como seleccionar la base de datos en la que va a trabajar, visualizar la estructura de una tabla o vista ya creada, así como listar las tablas que tiene una base de datos o las bases de datos que se encuentran en el SGBD.

Use

El comando USE le indica al SGBD que use la base de datos determinada como la base de datos por defecto para los comandos que se escriben a continuación. Se sigue utilizando esa base de datos hasta que finaliza la sesión o hasta que se realiza otro comando USE.

La sintaxis es la siguiente:

```
USE db_name;
```

db_name es el nombre de la base de datos con la cual se desea trabajar.

Describe

DESCRIBE provee información acerca de columnas en la tabla, sus campos, tipo de datos, extras, es decir, toda la información detallada de la tabla. Es una abreviación de ***SHOW COLUMNS FROM***. Este comando también permite visualizar la información de vistas.

La sintaxis de es la siguiente:

```
{DESCRIBE | DESC} tbl_name [col_name | wild];
```

col_name puede ser un nombre de columna y visualiza toda la información referente a ese campo de la tabla.

Show

SHOW proporciona información acerca de bases de datos, tablas, columnas o información de estado acerca del servidor.

La sintaxis es la siguiente:

```
SHOW [FULL] COLUMNS FROM tbl_name [FROM db_name] [LIKE 'pattern']
SHOW CREATE DATABASE db_name
SHOW CREATE TABLE tbl_name
SHOW DATABASES [LIKE 'pattern']
SHOW GRANTS FOR user
SHOW INDEX FROM tbl_name [FROM db_name]
SHOW PRIVILEGES
SHOW [FULL] PROCESSLIST
SHOW [GLOBAL | SESSION] STATUS [LIKE 'pattern']
SHOW TABLE STATUS [FROM db_name] [LIKE 'pattern']
SHOW [OPEN] TABLES [FROM db_name] [LIKE 'pattern']
SHOW TRIGGERS
```

Figura No. 27. Sintaxis SHOW.

A continuación encontrará ejemplos de aplicación de las sentencias útiles SQL.

```
USE informatica;
DESCRIBE detalle_factura;
DESCRIBE cliente EmailCliente;
DESCRIBE producto;
SHOW DATABASES;
SHOW CREATE DATABASE informatica;
SHOW CREATE TABLE factura;
```

Figura No. 28. Ejemplos de aplicación de las sentencias USE, DESCRIBE, SHOW.

ACTIVIDAD DE APRENDIZAJE

Propósito: Utilizar correctamente las sentencias SQL útiles que maneja el SGBD

Descripción:
Con el caso anteriormente desarrollado, aplique los siguientes comandos y describa el resultado que se obtiene con cada una.

COMANDO	DESCRIPCIÓN
SHOW CREATE DATABASE informatica;	
CREATE TABLE Categoria (idCategoria **INT NOT NULL AUTO_INCREMENT**, DesCategoria **VARCHAR(45) NULL**, **CONSTRAINT** Categoria **PRIMARY KEY** (idCategoria));	
DESCRIBE detalle_factura;	
USE informatica;	

Criterios de autoevaluación

- Identificar adecuadamente la función de cada una de las sentencias planteadas.
- Participar activamente en el foro con la solución de la actividad y realiza aportes a los trabajos de sus compañeros.

WEBGRAFÍA

- Manual de referencia MySQL. Sintaxis de SHOW. Recuperado el 08 de octubre de 2014 de: *http://goo.gl/bJGFm6*
- Manual de referencia MySQL. Sintaxis de USE. Recuperado el 08 de octubre de 2014 de: *http://goo.gl/kx7aVI*

ANOTACIONES

UNIDAD 2

SENTENCIAS DE MANIPULACIÓN DE DATOS (DML)

SESIÓN 4

FORMAS DE INSERTAR DATOS A LA BASE DE DATOS

Una vez codificada la base de datos, se procede a realizar la inserción de datos en la base de datos. En este momento empezamos a poner a prueba el diseño que se haya realizado. Se recomienda tener claro el diseño de la BD, ya que hay algunas recomendaciones para realizar este proceso. En esta sesión, trabajaremos las diferentes formas de insertar datos y algunas recomendaciones que se deben tener en cuenta para iniciar el proceso de ingresar datos a la BD.

¿QUÉ SABES SOBRE...?

1. ¿Qué es un archivo CVS? ¿Para qué sirve?

_____

_____

_____

_____

2. Complete la siguiente información sobre los tipos de datos que maneja el SGBD MySQL. Defina qué tipo de datos maneja (numérico, alfanumérico, fecha, otro), longitud máxima que maneja (si es necesario definirla).

Tipo de columna	Tipo de dato	Longitud máxima	Ejemplo
Long			
Decimal			
Datetime			
Text			

Year			
Binary			
Blob			
Char			

3. Describa en que consiste cada una de las siguientes reglas de integridad de una base de datos relacional.

Regla de integridad	Descripción
Regla de integridad de entidades	
Regla de integridad referencial	
Reglas de negocio	

1. INSERT

INSERT inserta nuevos registros en una tabla existente. Esta instrucción maneja tres opciones para realizar este proceso. Las opciones son:

INSERT… VALUES	Inserta filas con una lista de múltiples valores correspondientes a cada uno de los atributos de la tabla.
INSERT… SET	Inserta filas basándose en valores especificados explícitamente.
INSERT… SELECT	Inserta filas seleccionadas de otra tabla o tablas.

A continuación, conoceremos cómo se maneja cada una de las opciones anteriores. Recuerde que estamos trabajando el Modelo Relacional para el caso del Banco.

Nota:

Recuerde que la inserción de datos, al igual que la creación de tablas, tiene un orden específico. Se recomienda usar el mismo orden definido para la creación de las tablas, y de esta manera evitar errores de integridad referencial.

El orden de inserción de datos recomendado para el modelo del caso del banco, es el siguiente:

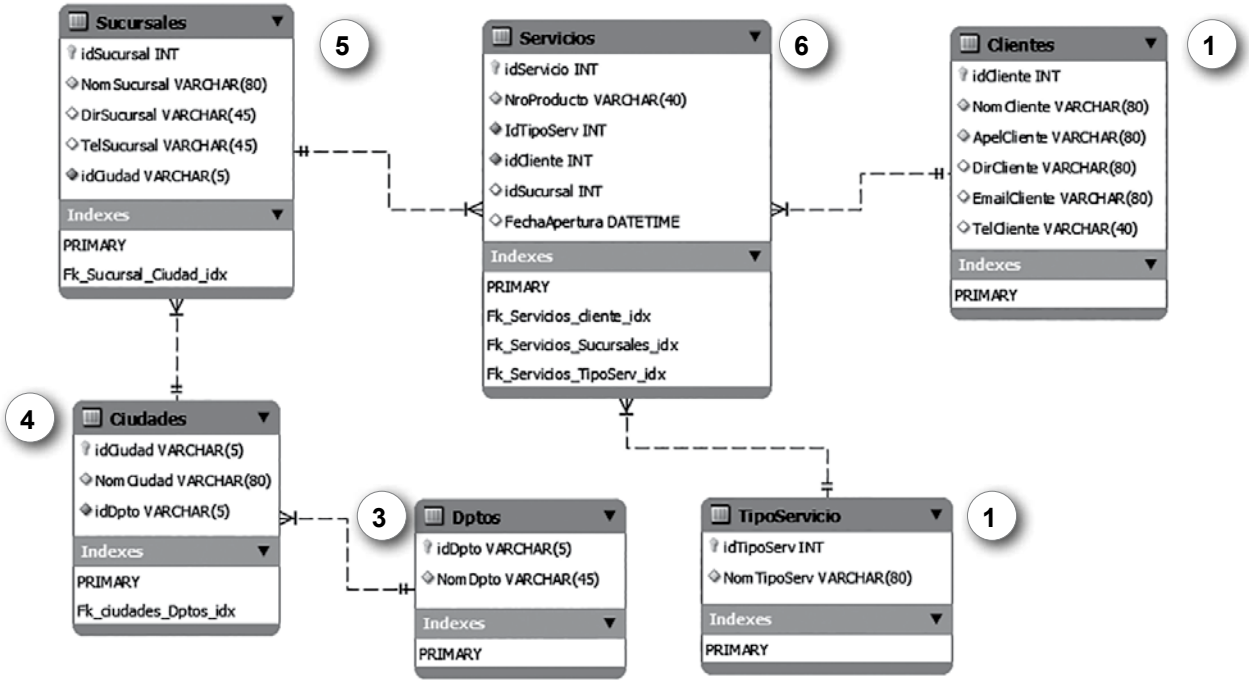


Gráfico No. 13. Modelo Relacional – Caso Banco. Orden de inserción de datos.

INSERT… VALUES

Inserta filas con una lista de múltiples valores correspondiente a cada uno de los atributos. La sintaxis es la siguiente:

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name [(col_name,...)]
    {VALUES | VALUE} ({expr | DEFAULT},...),(...),...
    [ ON DUPLICATE KEY UPDATE
      col_name=expr [, col_name=expr] ... ]
```

Figura No. 29. Sintaxis de la sentencia INSERT… VALUES

De acuerdo al Modelo Relacional para el caso del Banco, a continuación se muestra un ejemplo de INSERT… VALUES, listando los campos de la tabla.

```
INSERT INTO clientes (idCliente, NomCliente, ApelCliente, FechaNac,
DirCliente, EmailCliente, TelCliente)
VALUES (1020223567, 'Juan Camilo', 'Ortega Saenz', '21/09/1983', 'Calle 127 N°
17 - 44', 'jcortega@gmail.com', 3015689027);
```

Figura No. 30. Ejemplo de la sentencia INSERT… VALUES

La lista de atributos puede ser omitida, pero se debe conocer o tener claro el orden de los atributos de la tabla, tal como fueron creados.

Para insertar varios registros en la misma tabla, se puede manejar de la siguiente manera:

```
INSERT INTO clientes (idCliente, NomCliente, ApelCliente, FechaNac,
DirCliente, EmailCliente, TelCliente)
VALUES
(1020223567, 'Juan Camilo', 'Ortega Sáenz', '21/09/1983','Calle 127 N°17 -
44', 'jcortega@gmail.com', 3015689027),
(86920568, 'María Paz', 'Méndez Suarez', '09/15/1998','Cra 53 N°24 - 06',
'mpmendez@gmail.com', 3214477990),
(1234567890, 'Angélica', 'Pérez Arango', '27/12/1967','Cra 7 N°54 - 12',
'aperez@gmail.com', 318234567),
(1109765097, 'Mariana', 'Torres Ferrero', '01/02/1973','Calle 12 N°23 - 05',
'mtorres@gmail.com', 3056213456);
```

Figura No. 31. Insertar varios registros con INSERT… VALUES.

Nota:

Recuerde tener cuidado con la inserción de datos en cuyas tablas se tiene llaves foráneas. Los datos de estos atributos deben ser los que ya se han insertado en las tablas con las que se relaciona. En caso de escribir un dato diferente, generará un error de integridad referencial.

INSERT SET

Inserta filas basándose en valores especificados explícitamente. La sintaxis es la siguiente:

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name
    SET col_name={expr | DEFAULT}, ...
    [ ON DUPLICATE KEY UPDATE
      col_name=expr
        [, col_name=expr] ... ]
```

Figura No. 32. Sintaxis de la sentencia INSERT… SET.

A continuación se muestra un ejemplo de INSERT… SET.

```
INSERT INTO clientes
SET idCliente = 1020223567, NomCliente = 'Juan Camilo', ApelCliente='Ortega
Saenz', FechaNac = '21/09/1983', DirCliente='Calle 127 N° 17 - 44',
EmailCliente='jcortega@gmail.com', TelCliente = 3015689027;
```

Figura No. 33. Insertar registros con INSERT… SET.

INSERT… SELECT

Inserta filas seleccionadas de otra tabla o tablas. Hay veces que desea volcar un grupo de datos en una tabla y el contenido se encuentra en otra tabla, entonces lo que se tiene que hacer, es levantar el contenido de la tabla que contiene los datos a copiar he insertarlos en la nueva tabla. La sintaxis es la siguiente:

```
INSERT [LOW_PRIORITY | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name [(col_name,...)]
    SELECT ...
    [ ON DUPLICATE KEY UPDATE
      col_name=expr
        [, col_name=expr] ... ]
```

Figura No. 34. Sintaxis de la sentencia INSERT… SELECT.

A continuación se muestra un ejemplo de INSERT… SELECT. Para este ejemplo se utilizará la tabla temporal tmp_ClienteServicio que cuenta con la estructura que se ve en el gráfico No. 14.

```
INSERT INTO tmp_ClienteServicio
SELECT C.idCliente, NomCliente, ApelCliente, EmailCliente, TelCliente,
NroProducto, FechaApertura, idSucursal, NomTipoServ
FROM clientes C INNER JOIN servicios S
ON C.idCliente = S.idCliente
INNER JOIN tiposervicio TP
ON S.IdTipoServ = TP.IdTipoServ;
```

Figura No. 35. Insertar registros con INSERT… SELECT.

Gráfico No. 14. Modelo Relacional – Caso Banco. Tabla temporal tmp_Cliente

2. REPLACE

La sentencia REPLACE es una extensión de MySQL para el estándar SQL. Esta sentencia funciona como un INSERT, bajo las siguientes condiciones:

- Si el registro que desea insertar no existe, REPLACE inserta un nuevo registro.
- Si el registro que desea insertar ya existe, REPLACE elimina el registro viejo en primer lugar y, a continuación, insertar un nuevo registro.

La sintaxis es la siguiente:

```
REPLACE [LOW_PRIORITY | DELAYED]
    [INTO] tbl_name [(col_name,...)]
    {VALUES | VALUE} ({expr | DEFAULT},...),(...),...

OR:

REPLACE [LOW_PRIORITY | DELAYED]
    [INTO] tbl_name
    SET col_name={expr | DEFAULT}, ...

OR:

REPLACE [LOW_PRIORITY | DELAYED]
    [INTO] tbl_name [(col_name,...)]
    SELECT ...
```

Figura No. 36. Sintaxis de la sentencia REPLACE.

A continuación encontrará un ejemplo de la sentencia REPLACE.

```
REPLACE INTO clientes
SET idCliente = 1020223567, NomCliente = 'Juan Camilo', ApelCliente='Ortega
Saenz', FechaNac = '21/09/1983', DirCliente='Calle 127 N° 17 - 44',
EmailCliente='jcortega@gmail.com', TelCliente = 3015689027;
```

Figura No. 37. Insertar registros con REPLACE.

3. LOAD DATA INFILE

La sentencia LOAD DATA INFILE lee filas desde un fichero de texto a una tabla a gran velocidad. Se puede utilizar para leer archivos obtenidos de fuentes externas. Por ejemplo, muchos programas pueden exportar los datos en valores separados por comas (CSV) que tiene campos separados por comas y encerrados entre comillas dobles, con una línea inicial de nombres de columna. La sintaxis es la siguiente:

```
LOAD DATA [LOW_PRIORITY | CONCURRENT] [LOCAL] INFILE 'file_name.txt'
    [REPLACE | IGNORE]
    INTO TABLE tbl_name
    [FIELDS
        [TERMINATED BY '\t']
        [[OPTIONALLY] ENCLOSED BY '']
        [ESCAPED BY '\\' ]
    ]
    [LINES
        [STARTING BY '']
        [TERMINATED BY '\n']
    ]
    [IGNORE number LINES]
    [(col_name,...)]
```

Figura No. 38. Sintaxis de la sentencia LOAD DATA INFILE.

Para aplicar la sintaxis anterior, primero se debe tener listo el archivo con los datos en formato CVS. A continuación se explicará cómo construir el archivo desde la hoja de cálculo Excel. Se alistará el archivo para cargar los datos en la tabla clientes del Caso del Banco. Los pasos son los siguientes:

1. Se construye el archivo en Excel, donde se ubican los datos correspondientes al de los atributos al momento de crear la tabla como se ve en el siguiente gráfico. **Advertencia:** No se escriben los nombres de los atributos.

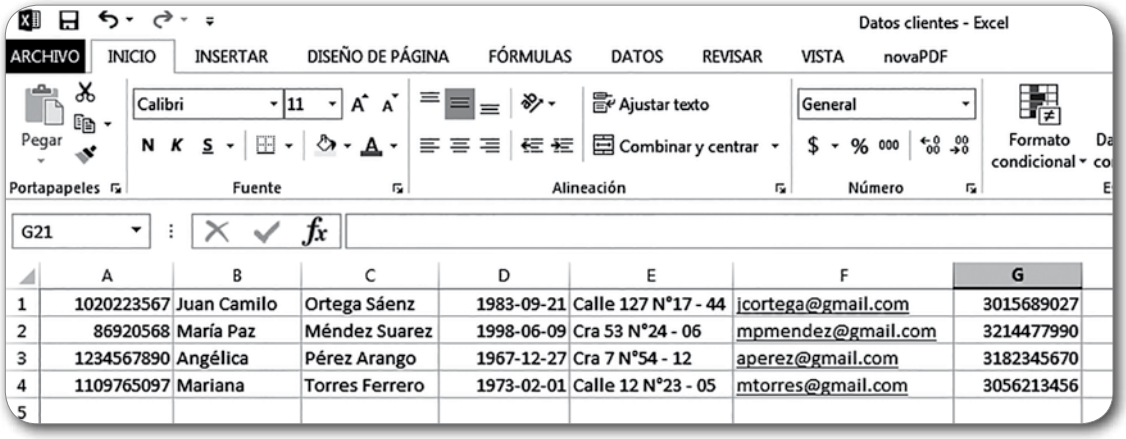


Gráfico No. 15. Datos de la tabla Clientes en Excel.

2. Se guarda el archivo con la opción ***Guardar como…*** y se selecciona el tipo de ***archivo CSV (delimitado por comas)***.

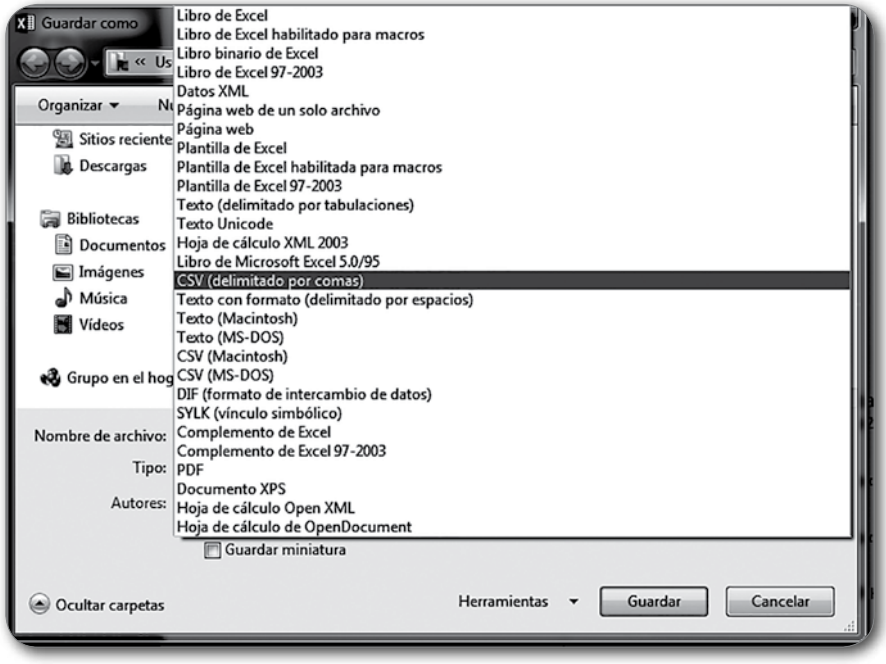


Gráfico No. 16. Tipo de archivo CSV (Delimitado por comas).

3. Se confirma que se desea conservar el nuevo formato y hace clic en ***Aceptar***.

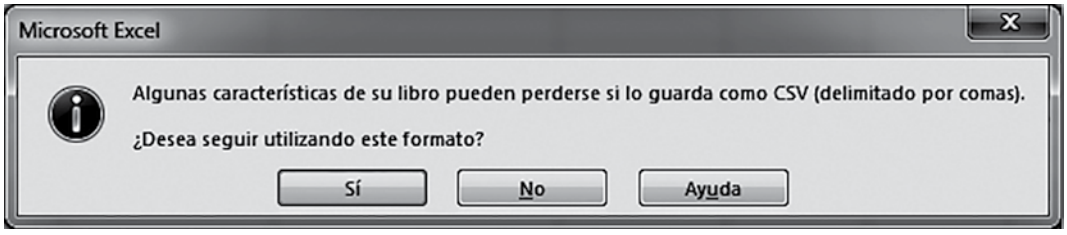


Gráfico No. 17. Confirmar el uso del nuevo formato.

4. En este momento ya se tiene listo el archivo para ser cargado a la base de datos.

Una vez se tiene el archivo listo con los datos, se aplica la siguiente instrucción para cargarlos a la base de datos.

```
LOAD DATA INFILE "C:\\DatosClientes.csv"
INTO TABLE banco.clientes
FIELDS TERMINATED BY ';';
```

Figura No. 39. Insertar registros con LOAD DATA INFILE.

En el siguiente gráfico se puede comprobar los datos cargados a la base de datos.

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

idCliente	NomCliente	ApelCliente	FechaNac	DirCliente	EmailCliente	TelCliente
86920568	María Paz	Méndez Suarez	1998-06-09	Cra 53 N°24 - 06	mpmendez@gmail.com	3214477990
1020223567	Juan Camilo	Ortega Sáenz	1983-09-21	Calle 127 N°17 - 44	jcortega@gmail.com	3015689027
1109765097	Mariana	Torres Ferrero	1973-02-01	Calle 12 N°23 - 05	mtorres@gmail.com	3056213456
1234567890	Angélica	Pérez Arango	1967-12-27	Cra 7 N°54 - 12	aperez@gmail.com	3182345670
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Gráfico No. 18. Datos en la base de datos.

ACTIVIDAD DE APRENDIZAJE

Propósito: Utilizar correctamente las sentencias de inserción de datos a la base de datos.

Descripción:
Para el caso del Banco, inserte los datos de toda la base de datos utilizando las diferentes sentencias explicadas en esta sesión. Tenga en cuenta lo siguiente:

1. Inserte mínimo 10 registros en cada tabla de la base de datos

2. Utilice cada una de las sentencias explicadas en la sesión y que sea posible aplicar al caso del Banco.

3. Recuerde el orden de inserción de datos que se debe tener en cuenta.

Criterios de autoevaluación

- Aplicar adecuadamente las sentencias de inserción de datos.
- Participar activamente en el foro con la solución de la actividad y realiza aportes a los trabajos de sus compañeros.

WEBGRAFÍA

- Manual de referencia MySQL. Sintaxis de LOAD DATA INFILE. Recuperado el 05 de noviembre de 2014 de: *http://goo.gl/gcsHxL*
- Manual de referencia MySQL. Sintaxis de REPLACE. Recuperado el 05 de noviembre de 2014 de: *http://goo.gl/R2gsQZ*

ANOTACIONES

UNIDAD 2

SENTENCIAS DE MANIPULACIÓN DE DATOS (DML)

SESIÓN 5

CONSULTAS CON FUNCIONES DE AGRUPAMIENTO

En SQL existen funciones que permiten contar registros, calcular sumas, promedios, desviación estándar, valores máximos, valores mínimos, entre otros. Estas funciones se denominan funciones de agrupamiento por que se aplican sobre conjuntos de registros, no con datos individuales. En esta sesión se trabajará las sentencias que permite realizar estas acciones, así como la sintaxis para las consultas básicas en SQL.

¿QUÉ SABES SOBRE...?

1. Describa las clausulas básicas para construir una consulta básica en SQL.

_____

_____

_____

_____

2. Explique la función de cada una de las siguientes sentencias SQL.

SENTENCIA	FUNCIÓN
ORDER BY	
HAVING	
BETWEEN	
LIKE	
GROUP BY	

3. Escriba cual es el resultado de cada una de las siguientes sentencias SQL.

SENTENCIA	DESCRIPCIÓN
SELECT * FROM TipoServicio;	
SELECT ApelCliente, NomCliente FROM clientes ORDER BY ApelCliente;	
SELECT NomSucursal, DirSucursal, TelSucursal FROM sucursales WHERE idCiudad=1;	

1. CONSULTAS BÁSICAS

Para ver los datos almacenados en las tablas de la base de datos, se usa la sentencia SELECT. Esta recupera las filas seleccionadas de las tablas indicadas.

La sentencia SELECT puede formarse de la combinación de las siguientes cláusulas:

CLÁUSULA	DESCRIPCIÓN
FROM	Permite especificar la tabla o tablas de las que se desea recuperar las filas.
ORDER BY	Permite organizar el resultado de la consulta de manera ASC o DESC.
WHERE	Permite definir los criterios de búsqueda sobre las filas generadas por la cláusula FROM.
GROUP BY	Permite agrupar las filas de la consulta, según las columnas seleccionadas.
HAVING	Permite definir las condiciones que deben cumplir los grupos resultantes de la cláusula GROUP BY.

La sintaxis básica de la sentencia SELECT es:

```
SELECT
      [ALL | DISTINCT | DISTINCTROW]
      col_name, [col_name2, col_name3, ...]
   [FROM table_references
      [WHERE where_definition]
      [GROUP BY \
        [ASC | DESC], ... [WITH ROLLUP]]
      [HAVING where_definition]
      [ORDER BY \
        [ASC | DESC],...]
```

Figura No. 40. Sintaxis de la sentencia SELECT.

Para construir las condiciones que debe cumplir los registros para ser mostrados, a veces, es necesario hacer uso de operadores y funciones de SQL. A continuación se listan los más utilizados.

OPERADOR O FUNCIÓN	DESCRIPCIÓN
BETWEEN	Especifica un rango determinado. ***Sintaxis:*** Col_name ***BETWEEN*** valor1 ***AND*** valor2
LIKE	Operador de comparación de cadena de caracteres.
IN	Hace que solo se muestren los registros cuyo campo uno de los valores indicados en la expresión o lista de valores.
IS NULL / IS NOT NULL	Establece como condición que un campo sea nulo o no.
IS TRUE / IS FALSE	Establece como condición que un campo tenga valor booleano verdadero o falso.
AS	Asigna un alias a un campo con expresión.
CAST()	Permite la conversión a un tipo de datos diferente. ***Sintaxis: CAST***(col_name) ***AS*** tipo_datos_nuevo
DATE_FORMAT()	Permite establecer un formato de fechas específico. ***Sintaxis: DATE_FORMAT***(Col_name,'%d/%m/%y')
NOW()	Devuelve la fecha y hora actuales del sistema.
DATEDIFF(Fecha1, Fecha2)	Permite calcular la diferencia de días entre fechas.
YEAR() / MONTH() / DAY()	Cada función, toma únicamente el año, mes o día de un campo tipo fecha.

A continuación encontrará algunos ejemplos de la sentencia SELECT combinado con algunas cláusulas, operadores y funciones.

El siguiente código, visualiza los clientes con cada servicio que tiene en el banco y el tiempo (cantidad de días) que tiene de haber solicitado dicho servicio.

```
SELECT C.idCliente, NomCliente, ApelCliente, S.NroProducto, NomTipoServ,
FechaApertura, DATEDIFF(NOW(), FechaApertura) AS Antigüedad
FROM clientes C INNER JOIN servicios S
ON C.idCliente = S.idCliente
INNER JOIN tiposervicio TS
ON S.IdTipoServ =TS.IdTipoServ;
```

Figura No. 41. Ejemplo de la sentencia SELECT.

El siguiente código, visualizar los productos con su respectiva sucursal de los productos que se abrieron en el primer semestre de 2014.

```
SELECT NroProducto, NomTipoServ, NomSucursal, NomCiudad
FROM tiposervicio TS INNER JOIN servicios S
ON TS.idTipoServ = S.IdTipoServ
INNER JOIN sucursales SU
ON S.idSucursal = SU.idSucursal
INNER JOIN ciudades C
ON SU.idCiudad = C.idCiudad
WHERE FechaApertura BETWEEN '2014/01/01' AND '2014/06/30';
```

Figura No. 42. Ejemplo de la sentencia SELECT con BETWEEN.

2. FUNCIONES DE AGRUPAMIENTO

SQL tiene funciones que permite realizar operaciones con los registros. Permite contar registros, calcular sumas, promedios, obtener valores máximos y mínimos, entre otros. Estas funciones se denominan ***Funciones de Agrupamiento*** por que trabajan con conjuntos de registros, no con datos individuales.

A continuación se describen las funciones de agrupamiento más utilizadas.

COUNT

Esta función permite averiguar la cantidad de registros devueltos por una consulta.

La sintaxis de la función es la siguiente:

```
COUNT (col_name) [AS Alias]
```

Figura No. 43. Sintaxis de COUNT.

Esta función se utiliza en el espacio donde se listan los campos a mostrar. Al trabajar la función ***COUNT***, se debe tener en cuenta que si utiliza ***COUNT(*),*** retorna la cantidad de registros de una tabla (incluyendo los que tienen valor ***null***). En cambio, si utiliza ***COUNT(col_name),*** retorna la cantidad de registros en los cuales el campo "*col_name*" no es nulo.

A continuación podrá encontrar un ejemplo del uso de la función ***COUNT()***.

```
SELECT COUNT(*) AS 'Cantidad Clientes'
FROM clientes
WHERE ApelCliente LIKE 'A%';
```

Figura No. 44. Ejemplo de la función COUNT.

El ejemplo anterior, permite contar la cantidad de clientes que tiene el banco, cuyo apellido inicia en la letra A.

SUM

La función **SUM** permite calcular la suma de un conjunto de valores o una expresión. La sintaxis de la función es la siguiente:

```
SUM (col_name) [AS Alias]
```

Figura No. 45. Sintaxis de SUM.

Se debe tener en cuenta que, si utiliza la función SUM en una instrucción SELECT y esta no encuentra filas coincidentes, la función SUM devuelve NULL, y no cero. La función SUM pasa por alto los valores NULL en el cálculo. Recuerde que esta función solo opera con campos numéricos.

A continuación podrá encontrar un ejemplo del uso de la función ***SUM()***.

```
SELECT SUM(TotalFact) AS 'Venta realizada'
FROM Ventas
WHERE FechaVenta BETWEEN '2014/07/01' AND '2014/07/31';
```

Figura No. 46. Ejemplo de la función SUM.

El ejemplo anterior, permite sumar el valor de la venta realizadas durante el mes de julio de 2014.

AVG

La función AVG es una función de agregado que permite calcular el valor medio de un conjunto de valores o una expresión. La sintaxis de la función AVG es el siguiente:

```
SELECT AVG(expression)
FROM tables
WHERE conditions;
```

Figura No. 47. Sintaxis de AVG.

Puede utilizar la cláusula DISTINCT dentro de la función AVG. Esta instrucción permite hallar el valor promedio del conjunto de valores, pero no tiene en cuenta los valores que se repiten.

A continuación podrá encontrar un ejemplo del uso de la función ***AVG()***.

```
SELECT AVG(expression)
FROM tables
[WHERE conditions];
```

Figura No. 48. Ejemplo de la función AVG.

MAX

La función MAX es utilizado para determinar el registro con un valor máximo de un conjunto de registros. La sintaxis de la función es el siguiente:

```
SELECT MAX(col_name) [AS Alias]
FROM tables
[WHERE conditions];
```

Figura No. 49. Sintaxis de MAX.

MySQL se puede optimizar funciones de agregado como MIN y MAX, mientras las columnas especificadas están indexados.

A continuación podrá encontrar un ejemplo del uso de la función ***MAX()***. Tenga en cuenta el gráfico 9, Modelo Relacional – Caso Tienda productos informáticos.

El siguiente ejercicio permite hallar el producto más costoso que tiene la tienda en su inventario.

```
SELECT DescProd, MAX(PrecioPod) As Precio
FROM Producto;
```

Figura No. 50. Ejemplo de la función MAX.

MIN

La función MIN se utiliza para averiguar el registro con valor mínimo de un conjunto de registros. La sintaxis de la función es el siguiente:

```
SELECT MIN(col_name) [AS Alias]
FROM tables
[WHERE conditions];
```

Figura No. 51. Sintaxis de MIN.

A continuación podrá encontrar un ejemplo del uso de la función ***MIN()***. Tenga en cuenta el gráfico 9, Modelo Relacional – Caso Tienda productos informáticos.

El siguiente ejercicio permite hallar el producto más barato o de menor precio que tiene la tienda en su inventario.

```
SELECT DescProd, MIN(PrecioPod) As Precio
FROM Producto;
```

Figura No. 52. Ejemplo de la función MIN.

3. GROUP BY – HAVING

La cláusula **GROUP BY** se utiliza con el comando **SELECT** para agrupar las filas en subgrupos por uno o más de los valores de las columnas o expresiones. Esta cláusula permite obtener subtotales. Una consulta con una cláusula **GROUP BY** se denomina ***consulta agrupada*** ya que agrupa los datos de la tabla origen y produce una única fila resumen por cada grupo formado. Las columnas indicadas en el **GROUP BY** se llaman ***columnas de agrupación***.

La sintaxis es la siguiente:

```
SELECT c1,c2,... cn, funcion_agregado(expresión)
FROM tabla
WHERE where_condiciones
GROUP BY c1, c2, ... cn;
```

Figura No. 53. Sintaxis de GROUP BY.

HAVING

La cláusula HAVING permite especificar condiciones de filtro para el grupo de filas o agregados. Esta cláusula se utiliza a menudo con la sentencia **GROUP BY**. Cuando se utiliza con esta sentencia, se puede aplicar una condición de filtro para las columnas que aparecen en la cláusula **GROUP BY**. Si el **GROUP BY** se omite en la cláusula, la cláusula

HAVING se comporta como la cláusula **WHERE**. Observe que la cláusula **HAVING** se aplica la condición a cada grupo de filas, mientras que la cláusula **WHERE** se aplica la condición de cada fila individual.

La sintaxis es la siguiente:

```
SELECT c1,c2,... cn, funcion_agregado(expresión)
FROM tabla
GROUP BY c1, c2, ... cn
HAVING condiciones_grupo;
```

Figura No. 54. Sintaxis de HAVING.

En el siguiente ejemplo, seguimos trabajando con la base de datos de la Tienda de productos informáticos, se pueden visualizar las facturas que han registrado ventas superiores a $200.000. Igualmente debe mostrar la cantidad de artículos vendidos en cada factura.

```
SELECT NroFactura,
       SUM(Cantidad) AS 'Cantidad Artículos',
       SUM(PrecioProd*Cantidad) AS total
FROM DetalleFactura DF
INNER JOIN Producto P
ON DF.idProd = F.idProd
GROUP BY NroFactura
HAVING total > 200000;
```

Figura No. 55. Ejemplo con las sentencias GROUP BY y HAVING.

ACTIVIDAD DE APRENDIZAJE

Propósito: Utilizar correctamente las sentencias de funciones de agrupamiento de acuerdo a los requerimientos de las consultas.

Descripción:
Teniendo en cuenta el caso de la tienda de productos informáticos, plantee una solución en SQL para cada uno de los siguientes informes solicitados:

1. Liste todas las facturas generadas en el mes de julio de 2014, donde se pueda observar la siguiente información: Número de factura, fecha, cantidad de artículos vendidos, total de la factura.

2. Genere un informe donde se pueda observar el promedio de ventas por mes durante el 2014.

3. Liste los clientes frecuentes del almacén. Los clientes frecuentes son aquellos que han comprado más de dos veces en el almacén.

Criterios de autoevaluación

- Aplicar adecuadamente las funciones de agrupamiento en cada informe.
- Participar activamente en el foro con la solución de la actividad y realiza aportes a los trabajos de sus compañeros.

WEBGRAFÍA

- Tutorial MySQL. Sintaxis de HAVING. Recuperado el 14 de noviembre de 2014 de: *http://goo.gl/lgTdjM*
- Tutorial MySQL. Sintaxis de GROUP BY. Recuperado el 14 de noviembre de 2014 de: *http://goo.gl/df9xZP*

ANOTACIONES

UNIDAD 2
SENTENCIAS DE MANIPULACIÓN DE DATOS (DML)

SESIÓN 6

CONSULTAS MULTITABLA

Las consultas básicas de SQL, tiene una limitación significativa, la cual solo permite traer información de la misma tabla. El verdadero potencial de SQL se alcanza cuando se combina el contenido de dos o más tablas. Supongamos que les solicitan generar un informe sobre los empleados de la empresa y la dependencia en la que trabajan; esta información se encuentra en dos tablas. Para extraer esta información, se tiene la opción de utilizar ***subconsultas*** o ***combinaciones***. En esta sesión se trabajará las diferentes combinaciones para extraer información de la base de datos.

¿QUÉ SABES SOBRE...?

1. Describa la información que arroja cada una de las siguientes sentencias.

SENTENCIA	FUNCIÓN
SELECT idCliente, NomCliente, ApelCliente **FROM** clientes **WHERE** ApelCliente LIKE 'a%';	
SELECT NomSucursales, COUNT(idServicio) AS 'Cantidad Servicios' **FROM** sucursales SU **INNER JOIN** Servicios SE **ON** SU.idSucursal = SE.idSucursal;	

SELECT NomSucursal, DirSucursal, TelSucursal, Nomciudad **FROM** Sucursales S **INNER JOIN** Ciudades C **ON** S.idCiudad = C.idCiudad;	
SELECT idCliente, NomCliente, ApelCliente, **COUNT**(idServicio) As 'Nro Servicios' **FROM** clientes C **INNER JOIN** servicios S **ON** C.idCliente = S.idCliente;	
SELECT NomSucursal, NomTipoServ, **COUNT**(idServicio) AS Cantidad **FROM** sucursales SU **INNER JOIN** Servicios SE **ON** SU.idSucursal = SE.idSucursal **INNER JOIN** TipoServicio TS **ON** SE.idTipoServ = TS.idTipoServ **WHERE** FechaApertura BETWEEN '2014/01/01' AND '2014/07/31' **GROUP BY** NomSucursal, NomTipoServ;	

1. JOIN

La sentencia SQL JOIN se utiliza para relacionar los datos de varias tablas. Esta cláusula permite listar los registros donde se encuentre coincidencia en ambas tablas en el atributo que las relaciona. Recuerde la relación entre las tablas es posible a través de las llaves foráneas.

La sintaxis es la siguiente:

```
SELECT col_name [, col_name2, ...]
FROM tbl_name1
[INNER | CROSS] JOIN tbl_name2
ON conditional_expr;
```

Figura No. 56. Sintaxis sentencia JOIN.

En la sentencia, se puede incluir los JOIN que sean necesarios, de acuerdo a la cantidad de tablas que desee combinar.

Para entender cómo funciona cada una de las opciones de JOIN, tenga en cuenta la siguiente información de algunas tablas del caso del banco.

Ciudades		
idCiudad	**NomCiudad**	**idDpto**
1	Paujil	3
2	Acacias	2
3	Anapoima	1
4	Zipaquirá	1

Dptos	
idDpto	**NomDpto**
1	Cundinamarca
2	Meta
3	Caquetá
4	Guajira

Tabla No. 1. Tablas Ciudades y Dptos con datos.

```
SELECT *
FROM Ciudades C
JOIN Dptos D
ON C.idDpto = D.idDpto;
```

Figura No. 57. Ejemplo de la sentencia JOIN

Al ejecutar la anterior sentencia, se obtiene el siguiente resultado.

idCiudad	NomCiudad	C.idDpto	D.idDpto	NomDpto
3	Anapoima	1	1	Cundinamarca
2	Acacias	1	1	Cundinamarca
4	Zipaquirá	2	2	Meta
1	Paujil	3	3	Caquetá

Tabla No. 2. Resultado de la sentencia – Figura No. 57.

2. INNER JOIN

Esta sentencia calcula el producto cruzado de todos los registros, es decir, cada registro en la tabla A es combinado con cada registro de la tabla B, pero sólo se muestran aquellos registros que satisfacen las condiciones que se han especificado. Este es el tipo de JOIN más utilizado, y por lo tanto es considerado el tipo de combinación predeterminado.

La sintaxis es la siguiente:

```
SELECT col_name [, col_name2, ...]
FROM tbl_name1
INNER JOIN tbl_name2
ON conditional_expr;
```

Figura No. 58. Sintaxis sentencia INNER JOIN.

Trabajando con los datos de la Tabla No 1, tenemos:

```
SELECT *
FROM Ciudades C
INNER JOIN Dptos D
ON C.idDpto = D.idDpto;
```

Figura No. 59. Ejemplo de la sentencia INNER JOIN

Al ejecutar la anterior sentencia, se obtiene el siguiente resultado.

idCiudad	NomCiudad	C.idDpto	D.idDpto	NomDpto
3	Anapoima	1	1	Cundinamarca
2	Acacias	1	1	Cundinamarca
4	Zipaquirá	2	2	Meta
1	Paujil	3	3	Caquetá

Tabla No. 3. Resultado de la sentencia – Figura No. 59

Como se puede observar, los resultados de las Tablas No. 2 y 3, los datos son iguales, lo cual indica que al utilizar JOIN o INNER JOIN se obtiene el mismo resultado.

3. CROSS JOIN

Esta sentencia muestra el producto cartesiano de todos los registros de las tablas que está cruzando. El código SQL para realizar este producto cartesiano enuncia las tablas que serán combinadas, pero no incluye algún predicado que filtre el resultado. El tamaño de un producto cartesiano es el número de filas en la primera tabla multiplicado por el número de filas de la segunda tabla.

La sintaxis es la siguiente:

```
SELECT col_name [, col_name2, ...]
FROM tbl_name1
CROSS JOIN tbl_name2;
```

Figura No. 60. Sintaxis sentencia CROSS JOIN.

Trabajando con los datos de la Tabla No 1, tenemos:

```
SELECT *
FROM Ciudades C
CROSS JOIN Dptos D;
```

Figura No. 61. Ejemplo de la sentencia CROSS JOIN

Al ejecutar la anterior sentencia, se obtiene el siguiente resultado.

idCiudad	NomCiudad	C.idDpto	D.idDpto	NomDpto
1	Paujil	3	1	Cundinamarca
2	Acacias	2	1	Cundinamarca
3	Anapoima	1	1	Cundinamarca
4	Zipaquirá	1	1	Cundinamarca
1	Paujil	3	2	Meta
2	Acacias	2	2	Meta
3	Anapoima	1	2	Meta
4	Zipaquirá	1	2	Meta
1	Paujil	3	3	Caquetá
2	Acacias	2	3	Caquetá
3	Anapoima	1	3	Caquetá
4	Zipaquirá	1	3	Caquetá
1	Paujil	3	4	Guajira
2	Acacias	2	4	Guajira
3	Anapoima	1	4	Guajira
4	Zipaquirá	1	4	Guajira

Tabla No. 4. Resultado de la sentencia – Figura No. 61.

4. LEFT JOIN

Esta sentencia permite combinar los registros de las tablas, pero el resultado contiene todos los registros de la tabla de la izquierda (la primera tabla que se menciona en la consulta), aun cuando no exista un registro correspondiente en la tabla de la derecha para uno de la izquierda.

La sentencia **LEFT JOIN** retorna la pareja de todos los valores de la tabla izquierda con los valores de la tabla de la derecha correspondientes, o retorna un valor nulo ***NULL*** en caso de no correspondencia.

La sintaxis es la siguiente:

```
SELECT col_name [, col_name2, ...]
FROM tbl_name1
LEFT JOIN tbl_name2
ON conditional_expr;
```

Figura No. 62. Sintaxis sentencia LEFT JOIN.

Trabajando con los datos de la Tabla No 1, tenemos:

```
SELECT *
FROM Ciudades C
LEFT JOIN Dptos D
ON C.idDpto = D.idDpto;
```

Figura No. 63. Ejemplo de la sentencia LEFT JOIN

Al ejecutar la anterior sentencia, se obtiene el siguiente resultado.

idCiudad	NomCiudad	C.idDpto	D.idDpto	NomDpto
3	Anapoima	1	1	Cundinamarca
2	Acacias	1	1	Cundinamarca
4	Zipaquirá	2	2	Meta
1	Paujil	3	3	Caquetá

Tabla No. 5. Resultado de la sentencia – Figura No. 63.

En este caso, en la tabla ciudades se encontraron correspondencia a todos los registros en la tabla Dptos.

5. RIGTH JOIN

Esta operación es inversa a la anterior; el resultado de esta operación contiene todos los registros de la tabla de la derecha (la segunda tabla que se menciona en la consulta), aun cuando no exista un registro correspondiente en la tabla de la izquierda para uno de la derecha.

La sentencia **RIGHT JOIN** retorna la pareja de todos los valores de la tabla derecha con los valores de la tabla de la izquierda correspondientes, o retorna un valor nulo ***NULL*** en caso de no correspondencia.

La sintaxis es la siguiente:

```
SELECT col_name [, col_name2, ...]
FROM tbl_name1
RIGHT JOIN tbl_name2
ON conditional_expr;
```

Figura No. 64. Sintaxis sentencia RIGHT JOIN.

Trabajando con los datos de la Tabla No 1, tenemos:

```
SELECT *
FROM Ciudades C
RIGHT JOIN Dptos D
ON C.idDpto = D.idDpto;
```

Figura No. 65. Ejemplo de la sentencia RIGHT JOIN

Al ejecutar la anterior sentencia, se obtiene el siguiente resultado.

idCiudad	NomCiudad	C.idDpto	D.idDpto	NomDpto
3	Anapoima	1	1	Cundinamarca
4	Zipaquirá	1	1	Cundinamarca
2	Acacias	2	2	Meta
1	Paujil	3	3	Caquetá
NULL	NULL	NULL	4	Guajira

Tabla No. 6. Resultado de la sentencia – Figura No. 65.

En este caso, en la tabla Dptos no encontró correspondencia a todos los registros en la tabla Ciudades, por lo tanto para el registro de Guajira, completa con NULL los demás atributos.

ACTIVIDAD DE APRENDIZAJE

Propósito: Interpretar y elegir adecuadamente las sentencias SQL ante los reportes solicitados por el usuario.

Descripción:

Teniendo en cuenta el caso de la tienda de productos informáticos, plantee una solución en SQL para cada uno de los siguientes informes solicitados:

1. Liste todas las facturas generadas en el mes de marzo de 2014, donde se pueda observar la siguiente información: Número de factura, fecha, Nombre y apellido del cliente, nombre de cada producto de la factura, con su precio unitario y la cantidad vendida.

2. Genere un informe donde se pueda observar los productos que están por agotarse y su respectivo proveedor. La alerta se genera con los productos que tiene 5 o menos productos disponibles.

3. Liste toda la información de los proveedores que surten los productos de la categoría 1.

Criterios de autoevaluación

- Aplicar adecuadamente las funciones de agrupamiento en cada informe.
- Participar activamente en el foro con la solución de la actividad y realiza aportes a los trabajos de sus compañeros.

WEBGRAFÍA

- INNER JOIN vs. CROSS APPLY. Recuperado el 19 de noviembre de 2014 de: *http://goo.gl/uihhBX*
- Uniones y composiciones en SQL. Recuperado el 19 de noviembre de 2014 de: *http://goo.gl/uf3gTH*

ANOTACIONES

UNIDAD 3
TRABAJANDO CON SUBCONSULTAS

SESIÓN 7

SUBCONSULTAS ANIDADAS

Como se vio en la sesión anterior, para extraer información que se encuentra almacenada en varias tablas, se puede realizar a través de combinaciones con la sentencia JOIN. Otra forma de extraer la información de múltiple tablas, es a través de las subconsultas, las cuales pueden darse de diferentes maneras. En esta sesión se trabajará las diferentes opciones que nos presenta las subconsultas para extraer información de las bases de datos.

¿QUÉ SABES SOBRE...?

1. Liste los operadores de comparación que permiten filtrar las filas seleccionadas, por medio de una condición.

_____

_____

_____

_____

2. Responda verdadero (v) o falso (f) a cada una de las siguientes afirmaciones:

   a. Se pueden construir condiciones múltiples basadas en los operadores lógicos AND, OR y NOT. ( )

   b. Las cláusulas WHERE y HAVING, tienen la misma función de filtrar la información que se va a mostrar en el informe solicitado. ( )

c..Cuando se desea limitar la búsqueda en atributos de caracteres, se utiliza '%' como comodín y representa cualquier cadena. ( )

d. La cláusula GROUP BY, permite agrupar registros que tengan la misma característica definida, pero esta característica solo la define un atributo. ( )

3. Construya la sentencia SQL para el siguiente informe solicitado. Teniendo en cuenta la base de datos del banco, liste todas las sucursales cuyo departamento inician con la letra A.

_____
_____
_____
_____
_____
_____
_____

1. SUBCONSULTAS

Una ***subconsulta*** o ***consulta interna*** o ***consulta anidada*** es una sentencia **SELECT** que aparece dentro de otra sentencia **SELECT** que se llama ***consulta principal***. La consulta interna se denomina ***consulta correlacionada***.

Las subconsultas se pueden encontrar en la lista de selección, en la cláusula **FROM**, **WHERE** o **HAVING**.

Las subconsultas se caracterizan por tener la misma sintaxis de una sentencia **SELECT**, exceptuando que esta va encerrada en paréntesis. Entre las restricciones que tienen las subconsultas se encuentran las siguientes: no puede contener las cláusulas **ORDER BY**, **BETWEEN** ni **UNION** y se pueden anidar máximo 16 subconsultas.

Subconsulta en la lista de selección

Cuando la ***subconsulta*** aparece en la lista de selección de la ***consulta principal***, en este caso la ***subconsulta***, no puede devolver varias filas ni varias columnas, de lo contrario se da un mensaje de error.

Para evitar este error, es necesario, dentro del cuerpo de una ***subconsulta***, hacer referencia al valor de una columna de la consulta principal, ese nombre de columna se denomina ***referencia externa***.

Teniendo en cuenta el caso de la tienda informática, en el siguiente ejemplo se puede visualizar la fecha en que cada cliente compró por primera vez en la tienda.

```
SELECT IdentCliente, NomCliente,
              (SELECT MIN(FechaFact) FROM factura  F WHERE
               F.IdentCliente = C.IdentCliente ) As 'Fecha'
FROM cliente C;
```

Figura No. 66. Ejemplo de subconsulta en la lista de selección

Como se puede observar en el código anterior, en el WHERE de la subconsulta, se relacionan los atributos de las tablas factura (de la subconsulta) y cliente (de la consulta principal).

Esta misma consulta con la sentencia JOIN quedaría de la siguiente manera:

```
SELECT C.IdentCliente, C.NomCliente, MIN(F.FechaFact) As 'Fecha'
FROM cliente C INNER JOIN factura F
ON F.IdentCliente = C.IdentCliente;
```

Figura No. 67. Ejemplo utilizando JOIN

Subconsulta en la cláusula FROM

En la cláusula FROM se puede escribir una sentencia SELECT encerrada entre paréntesis. Se puede decir que más que subconsulta sería una consulta en el FROM; se puede poner un nombre y manejarlo como una tabla.

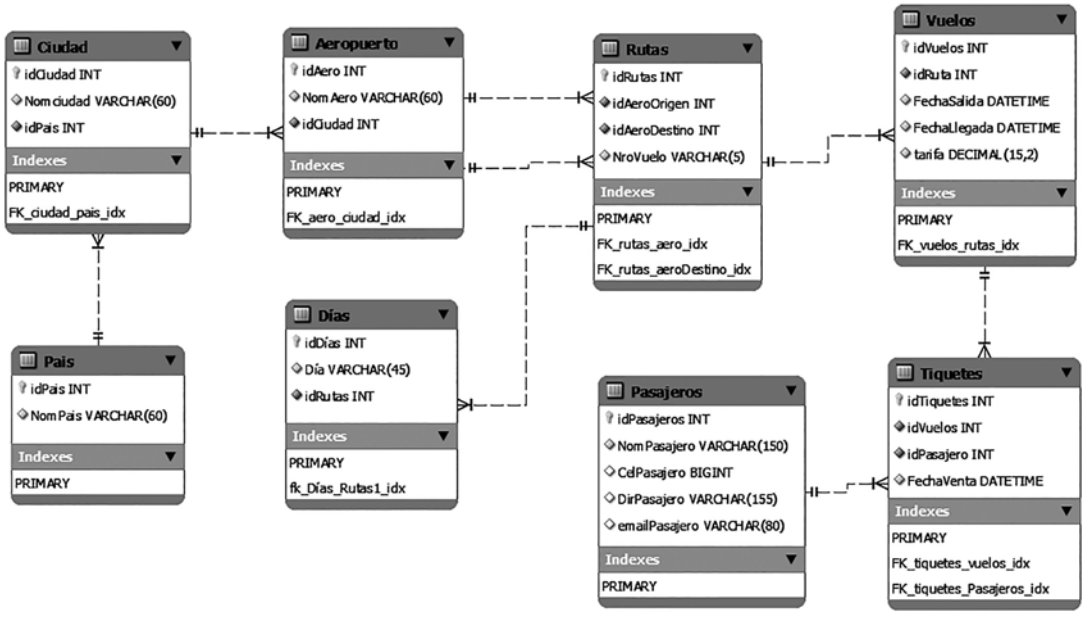


Gráfico No. 19. Modelo Relacional Caso Aerolínea

Teniendo en cuenta el Modelo Relacional de la aerolínea, se observa que entre aeropuerto y rutas hay doble relación; por lo tanto, para obtener el origen y destino de la misma tabla, en una consulta normal, trae el mismo origen y destino, para corregir este error, se trabaja una consulta en el FROM. Vemos el código:

```
SELECT Ru.idRutas, Ae.NomAero As Origen, DE.Destino, Ru.NroVuelo
FROM  (SELECT R.idRutas, A.NomAero AS Destino, R.NroVuelo
       FROM Rutas R INNER JOIN aeropuerto A
       ON R.idAeroDestino = A.idAero) AS DE
INNER JOIN Rutas Ru
ON DE.idRutas = Ru.idRutas
INNER JOIN aeropuerto Ae
ON Ru.idAeroOrigen = Ae.idAero;
```

Figura No. 68. Ejemplo Subconsulta en la cláusula FROM

Se debe tener en cuenta que en la "*nueva tabla*" del ***FROM***, se deja un atributo común para poder relacionarla. En este caso, el atributo ***idRutas***, es el atributo que permite relacionar la información con la tabla ***Rutas***.

Subconsulta en la cláusula WHERE y HAVING

Este tipo de subconsulta es el más utilizado. Las subconsultas en las cláusulas ***WHERE*** o ***HAVING***, se utilizan cuando los datos que se desean visualizar están en una tabla pero para seleccionar los registros de esa tabla, se necesita un dato que está en otra tabla.

Veamos el siguiente ejemplo:

```
SELECT DesCategoria
FROM categoria
WHERE idCategoria =  (SELECT idCategoria
                      FROM producto
                      HAVING MAX(PrecioProd));
```

Figura No. 69. Ejemplo Subconsulta en la cláusula WHERE y HAVING

La consulta anterior, permite listar la categoría donde su producto es el más costoso de la tienda informática.

2. OTRAS CLÁUSULAS PARA SUBCONSULTAS

Al trabajar subconsultas en las cláusulas WHERE y HAVING, hay diversos operandos para construirlas. Entre los operandos se encuentran los operadores de comparación (=, >, <, >=, <=, !=, <>), ANY, IN y EXIST.

ANY

La sentencia ANY compara el valor de la expresión con cada uno de los valores producidos por la subconsulta. La subconsulta debe devolver una única columna sino se produce un error.

```
SELECT DescProd
FROM producto
WHERE PrecioProd > ANY (SELECT MAX(PrecioProd)
                        FROM producto P
                        INNER JOIN categoria C
                        ON P.idCategoria=C.idCategoria
                        WHERE C.idCategoria =6);
```

Figura No. 70. Ejemplo Subconsulta en la cláusula ANY

La consulta anterior, permite listar los productos donde su precio es mayor al precio más alto de la categoría 6.

IN

La sentencia **IN**, evalúa si el valor de la expresión es uno de los valores incluidos en la lista de valores que produce la subconsulta. La subconsulta debe generar una única columna y las filas que sean. Si la subconsulta no produce ninguna fila, devuelve el valor de ***false***.

```
SELECT C.IdentCliente, C.NomCliente
FROM cliente C
WHERE C.IdentCliente IN (SELECT IdentCliente
                         FROM factura F
                         WHERE FechaFact BETWEEN '2014/01/01' AND
                                                 '2014/01/31');
```

Figura No. 71. Ejemplo Subconsulta en la cláusula IN

La consulta anterior, permite listar los clientes que compraron en el mes de enero de 2014. Recuerde que la restricción de la cláusula ***BETWEEN*** es en el ***WHERE*** de la ***consulta principal***.

EXISTS

La sentencia EXISTS (con la palabra reservada NOT opcional) se utiliza en comparaciones de verdad/falso; esta permite determinar si la subconsulta devuelve algún registro. En esta subconsulta se pueden tener varias columnas, ya que la sentencia se fija, no en los valores devueltos sino en si hay o no registros en la tabla resultado de la subconsulta.

Cuando se utiliza EXISTS, en la mayoría de los casos habrá que utilizar una referencia externa. Si no se utiliza una referencia externa, la subconsulta siempre devolverá todas las filas de la consulta principal o ninguna si la subconsulta no devuelve ninguna fila.

```
SELECT C.IdentCliente, C.NomCliente
FROM cliente C
WHERE EXISTS (SELECT * FROM factura F
              WHERE F.IdentCliente = C.IdentCliente);
```

Figura No. 72. Ejemplo Subconsulta en la cláusula EXISTS

La consulta anterior, se listan los clientes si han comprado en la tienda informática.

Se puede concluir, que un informe solicitado puede realizarse de varias maneras, solo debe tenerse en cuenta la cláusula de SQL que más se facilite en su manejo y sea más óptima en tiempo de ejecución con grandes registros.

ACTIVIDAD DE APRENDIZAJE

Propósito: Utilizar las diferentes opciones de subconsultas SQL en informes solicitados.

Descripción:
Teniendo en cuenta el caso de la aerolínea, plantee mínimo dos soluciones en SQL para cada uno de los siguientes informes solicitados utilizando subconsultas:

1. Liste todos los clientes que compraron tiquete en el vuelo número AV9778 Bogotá – San Andrés isla del día 16/diciembre/2014 a las 11:20 am.

2. Liste el aeropuerto del que salen el mayor número de rutas.

3. Liste todas las rutas (origen, destino, número vuelo) que salen los días lunes

Criterios de autoevaluación

- Aplicar adecuadamente las cláusulas que permiten generar subconsultas.
- Participar activamente en el foro con la solución de la actividad y realiza aportes a los trabajos de sus compañeros.

WEBGRAFÍA

- MySQL 5.5 Reference Manual. Subqueries with ANY, IN, or SOME. Recuperado el 20 de noviembre de 2014 de: *http://goo.gl/rnXOMX*
- MySQL 5.5 Reference Manual. Comparisons Using Subqueries. Recuperado el 20 de noviembre de 2014 de: *http://goo.gl/FoU4Gd*

- MySQL 5.5 Reference Manual. Subqueries with EXISTS or NOT EXISTS. Recuperado el 20 de noviembre de 2014 de: *http://goo.gl/bg2XT4*
- MySQL 5.5 Reference Manual. Optimizing Subqueries. Recuperado el 20 de noviembre de 2014 de: *http://goo.gl/On7PXA*

ANOTACIONES

UNIDAD 3
TRABAJANDO CON SUBCONSULTAS

SESIÓN 8

DIFERENCIANDO REGISTROS

Existen diversas formas de extraer la información de las bases de datos. Ya se ha trabajado las consultas simples, consultas multitabla, funciones de agregado y subconsultas. En esta sesión se sigue trabajando operadores que facilitan el proceso de extracción de información de la base de datos.

¿QUÉ SABES SOBRE...?

1. Relacione cada cláusula u operador con su función.

A.	GROUP BY	( )	Se utiliza para relacionar los datos de varias tablas.
B.	IN	( )	Permite renombrar los atributos, tablas o subconsultas.
C.	JOIN	( )	Se utiliza con el comando SELECT para agrupar las filas en subgrupos por uno o más de los valores de las columnas o expresiones.
D.	AS	( )	Evalúa si el valor de la expresión es uno de los valores incluidos en la lista de valores que produce la subconsulta.

2. Complete el siguiente cuadro realizando una descripción de la función de cada cláusula definida y planteando un ejemplo de aplicación.

CLÁUSULA	FUNCIÓN	EJEMPLO
BETWEEN		
HAVING		
INNER JOIN		
EXISTS		

A continuación se trabajarán los operadores UNION, ALL, DISTINCT, DISTINCTROW. Estas sentencias permiten personalizar las consultas de acuerdo a la información solicitada de la base de datos.

1. UNION

El operador UNION permite combinar dos o más conjuntos de resultados de varias tablas, o la misma tabla, en un único conjunto de resultados.

La sintaxis de es la siguiente:

```
SELECT column1, column2
FROM table_name
UNION [DISTINCT | ALL]
SELECT column1, column2
FROM table_name
UNION [DISTINCT | ALL]
...
```

Figura No. 73. Sintaxis del operador UNION

Para utilizar el operador UNION, se debe tener en cuenta las siguientes reglas:

- El número de columnas que aparece en todas las instrucciones SELECT deben ser iguales.

Las columnas debe estar en las posiciones correspondientes de cada instrucción SELECT y debe tener el mismo tipo de dato o por lo menos convertir a los tipos de datos correspondientes.

Por defecto, el operador UNION elimina las filas duplicadas del resultado, aunque no se use explícitamente el operador DISTINCT. En el caso de utilizar UNION ALL las filas duplicadas estarán en el resultado.

```
SELECT NomCliente, ApelCliente
FROM clientes C INNER JOIN Servicios se
ON C.idCliente = se.idCliente
WHERE se.idSucursal = 3
UNION
SELECT NomCliente, ApelCliente
FROM clientes C INNER JOIN Servicios se
ON C.idCliente = se.idCliente
WHERE se.idSucursal = 6;
```

Figura No. 74. Ejemplo del operador UNION

La consulta anterior, permite listar los clientes tienen servicios en la sucursal 3 y/o en la sucursal 6.

2. ALL

El operador ALL, adquiere una función de acuerdo donde se esté utilizando. Este operador se puede encontrar en la lista de selección o en el WHERE como subconsulta.

ALL en la lista de selección

El operador **ALL** en la lista de selección, permite revisar todos los atributos de la tabla. En ocasiones la palabra **ALL** es reemplazado por el símbolo de asterisco (*). Usualmente se usa para verificar la información de una tabla específica y se consulta todos los atributos y registros que existen en dicha tabla.

```
SELECT *
FROM Sucursales;
- - - - - - - - - -
SELECT ALL
FROM Sucursales;
```

Figura No. 75. Ejemplo del operador ALL en lista de selección

La consulta anterior, se lista toda la información de las sucursales del banco. Se muestra igualmente que el operador **ALL** es equivalente al símbolo asterisco (*).

ALL como subconsulta

El operador ALL evalúa la comparación con cada valor devuelto por la subconsulta. Esta subconsulta debe devolver una única columna sino se produce un error.

Cuando se realiza las comparaciones, se pueden obtener los siguientes resultados:

- Si todas las comparaciones individuales, producen un resultado verdadero, ALL devuelve el valor verdadero.
- Si la subconsulta no devuelve ningún valor, ALL devuelve el valor verdadero.
- Si la comparación es falso para algún valor de la columna, el resultado es falso.

```
SELECT DescProd
FROM producto
WHERE PrecioProd > ALL (SELECT AVG(PrecioProd)
                        FROM producto
                        GROUP BY IdCategoria);
```

Figura No. 76. Ejemplo del operador ALL en lista de selección

La consulta anterior, se lista todos los productos cuyo precio es mayor al precio promedio por categoría de producto.

3. DISTINCT

La cláusula **DISTINCT** se utiliza para eliminar los duplicados del conjunto de resultados. Esta cláusula sólo se puede utilizar con las instrucciones **SELECT**. En una tabla, una columna puede contener varios valores duplicados y a veces se desea una lista sin datos duplicados, es allí donde esta cláusula es muy útil.

La sintaxis es la siguiente:

```
SELECT DISTINCT columns
FROM table_name
WHERE where_conditions;
```

Figura No. 77. Sintaxis de la cláusula DISTINCT

Trabajando el caso de la tienda informática, se tiene la siguiente consulta donde se obtiene la lista de productos que se han vendido en la tienda.

```
SELECT DescProd
FROM producto P
INNER JOIN detalle_factura DF
ON P.idProd = DF.idProd;
```

Figura No. 78. Ejercicio con SELECT

El resultado obtenido es el siguiente:

DescProd
Disco externo Verbatim PhotoBank 80 GB
Disco externo Verbatim PhotoBank 80 GB
Pendrive Kingston DataTraveler ReadyFlash
Pendrive Kingston DataTraveler ReadyFlash
Acer Aspire 8920
Sony Vaio VGN-CR31S
Epson Stylus D120
HP Deskjet serie D4260

Tabla No. 7. Resultado de la sentencia – Figura No. 78.

Para eliminar los registros repetidos, se escribe la siguiente sentencia:

```
SELECT DISTINCT DescProd
FROM producto P
INNER JOIN detalle_factura DF
ON P.idProd = DF.idProd;
```

Figura No. 79. Ejercicio con SELECT

El resultado obtenido es el siguiente:

DescProd
Disco externo Verbatim PhotoBank 80 GB
Pendrive Kingston DataTraveler ReadyFlash
Acer Aspire 8920
Sony Vaio VGN-CR31S
Epson Stylus D120
HP Deskjet serie D4260

Tabla No. 8. Resultado de la sentencia – Figura No. 79.

4. DISTINCTROW

La cláusula **DISTINCTROW**, al igual que **DISTINCT**, se utiliza para eliminar los duplicados del conjunto de resultados. Se diferencia de **DISTINCT**, porque omite del resultado los registros repetidos (en su totalidad, no sólo en los campos seleccionados).

La sintaxis es la siguiente:

```
SELECT DISTINCTROW columns
FROM table_name
WHERE where_conditions;
```

Figura No. 80. Sintaxis de la cláusula DISTINCTROW

ACTIVIDAD DE APRENDIZAJE

Propósito: Utilizar las diferentes predicados de SQL para optimizar los resultados de los informes solicitados.

Descripción:

Teniendo en cuenta el caso de la aerolínea, plantee soluciones SQL para cada uno de los siguientes informes solicitados:

1. Liste todos los pasajeros que han comprado tiquetes en la aerolínea.
2. Liste los vuelos programados para el 2014/12/25 que salen de la ciudad de Cali.
3. Liste los vuelos superiores al promedio del valor de los tiquetes que salen de cada aeropuerto.

Criterios de autoevaluación

- Aplicar adecuadamente los operadores SQL que permiten generar consultas acertadas.
- Participar activamente en el foro con la solución de la actividad y realiza aportes a los trabajos de sus compañeros.

WEBGRAFÍA

- How to Use MySQL DISTINCT to Eliminate Duplicate Rows. Recuperado el 21 de noviembre de 2014 de: *http://goo.gl/Ai0xx7*
- MySQL 5.0 Reference Manual. DISTINCT Optimization. Recuperado el 21 de noviembre de 2014 de: *http://goo.gl/h52Pkg*
- Pildoritas de Bases de Datos: SQL. Predicados. Recuperado el 21 de noviembre de 2014 de: *http://goo.gl/ZmIB65*

ANOTACIONES

UNIDAD 3
TRABAJANDO CON SUBCONSULTAS

SESIÓN 9

ALGUNAS FUNCIONES ÚTILES

Cuando se trabaja con base de datos, al momento de generar informes, en ocasiones se necesita extraer una información específica de un atributo, por ejemplo, extraer el mes o año de una fecha, la cual está compuesta por el día, mes, año, hora, minutos o segundos. ¿Cómo extraer esa información de este dato? Para realizar este proceso, cada gestor de base de datos cuenta con una serie de funciones propias que facilitan varias tareas. En esta sesión se trabajará algunas de las funciones más trabajadas en los gestores de base de datos.

¿QUÉ SABES SOBRE...?

1. Mencione las funciones, en general, que conoce o ha utilizado del Gestor de Base de Datos MySQL.

_____

_____

_____

_____

2. Clasifique los siguientes tipos de datos de MySQL en: tipo texto, tipo binario, tipo numérico o tipo fecha-hora.

Char	_____	Blob	_____
Int	_____	Smallint	_____
Time	_____	Datetime	_____

Longblob	_____	Varchar	_____
Tinytext	_____	Float	_____
Double	_____	Text	_____
Year	_____	TimeStamp	_____

3. Defina el tipo de dato más adecuado y longitud para cada atributo de las siguientes tablas. Estas tablas pertenecen a un modelo sobre la información de las estaciones y rutas del metro. Tenga en cuenta los tipos de datos del SGBD MySQL (*http://goo.gl/8CJ1eG*).

LINEA	
IdLinea	_____
NomLinea	_____
TipoLinea	_____

ESTACIÓN	
idEstacion	_____
NomEstación	_____
NroOrden	_____
idLinea	_____

Las funciones propias de SQL, pueden ser utilizadas en varios puntos de los comandos SQL como: ORDER BY, HAVING, WHERE en las cláusulas SELECT, INSERT y UPDATE, o en la cláusula SET.

Las funciones más utilizadas se clasifican en estos grupos:

- Funciones numéricas
- Funciones de cadenas de caracteres
- Funciones de manejo de fechas

1. FUNCIONES NUMÉRICAS

Este grupo de funciones, realizan cálculos basados en valores de entrada proporcionados como parámetros a las funciones y devuelven valores numéricos.

A su vez estas funciones numéricas se pueden clasificar en dos grupos: las funciones matemáticas y funciones de redondeo.

A continuación se mostrará las funciones que manejan 3 Gestores de Base de Datos:

MySQL	ORACLE	SQL SERVER	DESCRIPCIÓN
ABS(n)	ABS(n)	ABS(n)	Calcula el valor absoluto del parámetro introducido.
MOD (m, n)	MOD(m, n)	m % n	Calcula el resto resultante de dividir m entre n.
POW(b, e) o POWER(b, e)	POWER(b, e)	POWER(b, e)	Devuelve el valor de elevar *b* a la *e*.
SQRT (p)	SQRT (p)	SQRT (p)	Devuelve la raíz cuadrada del valor *p*.
PI()	-	PI()	Devuelve el valor de la constante matemática PI.

Tabla No. 9. Funciones matemáticas

MySQL	ORACLE	SQL SERVER	DESCRIPCIÓN
ROUND(X,D)	ROUND(X,D)	ROUND(X,D)	Redondea el argumento *X* a *D* decimales
TRUNCATE (x, n)	TRUNC(x, n)	TRUNC(x, n)	Devuelve el número *x*, truncado a *n* decimales.
RAND()	dbms_random. value (min, max)	RAND()	Devuelve un valor aleatorio ente 0 y 1. Para Oracle, se especifica el rango en el cual se desean los números aleatorios.

Tabla No. 10. Funciones de redondeo

A continuación encontrará ejemplos de aplicación en el SGMS MySQL.

```
  SELECT Ru.idRutas, Ae.NomAero As Origen, DE.Destino, Ru.NumVuelo,
TRUNCATE((Tarifa-Tarifa*0.1),0) as 'Tarifa con Descuento'
FROM  (SELECT R.idRutas, A.NomAero AS Destino, R.NumVuelo
       FROM Rutas R INNER JOIN aeropuertos A
       ON R.idAeroDestino = A.idAero) AS DE
INNER JOIN Rutas Ru
ON DE.idRutas = Ru.idRutas
INNER JOIN aeropuertos Ae
ON Ru.idAeroOrigen = Ae.idAero
INNER JOIN vuelos v
ON v.IdRutas = ru.IdRutas;
```

Figura No. 81. Ejemplo de funciones numéricas MySQL

En el código anterior, se calcula el 10% de descuento a la tarifa de cada ruta de vuelo y el nuevo valor se muestra sin decimales.

2. FUNCIONES DE CADENA DE CARACTERES

Las funciones de cadena de caracteres permiten argumentos de tipo caracter y retornan caracteres o valores numéricos, de acuerdo a lo que sea solicitado con cada función. Estas funciones se pueden clasificar en dos grupos: funciones de manipulación y funciones de transformación.

A continuación se mostrará las funciones que manejan 3 Gestores de Base de Datos:

MySQL	ORACLE	SQL SERVER	DESCRIPCIÓN
CONCAT (x, y,…)	CONCAT (x, y)	CONCAT (x, y,…)	Concatena o une dos o más cadenas de caracteres.
CHAR_LENGTH (c)	LENGTH (c)	LEN (c)	Devuelve la longitud de una cadena de caracteres.
TRIM([{BOTH | LEADING | TRAILING} [remstr] FROM] c)	TRIM([{BOTH | LEADING | TRAILING} [remstr] FROM] c)	LTRIM(c) y RTRIM(c)	Quita los espacios o un carácter específico al inicio y/o final de la cadena de caracteres.
SUBSTR(t, n, m)	SUBSTR(t, n, m)	SUBSTRING(t, n, m)	Obtiene los *m* siguientes caracteres del texto a partir de la posición *n*.

Tabla No. 11. Funciones de manipulación

MySQL	ORACLE	SQL SERVER	DESCRIPCIÓN
UPPER (c)	UPPER (c)	UPPER (c)	Convierte a mayúsculas la cadena de caracteres.
LOWER (c)	LOWER (c)	LOWER (c)	Convierte a minúsculas la cadena de caracteres.

Tabla No. 12. Funciones de transformación

A continuación encontrará ejemplos de aplicación en el SGMS MySQL.

```
SELECT Ru.idRutas, CONCAT(c.NomCiudad,' - ',DE.Destino) as Ruta,
Ru.NumVuelo, TRUNCATE((Tarifa-Tarifa*0.1),0) as 'Tarifa con Descuento'
FROM  Destino DE
INNER JOIN Rutas Ru
ON DE.idRutas = Ru.idRutas
INNER JOIN aeropuertos Ae
ON Ru.idAeroOrigen = Ae.idAero
INNER JOIN vuelos v
ON v.IdRutas = ru.IdRutas
INNER JOIN ciudades c
ON c.IdCiudad = Ae.IdCiudad;
```

Figura No. 82. Ejemplo de funciones de cadena de caracteres MySQL

En el código anterior, se une los nombres de las ciudades origen y destino y se denomina ruta.

3. FUNCIONES DE MANEJO DE FECHAS

A continuación se trabajará las funciones más significativas de este tipo. Las funciones de manejo de fechas se ha clasificado en dos grupos: funciones para calcular fechas y funciones para obtener fechas.

A continuación se mostrará las funciones que manejan 3 Gestores de Base de Datos:

MySQL	ORACLE	SQL SERVER	DESCRIPCIÓN
DATEDIFF (enddate, startdate)	to_date('01/01/2015', 'dd/mm/yyyy') - to_date('01/01/2014', 'dd/mm/yyyy')	DATEDIFF (datepart , startdate , startdate)	Calcula la cantidad de días entre startdate – startdate.
MONTH (date)	EXTRACT(parte, date) *Parte:* year, month, day, hour, minute, second	MONTH (date)	Retorna el mes de la fecha.
DAYOFMONTH(date)		DAY (date)	Retorna el día del mes.
YEAR(date)		YEAR (date)	Retorna el año de la fecha.
DAYOFWEEK(date)	-	-	Retorna el día de la semana que corresponde a la fecha.

Tabla No. 13. Funciones para calcular fechas

MySQL	ORACLE	SQL SERVER	DESCRIPCIÓN
CURDATE ()	current_date o sysdate	-	Retorna la fecha actual
CURTIME ()	-	-	Retorna la hora actual
NOW()	systimestamp	SYSDATETIME ()	Retorna la fecha y hora actual.

Tabla No. 14. Funciones para obtener fechas

A continuación encontrará ejemplos de aplicación en el SGMS MySQL.

```
SELECT MONTH(FechaVenta), count(IdTiquete) AS Cantidad
FROM tiquetes
WHERE  YEAR(FechaVenta)=2014
GROUP BY MONTH(FechaVenta);
```

Figura No. 83. Ejemplo de funciones de manejo de fechas MySQL

En el código anterior, muestra un informe de la cantidad de tiquetes vendidos durante el año 2014. Esta lista muestra organizado mes a mes la cantidad de tiquetes vendidos.

ACTIVIDAD DE APRENDIZAJE

Propósito: Hacer uso apropiado de las funciones propias de SQL para la generación de consultas a nivel de base de datos.

Descripción:
Teniendo en cuenta el caso de la aerolínea, plantee soluciones SQL para cada uno de los siguientes informes solicitados:

1. Liste la cantidad de tiquetes vendidos en cada vuelo programado, durante el mes de octubre del 2014.

2. Consulte cual es el tiquete más viejo que se ha vendido en la ruta Cartagena-Bogotá y establezca la cantidad de días que se ha vendido dicho tiquete.

Criterios de autoevaluación

- Aplicar adecuadamente las funciones SQL que facilitan la generación de consultas.
- Participar activamente en el foro con la solución de la actividad y realiza aportes a los trabajos de sus compañeros.

WEBGRAFÍA

- Funciones de cadena (Transact-SQL). Recuperado el 24 de noviembre de 2014 de: *http://goo.gl/7VjzGv*
- Funciones integradas (Transact-SQL). Recuperado el 24 de noviembre de 2014 de: *http://goo.gl/mpVnI7*
- Oracle. Funciones String. Recuperado el 24 de noviembre de 2014 de: *http://goo.gl/8aa2d5*
- Oracle. Funciones matemáticas. Recuperado el 24 de noviembre de 2014 de: *http://goo.gl/AsQOBa*
- MySQL 5.5 Reference Manual. Functions and operators. Recuperado el 24 de noviembre de 2014 de: *http://goo.gl/9qSbYz*

ANOTACIONES

UNIDAD 4
OPTIMIZACIÓN DE PROCESOS EN EL SISTEMA GESTOR DE BASE DE DATOS (SGBD)

SESIÓN 10

OPTIMIZACIÓN DEL RENDIMIENTO

Actualmente, cuando se construye aplicaciones, independiente si orientada a web, de escritorio, entre otras, muchas veces se dedica la mayor parte del tiempo al desarrollo de la interfaz gráfica, y poco tiempo a la optimización de la base de datos. Seguramente en los inicios del uso del aplicativo, la base de datos no tendrá dificultades, pero cuando la base de datos crece, puede acarrear varios problemas como tiempos excesivos en las consultas. En esta sesión se trabajará sobre estrategias de optimización.

¿QUÉ SABES SOBRE...?

1. Defina que es el rendimiento de una base de datos. ¿Cómo se puede medir?

_____

_____

_____

_____

2. ¿Cómo identifica si su base de datos no está optimizada? ¿Qué problemas puede presentar?

_____

_____

_____

_____

3. Identifique 3 estrategias para optimizar una base de datos

_____

_____

_____

_____

1. ESTRATEGIAS PARA LA OPTIMIZACIÓN DEL RENDIMIENTO

El rendimiento de una base de datos, depende de varios factores como el diseño adecuado de la base de datos, creación correcta de las tablas, planteamiento correcto de consultas y ajustes en la configuración del gestor de base de datos.

La optimización en SQL pasa por tres componentes:

- Optimización del servidor SQL
- Optimización de la base de datos
- Optimización de las consultas

Optimización del servidor SQL

Es posible optimizar el funcionamiento del Gestor de Base de Datos MySQL, se puede realizar cambiando los valores de los parámetros del servidor.

Para mostrar los parámetros del servidor, se debe utilizar el comando:

```
show variables;
```

Este comando listará 328 variables del servidor.

Para mostrar el efecto de los parámetros sobre el servidor, se debe utilizar el comando:

```
show status;
```

Este comando listará 310 estados de los parámetros del servidor.

Hay herramientas de monitoreo que permiten ver los cambios en los parámetros en el servidor MySQL. Una herramienta es Mytopes un programa open source que permite la supervisión de bases de datos en MySQL. La aplicación tiene una interfaz de línea de comandos para monitorear en tiempo real los threads, las consultas por segundo, lista de procesos y el rendimiento de las bases de datos; además proporciona al administrador de la base de datos información para optimizar el servidor y manejar cargas pesadas.

Entre las variables que se pueden monitoreas, las más importantes son ***key_buffer_size*** y ***table_cache*** ya que son compartidas por todos los threads que corren sobre el servidor e influyen de manera considerable en el rendimiento.

Otras variables importantes son:

- ***key_buffer_size:*** memoria utilizada para las copias de seguridad de los índices MyISAM.
- ***table_cache:*** número de tablas que pueden ser abiertas simultáneamente.
- ***read_buffer_size:*** memoria utilizada para la copia de respaldo de los datos salidos de los full scan de las tablas.
- ***sort_buffer:*** memoria utilizada para la copia de respaldo de los datos de las tablas que serán ordenadas con un ORDER BY

Afinamiento de otros parámetros

Cuando se habla se sistemas operativos, el servidor MySQL obtiene un funcionamiento óptimo en SOLARIS, sin embargo, es posible optimizarlo en otros Sistemas Operativos cercano a su rendimiento ideal.

Otra recomendación, es verificar la velocidad de la red donde funciona el servidor. El uso de redes rápidas optimiza el tiempo de respuesta y optimiza la comunicación entre cliente/servidor.

Optimización de la base de datos

Uno de los aspectos fundamentales a tener en cuenta en las etapas de diseño y desarrollo de una aplicación es el del rendimiento futuro de la misma. Desde el momento en que se elige una configuración hardware y las aplicaciones de software para el desarrollo, el impacto sobre el rendimiento debe evaluarse cada vez que se realice cambios en el diseño y durante toda la fase de codificación. Es por esto, que es importante tener en cuenta las siguientes recomendaciones en cada una de las fases del desarrollo de la aplicación.

En la fase de diseño comienza a tenerse en cuenta las primeras apreciaciones sobre el rendimiento de la base de datos. Es importante conocer muy bien los datos que se desean almacenar, antes de decidir la estructura física de la base de datos o modificar una existente. Para evitar posibles problemas de rendimiento, se recomienda seguir las reglas asociadas al proceso de normalización de base de datos relacionales. Seguir estas recomendaciones permitirá tener un buen modelo de datos.

En ocasiones realizar la normalización completa al diseño, conlleva la excesiva división de tabla y por ende el uso excesivo de joins. Por lo anterior, se recomienda llevar el proceso de normalización hasta la tercera forma normal.

De igual manera, hay que cuidar el número de atributos definidos en cada tabla. Las tablas cargadas de descripciones (campos tipo texto), pueden influir negativamente en el rendimiento de la base de datos, en cambio los campos numéricos y tipo fecha no ocupan demasiado espacio.

Optimización de las consultas

MySQL permite analizar las consultas y conocer el tiempo y plan de ejecución. Esta información permite comprender lo que hace que las consultas sean lentas y optimizar la ejecución de éstas.

Para detectar las consultas lentas se debe:

1. Observar las consultas lentas durante su ejecución y los tiempos de respuesta anormales.

2. Hacer un benchmark: Probar las aplicaciones para ver qué componentes son los más lentos.

3. Verificar el Slow query log: es posible activar esta opción en MySQL configurando la variable --log-slow-queries

Una vez detectadas las consultas lentas, se ejecuta el comando EXPLAIN antes del SELECT. Este permite comprender la ejecución de la sentencia y por lo tanto conocer o intervenir para optimizar. Veamos la siguiente consulta:

```
EXPLAIN SELECT Ru.idRutas, CONCAT(c.NomCiudad,' - ',DE.Destino) as
Ruta, Ru.NumVuelo, TRUNCATE((Tarifa-Tarifa*0.1),0) as 'Tarifa con
Descuento'
FROM  Destino DE INNER JOIN Rutas Ru
ON DE.idRutas = Ru.idRutas
INNER JOIN aeropuertos Ae
ON Ru.idAeroOrigen = Ae.idAero
INNER JOIN vuelos v
ON v.IdRutas = ru.IdRutas
INNER JOIN ciudades c
ON c.IdCiudad = Ae.IdCiudad;
```

Figura No. 84. Ejemplo de sentencia SELECT con EXPLAIN

Al ejecutar esta sentencia se obtiene el siguiente resultado:

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	v	ALL	FK_VueloRuta	NULL	NULL	NULL	5	
1	SIMPLE	Ru	ALL	PRIMARY,FK_RutaAeroOrigen	NULL	NULL	NULL	5	Using where; Using join buffer
1	SIMPLE	r	eq_ref	PRIMARY,FK_RutaAeroDestino	PRIMARY	4	aerolinea.Ru.IdRutas	1	Using where
1	SIMPLE	a	eq_ref	PRIMARY,FK_AeroCiudad	PRIMARY	4	aerolinea.r.idAeroDestino	1	
1	SIMPLE	c	eq_ref	PRIMARY	PRIMARY	4	aerolinea.a.IdCiudad	1	
1	SIMPLE	Ae	eq_ref	PRIMARY,FK_AeroCiudad	PRIMARY	4	aerolinea.Ru.IdAeroOrigen	1	
1	SIMPLE	c	eq_ref	PRIMARY	PRIMARY	4	aerolinea.Ae.IdCiudad	1	

Gráfico No.20. Resultado de la ejecución del código de la figura 84

Para entender la información que se observa en el Gráfico No. 20, tenga en cuenta la siguiente tabla:

Columna	Definición
id	Identificador SELECT
select_type	Tipo de SELECT
table	Tabla de la fila de salida
type	Tipo de JOIN
possible_keys	Posibles índices para elegir
key	El índice que se ha elegido
key_len	Longitud de la clave
ref	Las columnas comparadas con el índice
rows	Estimación de las filas para ser examinado
Extra	Información adicional

Tabla No. 15. Columnas de la sentencia EXPLAIN

Nota:

Si desea profundizar sobre la información de estas columnas, lo invito a consultar el manual MySQL (*http://goo.gl/SoC10d*).

2. OPTIMIZAR LAS SENTENCIAS DML

Debido a que actualmente las aplicaciones acceden a la base de datos con múltiples operaciones DLM, la concurrencia es muy importante. Es por esto que es significativo conocer como acelerar las sentencias INSERT, UPDATE y DELETE.

INSERT

Para optimizar la velocidad de la sentencia INSERT, se recomienda combinar varias operaciones pequeñas en una sola operación. Lo mejor es realizar una única conexión, enviar los datos de muchas de las nuevas filas a la vez. Entre menos conexiones se realice a la base de datos, bajará el nivel de concurrencia a la base de datos.

Los métodos para mejorar la velocidad de esta sentencia, son los siguientes:

- Si va a insertar varias filas a la misma tabla, use la sentencia INSERT con múltiple lista de valores para insertar varias filas a la vez.

- Carga la tabla desde un archivo de texto. Para esto utilice la sentencia LOAD DATA INFILE que se trabajó en la sesión 4. Esta sentencia es 20 veces más rápida que la sentencia INSERT.

- Utilice el valor por defecto de los atributos. Solo ingrese datos cuando el valor sea diferente al valor por defecto.

UPDATE

La sentencia UPDATE se optimiza como una consulta de selección con la sobrecarga adicional de la escritura. La velocidad de la escritura de los datos depende de la cantidad de datos que se actualiza y el número de índices que se actualizan.

Por ejemplo, si se tiene una tabla con más de 25 millones de registros y necesita realizar una actualización de datos, se recomienda realizar un UPDATE de forma masiva. La sintaxis la encuentra a continuación.

```
SET ROWCOUNT 10000
WHILE EXISTS(SELECT * FROM TuTabla WHERE tucampo = 'x')
BEGIN
    UPDATE ...
    CONTINUE
END
```

Figura No. 85. Ejemplo UPDATE masivo

Esta sentencia le permite realizar actualizaciones de acuerdo a la cantidad de registros especificados, en ejemplo se definieron 10.000 registros. Con esto se puede realizar la actualización por partes y evitará tener mucho tiempo ocupada la base de datos con la misma sentencia.

DELETE

El tiempo para eliminar registros individuales es directamente proporcional al número de índices. Para eliminar los registros más rápidamente, puede aumentar el tamaño de la caché de los índices. Recuerde que esto se realiza modificando el parámetro del servidor que maneja el caché de los índices.

Para eliminar todos los registros de la tabla, se recomienda utilizar la sentencia TRUNCATE. No olvide que esta sentencia reinicia el contador que se tenga para una columna con propiedad AUTO_INCREMENT.

ACTIVIDAD DE APRENDIZAJE

Propósito: Analizar el rendimiento de una base de datos.

Descripción:
Seleccione una base de datos a la que tenga acceso y tenga mucho registro. Realice el siguiente análisis:

Ic 1. Plantee una actualización masiva para una de las tablas que considere contenga más registros. Analice la información allí y defina las condiciones de la actualización.

Pt 2. Analice la variable del sistema ***sort_buffer*** e indique cuál es su valor.

Criterios de autoevaluación

- Analizar adecuadamente la velocidad de la base de datos.
- Participar activamente en el foro con la solución de la actividad y realiza aportes a los trabajos de sus compañeros.

WEBGRAFÍA

- MySQL 5.5 Reference Manual. EXPLAIN Output Format. Recuperado el 25 de noviembre de 2014 de: *http://goo.gl/G6AaeE*
- Herramientas para la optimización de MySQL. Recuperado el 25 de noviembre de 2014 de: *http://goo.gl/Bec8kj*
- MySQL 5.5 Reference Manual. Optimization. Recuperado el 25 de noviembre de 2014 de: *http://goo.gl/Ol3Cyq*
- MySQL 5.5 Reference Manual. Server System Variables. Recuperado el 25 de noviembre de 2014 de: *http://goo.gl/aQqNux*

ANOTACIONES

UNIDAD 4

OPTIMIZACIÓN DE PROCESOS EN EL SISTEMA GESTOR DE BASE DE DATOS (SGBD)

SESIÓN 11

OPTIMIZACIÓN DE SENTENCIAS DE CONSULTA

La optimización de las consultas de la base de datos de cualquier aplicación, es de vital importancia. A la hora de medir el tiempo en el que carga la información en la aplicación, el tráfico del equipo se incrementa o el hardware tiene trabajo, se debe a un mal diseño de las consultas. En esta sesión se trabajará la optimización de las sentencias SELECT y las cláusulas que la acompañan.

¿QUÉ SABES SOBRE...?

1. Ten en cuenta el caso de la tienda informática y plantea la solución con sentencias SQL a cada uno de los casos planteados

Informe	Solución SQL
Liste la información de los pasajeros que compraron tiquetes en los vuelos 1 y 4.	
Liste la cantidad de vuelos que hay programados cada día de la semana.	
Liste las rutas programadas (vuelos que salen y llegan), de las ciudades de Bogotá y Cali.	

2. Si tuviera más de 12 millones de registros en las tablas de la base de datos, ¿Considera que las sentencias planteadas anteriormente se ejecutarían en poco tiempo? ¿Cuánto tiempo considera que demorarían? Justifique su respuesta.

_____

_____

_____

_____

_____

_____

Al verificar el rendimiento de las sentencias, cuanto más compleja es la configuración de los permisos, va afectar notablemente las mismas. Es por esto, que se recomienda ejecutar permisos (GRANT) simples, esto permite reducir la carga en la verificación de permisos en la ejecución de la sentencia. Otro problema para el rendimiento de la base de datos, es la forma de plantear las consultas. A continuación se listarán recomendaciones para identificar y mejorar las consultas lentas.

1. VELOCIDAD DE LAS CONSULTAS SELECT

Cuando se ha detectado una consulta SELECT… WHERE lenta y se desea hacerla más rápida, lo primero que hay que hacer es verificar si se puede agregar un índice. Todas las referencias entre tablas por lo general, deben tener índices. Para verificar esta información, puede utilizar el comando EXPLAIN y los índices que se usa la sentencia SELECT.

Para ayudar a optimizar las consultas MySQL, se utiliza la sentencia ANALYZE TABLE, este deba aplicarse a las tablas una vez que se ha cargado con los datos. Esta sentencia examina la distribución de las claves y las almacena en INFORMATION_SCHEMA. STATISTICS, reclama el espacio no utilizado y desfragmenta el archivo de datos.

La sentencia ANALYZE TABLE tiene la siguiente sintaxis:

```
ANALYZE [NO_WRITE_TO_BINLOG | LOCAL] TABLE
    tbl_name [, tbl_name] ...
```

Figura No. 86. Sintaxis de ANALYZE TABLE

La sentencia retorna un conjunto de resultados con las siguientes columnas:

Columna	Descripción
Table	El nombre de la tabla
Op	Siempre Analyze
Msg_type	Toma los valores de status, error, info, note o warning.
Msg_text	Información adicional

Tabla No. 16. Columnas de la sentencia ANALIZE TABLE

2. OPTIMIZANDO OTRAS SENTENCIAS

A continuación se enumerarán recomendaciones de optimización para diferentes cláusulas que se incluyen en las consultas.

WHERE

Para optimizar la cláusula WHERE, tenga en cuenta las siguientes recomendaciones:

- La cláusula WHERE es la más importante a la hora de la optimización.
- Nunca utilice SELECT *, porque el motor de base de datos debe leer primero la estructura de la tabla antes de ejecutar la sentencia.
- Especifique los campos que necesita mostrar, esto ahorrará espacio en la red.
- Sea cuidadoso con el uso del JOIN. Utilice siempre las llaves foráneas.
- No aplique JOIN en tablas que no necesita.
- Siempre trate de unir JOIN con los campos indexados.
- Si su consulta toma más de un segundo, considere la posibilidad de optimizarla.
- Plantee condiciones de consulta, en lo posible, a través de las llaves primarias, ya que estas se encuentran indexadas.
- Adicione índices en los campos que usa seguido para filtrar en la cláusula WHERE.
- Evite en lo posible el uso del operador NOT. Es más fácil buscar un valor exacto.
- En una columna indexada de tipo String, es mejor utilizar un comodín para buscar en ella en vez de la función Substr.
- Minimice el número de filas devueltas en tablas con gran cantidad de registros. Para esto utilice la cláusula LIMIT, el cual le permitirá limitar el número de registros visualizados.
- Los índices pierden velocidad cuando se usa el operador OR. Por lo tanto se recomienda utilizar la cláusula UNION en vez de OR. De esta manera se aprovecha al máximo los índices.

ORDER BY

Para optimizar la cláusula ORDER BY, tenga en cuenta las siguientes recomendaciones:

- Use la cláusula si solo si es necesario.
- Mantener el número de filas a ordenar al mínimo, filtrando los registros que son absolutamente necesarios.

- Mantener el número de columnas a ordenar al mínimo. Ordene solo las necesarias.
- Ordenar columnas con tipos de datos numéricos en lugar de tipos de datos carácter.
- Si se ordena por una columna constantemente, se debe considerar el crear un índice sobre esa columna para la mejora del rendimiento.

GROUP BY

Para optimizar la cláusula GROUP BY, tenga en cuenta las siguientes recomendaciones:

- No usar GROUP BY al menos sin hay una función de agregación en la consulta.
- Mantener el número de filas a devolver por la consulta tan pequeño como sea posible.
- Mantener el número de agrupaciones tan limitado como sea posible
- No agrupar columnas redundantes
- Si hay un JOIN en el mismo SELECT que tiene un GROUP BY, intente reescribir la consulta usado una SUBQUERY en lugar de usar un JOIN.
- Si se tiene que usar un JOIN, intente hacer el GROUP BY por columna desde la misma tabla que la columna o columnas sobre la cual la se usa la función.
- Considere el añadir un ORDER BY a la SELECT que ordena por la misma columna que el GROUP BY.

LIMIT

Para optimizar la cláusula LIMIT, tenga en cuenta las siguientes recomendaciones:

- Use la cláusula si solo necesita un número específico de registros.
- Utilice índices si necesita revisar todos los registros de la tabla.
- Utilice LIMIT con ORDER BY, de esta manera finaliza la clasificación cuando termina la cantidad de filas encontradas.
- Utilice LIMIT con DISCTICT, de esta manera se detiene tan pronto encuentre la cantidad de registros solicitados.

Recordemos las recomendaciones generales para optimizar las consultas SQL:

- Hay que optimizar dos tipos de instrucciones: las que consumen mucho tiempo en ejecutarse, o aquellas que no consumen mucho tiempo, pero que son ejecutadas muchas veces.

- El orden de ubicación las tablas en el FROM deberían ir en lo preferible de menor a mayor según el número de registros, de esta manera reducimos la cantidad de revisiones de registros que realiza el motor al unir las tablas a medida que se agregan.

- Use índices en las columnas que usen el WHERE y en las columnas que utilice en el ORDER BY.

- Los índices son muy interesantes en las búsquedas, pero ralentizan los insertados de datos. Hay que ser moderados con la creación de índices.

- Use la instrucción EXPLAIN para analizar los índices.

- Si quiere mostrar solo una línea como resultado de la consulta, debe especificar LIMIT 1. De esta forma el SGBD no continúa buscando.

- Defina tipos de datos que se adapten a los datos, no demasiado grandes.

- Utilice valores por defecto e inserte solo aquellos que no lo tienen. Los valores por defecto se insertan mucho más rápido.

- Generar y optimizar al máximo el plan de las vistas. Esto es importante porque el SQL de una vista, no se ejecuta mientras que la vista no es utilizada en una consulta.

- Utilizar siempre que sea posible las mismas consultas. La segunda vez que se ejecuta una consulta, se ahorrará mucho tiempo y optimización, así que se debe intentar utilizar las mismas consultas repetidas veces.

- Las consultas más utilizadas deben encapsularse en procedimientos almacenados. Esto es debido a que el procedimiento almacenado se compila y analiza una sola vez, mientras que una consulta ejecutada en la base de datos debe ser analizado, optimizado y compilado cada vez que se ejecuta.

- Los filtros de las consultas deben ser lo más específicos y concretos posibles. Es decir, es mucho más específico escribir WHERE campo = 'a' que WHERE campo LIKE '%a%'. se recomienda utilizar siempre consultas que filtren por la llave primaria u otros campos indexados.

- Evitar la condiciones IN (SELECT…) sustituyéndolas por JOINS. Cuando se utiliza un conjunto de valores en la cláusula IN, se traduce por una condición compuesta con el operador OR. Esto es lento, ya que por cada fila debe comprobar cada una de las condiciones simples. Suele ser mucho más rápido mantener una tabla con los valores que están dentro del IN, y hacer un JOIN normal.

A continuación encontrará un ejemplo de optimización.

```
SELECT Ru.idRutas, CONCAT(c.NomCiudad,' - ',DE.Destino) as Ruta,
Ru.NumVuelo, Tarifa
FROM  Destino DE INNER JOIN Rutas Ru
ON DE.idRutas = Ru.idRutas INNER JOIN aeropuertos Ae
ON Ru.idAeroOrigen = Ae.idAero INNER JOIN vuelos v
ON v.IdRutas = ru.IdRutas INNER JOIN ciudades c
ON c.IdCiudad = Ae.IdCiudad
WHERE Ru.idRutas IN (1,2,3);
```

Figura No. 87. Consulta SELECT para analizar

Ahora aplicando las recomendaciones de optimización quedaría de la siguiente manera: A continuación encontrará algunos ejemplos de optimización.

```
SELECT Ru.idRutas, CONCAT(c.NomCiudad,' - ',DE.Destino) as Ruta,
Ru.NumVuelo, Tarifa
FROM  Destino DE INNER JOIN Rutas Ru
ON DE.idRutas = Ru.idRutas INNER JOIN aeropuertos Ae
ON Ru.idAeroOrigen = Ae.idAero INNER JOIN vuelos v
ON v.IdRutas = ru.IdRutas INNER JOIN ciudades c
ON c.IdCiudad = Ae.IdCiudad
WHERE Ru.idRutas BETWEEN 1 AND 3;
```

Figura No. 88. Consulta SELECT optimizada

ACTIVIDAD DE APRENDIZAJE

Propósito: Interpretar y aplicar las recomendaciones para la optimización de consultas en cualquier SGBD.

Descripción:
Seleccione una base de datos del caso de la tienda informática y realice el siguiente análisis:

1. Mida el tiempo de ejecución de las siguientes consultas.

SENTENCIA	TIEMPO
SELECT DescProd FROM producto WHERE PrecioProd > ALL (SELECT AVG(PrecioProd) FROM producto GROUP BY IdCategoria);	

SELECT C.IdentCliente, C.NomCliente FROM cliente C WHERE C.IdentCliente IN (SELECT IdentCliente FROM factura F WHERE FechaFact BETWEEN '2014/11/01' AND '2014/11/30');	
SELECT DescProd FROM producto WHERE PrecioProd > ANY (SELECT MAX(PrecioProd) FROM producto P INNER JOIN categoria C ON P.idCategoria=C.idCategoria WHERE C.idCategoria =6 OR C.idCategoria =4);	
SELECT DISTINCTROW * FROM producto P INNER JOIN detalle_factura DF ON P.idProd = DF.idProd;	

Pt 2. ¿Qué recomendaciones aplicarías para optimizar consulta planteadas anteriormente? Escribe el código como debería utilizarse cada sentencia.

Criterios de autoevaluación

- Analizar adecuadamente la construcción de cada sentencia y plantear soluciones de optimización apropiadas.
- Participar activamente en el foro con la solución de la actividad y realiza aportes a los trabajos de sus compañeros.

WEBGRAFÍA

- MySQL 5.5 Reference Manual. Optimizing SELECT Statements. Recuperado el 26 de noviembre de 2014 de: *http://goo.gl/QxkhGh*
- El Blog de Sortea2. 10 consejos para optimizar MySQL. Recuperado el 26 de noviembre de 2014 de: *http://goo.gl/YcDO4R*
- MySQL 5.5 Reference Manual. How MySQL Optimizes WHERE Clauses. Recuperado el 26 de noviembre de 2014 de: *http://goo.gl/XCkVO1*
- ORACLE DBA - TUNNING. Tunning (optimización). Recuperado el 26 de noviembre de 2014 de: *http://goo.gl/X3Xwki*

ANOTACIONES

UNIDAD 4

OPTIMIZACIÓN DE PROCESOS EN EL SISTEMA GESTOR DE BASE DE DATOS (SGBD)

SESIÓN 12

ÍNDICES Y NUEVAS TECNOLOGÍAS DE BASE DE DATOS

La optimización de las consultas de la base de datos de cualquier aplicación, es de vital importancia. A la hora de medir el tiempo en el que carga la información en la aplicación, el tráfico del equipo se incrementa o el hardware tiene trabajo, se debe a un mal diseño de las consultas. En esta sesión se trabajará la optimización de las sentencias SELECT y las cláusulas que la acompañan.

¿QUÉ SABES SOBRE...?

1. ¿Describa para que sirven los índices en los libros? O ¿para qué sirve el directorio telefónico?

_____

_____

_____

_____

2. ¿Conoce otra tecnología diferente a la de base de datos relacionales para almacenar grandes volúmenes de información?

_____

_____

_____

_____

3. Investigue que Gestor de Base de Datos utilizan redes sociales como Facebook, Twitter, LinkedIn.

_____
_____
_____
_____

4. ¿Qué es base de datos NoSQL? Identifique 3 empresas colombianas o del estado que utilicen esta tecnología.

_____
_____
_____
_____

1. DEFINICIÓN Y TIPOS DE ÍNDICES

Los índices son objetos de base de datos, diseñados para mejorar el rendimiento de la consulta, están estructurados para facilitar una respuesta rápida de conjuntos de resultados. Estos objetos pueden asociarse a una o más columnas de tablas o vistas. Los índices, como los índices de los libros, sirven para agilizar las consultas a las tablas, evitando que el SGBD tenga que revisar todos los registros disponibles para devolver el resultado.

Normalmente, se crea todos los índices en una tabla en el momento de crearla con CREATE TABLE. Cuando se utiliza la sentencia CREATE INDEX, esta le permite añadir índices a tablas existentes.

Existen cinco tipos de índices que maneja el SGBD MySQL, estos son:

- **PRIMARY KEY:** Este índice se ha creado para generar consultas especialmente rápidas, debe ser único y no se admite el almacenamiento de NULL.

- **KEY o INDEX:** Son usados indistintamente por MySQL, permite crear índices sobre una columna, sobre varias columnas o sobre partes de una columna.

- **UNIQUE:** Este tipo de índice no permite el almacenamiento de datos repetidos.

- **FULLTEXT:** Permiten realizar búsquedas de palabras. Sólo pueden usarse sobre columnas CHAR, VARCHAR o TEXT

- **SPATIAL:** Este tipo de índices solo puede usarse sobre columnas de datos geométricos (spatial) y en el motor MyISAM.

Los índices FULLTEXT tiene limitaciones como:

- Solo busca por palabras completas.
- El índice no encontrará índices.
- No se indexan las palabras de menos de cuatro letras.
- No se indexan columnas que contengan menos de tres filas, ni palabras que aparezcan en la mitad o más de las filas.
- Las palabras separadas por guiones se cuentan como dos palabras.

En cambio, los índices ordinarios no tienen restricciones en cuanto a la existencia de valores idénticos o nulos. Una posibilidad interesante, si se desea crear un índice sobre columnas con tipo de datos CHAR y VARCHAR, es la de limitar el campo a indexar. Por ejemplo, cada entrada al atributo puede ser de hasta 40 caracteres y se puede indexar únicamente con los primeros 10 caracteres de cada uno.

2. CREACIÓN DE ÍNDICE

La sintaxis para la creación de índices es la siguiente:

```
CREATE [UNIQUE|FULLTEXT|SPATIAL] INDEX index_name
    [index_type]
    ON tbl_name (index_col_name,...)
```

Figura No. 89. Sintaxis CREATE INDEX

A continuación podrá observar ejemplos de creación de índices de los diferentes tipos.

```
CREATE UNIQUE INDEX index_NomAero
    ON Aeropuertos (nomAero);
```

Figura No. 90. Ejemplo creación de índice único

```
CREATE FULLTEXT INDEX index_NomPais
    ON Pais (NomPais);
```

Figura No. 91. Ejemplo creación de índice fulltext

```
CREATE INDEX index_NomPasajero
    ON Pasajeros (NomPasajero(5));
```

Figura No. 92. Ejemplo creación de índice ordinario

Entre las sentencias del SQL para la manipulación de índices, se tiene lo siguiente:

```
SHOW KEYS FROM tbl_name;  o
SHOW INDEX FROM tbl_name;
```

Figura No. 93. Sintaxis para mostrar los índices

Estas sentencias devuelven los siguientes campos:

CAMPO	DESCRIPCIÓN
Table	El nombre de la tabla.
Non_unique	0 si el índice no puede tener duplicados, 1 si puede.
Key_name	El nombre del índice.
Seq_in_index	El número de secuencia de columna del índice, empezando en 1.
Column_name	El nombre de columna.
Collation	El modo en que la columna se ordena en el índice. En MySQL, puede tener los valores 'A' (Ascending) o NULL (no ordenado).
Cardinality	El número de valores únicos en el índice. Este valor se actualizar mediante la ejecución de la o *myisamchk -a*. La cardinalidad se cuenta en base a las estadísticas almacenadas como enteros, de modo que no es necesario hacer aproximaciones para tablas pequeñas.
Sub_part	El número de caracteres indexados si la columna está indexada parcialmented. NULL si se indexa la columna completa.
Packed	Indica el modo en que se empaqueta la clave. NULL si no se empaqueta.
Null	Contiene *YES* si la columna puede contener *NULL*, '' si no.
Index_type	El método de índice usado (BTREE, FULLTEXT, HASH, RTREE).
Comment	Varios comentarios. Antes de MySQL 4.0.2 cuando se añadió la columna Index_type, *Comment* indica si un índice es *FULLTEXT*.

Tabla No. 17. Columnas de las sentencias SHOW KEYS FROM y SHOW INDEX FROM

```
DROP INDEX nombreIndice ON nombreTabla;  o
ALTER TABLE nombreTabla DROP INDEX nombreIndice;
```

Figura No. 94. Sintaxis para eliminar índices

```
ALTER TABLE nombreTabla DROP PRIMARY KEY;
```

Figura No. 95. Sintaxis para eliminar índices primarios

3. BASES DE DATOS NoSQL

El modelo de Base de Datos Relacional, es el modelo que se ha trabajado durante décadas, pero como es claro, la tecnología evoluciona a diario para brindar una mejor funcionalidad. En el caso de las Bases de Datos Relacionales, ha surgido un nuevo tipo de base de datos, conocido como NoSQL, el cual está llamando la atención y grandes empresas lo están implementando.

Durante décadas, la base de datos relacionales ha sido el modelo dominante para la gestión de base de datos. Pero, hoy en día las bases de datos NoSQL están ganando un espacio importante como un modelo alternativo de gestión de base de datos. Por ejemplo en Colombia, organismos del estado como el RUNT y la registraduría, están usando este nuevo enfoque de Bases de Datos, ya que los volúmenes de información que manejan, son bastante grande. A nivel mundial empresas como Facebook, Amazon, Twitter, LinkedIn y Google también usan bases de datos NoSQL.

¿QUE SON LAS BASES DE DATOS NoSLQ?

Las bases de datos NoSQL son sistemas de almacenamiento de información que no cumplen con el esquema entidad-relación, no imponen una estructura de datos en forma de tablas y relaciones entre ellas, por lo tanto son más flexibles, ya que permiten almacenar información en otros formatos como clave-valor, Mapeo de Columnas, Documentos o Grafos.

El término NoSQL aparece con la llegada de la web 2.0, hasta ese momento sólo subían contenido a la red aquellas empresas que tenían un portal, y era el administrador de la página el encargado se subir los contenidos, pero con la llegada de redes sociales como Facebook, Twitter o Youtube, cualquier usuario podía subir contenido, causando un crecimiento exponencial de los datos.

Es ahí cuando empiezan los problemas de gestión de la información de las bases de datos relacionales. Inicialmente utilizaron más máquinas, pero no fue suficiente. Es por esto que se empezó a trabajar en la crear sistemas pensados para un uso específico que con el paso del tiempo han dado lugar a soluciones robustas, apareciendo así el movimiento NoSQL.

Por lo tanto, cuando se habla de bases de datos NoSQL, se habla de estructuras que nos permiten almacenar información con miles de usuarios concurrentes y con millones de consultas diarias. El esquema que manejan las bases de datos NoSQL son formatos como clave–valor, mapeo de columnas o grafos.

CARACTERÍSTICAS

Las características de las bases de datos NoSQL son:

- Consistencia eventual
- Estructura distribuida
- Escalabilidad horizontal
- Tolerancia a fallos y redundancia
- No genera cuellos de botella
- Estructura dinámica

GESTORES DE BASES DE DATOS NoSQL

Cuando se habla de gestores de bases de datos NoSQL, la cosa se complica. Hoy día existen más de 100 sistemas de bases de datos NoSQL. Elegir uno de ellos puede ser complejo, ya que ninguno ha obtenido la reputación que sí han conseguido las bases de datos relacionales.

Las bases de datos NoSQL se pueden clasificar en cuatro tipos diferentes: orientadas a documentos, orientadas a columnas, de clave-valor y en grafo. A continuación se listará algunas características de estas y aplicaciones que cumplen con esas características.

Orientada a documentos

Son aquellas que gestionan datos semi estructurados. Es decir documentos. Estos datos son almacenados en algún formato estándar como puede ser XML, JSON o BSON.

En esta categoría se encuentra los sistemas:

Orientadas a columnas

Estas bases de datos permiten realizar consultas y agregaciones sobre grandes cantidades de datos. Funcionan de forma similar a las bases de datos relacionales, pero almacenando columnas de datos en lugar de registros.

En esta categoría se encuentra los sistemas:

De clave valor

Estas son las más sencillas de entender. Simplemente guardan registros que contienen una clave y su valor. Cuándo se quiere recuperar un dato, simplemente se busca por su clave y se recupera el valor.

En esta categoría se encuentra los sistemas:

En grafo

Basadas en la teoría de grafos utilizan nodos y aristas para representar los datos almacenados. Son muy útiles para guardar información en modelos con muchas relaciones, como redes y conexiones sociales.

En esta categoría se encuentra los sistemas:

ACTIVIDAD DE APRENDIZAJE

Propósito: Interpretar y aplicar adecuadamente el uso de los índices, así como las nuevas tecnologías de base de datos.

Descripción:

1. Trabaje con el caso de la tienda informática. Analice el modelo relacional de esta base de datos e identifique los índices que necesitaría para optimizar las consultas. Realice el código para la creación de los índices.

2. Realice un mapa conceptual donde se explique el concepto de las bases de datos NoSQL. Describa sus características y de ejemplos.

Criterios de autoevaluación:

- Analizar adecuadamente el modelo relacional y definir los índices apropiados para la optimización de la base de datos.
- Participar activamente en el foro con la solución de la actividad y realiza aportes a los trabajos de sus compañeros.

WEBGRAFÍA

- MySQL 5.5 Reference Manual. CREATE INDEX Syntax. Recuperado el 26 de noviembre de 2014 de: *http://goo.gl/QGRqnO*
- MySQL 5.5 Reference Manual. Examples of Fast Index Creation. Recuperado el 26 de noviembre de 2014 de: *http://goo.gl/7LlMqZ*
- SQL Server. CREATE INDEX (SQL Server Compact). Recuperado el 26 de noviembre de 2014 de: *http://goo.gl/ybJfF6*
- ORACLE DDL. CREATE INDEX, Recuperado el 26 de noviembre de 2014 de: *http://goo.gl/aJLhqG*
- Bases de datos NoSQL. Qué son y tipos que nos podemos encontrar. Recuperado el 26 de noviembre de 2014 de: *http://goo.gl/UCLzWT*

ANOTACIONES