Soluciones Ejecutive

ADMINISTRACIÓN DE

BASES DE DATOS

Microsoft SQL Server MySQL ORACLE Informix SOFTWARE

Indice

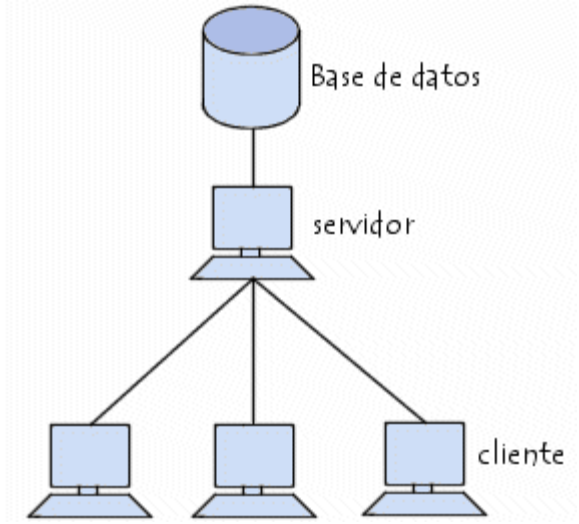
Objetivo

Brindar al participante los conocimientos y herramientas para instalar, organizar y manipular bases de datos en diferentes Sistemas manejadores de bases de datos; considerando todos los elementos internos y ajenos al servidor así como sus conexiones.

1. Conceptos básicos de Bases de Datos

¿Qué es una base de datos?

Una base de datos (cuya abreviatura es *BD*) es una entidad en la cual se pueden almacenar datos de manera estructurada, con la menor redundancia posible. Diferentes programas y diferentes usuarios deben poder utilizar estos datos. Por lo tanto, el concepto de base de datos generalmente está relacionado con el de red ya que se debe poder compartir esta información. De allí el término **base**. "Sistema de información" es el término general utilizado para la estructura global que incluye todos los mecanismos para compartir datos que se han instalado.



¿Por qué utilizar una base de datos?

Una base de datos proporciona a los usuarios el acceso a datos, que pueden visualizar, ingresar o actualizar, en concordancia con los derechos de acceso que se les hayan otorgado. Se convierte más útil a medida que la cantidad de datos almacenados crece.

Una base de datos puede ser local, es decir que puede utilizarla sólo un usuario en un equipo, o puede ser distribuida, es decir que la información se almacena en equipos remotos y se puede acceder a ella a través de una red.

La principal ventaja de utilizar bases de datos es que múltiples usuarios pueden acceder a ellas al mismo tiempo.

2. Control de usuarios

El objetivo de manejar un control de usuarios es que sólo puedan acceder a la información las personas y procesos autorizados y en la forma autorizada.

Técnicas:

• Identificación del usuario: El usuario tendrá que identificarse en el sistema para determinar si tiene los permisos correspondientes.

• Determinación de los accesos permitidos:

– Lista de autorizaciones (objeto y operaciones permitidas) por usuario: éste método permite determinar los objetos a los cuales el usuario puede entrar y que acciones puede llevar a cabo en cada uno.

– Niveles de autorización (menos flexible).- determina que tipo de acceso puede realizar el usuario, afectando a toda la base de datos.

• Gestión de autorizaciones transferibles: traspaso de autorizaciones de un usuario a otro:

Se pueden crear o brindar permisos transferibles a objetos comunes en donde el nivel de acceso sea bajo y no haya diferencia de jerarquía.

Es muy importante contar con una lista detallada de usuarios a los que se les dará el acceso a la base de datos, misma que identifique la posición y labores a realizar dentro del sistema. Al darle acceso a un usuario debemos identificarlo por código/clave de manera que podamos auditar el historial de cada usuario en cualquier momento y determinar si es el caso quien haya incurrido en faltas o mala operación de los datos.

Los dbms cuentan con herramientas que nos facilitan el control de usuarios dándonos información concisa de la actividad de cada usuario identificando cada acceso a los datos. En caso de no contar con dichas herramientas es imprescindible crear una capaza de realizar esta auditoria.

3. Conectividad a una base de datos

La conectividad es la interfaz para acceder a los datos en entornos heterogéneos de sistemas de gestión de bases de datos relacionales y no-relacionales. Esta puede llevarse a cabo mediante diversas formas; el primer agente que determina la forma en que se realizará la conectividad es el modo de trabajo es decir (web o Local). El segundo agente es el DBMS dado que se puede implementar de manera local o en red, con lo que debemos determinar cuales serán los parámetros de conexión.

Las interfaces de programación denotan el proceso de acceso y manipulación de los datos a una base de datos, partiendo de la aplicación. El siguiente esquema muestra 4 niveles o interfaces:

Niveles o Interfaces	
	Aplicación
	Interfaz de objetos de acceso a datos
	Interfaz de programación de aplicaciones (API)
	Bases de datos

La primera interfaz corresponde a la de **Aplicación**, la cual abarca y/o corresponde a cada uno de los programas clientes.

La Interfaz de Objetos de Acceso a Datos, se encuentra como punto medio entre las aplicaciones y las API's que llegan a ser necesarias para el acceso a las bases de datos. Entre las tecnologías que pertenecen a la Interfaz de Objetos de Acceso de Datos encontramos: DAO (Data Access Objects), ADO (ActiveX Data Objects), RDO (Remote Data Object), RDS (Remote Data Service) y MIDAS (Middle-tier Distributed Application Service). Su función es encapsular los componentes que se encuentran en la interfaz que corresponde a la de API's, con la finalidad de reducir el desarrollo de la aplicación y los costos de mantenimiento y deben situarse en todos los equipos que ejecuten la aplicación, ya que se encuentran casi de manera conjunta con la aplicación.

Por su parte, la **Interfaz de Programación de Aplicaciones** (Application Programming Interface, API), se encarga de mantener el diálogo con la base de datos, para poder llevar a cabo el acceso y manipulación de los datos. Algunos de los componentes que forman parte de esta interfaz son los siguientes: OLE DB, ODBC (Open Database Connectivity), JDBC (Java Data Base Connectivity), ISAPI (Internet Server Application Programming Interface) y CGI (Common Gateway Interface).

La función que tienen las API's, es la de ser una interfaz entre las aplicaciones y las bases de datos, llevando ésta tarea unas veces a través de los clientes y otros a través del servidor de base de datos. Esto quiere decir, que puede darse el caso de que el cliente conste de las tres primeras interfaces o niveles, o que se encuentren las dos últimas en el servidor. La interfaz correspondiente a la base de datos, es donde se encontrará el servidor y toda la información depositada en él.

Para poder accesar y manipular la información de una base de datos, es necesario llevar a cabo la instalación de ciertos API's o controladores, que son indispensables para efectuar la conectividad de los datos externos, y vincularlos a la aplicación para su correcta y adecuada utilización.

Las **API's** que se describen a continuación, son un claro ejemplo del proceso correspondiente a la conectividad de datos.

- **ODBC (Open Data Base Connectivity):** Esta tecnología proporciona una interfaz común para tener acceso a bases de datos SQL heterogéneas. ODBC está basado en SQL (Structured Query Language) como un estándar para tener acceso a datos. ODBC permite la conexión fácil desde varios lenguajes de programación y se utiliza mucho en el entorno Windows. Sobre ODBD Microsoft ha construido sus extensiones OLE DB y ADO. Los ODBC se pueden clasificar en 3 categorías:
  - Los ODBC's que permitan la realización de consultas y actualizaciones.
  - Los ODBC's que mediante ellos se pueda llegar a la creación de tablas en la base de datos.
  - Los ODBC's propios de los DBMS, los cuales se pueden llegar a manipular ciertas herramientas de administración.
- **CGI (Common Gateway Interface):** es una de las soluciones que se está utilizando más para la creación de interfaces Web/DBMS. Entre las ventajas de la programación CGI, destaca la sencillez, ya que es muy fácil de entender, además de ser un lenguaje de programación independiente, ya que los escritos CGI pueden elaborarse en varios lenguajes. También es un estándar para usarse en todos los servidores Web, y funcionar bajo una arquitectura independiente, ya que ha sido creado para trabajar con cualquier arquitectura de servidor Web. Como la aplicación CGI se encuentra funcionando de forma independiente, no pone en peligro al servidor, en cuanto al cumplimiento de todas las tareas que éste se encuentre realizando, o al acceso del estado interno del mismo. Pero el CGI presenta cierta desventaja en su eficiencia, debido al que el servidor Web tiene que cargar el programa CGI y conectar y desconectar con la base de datos cada vez que se recibe una requisición. Además, no existe un registro del estado del servidor, sino que todo hay que hacerlo manualmente.
- **ISAPI (Internet Server Application Programming Interface):** Es la interfaz propuesta por Microsoft como una alternativa más rápida que el CGI, y está incluida en el Servidor Microsoft Internet Information (IIS). Así como los escritos CGI, los programas escritos usando ISAPI habilitan un usuario remoto para ejecutar un programa, busca información dentro de una base de datos, o intercambia información como otro software localizado en el servidor. Los programas escritos usando la interfaz ISAPI son compilados como bibliotecas de enlace dinámico (DLL - Dinamic Link Library), ya que son cargados por el servidor Web cuando éste se inicia. Dichos programas se vuelven residentes en memoria, por lo que se ejecutan mucho más rápido que las aplicaciones CGI, debido a que requieren menos tiempo de uso de CPU al no iniciar procesos separados. Uno de los programas ISAPI más usados es el HTTPODBC.DLL que se usa para enviar y/o devolver información hacia y desde las bases de datos, a través de ODBC. Además, ISAPI permite realizar un procesamiento previo de la solicitud y uno posterior de la respuesta, con lo cual manipula la solicitud/respuesta HTTP. Los filtros ISAPI pueden utilizarse para aplicaciones tales como autenticación, acceso o apertura de sesión.
- **NSPAI**. es la API propuesta por Netscape para extender la funcionalidad de sus servidores.
- **DBI (PERL):** Perl es uno de los lenguajes más utilizados para programación en la Web y proporciona su propia interfaz de acceso a datos, llamada DBI (*DataBase Interface*). Es especialmente utilizado bajo plataformas Linux/Unix, solucionando las complejidades de ODBC en estos sistemas. DBI actúa como una abstracción para un conjunto de módulos *DBD* (*DataBase Driver*). Cada módulo DBD actúa como manejador de un sistema gestor de base de datos distinto. Existen módulos para prácticamente cualquier SGBD (Oracle, Informix, MySQL, etc.) y puentes hacia otras tecnologías como ADO, JDBC ...
- **JDBC (Java Data Base Connectivity)**: se trata del estándar para la conectividad entre el lenguaje Java y un amplio rango de sistemas gestores de bases de datos. Los JDBC pueden desenvolverse tanto en un nivel cliente, esto es, trabajando del lado de la aplicación, o en el servidor directamente relacionado con la base de datos. Cuando se

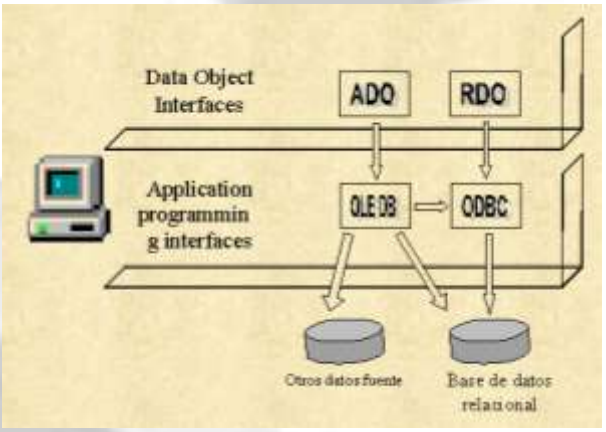
encuentre a nivel cliente, trabajará con la tecnología ODBC para acceso a los datos. Hay diversos tipos de controladores JDBC:

- El puente JDBC-OBDC: fue uno de los primeros controladores disponibles, implementa un enlace para utilizar un controlador ODBC desde Java. Con el tiempo han surgido controladores JDBC específicos para cada base de datos que mejoran el rendimiento del puente JDBC-ODBC.
- Controladores Java parcialmente nativos: usan tanto código Java como binario específico de cada plataforma.
- Controladores JDBC-Net de Java puro: son controladores escritos completamente en Java que entienden un protocolo de red estándar (HTTP, etc.) y permiten comunicarse con un servidor de acceso a bases de datos, que es el que finalmente provee el acceso al SGBD específico (posiblemente con ODBC).
- Controladores de protocolo nativo en Java puro: escritos en Java puro, utilizan el protocolo específico de la marca del SGBD.

- **SQL LINKS**: se trata de controladores que se encargan de realizar la comunicación remota entre la aplicación y los servidores remotos de bases de datos, permitiendo una comunicación casi directa y muy rápida. Los ha desarrollado la empresa Inprise y permiten conexiones con otros servidores de bases de datos como Interase, Oracle, Sybase, Informix, Microsoft SQL Server, etc.

Las 2 tecnologías más importantes de conectividad a la la base de datos son **ADO** y **JDBC**

Existen varios niveles o interfaces para lograr la comunicación o acceso a la base de datos a través de la aplicación. El siguiente esquema muestra 2 de los principales niveles, dentro de los cuales se encuentra ADO

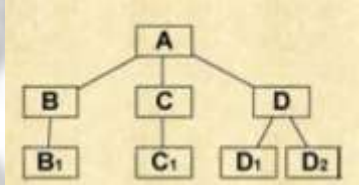


Por lo general, las interfaces de objetos de datos son más fáciles de usar que las APIS, aunque las APIs ofrecen más funcionalidades. ADO (*ActiveX Data Objects*) es la interfaz de objetos de datos para OLE DB, y RDO (*Remote Data Objects*) es la interfaz para el objeto ODBC.

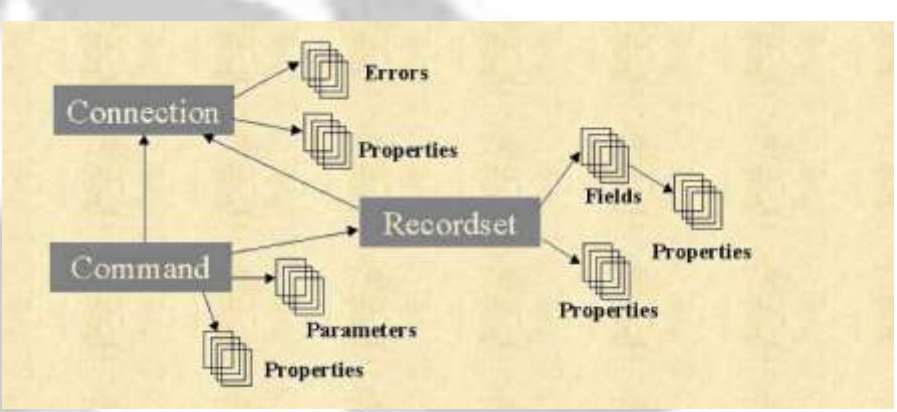
ADO encapsula el API OLE DB en un modelo objeto simple que reduce el desarrollo, mantenimiento y costo de la aplicación. Es muy fácil de usar, utiliza lenguajes de programación como Visual Basic, Java, C++, VBScript y JScript, puede accesar datos desde cualquier recurso OLE DB y además, es extensible. Es la interfaz utilizada por Microsoft.

El modelo ADO, basado en el modelo de objetos, define una jerarquía de objetos programables que pueden ser usados por desarrolladores de páginas Web para acceder a la información almacenada en una base de datos. Una jerarquía es un grupo de objetos relacionados que

trabajan juntos para un mismo propósito. Por ejemplo, en la siguiente figura, cada caja representa un objeto, y cada línea representa una asociación directa entre ellos.



ADO está compuesto de siete objetos, algunos de alto nivel como Connection, Command y Recordset, que pueden ser creados y eliminados por el usuario y otros con distintas funcionalidades como designar propiedades de conexión, definir sentencias y ejecutarlas, optimización de consultas, etc. Estos elementos se representan en la siguiente figura:



Cada uno de los objetos anteriores contiene una colección de objetos Property. El objeto Property permite a ADO mostrar dinámicamente las capacidades de un objeto específico.

ADO permite diseñar sitios web que pueden acceder repetidamente a la misma base de datos usando una misma búsqueda u otra similar. Se pueden compartir conexiones y esto significa una menor carga de trabajo para el servidor de la base de datos, un tiempo de respuesta más rápida y más accesos a página con éxito.

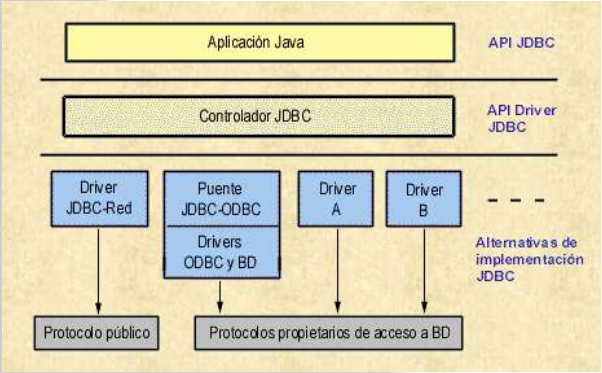
Existe un componente llamado RDS (*Remote Data Service*) que ofrece el ambiente de Acceso Universal a Datos, ya sea desde Internet o la World Wide Web, creando un marco de trabajo que permite una interacción fácil y eficiente con los datos fuente OLE DB tanto en Intranets corporativas o en Internet. RDS ofrece la ventaja de obtener por el lado del cliente resultados de datos, actualización y soporte para controles ADO y ofrece el modelo de programación OLE DB/ADO para manipular datos de las aplicaciones del cliente.

JDBC

JDBC o *Java Data Base Connectivity*, creado por la empresa Sun, es la API estándar de acceso a bases de datos con Java. Sun optó por crear una nueva API en lugar de utilizar ODBC, porque esta última presentaba algunos problemas desde ciertas aplicaciones Java. ODBC es una interfaz escrita en lenguaje C, que al no ser un lenguaje portable, hacía que las aplicaciones Java también perdiesen la portabilidad. Además, ODBC ha de instalarse manualmente en cada máquina, mientras que los controladores (*drivers)* JDBC que están escritos en Java son automáticamente instalables y portables. El nivel de abstracción al que trabaja JDBC es más alto que el de ODBC y, de esta forma, se pueden crear librerías de más alto nivel,

Para trabajar con JDBC es necesario tener controladores que permitan acceder a las distintas bases de datos. Sin embargo, ODBC sigue siendo hoy en día la API más popular para acceso a

Bases de Datos, por lo que: Sun se ha visto obligada a diseñar un puente que permite utilizar la API de JDBC en combinación con controladores ODBC.



Las tecnologías que se emplean para la conectividad entre los datos y la aplicación, se han convertido en un factor muy importante a la hora de desarrollar un proyecto que cuente con funcionalidad de acceso a datos. A continuación se muestra un cuadro comparativo de las dos tecnologías más importantes en este sentido: ActiveX Data Objects (ADO) y Java Data Base Connectivity (JDBC).

ADO	JDBC
• Tecnología elaborada por Microsoft • Tiene la principal función de realizar la solicitud de los datos a la base de datos. • Esta solicitud la realizará mediante la tecnología OLE DB, la cual estará en contacto de manera directa con la base de datos. • La tecnología OLE DB sólo se empleará cuando el DBMS pertenezca de igual manera a Microsoft, como es SQL Server. • ADO encapsulará a ciertos objetos de OLE DB, para que de ésta manera se realice la conexión con la base de datos. • Para realizar la gestión de acceso a bases de datos heterogéneas por parte de ADO, éste hará uso de ciertos objetos de la tecnología RDO (Remote Data Objects). • RDO dependerá de los ODBC's para poder efectuar la conexión a la base de datos y con esto el acceso a la información. • ADO podrá encontrarse trabajando en una página web en conjunto con código HTML; esto será posible mediante un mecanismo de introducción de instrucciones como es el VBscript. • Los objetos que conforman al ADO, no son compatibles con otros lenguajes, solo por aquellos que pertenecen a la empresa Microsoft como son: Visual C++, Visual Basic, Visual Java, etc.	• Tecnología hecha por Sun Microsistems. • Tiene la función de ser un gestor para la aplicación con respecto a la base de datos. • Por primera vez el JDBC fue empleado, tomando como intermediario entre él y la base de datos al ODBC. • Como modelo cliente/servidor, el JDBC se encontrará trabajando en el equipo cliente, conectándose directamente con la base de datos. • Como modelo de tres capas, el JDBC se encontrará en una capa intermedia, donde todos los usuarios pasarán por él para poder accesar a la base de datos. • Existen módulos JDBC que son propios de los fabricantes de DBMS, que son utilizados para el rápido acceso a la información de las bases de datos de los mismos. • JDBC no se encontrará ligado a trabajar con alguna tecnología en específica, ya que se elaboró con la finalidad de ser portable. • En aplicaciones Web, JDBC se encontrará laborando en conjunto con código HTML, mediante el mecanismo del Java script. • JDBC se elaboró con la finalidad de poder ser compatible y portable para poder ser empleado en aplicaciones y para la conexión con bases de datos.

4. DBA

La información es uno los activos más valiosos de la empresa, es indispensable contar con una persona -el administrador de datos- que conozca la información, y las necesidades de la empresa en este aspecto, *en un nivel gerencial superior*. Así la labor del administrador de datos es decidir en primer término cuáles datos deben almacenarse en la base de datos, y establecer políticas para mantener y manejar los datos una vez almacenados. El administrador de datos es por lo general, un gerente, no un técnico. El técnico responsable de poner en práctica las decisiones del administrador de datos es el *administrador de bases de datos*(DBA, database administrator).

El alcance de la actividad de la Administración de Datos es la organización completa (empresa, institución u otro organismo), mientras que el alcance de la Administración de Bases de Datos queda restringido a una Base de Datos en particular y a los sistemas que los procesan. La Administración de la Base de Datos opera dentro de un marco proporcionado por la Administración de Datos facilitándose de esta manera el desarrollo y el uso de una Base de Datos y sus aplicaciones. Las siglas DBA suelen utilizarse para designar tanto la función Administración de Base de Datos como al titulo del puesto administrador de Base de Datos.

En los distintos niveles y aplicaciones de Base de Datos existe la función DBA, aunque varia en complejidad. Esta es más sencilla cuando se trata de una Base de Datos Personal que cuando se refiere a una Base de Datos de grupos de trabajo, y esta a su vez es más sencilla que en una Base de Datos Organizacional. En una Base de Datos Personal comúnmente el mismo usuario es el Administrador de la Base de Datos; las Bases de Datos de grupos de trabajo requieren de una o dos personas que normalmente no se dedican a esta función de tiempo completo puesto que tienen otras responsabilidades dentro o fuera de la organización. En las Bases de Datos Organizacionales, que comúnmente permiten el acceso a decenas e incluso centenas de usuarios, se requiere de un administrador de Base de Datos de tiempo completo; lo anterior debido al alto volumen de procesos que deben desarrollarse, controlarse y supervisarse.

Un Administrador de Base de Datos de tiempo completo normalmente tiene aptitudes técnicas para el manejo del sistema en cuestión a demás, son cualidades deseables nociones de administración, manejo de personal e incluso un cierto grado de diplomacia. La característica más importante que debe poseer es un conocimiento profundo de las políticas y normas de la empresa así como el criterio de la empresa para aplicarlas en un momento dado.

Así, el DBA, a diferencia del administrador de datos, es un profesional en procesamiento de datos. La tarea del DBA es crear la base de datos en sí y poner en vigor los controles técnicos necesarios para apoyar las políticas dictadas por el administrador de datos. El DBA se encarga también de garantizar el funcionamiento adecuado del sistema y de proporcionar otros servicios de índole técnica relacionados. El DBA cuenta por lo regular con un grupo de programadores de sistemas y otros asistentes técnicos.

La responsabilidad general del DBA es facilitar el desarrollo y el uso de la Base de Datos dentro de las guías de acción definidas por la administración de los datos.

El DBA es responsable primordialmente de:

- Administrar la estructura de la Base de Datos
- Administrar la actividad de los datos
- Administrar el Sistema Manejador de Base de Datos
- Establecer el Diccionario de Datos
- Asegurar la confiabilidad de la Base de Datos
- Confirmar la seguridad de la Base de Datos

- **Administración de la estructura de la Base de Datos**

La administración de la estructura de la Base de Datos incluye participar en el diseño inicial de la misma y su puesta en practica así como controlar, y administrar sus requerimientos, ayudando a evaluar alternativas, incluyendo los DBMS a utilizar y ayudando en el diseño general de BD. En los casos de grandes aplicaciones de tipo organizacional, el DBA es un gerente que supervisa el trabajo del personal de diseño de la BD.

Una vez diseñada la BD, es puesta en practica utilizando productos del DBMS, procediéndose entonces a la creación de los datos (captura inicial). El DBA participa en el desarrollo de procedimientos y controles para asegurar la calidad y la alta integridad de la BD.

Los requerimientos de los usuarios van modificándose, estos encuentran nuevas formas o métodos para lograr sus objetivos; la tecnología de la BD se va modificando y los fabricantes del DBMS actualizan sus productos. Todas las modificaciones en las estructuras o procedimientos de BD requieren de una cuidadosa administración.

- **Implicaciones por la modificación de los esquemas**

Las solicitudes de modificación son inevitables una vez que el sistema ha entrado en operación, pueden aparecer solicitudes de nuevos requerimientos o estos pueden resultar de una comprensión inadecuada de los mismos. En cualquier caso, deberán efectuarse modificaciones en relación con toda la comunidad de la BD, ya que el impacto de tales alteraciones será resentido por mas de una aplicación. En algunos casos, pueden darse modificaciones que presentan efectos negativos para algunos usuarios; estos casos deberán ser tratados esgrimiendo como argumento los beneficios globales que serán obtenidos de tales alteraciones.

Una administración eficaz de la BD debe incluir procedimientos y políticas mediante las cuales los usuarios puedan registrar sus necesidades de modificaciones, y así la comunidad podrá analizar y discutir los impactos de dichas modificaciones, determinándose entonces la puesta o no en practica de tales alteraciones.

En razón del tamaño y complejidad de una BD y de sus aplicaciones, las modificaciones pudieran tener resultados inesperados. El DBA debe estar preparado para reparar la BD y reunir suficiente información para diagnosticar y corregir el problema provocado por la falla. Después de un cambio la BD es más vulnerable a fallas.

- **Documentación**

La responsabilidad final de un DBA en la administración de la estructura de una BD es la DOCUMENTACIÓN. Es de suma importancia saber que modificaciones han sido efectuadas, como fueron realizadas y cuando fueron establecidas. Una modificación sobre la estructura de la BD pudiera ocasionar un error que no apareciera a corto plazo; una vez que este surja, sin la documentación adecuada sobre las modificaciones realizadas, él diagnostico resultaría extremadamente complicado. En estos casos, se haría necesario una secuencia de rejecuciones para intentar detectar el punto en conflicto; el riesgo de este procedimiento radica en que es posible afectar la información contenida en la BD. Para identificar un cambio es de suma importancia mantener un registro de los formatos de prueba y de las ejecuciones de las pruebas efectuadas. Si se utilizan procedimientos de prueba formatos de pruebas y métodos de registro estandarizados, el registro de los resultados de la prueba no consumirá tiempo excesivo.

Comúnmente el tiempo de la documentación es tedioso y esto ocasiona que algunos DBA tienden a reducir o abreviar la información que se registra en ella e incluso llegan a desatenderla. Cuando ocurre un siniestro, la documentación completa y organizada puede ser la diferencia entre resolver o no un problema de extrema importancia y en la mayoría de los casos, que implica costos cuantiosos a la empresa.

La tarea de la documentación es cada vez más ligera y precisa cuando se utilizan DBMS que integran herramientas CASE para las tareas de diseño, mantenimiento y documentación. Estas mismas herramientas CASE proporcionan en la, mayoría de los casos la facilidad de generar y mantener en forma automática el Diccionario de Datos.

Una razón más para documentar consiste en la necesidad de mantener organizados datos históricos. Ocurre comúnmente que se desea realizar una consulta sobre los respaldos para conocer el estado que guardaba la información en un periodo determinado que transcurrió previamente. Los registros de modificación existentes en la documentación permitirá resolver problemas de incompatibilidad entre las estructuras que eran vigentes en el periodo de respaldo y las que lo son ahora; permitirá también el desarrollo de módulos de ajuste que faciliten la traducción de formatos y/o escalas para valores almacenados.

En los casos de caídas del sistema se presenta una situación parecida; los respaldos son requeridos y habrá de verificarse su estructura; formato y escala para integrarlos a la operación del sistema.

- **Administración de la actividad de datos**

Aunque el DBA protege los datos, no los procesa. El DBA no es usuario del sistema, en consecuencia, no administra valores de datos; el DBA administra actividad de datos. Dado que la BD es un recurso compartido, el DBA debe proporcionar estándares, guías de acción, procedimientos de control y la documentación necesaria para garantizar que los usuarios trabajan en forma cooperativa y complementaria al procesar datos en la BD.

Como es de suponerse, existe una gran actividad al interior de un DBMS. La concurrencia de múltiples usuarios requieren de estandarizar los procesos de operación; el DBA es responsable de tales especificaciones y de asegurarse que estas lleguen a quienes concierne. Todo el ámbito de la BD se rige por estándares, desde la forma como se capture la información (tipo, longitud, formato), como es procesada y presentada. El nivel de estandarización alcanza hasta los aspectos más internos de la BD; como sé accesa a un archivo, como se determinan los índices primarios y auxiliares, la foliación de los registros y demás.

Debe procurarse siempre que los estándares que serán aplicados beneficien también a los usuarios, privilegiando siempre la optimización en la operación del DBMS y el apego de las políticas de la empresa.

Una administración de BD efectiva deberá disponer siempre de este tipo de estándares; entre las funciones del DBA se encuentra la de revisarlos periódicamente para determinar su operatividad, y en su caso ajustarlos, ampliarlos o cancelarlos. Es también su responsabilidad el que estos se cumplan.

Cuando se definen estándares sobre la estructura de la BD, estos deben registrarse en una sección del diccionario de datos a la que todos aquellos usuarios relacionados con ese tipo de proceso pueden acceder.

Otro de los aspectos que el administrador debe atender es el de coordinar las nuevas propuestas para realizar ajustes en los derechos de acceso a datos compartidos y aplicaciones específicamente propuestas serían analizados en conjunto con los supervisores o directivos de las áreas involucradas para determinar si procede pudieran aparecer problemas cuando dos o más grupos de usuarios quedan autorizados para notificar los mismos datos. Uno de tales conflictos es el de la actualización perdida; este ocurre cuando el trabajo de un usuario queda sobrescrito sobre por el de un segundo usuario. El DBA queda responsabilizado para identificar la posible ocurrencia de dichos problemas así como de crear normas y procedimientos para su eliminación.

Se obtendrán este tipo de garantías cuando el DBMS sea capaz de implementar las restricciones aplicables al acceso concurrente, y este sea utilizado adecuadamente por programadores y usuarios; para borrar lo anterior, se hace indispensable el apego a los estándares el seguimiento de instructivos y manuales y las reglas establecidas para los diversos procesamientos y procedimientos que se llevan acabo.

Entre las alternativas mas utilizadas por el DBA para tratar de resolver o minimizar este problema se encuentran las siguientes:

a)Restringir el acceso a los procedimientos para ciertos usuarios.
b)Restringir al acceso a los datos para ciertos usuarios procedimientos y/o datos.
c)Evitar la coincidencia de horarios para usuarios que comparten.

Las técnicas de recuperación son otra función esencial del DBA al administrar la actividad de datos. A pesar de que el DBMS lleva a cabo una parte del proceso de recuperación, los usuarios determinan en forma critica la operatividad de esos sistemas de protección. El DBA debe anticipar fallas y definir procedimientos estándares de operación; los usuarios deben saber que hacer cuando el sistema este caído y que es lo primero que debe realizarse cuando el sistema este puesto en marcha nuevamente. El personal de operación deberá saber como iniciar el proceso de recuperación de la BD que copias de seguridad utilizar; como programar la rejecución del tiempo perdido y de las tareas pendientes; es importante también establecer un calendario para llevar a cabo estas actividades sin afectar a otros sistemas dentro de la organización que hagan uso de los mismos recursos de computo. Destacan por su importancia en el proceso de recuperación y a su vez en la atención que prestan a otros sectores de la organización. Los dispositivos de comunicación remota, los sistemas de interconexión y otros accesorios de uso compartido.

El DBA es el responsable de la publicación y mantenimiento de la documentación en relación con la actividad de los datos, incluyendo los estándares de la BD, los derechos de recuperación y de acceso a la BD, los estándares para la recuperación de caídas y el cumplimiento de las políticas establecidas. Los productos DBMS más populares que se encuentran en el mercado proporcionan servicios de utilerias para ayudar al DBA en la administración de los datos y su actividad. Algunos sistemas registran en forma automática los nombres de los usuarios y de las aplicaciones a las que tienen acceso así como a otros objetos de la BD. Incorpora también utilerias que permitan definir en el diccionario de datos las restricciones para que determinadas aplicaciones o módulos de ellas solo tengan acceso a segmentos específicos de la BD.

6. Técnicas de diseño lógico de bases de datos relacionales

Es posible comenzar el diseño de una base de datos desde distintos puntos del ciclo de vida del desarrollo de sistemas:

- Comenzar por el análisis: creando un modelo entidad relación a partir del cual obtendremos un diseño de base de datos
- Comenzar por un nuevo diseño: es posible comenzar directamente por la fase de diseño, pero exige algún modo de análisis previo.
- Capturar un diseño de base de datos existente.

Planificación de una base de datos relacional

- Tablas/columnas
- Valores permitidos o rangos para las columnas
- Dominios
- Integridad de datos
- Valores autogenerados: secuencias, valores por defecto
- Vistas e índices
- Desnormalización

Diagrama

Antes de comenzar la fase de diseño es necesario comprobar la calidad del análisis realizado

Entidades:

- Cada entidad debe tener su nombre y descripción
- Cada entidad debe tener al menos un identificador único

Atributos

- Cada atributo debe tener su nombre y descripción
- Cada atributo debe tener tamaño, formato y opcionalidad
- Cada atributo debe contar con reglas de validación: dominios, valores permitidos, valores por defecto, reglas de derivación.

Relaciones

- Toda entidad debe estar relacionada
- Cada relación debe tener un nombre significativo

Creación del modelo lógico

El primer paso que hay que realizar en el diseño de una base de datos a partir de un análisis es convertir la información del modelo en un primer diseño de datos.

- El modelo de datos es un diagrama que muestra los elementos de información y el modo en que se relacionan, así como la documentación de los mismos.

- Un modelo de información no es una base de datos, no un diseño de base de datos.
- Es un esquema conceptual, independiente, de los datos necesarios en el sistema
- Es posible mostrar el modelo de datos como un esquema entidad-relacion

Componentes del diseño de datos

- Definición de tablas: las tablas representan las unidades básicas de almacenamiento en una base de datos
- Las tablas contienen definiciones de columnas (descripción, código, fecha,etc…)
- Cada columna tiene un tipo de dato (integer, char, text, etc…)

Restricciones de integridad

Las restricciones de integridad se utilizan para definir reglas de negocio sobre las tablas

Integridad de clave primaria: toda tabla debe tener una y solo una restricción de clave primaria, de esta manera podemos garantizar:

- Que no puede existir dos veces el mismo valor en clave primaria dentro de la tabla
- Que la clave primaria no puede contener valor nulo.

Integridad referencial (clave foranea)

- Una clave foránea debe referenciar una clave primaria o clave única de otra tabla.
- Las columnas que componen la clave foránea, deben tener el mismo tipo de datos y longitud que las columnas de la clave primaria con que se relacionan.
- Las columnas de clave foránea deben heredar la opcionalidad de la relación original

7. Diseño físico de bases de datos

La fase de diseño físico consiste en la toma de decisiones sobre los aspectos de representación interna y organización de los datos de manera que garanticen un buen rendimiento espacial y temporal.

El objetivo del diseño físico de las bases de datos tiene como objetivos:

- Disminuir los tiempos de respuesta
- Minimizar el espacio de almacenamiento
- Conseguir la máxima seguridad de los datos
- Optimizar el consumo de recursos

Herramientas que los DBMS ofrecen para realizar el diseño físico:

- Elección de la organización de los datos
- Definición de índices
- Definición de registros físicos
- Asignación de ficheros a dispositivos de almacenamiento
- Asignación de espacios de almacenamiento intermedios(bufer, memoria, etc…)

En el estado actual de desarrollo de los DBMS, el diseño físico es dependiente del sistema de gestión sobre el que se va a hacer la implementación.

Factores que influyen en el diseño físico

- Debe estar guiado por la naturaleza de los datos y además por el uso esperado de éstos.
- Para cada una de las relaciones hay que estimar el tamaño de las columnas y las previsiones de crecimiento de las ocurrencias a lo largo de la vida del sistema.
- Se debe tener una lista de las consultas que se van a realizar y la frecuencia de ejecución de éstas.
- Se debe tener una lista de las transacciones a ejecutar y la frecuencia de éstas.

Para cada consulta o transacción será necesario indentificar:

- Relaciones a las que se accede
- Atributos que son obtenidos
- Especialmente, los atributos que forman parte de la selección o de las concatenaciones si las hubiera.
- Tipo o tipos de acciones en cada transacción(insertar, borrar o actualizar)
- Es necesario conocer las características del DBMS y del equipo en el que será instalado

Una vez identificados los elementos anteriores, partiremos del diseño lógico de la base de datos(modelo de datos) para la implementación del diseño físico en donde podremos percatarnos de algunas incidencias en las que el modelo de datos haya incurrido que afecten el funcionamiento real de la base de datos; es por eso que la fase de implementación comienza en el diseño físico; mismo que nos permite visualizar por primera vez la base de datos en cuanto a su estructura.

8. Estructura de Datos

Es un esquema que representa el diseño de una base de datos de red. Este modelo se basa en representaciones entre registros por medio de ligas, existen relaciones en las que participan solo dos entidades(binarias) y relaciones en las que participan más de dos entidades (generales) ya sea con o sin atributo descriptivo en la relación.

La forma de diagramado consta de dos componentes básicos:

Celdas: representan a los campos del registro.

Líneas: representan a los enlaces entre los registros.

su representación gráfica se basa en el acomodo de los campos de un registro en un conjunto de celdas que se ligan con otro(s) registro(s)

La estructura de una base de datos hace referencia a los tipos de datos, los vínculos o relaciones y las restricciones que deben cumplir esos datos (integridad de datos y redundancia de datos).

La estructura de una base de datos es diseñada o descripta empleando algún tipo de modelo de datos.

Un ejemplo a modo de descripción de la estructura de una base de datos puede ser:

ALUMNO: numero de alumnno (entero de 6 números), nombre (cadena de 30 caracteres), apellido (cadena de 30 caracteres), año de nacimiento (entero de 4 números), especialidad (entero de 3 números).

ESPECIALIDAD: numero de especialidad (entero de 3 números), nombre de especialidad (cadena de 30 caracteres).

9. Optimización de bases de datos

Cuanto mejor mantengamos nuestra base de datos, mejor rendimiento obtendremos de las consultas que realicemos sobre la misma (los resultados se obtendrán más rápidamente, y en consecuencia, se podrán mostrar antes).

Siempre es necesario **dedicar un tiempo al diseño** de nuestra base de datos. Indicar bien las tablas, campos y sus relaciones, en función de las necesidades que tengamos, puede facilitarnos el mantenimiento y garantizarnos un rendimiento adecuado a nuestras necesidades.

Para conseguir un buen diseño de las tablas que integrarán nuestra base de datos suele utilizarse un **Modelo Relacional**, donde se extraen los elementos, propiedades y relaciones entre los mismos, que se traducen en la base de datos en tablas, sus campos, índices y claves relacionadas.

Una vez extraídas las tablas, del modelo relacional, lo principal para un buen funcionamiento de nuestra base de datos es disponer de los **índices correctos** en las tablas, sobre los que trabajará el DBMS para extraer el resultado de las mismas.

- La indexación, tanto de claves primarias como extranjeras, se puede obtener del modelo relacional.
- Las claves primarias identifican unívocamente a cada elemento de una tabla.
- Las claves extranjeras marcan las relaciones entre tablas.

Disponer de índices en los campos adecuados optimizará sus resultados:

- Para mejorar una consulta (SELECT), hay que crear un índice sobre los campos que son utilizados en las búsquedas (los que aparecen en las cláusulas **WHERE o JOIN**).
- Utilice índices sobre campos con valores únicos. Los índices funcionan peor si el campo tiene valores duplicados.
- Trate de que los índices sean cortos. Si indexa un campo de texto, evite hacerlo sobre campos de longitud variable, y acorte siempre el tamaño del índice a lo que considere más adecuado. Por ejemplo, si un campo CHAR tiene 200 caracteres y sabe que los valores se distinguen en los primeros 20 caracteres, indexe sólo hasta dicho tamaño de campo. Ahorrará espacio y ganará velocidad de respuesta.
- No cree índices innecesarios. Estos se actualizan con cada cambio en la tabla asociada y pueden ralentizar las modificaciones de la misma.

Sea coherente con los **tipos de campos** en sus tablas y elija siempre los más adecuados:

- Utilice los mismos tipos de campos para el mismo tipo de información en distintas tablas. Si necesitara cruzar tablas con campos del mismo tipo ganará en rapidez.
- Evite en lo posible el uso de campos de tamaño variable. Los campos de longitud fija (como CHAR) son más eficientes que los de longitud variable (VARCHAR, BLOB o TEXT).
- Utilice campos numéricos frente a campos de texto.
- Trate de usar campos que no puedan tener valores nulos (**Not Null**). Los valores nulos ralentizan las lecturas.

Mantenga siempre sus **tablas** con la información necesaria (ni más ni menos):

- Si crea tablas con el atributo **row_format**, use el tipo **fixed**, en vez de **dynamic**, ya que las tablas se consultarán de modo más rápido.

- Haga “limpieza” cada cierto tiempo. Si observa que sus tablas tienen muchos registros (han crecido mucho de tamaño), analice sus datos para comprobar si algunos registros están anticuados y pueden eliminarse o archivarse. Reducir el tamaño de sus tablas mejorará su rendimiento.
- Cuando haya hecho varios cambios o eliminaciones en alguna tabla ejecute la sentencia **OPTIMIZE TABLE**, que reparará y ordenará la tabla para mejorar su rendimiento.

Sintaxis de OPTIMIZE TABLE

OPTIMIZE [LOCAL | NO_WRITE_TO_BINLOG] TABLE ***tbl_name*** [, ***tbl_name***] ...

OPTIMIZE TABLE debe usarse si ha borrado una gran parte de la tabla o si ha hecho varios cambios en una tabla con registros de longitud variable (tablas que tienen columnas **VARCHAR**, **BLOB**, o **TEXT**). Los registros borrados se mantienen en una lista enlazada y operaciones **INSERT** posteriores reúsan posiciones de antiguos registros. Puede usar **OPTIMIZE TABLE** para reclamar el usuario no usado y para defragmentar el fichero de datos.

En la mayoría de inicializaciones, no necesita ejecutar **OPTIMIZE TABLE** para nada. Incluso si hace muchas actualizaciones a registros de longitud variables, no es probable que necesite hacerlo más de una vez a la semana o mes y sólo en ciertas tablas.

Actualmente, **OPTIMIZE TABLE** funciona sólo en tablas **MyISAM**, **BDB** y **InnoDB** .

- Si la tabla ha borrado o dividido registros, repara la tabla.
- Si las páginas índice no están ordenadas, ordéna las tablas.
- Si las estadísticas no están actualizadas (y la reparación no puede hacerse ordenando el índice), actualiza las talbas.

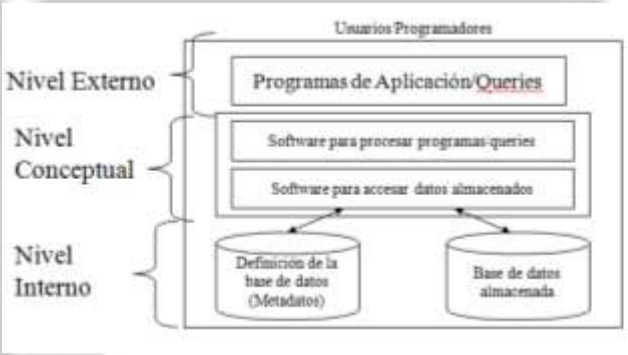
10. DBMS

Es un software de sistemas que tiene como propósito general facilitar el proceso de definir, construir y manipular bases de datos que se utilizan para diferentes tipos de aplicaciones.

Componentes de software principales asociados a un DBMS :

- Generadores de aplicación.
- Lenguajes de cuarta generación (4GL).
- Software de consulta a la base de datos.
- Generadores de reportes y pantallas.

Ejemplos de software: Informix 4GL, Oracle, SQL server...



Esquema del ambiente de un DBMS
Los niveles están conectados lógicamente

PROCESOS RELACIONADOS CON LOS DBMS:

*Definir la base de datos significa la declaración de:

- Los tipos de datos
- La estructura
- Las restricciones de los datos a ser almacenados en la base de datos.

* Crear o construir la base de datos: es el proceso de almacenar los datos en algún medio de almacenamiento, esto es controlado por el DBMS.

* Manipular una base de datos incluye funciones como:

- Consultar la base de datos para obtener algunos datos específicos.
- Actualizar la base de datos para reflejar cambios en el minimundo.
- Generar reportes de los datos.
- Eliminar algunos datos

CAPACIDADES QUE DEBE OFRECER UN DBMS:

- Control de redundancias.

- Restricción de accesos no autorizados
- Proporcionar múltiples interfaces de usuario.
- Representar relaciones complejas entre datos.
- Forzar el uso de restricciones de integridad.
- Proporcionar métodos de respaldos y recuperación.

CUANDO NO UTILIZAR UN DBMS

* La base de datos y aplicaciones son simples, bien definidas y se requieren pocos cambios.

* No es necesario el acceso de múltiples usuarios a los datos.

Vista Externa: Nivel mas alto, visto por el programador de aplicaciones o el usuario, en esta vista solo porciones de la base de datos son de interés para el usuario o programador de aplicaciones, se representa por el esquema externo.

Vista Conceptual o global : En esta vista se incluyen todas las entidades de la base de datos y las relaciones entre ellas. La vista conceptual representa la base de datos entera, definida por el esquema conceptual.

Vista Interna: Nivel mas bajo de abstracción de la base de datos, contiene la definición del almacenamiento de registros, el método de representación de datos y el acceso utilizado, expresado por el esquema interno.

11. Implementar y administrar sistemas manejadores de bases de datos

Consideraciones al elegir un DBMS

- Número de usuarios
- Número de transacciones
- Cantidad de datos para almacenar
- Consistencia en la información
- Presupuesto
- Experiencia propia o externa

MySQL

MySQL, es una manejador de bases de datos relacional bastante robusto, de código abierto bajo la licencia GPL el cual se ha convertido en el más popular hoy en día.

Su origen se debió a la búsqueda por parte de los fundadores de crear un manejador de bases de datos que fuera "rápido", todavía más rápido que mSQL. Así surgió MySQL, primero como un producto de la empresa y después como software de dominio público.

El nombre de My se debe probablemente a que la hija del cofundador Monty Widenius recibía ese sobrenombre, aunque a ciencia cierta nunca se ha revelado el origen. Por otro lado en el año 2002 MySQL tuvo un logo más original que el simple nombre, incluyendo un delfín, el cual a través de una encuesta en la página web recibió su nombre: "Sakila", de origen africano.

Por qué usar MySQL ?

Es importante resaltar que no se trata de una herramienta de juguete o aprendizaje, MySQL es un manejador que puede competir con sus famosas contrapartes comerciales: Oracle, DB2, Informix, Sybase.

Básicamente los motivos por los cuales se podría optar por usar MySQL en lugar de otro manejador serían:

- Es gratis
- Es extensible
- Es robusto
- Es rápido
- No requiere de una gran número de recursos para funcionar (obviamente para aplicaciones a gran escala es mejor contar con una buena infraestructura)
- Es fácil de administrar

Instalación Básica

MySQL posee varias versiones. Los cambios en la versión 4 permiten tener mayor funcionalidad, teniendo por ejemplo queries anidados y búsquedas a texto completo.

MySQL posee 4 tipos de tablas (y en consecuencia de índices):

Index	Versión
MyISAM	Standard, Max

Heap	Standard, Max
BerkeleyDB	Max
Innodb	Max

Tabla 3.1 Indexamiento en MySQL

MyISAM se basa en un indexamiento por bloques, Heap es una tabla que existe solo en memoria, mientras que BDB e InnoDB utilizan B-Trees.

Se puede observar que existen 2 variantes del software, la Standard y la Max; la diferencia radica en el soporte de transacciones que es posible en la versión Max gracias a los módulos de Berkeley DB e InnoDB incorporados en ella.

Es importante resaltar que aunque esta funcionalidad esta disponible no es la configuración por default, salvo que se indique lo contrario, siempre tipo de indexamiento por defecto será MyISAM.

El equipo MySQL recomienda bajar los binarios compilados por ellos para evitar cualquier tipo de problema, de manera que en la sección de "Database Server" se puede bajar el binario de la versión deseada.

Una vez descargado el software se procede a desempaquetarlo (.tgz, zip) o bien ejecutar el .exe correspondiente.

Dichos directorios contenidos en un directorio que por lo general lleva el mismo nombre 'mysql' contiene una estructura de la siguiente manera:

- bin: programas ejecutables, mysql, mysqld, mysqldump, myisamchk, mysqlbinlog.
- include, lib, libexec: librerías y encabezados para programar en C/C++
- mysql-test, sql-bench: pruebas y benchmarks
- var ó data: estructura de todas las bases y datos de las tablas tipo MyISAM y Berkeley DB.
- man: páginas de manual
- share: información en distintos idiomas
- support-files: archivos de configuración y scripts de arranque automático

```
(root) [mysql]> show databases;
+------------+
| Database   |
+------------+
| mysql      |
+------------+
1 rows in set (0.12 sec)

(root) [mysql]> show tables;
+----------------+
| Tables_in_mysql |
+----------------+
| columns_priv    |
| db              |
| func            |
| help_category   |
| help_relation   |
| help_topic      |
| host            |
| tables_priv     |
| user            |
+----------------+
9 rows in set (0.10 sec)
```

- Tabla 3.4 Tablas de administración del dbms
- Para cada tabla se puede emplear el comando 'desc' o 'describe' para analizar la estructura de cada tabla y apreciar la relación que tiene con las demás.
- Para dar de un alta un usuario, ejemplo de la tabla 3.5, se debe crear el usuario dentro de la tabla 'user', crear la base de datos y posteriormente asociar dicho usuario con la base en la tabla 'bd', todo lo anterior utilizando instrucciones de SQL tradicionales.

```
(root) [mysql]> desc user;
+-----------------------+-----------------------------------+-------------------+------+-----+---------+-------+
| Field                 | Type                              | Collation         | Null | Key | Default | Extra |
+-----------------------+-----------------------------------+-------------------+------+-----+---------+-------+
| Host                  | varchar(60) binary                | binary            |      | PRI |         |       |
| User                  | varchar(16) binary                | binary            |      | PRI |         |       |
| Password              | varchar(45) binary                | binary            |      |     |         |       |
| Select_priv           | enum('N','Y')                     | latin1_swedish_ci |      |     | N       |       |
| Insert_priv           | enum('N','Y')                     | latin1_swedish_ci |      |     | N       |       |
| Update_priv           | enum('N','Y')                     | latin1_swedish_ci |      |     | N       |       |
| Delete_priv           | enum('N','Y')                     | latin1_swedish_ci |      |     | N       |       |
| Create_priv           | enum('N','Y')                     | latin1_swedish_ci |      |     | N       |       |
| Drop_priv             | enum('N','Y')                     | latin1_swedish_ci |      |     | N       |       |
| Reload_priv           | enum('N','Y')                     | latin1_swedish_ci |      |     | N       |       |
| Shutdown_priv         | enum('N','Y')                     | latin1_swedish_ci |      |     | N       |       |
| Process_priv          | enum('N','Y')                     | latin1_swedish_ci |      |     | N       |       |
| File_priv             | enum('N','Y')                     | latin1_swedish_ci |      |     | N       |       |
| Grant_priv            | enum('N','Y')                     | latin1_swedish_ci |      |     | N       |       |
| References_priv       | enum('N','Y')                     | latin1_swedish_ci |      |     | N       |       |
| Index_priv            | enum('N','Y')                     | latin1_swedish_ci |      |     | N       |       |
| Alter_priv            | enum('N','Y')                     | latin1_swedish_ci |      |     | N       |       |
| Show_db_priv          | enum('N','Y')                     | latin1_swedish_ci |      |     | N       |       |
| Super_priv            | enum('N','Y')                     | latin1_swedish_ci |      |     | N       |       |
| Create_tmp_table_priv | enum('N','Y')                     | latin1_swedish_ci |      |     | N       |       |
| Lock_tables_priv      | enum('N','Y')                     | latin1_swedish_ci |      |     | N       |       |
| Execute_priv          | enum('N','Y')                     | latin1_swedish_ci |      |     | N       |       |
| Repl_slave_priv       | enum('N','Y')                     | latin1_swedish_ci |      |     | N       |       |
| Repl_client_priv      | enum('N','Y')                     | latin1_swedish_ci |      |     | N       |       |
| ssl_type              | enum('','ANY','X509','SPECIFIED') | latin1_swedish_ci |      |     |         |       |
| ssl_cipher            | blob                              | binary            |      |     |         |       |
| x509_issuer           | blob                              | binary            |      |     |         |       |
| x509_subject          | blob                              | binary            |      |     |         |       |
| max_questions         | int(11) unsigned                  | binary            |      |     | 0       |       |
| max_updates           | int(11) unsigned                  | binary            |      |     | 0       |       |
| max_connections       | int(11) unsigned                  | binary            |      |     | 0       |       |
+-----------------------+-----------------------------------+-------------------+------+-----+---------+-------+
31 rows in set (0.00 sec)

(root) [mysql]> desc db;
+-----------------------+---------------+-------------------+------+-----+---------+-------+
| Field                 | Type          | Collation         | Null | Key | Default | Extra |
+-----------------------+---------------+-------------------+------+-----+---------+-------+
| Host                  | char(60) binary | binary          |      | PRI |         |       |
| Db                    | char(64) binary | binary          |      | PRI |         |       |
| User                  | char(16) binary | binary          |      | PRI |         |       |
| Select_priv           | enum('N','Y')   | latin1_swedish_ci |      |     | N       |       |
| Insert_priv           | enum('N','Y')   | latin1_swedish_ci |      |     | N       |       |
| Update_priv           | enum('N','Y')   | latin1_swedish_ci |      |     | N       |       |
| Delete_priv           | enum('N','Y')   | latin1_swedish_ci |      |     | N       |       |
| Create_priv           | enum('N','Y')   | latin1_swedish_ci |      |     | N       |       |
| Drop_priv             | enum('N','Y')   | latin1_swedish_ci |      |     | N       |       |
| Grant_priv            | enum('N','Y')   | latin1_swedish_ci |      |     | N       |       |
| References_priv       | enum('N','Y')   | latin1_swedish_ci |      |     | N       |       |
| Index_priv            | enum('N','Y')   | latin1_swedish_ci |      |     | N       |       |
| Alter_priv            | enum('N','Y')   | latin1_swedish_ci |      |     | N       |       |
| Create_tmp_table_priv | enum('N','Y')   | latin1_swedish_ci |      |     | N       |       |
| Lock_tables_priv      | enum('N','Y')   | latin1_swedish_ci |      |     | N       |       |
+-----------------------+---------------+-------------------+------+-----+---------+-------+
(root) [mysql]> insert into user (host,user,password) values ('%','carlos',password('lolo'));
Query OK, 1 row affected (0.07 sec)

(root) [mysql]> select * from user where user='carlos';
+------+--------+-------------------------------------------+-------------+-------------+-------------+-------------+
| Host | User   | Password                                  | Select_priv | Insert_priv | Update_priv | Delete_priv |
+------+--------+-------------------------------------------+-------------+-------------+-------------+-------------+
| %    | carlos | *87f0212af7420ce3e0b2b8992eb42dda4be54c9125b3 | N           | N           | N           | N           |
+------+--------+-------------------------------------------+-------------+-------------+-------------+-------------+
1 row in set (0.00 sec)

(root) [mysql]> create database prueba;
Query OK, 1 row affected (0.00 sec)

(root) [mysql]> insert into db values ('%','prueba','carlos','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y');
Query OK, 1 row affected (0.00 sec)

(root) [mysql]> select * from db where user='carlos';
+------+--------+--------+-------------+-------------+-------------+-------------+
| Host | Db     | User   | Select_priv | Insert_priv | Update_priv | Delete_priv |
+------+--------+--------+-------------+-------------+-------------+-------------+
| %    | prueba | carlos | Y           | Y           | Y           | Y           |
+------+--------+--------+-------------+-------------+-------------+-------------+
1 row in set (0.09 sec)

(root) [mysql]> flush privileges;
Query OK, 0 rows affected (0.07 sec)
```

12. Lenguaje de manipulación de datos DML

Lenguaje de Manipulación de Datos (**D**ata **M**anipulation **L**anguage, DML) es un lenguaje proporcionado por el sistema de gestión de base de datos que permite a los usuarios de la misma llevar a cabo las tareas de consulta o manipulación de los datos, organizados por el modelo de datos adecuado. El lenguaje de manipulación de datos más popular hoy día es SQL, usado para recuperar y manipular datos en una base de datos relacional.

Cuando se quiere agregar, actualizar o eliminar datos de una base de datos, se ejecutan sentencias DML. Una colección de sentencias DML que forman una unidad lógica de trabajo es llamada transacción.

Considere una base de datos de un banco. Cuando un cliente del banco transfiere dinero de su cuenta de ahorros a su cuenta de cheques, la transacción puede consistir de tres operaciones separadas:

1. Decrementar la cuenta de ahorros.
2. Incrementar la cuenta de cheques.
3. Registrar la transacción en la bitácora de transacciones.

El servidor de Oracle puede garantizar que las tres sentencias SQL sean ejecutadas para mantener las cuentas en un correcto balance. Cuando algo impide que una de las sentencias en la transacción sea ejecutada, las otras sentencias de la transacción pueden ser desechas.

INSERT

Se puede añadir nuevas filas(registros) a una tabla con el uso de la sentencia INSERT.

Sintaxis:

INSERT INTO tabla (campo1,campo2,…)VALUES(dato1,dato2,….)

tabla es el nombre de la tabla

campos es el nombre del campo de la tabla a ser poblada

values es el valor del correspondiente campo

Nota: en esta sentencia con la cláusula VALUES se agrega solamente una fila a la vez a la tabla

Encierre los caracteres y fechas entre comillas sencillas; esto no es recomendado para valores numéricos.

Los valores numéricos no deben ser encerrados entre comillas sencillas, puesto que la conversión implícita puede tomar lugar para valores numéricos asignados a una columna con tipo de dato NUMBER sin ser necesario.

UPDATE

Actualizando filas

Se pueden modificar filas existentes con el uso de la sentencia UPDATE.

Sintaxis:

UPDATE tabla SET campo=valor,campo2=valor2 WHERE condición

Tabla: es el nombre de la tabla

columna: es el nombre de la columna en la tabla a poblar

valor: es el valor correspondiente o sub consulta para la columna

condición: identifica las filas que serán actualizadas y se compone de nombres de columnas, expresiones, constantes, sub consultas y operadores de comparación.

Confirme la operación de actualización consultando en la tabla las filas modificadas.

Nota: en general, utilice llaves primarias para identificar una fila en particular. Utilizando otras columnas se puede inesperadamente causar la actualización de diversas filas. Por ejemplo, identificar una fila determinada en la tabla EMPLEADOS por medio de los apellidos es peligroso puesto que más de un empleado puede tener el mismo apellido.

DELETE

Eliminando filas

Se pueden eliminar filas existentes utilizando la sentencia DELETE.

Sintaxis

DELETE FROM tabla WHERE condición

Tabla: es el nombre de la tabla

Condición: identifica las filas que serán eliminadas y esta compuesta de nombre de columnas, expresiones, constantes, subconsultas y operadores de comparación

Nota: si no hay filas eliminadas, el mensaje “0 rows deleted” es mostrado. Es muy importante generar condiciones exactas pues al igual que al actualizar si nuestra condición es muy ambigua podríamos eliminar más de un registro de manera accidental.

SELECT

Consulta de registros

Sintaxis

SELECT campo,campo2 FROM tabla WHERE condición

Tabla: la tabla de la cual queremos visualizar datos

Campo: los campos que deseamos visualizar

Condición: conjunto de instrucciones que filtran la información de modo que solo veamos los registros necesarios.

Dentro de la sentencia SELECT nos encontramoes con una serie de palabras reservadas del lenguaje que nos permiten manipular la consulta a nuestras necesidades.

ALL, *	Indica que queremos seleccionar todos los valores.Es el valor por defecto y no suele especificarse casi nunca.
DISTINCT	Indica que queremos seleccionar sólo los valores distintos.
FROM	Indica la tabla (o tablas) desde la que queremos recuperar los datos. En el caso de que exista más de una tabla se denomina a la consulta "consulta combinada" o "join". En las consultas combinadas es necesario aplicar una condición de combinación a través de una cláusula **WHERE**.
WHERE	Especifica una condición que debe cumplirse para que los datos sean devueltos por la consulta. Admite los operadores lógicos **AND** y **OR**.
GROUP BY	Especifica la agrupación que se da a los datos. Se usa siempre en combinación con funciones agregadas.
HAVING	Especifica una condición que debe cumplirse para los datos Especifica una condición que debe cumplirse para que los datos sean devueltos por la consulta. Su funcionamiento es similar al de **WHERE** pero aplicado al conjunto de resultados devueltos por la consulta. Debe aplicarse siempre junto a **GROUP BY** y la condicion debe estar referida a los campos contenidos en ella.
ORDER BY	Presenta el resultado ordenado por las columnas indicadas. El orden puede expresarse con**ASC** (orden ascendente) y **DESC** (orden descendente). El valor predeterminado es **ASC.**

Para la gran mayoría de DBMS al contar con un entorno visual es muy fácil poder ver la estructura de las tablas y de la base de datos en si; sin embargo para algunos lenguajes de uso frecuente como el MySQL tenemos que hacer referencia a 2 sentencias de vital importancia para un DBA.

SHOW

La sentencia **SHOW** tiene una sintaxis sencilla y en nuestro ejemplo siguiente, lo utilizamos en tres diferentes formas:

```
SHOW DATABASES;
SHOW TABLES;
SHOW COLUMNS FROM nombre_de_tabla;
```

Lo que hacen estas sentencias es, en su respectivo orden:

Muestra las bases de datos que están guardadas en MySQL
Muestra las tablas contenidas en una base de datos específica.
Muestra las columnas contenidas en una tabla específica.

DESCRIBE

DESCRIBE proporciona información acerca de columnas en una tabla. Es una abreviación de **SHOW COLUMNS**

```
mysql> DESCRIBE city;
+------------+----------+------+-----+---------+----------------+
| Field      | Type     | Null | Key | Default | Extra          |
+------------+----------+------+-----+---------+----------------+
| Id         | int(11)  |      | PRI | NULL    | auto_increment |
| Name       | char(35) |      |     |         |                |
```

```
| Country    | char(3)  |      | UNI |         |                |
| District   | char(20) | YES  | MUL |         |                |
| Population | int(11)  |      |     | 0       |                |
+------------+----------+------+-----+---------+----------------+
```

13. Indexación de datos

El **índice** de una base de datos es una estructura de datos que mejora la velocidad de las operaciones, permitiendo un rápido acceso a los registros de una tabla en una base de datos. Al aumentar drásticamente la velocidad de acceso, se suelen usar sobre aquellos campos sobre los cuales se hacen frecuentes búsquedas.

El índice tiene un funcionamiento similar al índice de un libro, guardando parejas de elementos: el elemento que se desea indexar y su posición en la base de datos. Para buscar un elemento que esté indexado, sólo hay que buscar en el índice dicho elemento para, una vez encontrado, devolver el registro que se encuentre en la posición marcada por el índice.

Los índices pueden ser creados usando una o más columnas, proporcionando la base tanto para búsquedas rápidas al azar como de un ordenado acceso a registros eficiente.

Los índices son construidos sobre árboles B, B+, B* o sobre una mezcla de ellos, funciones de cálculo u otros métodos.

El espacio en disco requerido para almacenar el índice es típicamente menor que el espacio de almacenamiento de la tabla (puesto que los índices generalmente contienen solamente los campos clave de acuerdo con los que la tabla será ordenada, y excluyen el resto de los detalles de la tabla), lo que da la posibilidad de almacenar en memoria los índices de tablas que no cabrían en ella. En una base de datos relacional un índice es una copia de una parte de la tabla.

Algunas bases de datos amplían la potencia del indexado al permitir que los índices sean creados de funciones o expresiones. Por ejemplo, un índice puede ser creado sobre la función upper(apellido), que almacenaría en el índice solamente las versiones mayúsculas del campo *apellido*. Otra opción a veces soportada, es el uso de índices "filtrados", donde las entradas del índice son creadas solamente para los registros que satisfagan una cierta expresión condicional. Un aspecto adicional de flexibilidad es permitir la indexación en funciones definidas por el usuario, también como expresiones formadas de un surtido de funciones incorporadas. Todos estos refinamientos de la indexación son soportados en la gran mayoría de lenguajes de programación.

Los índices pueden ser definidos como únicos o no únicos. Un índice único actúa como una restricción en la tabla previniendo filas idénticas en el índice. Para crear índices únicos es muy común aplicar el atributo de autoincremento en estos campos para que nunca se repitan los valores.

14. Respaldo de datos

En todo sistema manejador de abses de datos existe la posibilidad de que ocurran fallas que generen pérdida de información, estas fallas pueden ocurrir por:

Errores del usuario: por ejemplo.

- Actualización indebida de una tabla
- Fallas en el hardware: Como por ejemplo un aterrizaje de discos
- Fallas en el software: errores en el código de una aplicación

Todos los manejadores de bases de datos ofrecen mecanismos de respaldo, que permite hacer copias totales, parciales o incrementales de la base de datos.

Los respaldos pueden ser:

En línea: mientras se está respaldando, los datos siguen estando disponibles para los usuarios. Esto es muy útil para bases de datos que deben estar en servicio todo el tiempo.

Fuera de línea: Requiere que la base de datos esté fuera de servicio, mientras se está respaldando. Este método particularmente suele ser el más utilizado debido a que previene errores de integridad así como de concurrencia de datos.

15. Recuperación de datos

El sistema de recuperación que ofrecen los manejadoes de bases de datos, permite, que en el caso de que ocurra una falla, la base de datos pueda ser restaurada, con un mínimo impacto para el usuario; es decir, que se puedan recuperar todas las transacciones que se hayan hecho, hasta momentos antes de que la falla haya ocurrido.

La recuperación de datos puedes ser:

Estática: Es decir la base de datos se restaura hasta el estado en que se encontraba cuando se tomó el último respaldo o copia.

Dinámica: No solo restaura la base desde la copia más reciente que se tenga, sino que también es capaz de recuperar las transacciones que se hayan hecho desde entonces.(Cabe mencionar que no todos los DBMS cuentan con esta habilidad).