**BASE DE DATOS**

Sistema computarizado cuya finalidad es almacenar información y permitir a los usuarios recuperar y actualizar esa información en base a peticiones.

**Componentes de una BD**

* Datos: Es una entrada del sistema de información. Pueden ser integrados y compartidos.

*Integrados* porque se puede imaginar una BD como una unificación de varios archivos con redundancia (datos repetidos) mínima entre ellos.

*Compartidos* porque las piezas individuales de la BD son utilizadas por diferentes usuarios al mismo tiempo.

* HW: Unidades de procesamiento y unidades de almacenamiento principal y secundario.
* SW: Abarca el SO y el motor de BD.

*Motor de BD*(DBMS): Es una pieza de SW ubicada entre el SO y los programas de aplicación de usuarios. Procesa todas las peticiones (agregar, eliminar, modificar y leer datos), ocultando a las aplicaciones los detalles de implementación.

* Usuarios: Hay varios tipos:

*Programadores*: Codifican aplicaciones de alto nivel para los usuarios finales. Estas aplicaciones son las que generan peticiones.

*Usuario final*: Utilizan las aplicaciones desarrolladas por los programadores que finalmente accederán en forma concurrente a la BD.

*Administrador de BD* (DBA): Personal técnico responsable del funcionamiento y disponibilidad de la BD.

*Administrador de datos* (DA): Persona no técnica responsable de las necesidades de información de la BD. Decide qué datos hay que guardar en la BD.

**Beneficios** con respecto a los archivos.

* *Recuperación*: La diferencia fundamental con los archivos convencionales es que al recuperar información, las BD no devuelven solamente los datos, como los archivos, sino también su estructura.
* *Compactación*: Los archivos de BD son administrados por el DBMS, quien resuelve las problemáticas de duplicación de datos y administración del espacio vacío.
* *Velocidad*: Las BD son más rápidas para obtener la información del negocio requerido, es decir, en el código se usarán pocas sentencias para acceder a los datos. Sería más rápido acceder directamente al SO pero lleva mucho más código.
* *Productividad*: Los trabajadores se concentran en la información de negocio requerida y no en los detalles técnicos que se necesitan para obtenerla.
* *Información actualizada*: Los cambios realizados sobre una BD se hacen visibles al instante para el usuario que la requiere.
* *Seguridad*: Las BD son más potentes que los sistemas de archivos ya que permiten mayor granularidad (nivel de detalles) en la regla de seguridad sobre los datos. Es decir que se le puede dar seguridad a cada campo de los registros.
* *Manejo centralizado de la información*: Los modelos de datos se crean y administran en forma centralizada.
* *Integridad*: Es el grado de corrección de los datos, coherente entre los datos y con la realidad.
* *Transaccionabilidad*: Manejar transacciones como unidades atómicas de trabajo, es decir, se hace todo o nada.
* *Recuperación ante fallos*: Las BD están preparadas para reponerse automáticamente ante cualquier fallo de un sistema computarizado.

**Tareas del DBA**

* *Respaldo*: Asegura que la BD tenga un mecanismo de respaldo (backup).
* *Integridad*: Ofrece reglas de integridad para la BD.
* *Seguridad*: Brinda controles de acceso físicos (servidores) y lógicos (usuarios y datos) a los datos.
* *Disponibilidad*: Cumple los contratos de servicio definidos.
* *Desempeño*: Ofrece buena performance, rapidez.
* *Soporte*: Brinda asesoramiento en materia de BD. El programador puede hacerle consultas.

**Independencia de datos** (Importante en empresas grandes y por la centralización)

Es la inmunidad o robustez que tienen las aplicaciones para soportar cambios en la representación y acceso a los datos. (que el usuario no se entere de los cambios)

Hay 3 tipos:

* Independencia de dispositivos: Capacidad de soportar cambios en los almacenamientos físicos. Lo da el HW y el SO.
* Independencia física: Las aplicaciones deben soportar cambios en los archivos. Lo brinda el DBMS.
* Independencia lógica: Las aplicaciones deben soportar cambios en el modelo de datos (tablas). Lo da el DBMS y hay que cumplir el modelo de datos.

**Modelo de datos**

Determina el modo de organizar, manipular y almacenar los datos.

Tiene 3 aspectos:

* *Estructural*: La estructura para la representación de los datos.
* *Manipulación*: Cómo se procesan los datos para obtener la información.
* *Integridad*: Garantizar la corrección de los datos.

//PONER LO DE LOS TPS!

**BD RELACIONALES** (se usa hoy en día)

* Aspecto estructural: Los datos se estructuran solamente en tablas.
* Aspecto de manipulación: Define operaciones donde sus operandos y resultados son tablas. Esto permite encadenar operaciones.
* Aspecto de integridad: Define reglas de integridad implementadas como restricciones sobre las tablas (clave primaria).

Estos aspectos están demostrados matemáticamente: El de manipulación se demuestra con el álgebra relacional y el cálculo relacional. El estructural con la teoría de conjuntos. (R = {(a,b), (c,d)} )

**Operaciones** que define el álgebra relacional (todos los resultados son tablas nuevas):

|  |  |  |
| --- | --- | --- |
| a | b | c |
|  |  |  |
|  |  |  |

* Proyección: Operación unaria que tiene como operando una tabla y produce como resultado la misma tabla pero con menos campos. Adicionalmente, la operación requiere un parámetro que es una lista de campos.

=

|  |  |
| --- | --- |
| a | c |
|  |  |
|  |  |

a,c

* Selección: Operación unaria que tiene un operando y produce como resultado la misma tabla pero con menos registros. Adicionalmente posee un parámetro que es una condición.

|  |  |  |
| --- | --- | --- |
| a | b | c |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| a | b | c |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

=

a>5

* Producto cartesiano: Operación binaria que posee dos operandos y el resultado es una tabla con la suma de los campos de los operandos y que contiene todas las combinaciones posibles entre los registros.

|  |  |  |
| --- | --- | --- |
| a | b | c |
|  |  |  |
|  |  |  |
|  |  |  |

|  |  |
| --- | --- |
| d | e |
|  |  |
|  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| a | b | c | d | e |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

=

x

* Reunión: Operación binaria que tiene dos operandos y un parámetro que es una lista de campos. El resultado es el producto cartesiano de los operandos seguido de una selección en donde se igualan los campos del parámetro. (Une las tablas de costado)
* Unión: Operación binaria cuyos operandos deben tener la misma cantidad y calidad de campos. El resultado es una tabla con los mismos campos que sus operandos pero con la sumatoria de sus registros. (Une las tablas una arriba de otra)
* Intersección: Operación binaria, con la restricción de que los operandos deben tener la misma cantidad y calidad de campos y el resultado es una tabla con los mismos campos que sus operandos pero con los registros comunes a ambos.
* Diferencia: Operación binaria con la restricción de igual cantidad y calidad de campos en sus operandos y el resultado es una tabla con los mismos campos que sus operandos pero con los registros que pertenecen al primero y no pertenecen al segundo.
* División: Es la división de conjuntos de sus operandos.

El motor de BD tienen la inteligencia para simplificar las operaciones que se codifiquen, por lo tanto, se puede escribir de la forma que mejor se entienda.

Ejemplo modelo de datos y operaciones:

|  |
| --- |
| CLIENTES |
| Dni-Cliente |
| Nombre-Cliente |
| Direc-Cliente |
| Local-Cliente |

|  |
| --- |
| FACTURAS-DET |
| Nro-Fact |
| Cod-Prod |
| Cantidad |

|  |
| --- |
| PRODUCTOS |
| Cod-Prod |
| Nombre-Prod |
| Precio-Unit |

|  |
| --- |
| FACTURAS |
| Nro-Fact |
| Fecha-Fact |
| Dni-Cliente |

//ESCRIBIR EJEMPLOS

**SQL** (Standard Query Language)

Surge en la década del 70 como la propuesta para manipular las BD Relacionales, por lo tanto está basado en el álgebra relacional.

Está estandarizado: - 92’ ANSI – SQL: SQL92 (en uso)

- 99’ SQL3

Tiene operaciones set at time que traba siempre sobre conjuntos o tablas, en contra de row at time que trabaja sobre registros.

Es un lenguaje HOST: está metido en un lenguaje de programación.

C#

-

-

-

-

-

SQL

SQL

Server

Devuelve

tabla

**Sentencias**

* DDL (Data Definition Language): Define el modelo de datos.

. Create table: crea

. Alter table: modifica estructura

. Drop table: borra

* DML (Data Manipulation Language): Manipula los registros de la tabla.

. Insert

. Update: actualiza

. Delete

. Select: Lee

**Sentencia ‘select’** (sentencia de consulta)

Tiene 3 cláusulas fundamentales:

* *Select* = Proyección
* *From* = Producto cartesiano x
* *Where* = Selección

Orden de ejecución:

3 → *Select* <campos>

1 → *From* <tablas>

2 → *Where* <condición>

Ejemplo:

→ Mostrar el DNI y nombre de los clientes que hayan comprado en una fecha determinada.

*Select* Dni-Cliente, Nombre-Cliente

*From* Clientes, Facturas

*Where* Fecha-Fact = 5/8 and Clientes.Dni-Cliente = Facturas.Dni-Cliente

**Distinct** (para eliminar duplicado – usarlo solo si es necesario)

Ejemplos *select*:

→ Los clientes de Rafaela:

*Select* Dni\_Cliente, Nombre //No se pone el *distinct* porque no podría haber duplicados

*From* Clientes

*Where* Localidad\_Cliente = ‘Rafaela’

→ Los clientes (DNI, nombre y dirección) que me compraron el producto 4 hoy:

*Select* *distinct* Clientes.DNI\_Cliente, Nombre\_Cliente, Direccion\_Cliente

*From* Clientes, Facturas, Facturas\_Det

*Where* Clientes.DNI\_Cliente=Factura.DNI\_Cliente and Facturas.Num\_Factura=Facturas\_Det.Num\_ Factura and Fecha\_Factura=12/8 and Cod\_Producto=4

**Comodín**

→ Listado de todos los datos de los productos de la empresa:

*Select* **\*** //En vez de poner todo, pongo ‘***\**’** que muestra todos los campos. Usar cuando no hay where.

*From* Productos

**Alias**

→ Clientes (DNI) de Rafaela que me compraron este año

*Select distinct* **C**.DNI\_Cliente **as DNI** //Alias para los campos para que devuelva como nombre el alias.

*From* Clientes **C**, Facturas **F** //Alias para tablas, para acortar código.

*Where* **C**.DNI\_Clientes=**F**.DNI\_Cliente and Localidad=Rafaela and Fecha\_Factura>=1/1/2015

**Constantes**

→ Poner UCSE a todos los clientes de Rafaela

*Select* DNI\_Clientes, **‘UCSE’** as Universidad //Fijo para todos los registros.

*From* Clientes

*Where* Localidad=‘Rafaela’

→ Precio total del producto 5 en la factura número 1234

*Select* Precio\_Unitario \* Cantidad as Subtotal //Campo calculado

*From* Productos P, Facutras\_Det FD

*Where* P.Cod\_Producto=5 and P.Cod\_Producto=FD.Cod\_Producto and Num\_Factura=1234

→ Cantidad vendida del producto 6 en la factura 2222. Mostrar código de producto y cantidad.

*Select* Cod\_Producto, Cantidad

*From* Facturas\_Det

*Where* Cod\_Prodcuto=6 and Num\_Factura=2222

**CONECTORES**

*Lógicos*: Unen dos condiciones transformando la expresión en una condición compuesta. *And, not, or*.

*De inclusión*: Para simplificar las sentencias con lógicos. |In/not in|

|= > < |, |all any|

Nro\_Fact **in** (1234, 2231, 4371) -igual a → Nro\_Fact = 1234 **or** Nro\_Fact = 2231 **or** Nro\_Fact = 4371

Nro\_Fact **not in** (1234, 2231, 4371) -igual a → Nro\_Fact 1234 **and** Nro\_Fact 2231 **and** Nro\_Fact 4371

Nro\_Fact **> all** (1234, 2231, 4371) -igual a → Nro\_Fact > 4371

Nro\_Fact **> any** (1234, 2231, 4371) -igual a → Nro\_Fact > 1234

**SUBCONSULTA**

Es una sentencia *select* completa embebida en la cláusula *where* de otra sentencia *select* completa utilizando conectores de inclusión. La tabla que genera la subconsulta se desecha por lo que no se pueden usar sus campos para mostrar.

*Select*

*From*

*Where* … in **(*Select***

***From***

***Where*)**

Dos tipos:

* *Anidadas*: Son aquellas que no hacen referencia a ningún objeto de la consulta exterior. El orden de procesamiento lógico es: primero se ejecuta la subconsulta una vez y luego se ejecuta la consulta exterior una vez. //Usar esta.
* *Correlacionadas*: Son aquellas que hacen referencia a objetos de la consulta externa. El orden de procesamiento lógico es: primero se ejecuta la consulta externa una vez, y por cada registro a validar en su cláusula *where* se ejecuta la subconsulta.

Ejemplo anidada:

→ Los clientes que me compraron este mes.

*Select* DNI\_Cliente, Nombre\_Cliente

*From* Clientes

*Where* DNI\_Clientes in (*Select* DNI\_Clientes

*From* Facturas

*Where* Fecha 1/8)

Tiene mejor desempeño la subconsulta porque genera tablas más chicas.

Para las inclusiones (in) se puede usar cualquier forma. Las no inclusiones, únicamente con subconsulta. Cuando hay que proyectar campos de diferentes tablas, únicamente se resuelve con producto cartesiano.

Si quiero mostrar un campo de la tabla que se usa en el subconjunto, no podría por lo que tendría que hacerlo con producto cartesiano.

Si quiero los clientes que no me compraron este mes o que nunca me compraron, tendría que usar si o si subconjunto.

Cosas a tener en cuenta:

* No poner distinct en la subconsulta ni \*.
* Tener en cuenta que los campos del where sean iguales que los campos del select de la subconsulta (tipo y cantidad).
* No usar distinto en el producto cartesiano.
* Puede haber varios niveles de subconsulta.
* En el where principal puede haber una subconsulta y otras condiciones.

Ejemplo correlacionada:

→ Los clientes que no me compraron este mes.

*Select* DNI\_Cliente, Nombre\_Cliente

*From* Clientes

*Where* (TipoDNI, NumDNI\_Clientes) not in (*Select* TipoDNI, NumDNI\_Clientes

*From* Facturas

*Where* Fecha ≥ 1/8 and Clientes.Localidad = ‘Sunchales’)

EJERCICIOS

→ Los productos no vendidos este mes.

**Con subconjunto**

*Select* \*

*From* Productos

*Where* Cod\_Producto not in (*Select* Cod\_Prodcuto

*From* Facturas\_Det

*Where* Num\_Factura in (*Select* Num\_Factura

*From* Facturas

*Where* Fecha 1/8))

**Con producto cartesiano**

*Select* \*

*From* Productos

*Where* Cod\_Producto not in (*Select* Cod\_Prodcuto

*From* Facturas\_Det FD, Facturas F

*Where* FD.Num\_Factura = F.Num\_Factura and Fecha 1/8)

//El not in no se puede hacer con producto cartesiano.

→ Los clientes que nunca me compraron.

*Select* \*

*From* Clientes

*Where* DNI\_Cliente not in (*Select* DNI\_Cliente

*From* Facturas)

**GRUPO**

Un grupo es un conjunto de registros que tienen valores iguales en uno o más campos. Esos campos son los que definen el grupo.

Grupo

Fecha-Fact = 5/8

Grupo

Cod-Cliente = 3

|  |  |  |
| --- | --- | --- |
|  | FACTURAS |  |
| Nro-Fact | Cod-Cliente | Fecha-Fact |
| 1 | 3 | 5/8 |
| 2 | 2 | 5/8 |
| 3 | 1 | 5/8 |
| 4 | 3 | 5/8 |
| 5 | 3 | 6/8 |
| 6 | 2 | 6/8 |
| 7 | 4 | 6/8 |
| 8 | 4 | 6/8 |

Objetivo: sacar medidas resumen, estadísticas. Agrupar para sacar información del grupo.

SQL - Orden de ejecución:

4- Select

1- From

2- Where

**3- Group by <campos>** (Agrupa. Es más lento que el distinct)

Ejemplo:

*Select* Cod-Cliente

*From* Facturas

*Where* Fecha 5/8

*Grupo By* Cod-Cliente

Como resultado sale lo siguiente, y es lo que se le pasa al select:

|  |
| --- |
| Cod-Cliente |
| 3 |
| 2 |
| 1 |
| 4 |

**Funciones de resumen**:

Son funciones que se utilizan en las cláusulas posteriores en el orden del procesamiento lógico del Group By y permiten extraer medidas de resumen de cada uno de los grupos. Es necesario que esté el group by

Las medidas de resumen son:

* COUNT (\*): Cuenta la cantidad de registros de cada grupo. Cuanta nulos.

COUNT(<campo>): No cuenta nulos.

COUNT(Distinct <campo>): Cuenta la cantidad de campos distintos. (Ej: cantidad de vendedores en un detalle)

* SUM (<campo>): Sumatoria del <campo> de cada grupo.
* AVG (<campo>): Promedio del <campo> de cada grupo.
* MAX (<campo>): Máximo del <campo> de cada grupo.
* MIN (<campo>): Mínimo del <campo> de cada grupo.

Ejemplo:

→ Cantidad de ventas de cada cliente a partir del 5/8

*Select* Cod-Cliente, **count**(\*) as Cantidad-Ventas

*From* Facturas

*Where* Fecha-Fact 5/8

*Grupo By* Cod-Cliente

El resultado del group by es todo esto, y es lo que se muestra en el select:

|  |  |
| --- | --- |
| Cod-Cliente | Cantidad-Ventas |
| 3 | 3 |
| 2 | 2 |
| 1 | 1 |
| 4 | 2 |

El motor de base de datos reconoce las funciones resumen que se escriben en todas las sentencias y las calcula al momento de agrupar.

→ Para sacar además, la fecha de la última venta

*Select* Cod-Cliente, count(\*) as Cantidad-Ventas, **max**(Fecha-Fact) as Ultima-Venta

*From* Facturas

*Where* Fecha-Fact 5/8

*Grupo By* Cod-Cliente

→ Si quiero mostrar el nombre del cliente, hay que hacer producto cartesiano y agregar el campo en el group by.

Agrupar siempre por las claves.

→ Cantidad de productos que se vendieron este año. Mostrar Cod-Prod, Nom-Prod, Cantidad total vendida y monto facturado de cada producto.

*Select* P.Cod-Prod, Nom-Prod, sum(Cantidad) as Cant-Total, sum(Cantidad\*Precio-Unitario) as Monto-Total

*From* Facturas F, Facturas-Det FD, Productos P

*Where* P.Cod-Prod = FD.Cod-Prod and F.Num-Fact = FD.Num-Fact and Fecha-Fact 1/1/2015

*Grupo By* P.Cod-Prod, Nom-Prod

→ Cantidad de clientes que tengo registrados

*Select* count(\*)

*From* Cliente

→ Cantidad de ventas totales de este año

*Select* count(\*) as Cant-Total-Ventas

*From* Facturas

*Where* Fecha-Fact 1/1/2015

**HAVING**

Filtra lo del Group By

**ORDER BY** <campo> ASC|DESC

Ordena según el campo o función sobre un campo

**INTO**

Pone el resultado de una consulta en otro lado. Almacena en una tabla temporal.

Orden de ejecución

6-Select

7-Into

1-From

2-Where

3-Group By

4-Having

5-Order By

**DML**

**Insert**

*Insert [Clientes] (id, nombre, status)*

*Values (1, ‘yo’, 0)*

**Update**

*Update Clientes*

*Set nombre = ‘El’*

*Where status = 0*

**Delete**

*Delete Clietes*

*Where status = 0*

**DDL**

**Create**

*Create table Cliente(id int not null, Nombre Varchar(50) null, status char(1) null)*

**Alter**

*Alter table Clientes add Apellido Varchar(50)* //agrega un campo

*Alter table Clientes alter column Apellido char (100)* //modifica un campo

*Alter table Clientes drop column Apellido* //elimina

**Drop**

*Drop table [Clientes]*

**PARCIAL - 21/10**

Ejercicios de Max y Min, subconsulta con Not In y count con campo o \* o distinct

Prod cart con toda la cosa

Subconsulta se usa con max y min

Al usar MAX y MIN se puede usar ‘=’ antes de la subconsulta, sino usar IN

//BUSCAR TIPO DE DATOS DE SQL

App

DBMS

SO

BD en disco.

SQL

**ANSI-SPARC**

Divide la estructura de la BD en 3 niveles:

* Externo: Administra las diferentes visiones o vistas que tienen los diferentes usuarios o roles que utilizan las aplicaciones. Se refiere a la forma en que un usuario individual ve los datos. Pueden existir varias visiones o vistas y cada una abarca solamente una porción del modelo lógico que corresponde al perfil del usuario interesado. (Que no todos vean todo por seguridad)

• MAPPING E-C

* Conceptual: Administra la estructura lógica de los datos. Mantiene la representación del *modelo de datos* lógico, manejando cuestiones de estructuras, de seguridad, de manipulación y de integridad. Este nivel esconde los detalles del almacenamiento físico y se concentra en describir entidades, atributos, tipos de datos y relaciones.

•MAPPING I-C

* Interno: Es el de mayor relación con el SO y contiene la administración de los *archivos convencionales* en MS. Administra la estructura interna de los archivos, la organización interna y los métodos de acceso.

MAPPING Interno-Conceptual: Administra la relación entre el nivel conceptual y el interno, es decir, como se distribuyen los registros de las tablas del nivel conceptual en los diferentes archivos físicos del nivel interno. Garantiza la independencia física de datos.

MAPPING Externo-Conceptual: Administra la relación entre el modelo de datos conceptual y las diferentes vistas del nivel externo. Garantiza la independencia lógica de datos.

**Cosas a tener en cuenta al definir el nivel interno**:

→ Espacio de almacenamiento.

* Tamaño.
* Tasa de crecimiento.
* Velocidad de acceso: velocidad de discos, concurrencia de discos, distribución de los datos en los archivos (mapping i-c), RAID(\*).

*//si se dividen los archivos en diferentes discos, se aumenta la velocidad de búsqueda.*

(\*)RAID: Arreglo de disco físico.

Permite que varios discos trabajen como uno.

Se puede hacer mediante SW (con el SO) o HW (con placas especiales entre disco y BD)

*Tipos*:

* RAID 0: divide todo lo que se almacena en una unidad en los discos disponibles.
* Duplica la velocidad de acceso.
* No tiene redundancia. Si se rompe un disco, se pierde el archivo.
* RAID 1: todo se almacena en todos los discos (duplicado).
* Redundancia. Mejora el desempeño de lecturas.
* Pérdida de la mitad del almacenamiento físico. Pequeña penalidad en la escritura.
* RAID 5: Para 5 discos o más. Usa uno para paridad.
* Mejora en 2/3 la velocidad de acceso. Tiene redundancia.
* Mínimo 3 discos para armar el RAID. Pierdo 1/3 de espacio de almacenamiento.
* RAID 10: Combinación entre 0 y 1. Divide los discos a la mitad: arriba usa RAID 0 y abajo 1
* Duplica la velocidad de acceso. Tiene redundancia.
* Mínimo 4 discos para armar el RAID. Se pierde el 50% del espacio de almacenamiento.

Evolución de Librerías de acceso a datos:

* DBLibrary: librerías que proporcionaba directamente el fabricante del DBMS. Cada librería tenía su forma particular de usarse y las aplicaciones no tenían portabilidad.
* ODBC: librería fabricada por Microsoft cuyo objetivo es la portabilidad en las aplicaciones cliente. Se basa en una capa de librerías genéricas conectadas a una serie de drivers. Estos drivers son los provistos por los fabricantes de DBMS. La programación era de muy bajo nivel.

ODBC

drivers

drivers

drivers

* OLEDB: fabricado por Microsoft con la misma arquitectura y objetivo que ODBC pero incorpora características de la programación orientada a objetos. La programación sigue siendo de bajo nivel. Los drivers se llaman providers pero son lo mismo.
* DAO (Data Access Object): fabricada por Microsoft que encapsula ODBC permitiéndole al programador programar en un nivel más alto. (se pone primero DAO, después ODBC y después las librerías) DAO se conecta directamente con BD Access.

DAO

ODBC

drivers

drivers

drivers

* RDO (Remote Data Object): evolución de DAO agregándole características como por ejemplo soporte para BD distribuidos. Usa ODBC también.
* ADO (Active Data Object): Encapsula OLEDB brindándole al programador una interface de alto nivel.

ADO

OLEDB

providers

providers

providers

* JDBC: mismas características de ODBC pero para Java y con interfaces de alto nivel. Es un lenguaje interpretado. Los de antes eran compilados.
* ADO.NET: evolución de ADO para la plataforma .NET de Microsoft. Tiene OLEDB también.
* Mapeadores objeto relacional: son librerías que funcionan por encima de las librerías tradicionales de alto nivel. Su objetivo es mapear automáticamente los objetos de negocio del programa a las tablas de la BD relacional. Los mapeadores generan automáticamente la sentencia de SQL. Ejemplos: NHibernet (para .net), Hibernet (java), Linq (Microsoft), Entity framework (Microsoft), Data nucleus.

ADO.NET

OLEDB

providers

providers

providers

NHibernate

C#

**RECOVERY**

Transacción: unidad de trabajo lógica que comprende por lo regular a varias sentencias SQL. Es una unidad de trabajo porque todas las operaciones dentro de la transacción se toman como un todo y también es una unidad de recuperación porque si algunas de las operaciones falla, se vuelve toda la transacción atrás. Las transacciones empiezan con un BEGIN TRANSACTION y pueden finalizar con un COMMIT si la transacción terminó correctamente o un ROLLBACK si la transacción debe volverse atrás o deshacerse.

Una sola sentencia se considera transacción porque puede que tenga que actualizar varios registros y por si hay algún fallo de energía.

BEGIN TRANSACTION

UPDATE cuenta1

UPDATE cuenta2

IF OK

COMMIT

ELSE

ROLLBACK

Propiedades de las transacciones (ACID): (FINAL!!!)

* *Atomicidad*: las transacciones son atómicas, se ejecuta todo o nada.
* *Consistencia*: dejan siempre a las BD en un estado consistente. Cuando arranca una transacción, la BD está en un estado consistente. Cuando finaliza, la consistencia debe seguir estando garantizada. Esto último no ocurre en el medio del procesamiento de una transacción.
* *Aislamiento*: Las transacciones están aisladas entre sí, es decir, una transacción no interfiere en el procesamiento de otra y ambas mantienen la consistencia de la BD. (concurrencia)
* *Durabilidad*: Una vez que una transacción es confirmada (COMMIT), las modificaciones que realizó en la BD son permanentes más allá de que ocurra una caída posterior.

Recuperación de transacciones: Cuando una transacción es confirmada, se establece un COMMIT POINT o punto de confirmación, y corresponde al final de una unidad lógica de trabajo y a un estado de consistencia de la BD. En cambio, cuando una transacción de revertida (ROLLBACK) todas las modificaciones deben volverse al estado en que se encontraban en el COMMIT POINT anterior.

Log transacciones (T-Log): El DBMS, para cada BD, mantiene en disco un log de transacciones en donde va almacenando secuencialmente todas las operaciones que se realizan sobre los datos, como así también la información necesaria para revertirlas. Se utiliza el protocolo de escritura adelantada que dice que antes de realizar cualquier modificación en la BD se debe persistir la operación en el T-Log. Cada BD tiene los archivos donde están las tablas y un archivo para T-LOG (siempre tiene que estar en un disco aparte por la velocidad). Las BD primero se escriben en el LOG y después en la tabla.

La BD usa la RAM como caché (deja las tablas en RAM para ser más rápido)

Protocolo de escritura adelantada: Escribe primero en LOG que está en disco, después los pasa a RAM donde están las tablas y después por detrás se actualizan las tablas en disco (check point). Esto garantiza la recuperación transacciones (recovery).

El Log tiene una parte activa y otra pasiva. La activa contiene las operaciones de las transacciones que aún no finalizaron. En cambio, la pasiva, contiene las operaciones de aquellas transacciones que ya finalizaron. No guardar la parte pasiva o hacerle backup para que no ocupe espacio.

||||||||||||| ||||||||||||| ||||…..

T1 terminada

T2 terminada

T3 ejecutándose

Pasivas

Activa

CHECK POINT: Son momentos regulares en el tiempo en que el motor de BD persiste en disco todas las modificaciones existentes en los buffers de memoria RAM. Adicionalmente se registra en el T-Log el momento en que se realizó ese check point.

m

m

||||||||||||| ||||||||||||| ||||…..

RECOVERY:

* *Falla Local*: Se ejecuta cuando una transacción en particular hizo rollback e implica deshacer los cambios únicamente de esa transacción. Para eso debe retroceder en el T-Log a partir de la ejecución del rollback e ir revirtiendo todos los comandos ejecutados por esa transacción hasta el comienzo de la misma. Deshace todos los cambios en memoria, sin importar si las transacciones están en disco o en memoria.
* *Falla Global*: Es la que afecta al motor de BD completo en donde hay que deshacer los comandos de las transacciones que aún no finalizaron y rehacer los comandos de aquellas transacciones que finalizaron correctamente pero sus modificaciones no fueron persistidas en disco.
* *Soft Crash* (caídas suaves): Son caídas que no comprometen los medios de almacenamiento secundario y únicamente se pierde el contenido de la RAM.

Recuperación de la BD ante un Soft Crash:

---------------------------------------------------------------------------

-----------------------------------------------

||||

T1

T2

T3

T4

T5

|  |  |
| --- | --- |
| Rehacer | Deshacer |
|  | T2 |
|  | T3 |

|  |  |
| --- | --- |
| Rehacer | Deshacer |
| T3 | T2 |
|  | T5 |
|  | T4 |

|  |  |
| --- | --- |
| Rehacer | Deshacer |
| T3 | T2 |
| T4 | T5 |

El motor de BD al iniciar ejecuta un proceso de Recovery general para dejar la BD en un estado consistente. Durante el Recovery los datos no son accesibles. El proceso inicia generando dos listas vacías: una de transacciones a rehacer y otra de transacciones a deshacer. Luego se posiciona en el T-Log en el momento del último Check Point ejecutado. Las transacciones activas en ese momento las coloca en la lista de deshacer.

Comienza a recorrer hacia adelante desde el el último check point, cuando encuentra un commit transfiere esa transacción de la lista de deshacer a rehacer. En cambio, cuando encuentra un Begin coloca esa transacción en la lista de deshacer.

Al finalizar el proceso elimina de la lista de deshacer las transacciones que empezaron a posteriori del último check point y nunca terminaron (acá elimina T5). Al final del proceso se ejecuta el rollback de las transacciones de la lista de deshacer y ejecuta nuevamente las transacciones de la lista de rehacer teniendo en cuenta la necesidad de ejecutar rollback parciales

* *Hard Crash* (caídas fuertes): Son caídas en donde hay compromiso en los medios secundarios. Ej: Se rompe un disco.

Para recuperarse de esto, implica restaurar las copias de seguridad que se han realizado en otro soporte. Una vez restaurada, si el medio de almacenamiento en donde estaba el T-Log no fue comprometido se lo utiliza para ejecutar el proceso de recovery normal con la salvedad de que se inicia en el último check point anterior de la copia de seguridad. De esta forma no hay pérdida de datos.

En cambio, si el medio del almacenamiento del Log sí se vio comprometido, los datos cargados en horarios sin copia de seguridad se perdieron.