Resumen 1er parcial – Base de datos I

Clase 1:

Evolución desde sistemas de archivo:

Algo que se tiene que recordar de las bases de datos es que las mismas se almacenan dentro de archivos. Esto quiere decir que estan sujetas a las ventajas y desventajas que tienen los diferentes medios de almacenamiento. Y cuando se habla de BD s se destacan 2 medios de almacenamiento, cada uno con sus caracteristicas, que son:

- Almacenamiento en cintas magneticas:
    - Es muy bueno para almacenar grandes volumenes de informacion por su capacidad. Buenas grabando informacion.
    - Su tipo de acceso es de forma secuencial, que quiere decir que si existe un registro N en el medio de la cinta para poder acceder a el se tienen que leer los N – 1 registros anteriores. Malas recuperando informacion.
    - No otorga acceso en linea, esto quiere decir que para poder utilizarlo un operador externo tiene que cargar la cinta.

Todas estas caracteristicas hace que las cintas magneticas no sea una buena opcion para el trabajo diario pero si cuando se necesita almacenar una BD durante mucho tiempo (Backups).

- Almacenamiento en discos rigidos:
    - Su tipo de acceso es de forma directa (tambien conocido como aleatorio). Esto quiere decir que se puede acceder directamente a un registro si se conoce la clave de acceso al mismo. Esta clave de acceso es un numero que la unidad de disco genera automaticamente mediante algun algoritmo determinado.
    - Otorga acceso en linea, esto quiere decir que no se involucra a un tercero cuando es necesario grabar o recuperar un registro.

Cuando se graba un registro hay que tener en cuenta algunas particularidades.

    - Su capacidad de almacenamiento no es tan grande como el de las cintas magneticas.

Todas estas caracteristicas hacen que los discos rigidos sean idealeas para el trabajo diario. Actualmente es el medio de almacenamiento mas utilizado de todos.

Registros:

Los datos se almacenan en registros que son una colección de elementos o valores de información relacionada, donde a cada valor se lo considera un campo del registro y esta compuesto de 1 o mas bytes. Esxisten 2 tipos de registros:

Archivos:

Un archivo es una secuencia de registros uno detras del otro. Los mismos pueden estar compuestos de 2 tipos de registros:

- De logitud fija: Todos los registros son del mismo tipo y tienen el mismo numero de bytes.
- De logitud variables: Los registros no tiene el mismo numero de bytes. Esto se puede dar porque:
    - Los registros son del mismo tipo pero alguno de sus capos es variable, como por ej. nombre del cliente
    - Los registros son del mismo tipo pero alguno de los valores de los campos tiene un valor diferente para cada registro.
    - El archivo es mixto. Esto quiere decir que almacena registros de diferentes tipos.

Grabar registros en el disco:

Los discos se digiden en bloques porque esa es la convencion que se toma para transferir informacion con la memoria RAM. Por este comportamiento los registros de un archivo se deben guardar en bloques de disco. Esto genera 2 formas de guardar los registros:

- Guardar como registro extendido: Esto quiere decir que si un registro supera al tamaño de un bloque en bytes, lo que se debe hacer es utilizar N bloques para guardar ese registro, esto quiere decir que el tamaño del registro tiene que ser menor o igual al tamaño de un bloque por N. Para seguir la pista al registro si los bloques no son consecutivos se agrega un apuntador a la siguiente parte del registro al final del bloque. Este tipo de almacenamiento se puede utilizar para registros de longitud variable y ni variable
- Guardar como registro no extendido: Esto quiere decir que no se permite que un registro sea mas grande que el tamaño de un bloque. Este tipo de almacenamiento se utiliza con registros de logitud no variable.

Organizacion logica de los archivos:

Cuando se trabaja con archivos de registros existen 2 formas de organizarlos logicamente, estas son:

- Archivos de monticulo o pila: En estos casos los registros se agregan siempre al final del organización es rapida en la incercion pero mala en la busqueda y eliminacion.
- Archivos ordenados: Consiste en tomar un campo de un registro y usarlo como clave de ordenamiento. Estos tipos de archivos son lentos para insertar y eliminar porque se debe buscar primero el registro y luego insertarlo o eliminarlo, lo que conlleva tener que reorganizar todo el archivo cada vez que se inserta o elimina. Para mejorar esto se suele utilizar un archivo maestro y otro secundario o de desvorde donde se colocan los registros a eliminar o insertar y reorganizarlo periodicamente. Las busquedas son mas rapidas.

Tecnica de dispercion, de Hashing o de algoritmo clave direccion:

Este tipo de tecnica es otra forma de tener acceso a un registro dentro de un archivo. Consiste utilizar un campo clave del registro (clave de dispercion) y aplicarle una funcion de hashing para obtener la direccion fisica del bloque donde se encuentra ese registro. Este tipo de tecnica permite un acceso mucho mas rapido a un registro en particular. Para que se pueda utilizar esta tecnica la funcion o algoritmo a utilizar tiene que cumplir los sigueintes requisitos:

- Repetible: Para un valor determinado de clave de registro el algoritmo siempre tiene que devolver el mismo valor de direccion de bloque
- Distribuido: Guardar los registros de la forma mas distribuida posible para facilitar la incersion de nuevos registros de forma ordenada.

Algo importante a tener en cuenta es que se puede dar el caso de que dos valores diferentes de campos de registro generen el mismo valor de dirección de bloque, a estos casos se lo conocen como coliciones. Para resolver estas coliciones se utiliza el area de desvorde. Para elegir esta nueva area se pueden utilizar las siguientes tecnicas:

- Direccionamiento abierto: Se parte de la direccion obtenida por el algoritmo de dispercion y se busca secuencialmente la siguiente direccion vacia.
- Encadenamiento: Consiste en agregar al final del registro un nuevo campo que es un apuntador a un bloque diferente donde se puede guardar el registro que genero el mismo valor en el algoritmo de dispercion. Esta nueva direccion es el area de desvorde.
- Dispercion multiple: Cuando se produce la colicion se utiliza un segundo algoritmo de dispercion, si este resulta en otra colicion se puede aplicar otro algoritmo de dispercion (y asi susecivamente) hasta que se decide utilizar direccionamiento abierto en ultimo lugar.

Existen 2 tipos diferentes de dispercion que son:

- Dispercion estatica: El archivo tiene que tener un numero fijo de registros y no se puede superar ese numero. En estos casos el algoritmo debe ser acotado porque no puede pasarse de la cantidad maxima de registros.
- Dispercion dinamica: El archivo no tiene un tamaño fijo de registros. En estos casos se utiliza una tecnica muy parecida a la indexacion pero que esta basada en crear un archivo con los valores resultantes del algoritmo de dispercion y no con los valores del campo clave como en la indexación.

Indices:

Un indice es una lista de valores ordenada alfabeticamente la cual se puede consultar cuando se quiere buscar un termino en particular.

En base de datos los indices son archivos que contienen 2 campos, uno es el valor de un campo clave de un registro y el otro puede ser la direccion del bloque (si son no densos) donde esta ese registro o la direccion del registro en si (si son densos). Como los valores en el archivo del indice estan ordenados se puede utilizar la busqueda binaria de forma mas eficiente.

indices de un nivel:

Existen varios tipos de indices que son:

- Indice primario: Son los que usan un campo de ordenamiento de registros en un archivo y ademas ese campo es clave (no existen 2 valores iguales para 2 o N registros diferentes).

Los registros del archivo de indices estan compuestos por:

    - El valor del campo de clave del primer registro del bloque
    - La direccion de ese bloque

Como el archivo de indice es mucho mas pequeño que el original la busqueda binaria se acelera mucho y como los valores son clave dentro del bloque tambien se puede usar busqueda binaria.

Este tipo de indice es no denso.

- Indice de agrupamiento: Son los que usan un campo de ordenamiento de registros en un archivo pero ese campo no es clave (pueden existir 2 valores iguales para 2 o N registros diferentes).

Los registros del archivo de indices estan compuestos por:

    - El valor del campo de ordenamiento del primer registro del bloque.
    - La direccion de ese bloque pero se puede organizar de 2 formas:

        - El valor del primer registro del bloque puede no coincidir con el valor en el archivo indice, esto quiere decir que en ese bloque esta el primer valor que aparece en el archivo pero no precisamente tiene que ser el del primer registro del bloque.
        - El primer valor del registro del bloque coincide con el valor del archivo de indice. Esto se logra acomodando los registros del archivo en los bloques para que el primer registro de cada bloque coincida con el valor del registro del archivo de indice. Esto supone un consumo mayor de espacio porque aumenta la cantidad de bloques en el disco.

Como el archivo de indice es mucho mas pequeño que el original la busqueda binaria se acelera mucho pero como los valores no son clave dentro del bloque se tiene que usar busqueda lineal una vez dentro del mismo.

Este tipo de indice es no denso.

- Índice secundario: Son los que usan un campo que no es de ordenamiento de registros en un archivo. Existen 2 tipos de indices secundarios:
    - Por campo clave: A este tipo de indice secundario se lo conoce como clave secundaria porque utiliza un campo de registro que es clave (no existen 2 valores iguales para 2 o N registros diferentes) en el archivo de datos

Los registros de este archivo de indice estan compuestos por:

        - Todos los valores de los registros del archivo de datos
        - La direccion del bloque donde esta ese registro o la direccion del registro.
    - Por campo no clave: Este tipo de indice secundario no utiliza un campo de registro que es clave (existen 2 valores iguales para 2 o N registros diferentes) en el archivo de datos.

Los registros de este archivo de indices se pueden implementar de 3 formas diferentes:

- Implementacion 1:

Repetir todos los valores de los registros del archivo de datos por cada direccion diferente de registro. Este tipo de indice seria denso

- Implementacion 2:

Utilizar registros de logitud variable en el archivo de indice para ir agregando las diferentes direcciones por cada valor del campo del archivo de datos.

- Implementacion 3: Utilizar un archivo de indice sin repetir los valores de los campos y un apuntador a un archivo intermedio que contiene los registros con las direcciones de los bloques del archivo de datos. En este caso se utilizan 3 archivos y esto hace al indice no denso.

Este tipo de indice acelera mucho las busquedas de los campos que no son de ordenamiento porque hace que se le pueda aplicar la busqueda binaria en el archivo de indice.

Indices multinivel:

Los indices de multinivel consiste en tomar un archivo de indice y a partir de este crear un nuevo archivo de indice del archivo original. Esto quiere decir que se crea un archivo indice del indice del archivo de datos. Esta particion en multiniveles de indices acelera notablemente la lectura de los bloques porque a diferencia de la busqueda binaria que (que descartaba la mitad de un archivo de indice) las busquedas en archivos multiindice reducen mas de la mitad del archivo.

Para este tipo de indices se suelen utilizar estructuras de arboles B y B mas para implementarlos.

Conceptos básicos de Base de Datos:

Una base de datos se define como:

“Es un conjunto de datos y las relacciones que existen entre esos datos”

Las Bds nacieron para solucionar problemas concretos que aparecen cuando se trabajan con archivos de registros puros. Estos problemas son:

- Concurrencia: Mas de una persona neceita utilizar el archivo para consultarlo o modificarlo.
- Redundancia: Se tienen muchos archivos diferentes con la misma informacion
- Seguridad: No se tiene un control sobre que personas o que procesos pueden utilizar el archivo.
- Relaciones entre los datos: No existen relaciones puras entre los datos de un archivo y los datos de otro archivo diferente.
- Inconcistencias: Esto es un sub-problema al tener redundancia de informacion y datos no relacionados. Cuando se tiene la misma informacion en archivos diferentes sin relacion se suelen tener datos desactualizados en alguno de los archivos.
- Reglas de integridad: No se tenian reglas que definieran que los valores ingresados sean del tipo correcto para el campo de un registro.

Beneficios:

Las Bds nacieron para solucionar estos inconvenientes que se daban cuando se trabajaba con archivos de registro. La soluciones que se utiliza y que todos los DBMS (Data Base Management System) (Sistema de administracion de base de datos) utilizan actualmente es:

“Todas las transacciones a los archivos de la base de datos no se hacen directamente al disco, si no que se hacen al software de DBMS y este luego es el que se encarga de hablar con el disco o el SO. Esto permite tener una cola con las diferentes transacciones que se quieran realizar en los archivos de la BD.”

Con este cambio de enfoque se logro solucionar los problemas de concurrencia, redundancia, inconsistencias, seguridad y reglas de integridad que se tenia al trabajar con archivos de registros porque dentro del software de DBMS existen diferentes modulos que analizan la informacion que se desea guardar y se conoce como es la estructura de la BD donde se quiere guardar esa informacion.

Otra gran ventaja que se tiene al utilizar los DBMS es que comenzaron a implementar diferentes esquemas de relacion entre los datos.

Clase 2:

Distintos modelos de Base de Datos:

A lo largo del tiempo fueron surgiendo distintos tipos de modelos de base de datos, estos se pueden enumerar en:

- Jerarquico (Arbol): Existe una relacion entre padre e hijo. Cada tipo de registro tiene diferentes instancias y cada instancia esta relacionada con otro registro diferente mediante un SET que es un puntero al registro relacionado. Como su organización esta hecha en forma de arbol existe un unico nodo padre que es raiz del arbol, este nodo es por donde se debe entrar cuando se quieren realizar diferentes consultas. Tambien pueden existir diferentes nodos que sean hijos de un padre pero a su vez ese nodo sea padre de un hijo que esta mas abajo en la altura del arbol, a este tipo se lo denomina nodo hijo/padre.

  Esta organización de BD comienza a tener en cuenta las relaciones que existen entre diferentes registros pero tenia muchas limitaciones porque la navegacion se tenia que realizar registro a registro y solo permitia relaciones de tipo 1 a muchos o 1 a 1.

  Para solucionar estas limitaciones se opto por extender el modelo jerarquico para poder crear “hijos virtuales” que consisten en guardar en un hijo nuevo los punteros al padre.

- Red (Plex): Es una ampliacion al modelo jerarquico que soluciona los problemas de cardinalidades. Ya no se tiene un unico punto de entrada como en el modelo jerarquico si no que se puede entrar para consultar desde cualquier registro. Tambien tenia la ventaja que los registros estaban doblemente enlazados entonces se podia navegar hacia arriva o hacia abajo dentro del arbol.

- Relacional:

Este modelo se puede dividir en 2 tipo:

  - Puro: Es el modelo mas tradicional y utilizado de la actualidad. Esta basado en el modelo relacional version 2 que propuso Codd en 1972 y en las extenciones que se le hicieron luego.

    - Orientado a objetos: Consiste en una base de datos relacional que en sus tablas guardan las relaciones y los diferentes estados de los objetos para poder ser reconstruidos luego.

- Documentales: Consiste en un motor de indexacion y busqueda para documentos. Este tipo de BD tiene que tener ciertas caracteristicas:

    - No tienen que tener un limite predeterminado para el tamaño del documento

    - Se tiene que poder buscar por cualquier palabra dentro del documento.

    Su funcionamiento es dividir al archivo de documento en diferentes partes y mediante una referencia en la pagina la vinculaba con un puntero a otra hoja.

    Cuando se quiere vincular palabras funciona de la misma forma pero lo que se hace es crear una lista de referencias a todas las hojas donde aparece esa palabra.

Niveles dentro de un Sistema de Base de Datos:

Estos niveles se dividen en 3 maneras.

Nivel externo, conceptual e interno:

- Externo: Es la vision que tiene el usuario de los datos de la BD (Es el formato de presentación de los datos). Hay que tener en cuenta que puede existir mas de un nivel externo, esto es posible porque al tener independecia del nivel conceptrual se pueden crear diferentes software o diferentes reportes que muestren con diferentes formato los datos de una misma BD.

Aqui es donde generalmente trabajan los usuarios finales que utilizan el software de formato externo y los desarrolladores de software en si.

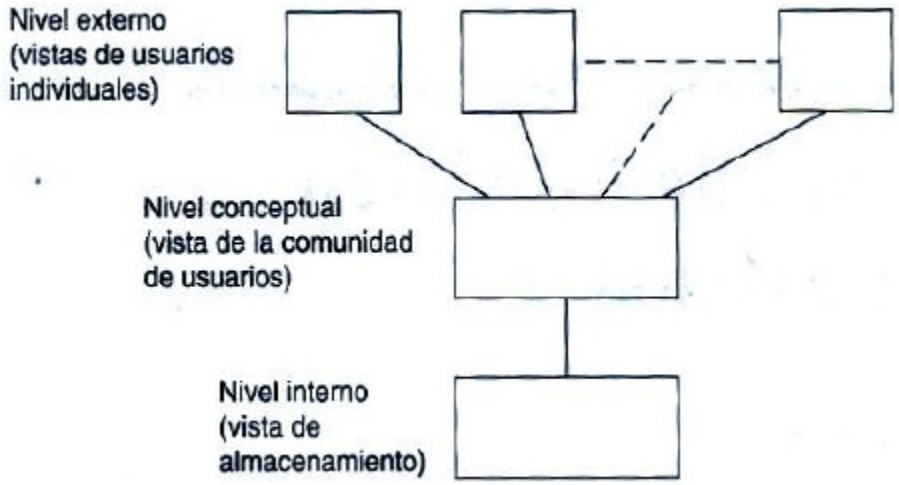
- Conceptual: Es la definicion logica de la BD (Es el modelo logico)

Aqui en donde generalmente trabajan los desarrolladores del software de formato externo y los administradores de bases de datos (D.B.A)

- Interno: Es la definicion a nivel fisico del modelo conceptual. Es que se encarga del almacenamiento fisico de los datos (Son los archivos, los tipos de archivos (binarios, texo), etc)

Aqui es donde generalmente trabajan los D.B.A

Imagen de los niveles:



Transformaciones entre los niveles:

Las transformaciones o correspondencias entre niveles permite tener una independencia de los datos con sus partes logicas y fisicas. Esto permite realizar las siguientes correspondencias o tranformaciones entre niveles:

- Externo / Conceptual – Conceptual / Externo: Esto permite que la misma estructura interna de los datos se pueda ver de diferentes formas según el software de formato externo. Por ej. Se pueden utilizar una aplicación desarrollada en java y otra en C# si se esta utilizando ODBC o JDBC para comunicarce con la misma BD. El ODBC o JDBC es el nivel conceptual.

- Interno / Conceptual – Conseptual / Interno: Se puede modificar la estructura interna si necesidad de modificar los diferentes softwares de formateo. Por ej. se puede pasar de un motor de mysql a uno de oracle si el software esta utilizando ODBC o JDBC para comunicarse con la BD. El ODBC o JDBC es el nivel conceptual.

Recursos de hardware que un DBMS necesita para funcionar:

Estos recursos son:

- Procesador
- Memoria
- Almacenamiento:

En cuanto al almacenamiento se deben tener precente dos tipos de los mismos:

    - Primario: Es donde se almacena la BD y hace que la misma funcione (Archivos, indices, memoria principal,etc)
    - De maniobra: No contiene informacion que se consulta pero esta reservado para tareas administrativas como re-indexar los archivos, tareas de backup, etc.

Formas de hacer un backup:

Existen 3 formas clasicas de hacer un backup:

- Full: Se guarda periodicamente el contenido de todo los archivos
- Incremental: Se guardan los cambios realizados en el archivo durante el dia anterior mas todos los cambios que se tenian guardados anteriormente. Es agregar secuencialmente en un backup anterior, al final del mismo, el nuevo.
- Parcial: Se guardan los cambios realizados en el archivo durante el dia anterior.

Componentes de un DBMS:

El DBMS es lo que se conoce como el motor de la base de datos (MySql, Oracle, MS SQL, etc). El mismo esta compuesto por diferentes componentes:

- DDL: Lenguaje de descripcion de datos (Es la metadata)
- DML: Lenguaje de manipulacion de datos (Generalmente es el SQL relacional)

- DCL: Lenguaje de control de datos (Es el que otorga permisos de acceso dentro de la BD)
- Modulos de procesadores de lenguaje: Realiza el analisis semantico y sintactico de la expreciones que se utilizan en los lenguajes DDL, DML y DCL
- Modulo de recuperacion: Es el que se encarga de realizar tareas de recuperacion de informacion (si es que es posible) cuando se corrompen los datos de la BD.
- Modulo de manejo de concurrencias: Es la cola de transacciones de la BD.
- Modulo de seguridad de accesos: Este modulo trabaja con el lenguaje DCL para garantizar que los usuarios tengan acceso a los datos o de restringirlo
- Modulo de backup: Es el que se encarga de hacer los backups de la BD.
- Modulo de reorganizacion: Es el que se encarga de las tareas de re-indexado de los archivos de la BD.

Utilitarios incluidos en los DBMS:

Estas son la diferentes herramientas o utilidades que vienen incluidas (o deverian venir) dentro de todos los software de DBMS. Se pueden dividir en 2 secciones:

- Herramientas de la seccion frontal: Estas son las que utilizan los usuarios finales o los programadores para poder tener acceso a los datos. Estos accesos, y las diferentes relgas que se aplican a ellos, siempre se realizan a travez del DBMS. Estas herramientas puden ser:
    - Generadores de reportes: Son herramientas que se conectan a la BD para obtener diferentes informes en base a los datos almacenados.
    - Generadores de graficos: Tambien son herramientas de reportes pero muestran la informacion graficamente (Graficos de torta, de barras, etc)

    - Generadores de codigo: Herramientas que pueden generar sentencias SQL para realizar diferentes consultas.
    - Herramientas CASE: Son las tipicas herramientas CASE.
- Herramientas de la sección posterior: Estas son las que suele utilizar los DBAs para realizar diferentes tareas administrativas sobre la BD. Hestas herramientas pueden ser:
    - Herramientas de carga de datos: Se parte de un archivo externo a la BD y el DBMS debe bloquear la estructura donde se estan cargando los datos (una tabla en particular, una BD, etc). Para poder realizar la carga la estructura debe existir y estar vacia, esto impica que el DBMS se tiene que estar ejecutando en modo exclucivo para que nadie pueda acceder a el.
    - Herramientas de recarga de datos: Se parte de un archivo externo a la BD y se carga en una nueva BD que tiene que estar vacia (sinestructuras ni datos generados). El DBMS se encarga de validar la estructura de los datos. Generalmente este tipo de herramientas tienen asociados algun tipo de archivo de error.
    - Herramientas de estadisticas: Se refieren a la estructura de la BD y a diferente informacion en cuanto a su uso. Se utilizan para ver si el rendimiento de la BD es el adecuado.
    - Herramientas de reorganizacion de la BD: Generalmente se utilizan para mejorar el rendimiento de la BD (generar nuevamente los indices, etc) o para recuperar espacio en el diso si la BD pesa demaciado y se cuenta con poco espacio. Dependiendo de la tarea la reorganizacion se puede hacer de 2 formas:
        - En caliente: Se realiza “on the fly”, no hace falta sacar a los usuarios y la BD se puede seguir utilizando.
        - En frio: En necesario sacar a los usuarios para que nadie excepro el DBMS pueda tener acceso a la BD.

    - Herramientas de optimizacion: Estas trabajan en conjunto con las herramientas de estadistica para poder mejorar y acelerar los tiempos de respuesta de las consultas a la BD.

Tipos de usuario:

Existen 3 grandes grupos de usuarios que utilizan los DBMS directa o indirecamente, esto son:

- Programadores de aplicaciones: Son los responsables de escribir los programas de aplicaciones de bases de datos en algun lenguaje de programacion determinado (Java, C++, PHP, etc). Esos programas acceden a la BD haciendo solicitudes apropiadas al DBMS.
- Usuarios finales: Estos son los que interacturan con la BD desde sus terminales en linea o desde sus estaciones de trabajo mediante las aplicaciones desarrolladas por los programadores anteriormente mencionados o mediante las diferentes aplicaciones que viene integradas con el software del fabricante del DBMS (Generalmente para poder utilizar esas interfazes hay que tener un poco de conocimiento tecnico)
- Administrador de datos (D.A.): Es la persona que toma las desiciones de estrategia y politicas con respecto a los datos dentro de la organizacion
- Administrador de base de datos (D.B.A): Es la persona que proporciona el apoyo tecnico necesario para implementar las diferentes deciciones tomadas por el D.A. Es el responsable del control general del sistema de BD a nivel tecnico.

Rol del DBA, funciones y responsabilidades:

Las funciones que puede tener un DBA dependen siempre de los lineamientos de las organizaciones. A pesar de esto un DBA siempre deveria poder relizar alguna de las siguientes:

- Definir el esquema conceptual: Una vez que el DA decidio el contenido de la BD en un nivel abstracto, entonces el DBA debe crear el esquema conceptual (o logico) correspondiente mediante el lenguaje DDL conceptual del DBMS.

- Definir el esquema interno: El DBA también debe decidir la forma en que van a ser representados los datos fisicamente en la base de datos almacenada. Una vez realizado el diseño físico, el DBA deberá crear la definición de la estructura de almacenamiento correspondiente (es decir, el esquema interno), utilizando el DDL interno. Además, también deberá definir la transformación conceptual/interna asociada.
- Establecer un enlace con los usuarios: Es asunto del DBA enlazarse con los usuarios para asegurar que los datos necesarios estén disponibles y para escribir (o ayudar a escribir) los esquemas externos necesarios, utilizando el DDL externo aplicable. Además, también es necesario definir las transformaciones externas/conceptual correspondientes. Otros aspectos de la función de enlace con los usuarios incluyen la asesoría sobre el diseño de aplicaciones, una capacitación técnica, ayuda en la determinación y resolución de problemas, así como otros servicios profesionales similares.
- Definir las restricciones de seguridad y de integridad: Las restricciones de seguridad y de integridad pueden ser vistas como parte del esquema conceptual. El DDL conceptual debe incluir facilidades para especificar dichas restricciones.
- Definir las politicas de vaciado y recarga: En el caso de que se produzca un daño en cualquier parte de la base de datos (ocasionado, por ejemplo, por un error humano o por una falla en el hardware o en el sistema operativo) resulta esencial poder reparar los datos afectados con el mínimo de demora y con tan poco efecto como sea posible sobre el resto del sistema. El DBA debe definir e implementar un esquema apropiado de control de daños que comprenda la descarga o "vaciado" periódico de la base de datos en un dispositivo de almacenamiento de respaldo y la recarga de la base de datos cuando sea necesario, a partir del vaciado más reciente.

- Supervisar el rendimiento y responder a los requerimientos cambiantes: El DBA es el responsable de organizar el sistema de tal manera que se obtenga el rendimiento "ideal para la organizacion" y de hacer los ajustes apropiados (es decir, afinar) conforme las necesidades cambien. Por ejemplo, podría ser necesario reorganizar de vez en cuando la base de datos almacenada para asegurar que los niveles de rendimiento se mantengan aceptables. Todo cambio al nivel interno de almacenamiento físico debe estar acompañado por el cambio correspondiente en la definición de la transformación conceptual/interna, de manera que el esquema conceptual permanezca constante.

Importancia del DBA en la organización:

El DBA es importante porque como se menciono en sus funciones es responsable que el software de DBMS funcione adecuadamente y porque es el que administra todos los datos dentro de una organización.

Clase 3:

El Modelo Relacional:

Este modelo fue creado por Codd y esta basado en conjuntos, por eso se aplican ciertas propiedades de los mismos.

El modelo final se creo en los 70 y se lo conoce como modelo relacional version 2 y esta basado en las siguientes propiedades:

- Presentacion (Percepcion): La forma mas facil de ver la informacion es si esta tabulada en filas y columnas.
- Consistencia: Deben existir reglas de integridad
- Lenguaje natural: Se debe poder acceder a la informacion utilizando un lenguaje natural

Estructura:

Atributos, dominios, tuplas y relaciones:

La definiciones de estas 4 caracteristicas del algebra relacional son:

- Atributos: Es la menor unidad de informacion con sentido propio que no posee estructura desde el punto de vista del modelo. El significado de cada atributo tiene que estar definido en el diccionario de datos. Hay que tener en cuenta que los atributos pueden ser:
    - Sin estructura (Atomicos): Estos son los tipos de atrbutos que se pueden definir en el modelo. Son atomicos porque no son compuestos. No son la suma de 2 atributos como por ej. DNI+NOMBRE
    - Con estructura (No son atomicos): Estos sipos de atributos no se aceptan en el modelo porque son compuestos. Son la suma de 2 atributos. Para poder utilizarlos en el modelo este atributo compuesto se debe descomponer en 2 atributos atomicos, por ej. DNI+NOMBRE se puede descomponer es 2 atributos, uno DNI y otro NOMBRE.

Al conjunto de todos los atributos se lo conoce como cabezera de una relacion.

El grado de la cabezera de una relacion es el numero de atributos que tiene esa cabezera. Este valor es siempre constante para una relacion.

En una BD relacional a este concepro se lo conoce como columna aunque se lo utiliza de otra forma.

- Dominios: Un dominio es un conjunto de valores legales que puede tomar un atributo. Es lo que se conoce como tipo de datos que almacena ese atributo, por ej. Enteros (INT), String (Cadena de caracteres), Date (fechas), etc. El dominio conoce todos los valores de sus atributos y define que conceptos se pueden aplicar a esos atributos. En resumen un dominio define:
    - Que es lo que puede contener un atributo
    - Que operaciones se pueden hacer dentro del dominio de un atributo.
    - Que operaciones se pueden hacer hacer en otro dominio.
    - Que comparaciones se pueden hacer dentro del dominio de un atributo.

    - Que comparaciones se pueden hacer en otro dominio.

- Tublas: Es un conjunto de valores de atributos donde cada uno de esos valores esta vinculado con un atributo de la cabezera de una relacion.

Se conoce como cuerpo de una relacion a un conjunto de tuplas.

La cardinalidad de una relacion es la cantidad de tuplas que tiene esa relacion. Este numero no es constante y puede variar dentro de la misma relacion.

En una BD relacional a este concepro se lo conoce como fila aunque se lo utiliza de otra forma.

- Relacion: Una relacion es la union de una cabezera de atributos con un cuerpo de tuplas. Dentro del algebra relacional las relaciones tienen que cumplir ciertas propiedades:
    - No pueden existir tuplas duplicadas (Esto se derifa de la teoria de conjuntos)
    - Las tuplas no estan ordenadas (Esto se deriva de la teoria de conjuntos)
    - No pueden existir atributos duplicados (Esto quiere decir que la cabecera no puede tener 2 nombre de atributos iguales y que tengan el mismo dominio).
    - Los atributos no estan ordenados.
    - Todos los atributos tienen que ser atomicos.

Existen diferentes tipos de relaciones que se pueden enumerar como:

- Relacion base: Es una relacion autonoma con nombre propio que no depende de ninguna otra para existir, tiene una unica manera de ser identificada y la misma tiene datos asociados.
- Relacion derivada: Es una relacion que existe a partir de otras relaciones, posee un nombre propio pero no tiene datos asociados a la misma si no que toma los datos de otras relaciones y los muestra como un resultado.

Definición y características:

Las caracteristicas del modelo relacional son:

- Percepcion: Es la forma en la que se muestra la informacion. La misma se muesta en una forma tabular (Esta organizada en filas y columnas).
- Restricciones:
- Lenguaje natural: Es el algebra relacional de Codd

Catálogo o diccionario del sistema:

Todo DBMS debe proporcionar una función de catálogo o diccionario. El catálogo es el lugar donde, entre otras cosas, se guardan los diversos esquemas (externo, conceptual, interno) y todas las transformaciones correspondientes (externa/conceptual, conceptual/interna). En otras palabras, el catálogo contiene información detallada (a veces denominada información de descriptores o metadatos) respecto de los distintos objetos que son de interés para el propio sistema. Ejemplos de dichos objetos son los índices, los usuarios, las restricciones de integridad, las restricciones de seguridad, etc. La información de descriptores es esencial para que el sistema haga bien su trabajo. Los DBMS actuales crean el catalogo como una BD mas del sistema, asi que a la misma se la puede consultar como una BD comun y corriente mas.

Claves candidatas y primaria:

Claves candidatas (C.C): Dentro de la cabecera de una relacion existen atributos especiales (pueden ser mas de un atributo) que se los conocen como claves candidatas. Estas tienen un papel fundamental en la identidad de una relacion y deben tener las siguientes propiedades:

- Unicidad (Unico): No deben existir dos tuplas diferentes con el mismo valor en los atributos que sean clave candidata.
- Minimalidad (Minimo, ireducible): Si una clave candidata esta compuesta por mas de un atibuto y yo saco de la cabecera alguno la tupla tiene que dejar de ser identificable (pierde su identidad). Si se saca alguno de los atributos y la tupla sigue siendo identificable (no pierde su identidad) ese sub-conjunto no era una clave candidata.

Clave primaria (P.K.): Si en una relacion existen mas de una clave candidata lo que se debe hacer es elegir una de ellas. Esta clave que se

eligio se la conoce como clave primaria (P.K.) y como es una clave candidata cumple con todas sus propiedades.

Función e importancia:

Características:

Claves externas:

Las claves externas o foraneas se utilizan para lograr tener una redundancia controlada. Si bien es verdad que se debe intentar reducir la redundancia lo mas posible, este concepro es util porque permite dejar de utilizar punteros que necesiten conocer como era la estructura del modelo para poder funcionar.

La gran ventaja de utilizar claves foraneas es que le da al modelo una integridad referencial. Esto quiere decir que por ej. queremos cargar una tupla en una relacion y uno de las campos de la cabecera es una clave foranea de otra relacion diferente, esto limita los valores que se pueden cargar a valores que solamente existan en la tabla relacionada. Esta propiedad es la que genera la identidad referencial.

En conclucion podemos definir a la clave foranea como:

“Un sub-conjunto de atributos (k) de una relacion A es clave foranea de la relacion si concuerda con un sub-conjunto de atributos de una relacion B en donde ese sub-conjunto es una clave primaria de B”

Esto quiere decir que un atributo de una relacion A es clave foranea de de la misma si existe otro atibuto en una relacion B que tiene el mismo nombre y el mismo dominio que el atributo de la relacion A y a demas es una clave primaria en la relacion B.

Concepto, ventajas y desventajas:

Actualizaciones en cascada.

Clase 4:

Álgebra Relacional

Operadores del AR: Estos nos permiten realizar tareas de consultas sobre las relaciones pero su particularidad es que no generan información nueva, solo la muestran de diferente forma en una relación resultado. Esto permite que estas nuevas relaciones tengan herencia sobre los atributos de las relaciones originales, lo que quiere decir que si un atributo es clave primaria de una relacion original tambien es clave primaria en la relacion resultado, (Cuidado, esto es cierto si el operador no quita la clave primaria de los atributos en la relacion resultado) por ej. si se mantiene la cabezera igual despues de utilizar el operador de proyeccion.

Los operadores del AR son los siguientes:

Tradicionales	**Especiales**
Unión	Proyección
Intersección	Restricción
Diferencia	Reunión
Producto Cartesiano (Extendido)	División

Definición de los operadores:

- **Unión:** Regresa una relación que contiene todas las tuplas que aparecen en una o en las dos relaciones especificadas o mejor dicho, “dadas dos relaciones A y B del mismo tipo, la unión de dichas relaciones (A UNION B) da como resultado una relación del mismo tipo con un cuerpo que contiene todas las tuplas, que aparece en A, en B o en ambas”. Como el resultado es una relacion nueva se cumple la propiedad de cierre.

Para que la unión se pueda efectuar se tienen que cumplir las siguientes condiciones:

  - Las relaciones A y B tienen que tener igual grado (La misma cantidad de atributos).

- Todos los atributos de las relaciones A y B tienen que tener el mismo nombre.
- Todos los atributos de las relaciones A y B tienen que tener el mismo dominio.

Esto quiere decir quiere decir que las cabeceras de las relaciones A y B tiene que ser iguales.

Ejemplo:

Relación A

ID	DNI	NOMBRE
1	30270067	Juan
5	20270067	Beto
10	10270067	Master

Relación B

ID	DNI	NOMBRE
1	30270067	Juan
8	50270067	Dante
10	10270067	Master

Relación resultado de A UNION B

ID	DNI	NOMBRE
1	30270067	Juan
5	20270067	Beto
10	10270067	Master
8	50270067	Dante

El resultado es una nueva relación (Cumple la propiedad de cierre) y también mantiene la propiedad de eliminar las tuplas duplicadas.

Propiedades:

- La unión es asociativa:

  A UNION (B UNION C) = (A UNION B) UNION C

- La unión es conmutativa:

  A UNION B = B UNION A

- El grado (cantidad de atributos en la cabecera) de la relación resultado es igual al grado de las relaciones A y B.

  $$G_R = G_A = G_B$$

- La cardinalidad (cantidad de tuplas de una relación) de la relación resultado esta entre la cardinalidad mas chica entre las 2 relaciones a utilizar y la suma de las cardinalidades de las 2 relaciones a utilizar.

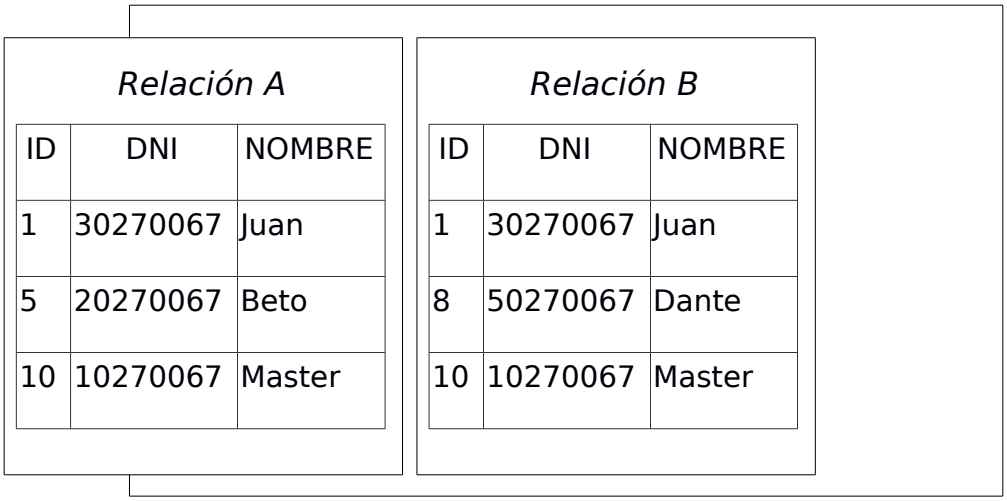
  $$C_{Menor} \leqslant C_{Resultado} \leqslant (C_A + C_B)$$

- **Intersección:** Regresa una relación que contiene todas las tuplas que aparecen en las dos relaciones especificadas (en ambas, no en una u otra) o mejor dicho, “Dadas dos relaciones A y B del mismo tipo, entonces la intersección de esas dos relaciones (A INTERSECCION B) es una relación del mismo tipo con un cuerpo que consiste todas las tuplas T, tal que T aparecen tanto en A como en B”. Como el resultado es una relación nueva se cumple la propiedad de cierre.

Para que la intersección se pueda efectuar se tienen que cumplir las siguientes condiciones:

- Las relaciones A y B tienen que tener igual grado (La misma cantidad de atributos).
- Todos los atributos de las relaciones A y B tienen que tener el mismo nombre.
- Todos los atributos de las relaciones A y B tienen que tener el mismo dominio.

Ejemplo:



Relación A

ID	DNI	NOMBRE
1	30270067	Juan
5	20270067	Beto
10	10270067	Master

Relación B

ID	DNI	NOMBRE
1	30270067	Juan
8	50270067	Dante
10	10270067	Master

Relación resultado de A INTERSECCION B

ID	DNI	NOMBRE
1	30270067	Juan
10	10270067	Master

El resultado es una nueva relación (Cumple la propiedad de cierre) y también mantiene la propiedad de eliminar las tuplas duplicadas.

Propiedades:

- La intersección es asociativa:

A INTERSECCION (B INTERSECCION C) = (A INTERSECCION B) INTERSECCION C

- La intersección es conmutativa:

A INTERSECCION B = B INTERSECCION A

- El grado (cantidad de atributos en la cabecera) de la relación resultado es igual al grado de las relaciones A y B.

$$G_R = G_A = G_B$$

- La cardinalidad (cantidad de tuplas de una relación) de la relación resultado esta entre 0 y la cardinalidad mas chica entre las 2 relaciones a utilizar.

$$0 \leqslant C_{Resultado} \leqslant C_{Menor}$$

- **Diferencia:** Genera una relación que contiene todas las tuplas que aparecen en la primera pero no en la segunda de las dos relaciones especificadas o mejor dicho, ”Dadas dos relaciones A y B del mismo tipo, entonces la diferencia entre esas dos relaciones (A DIFERENCIA B, en ese orden) es una relación del mismo tipo con un cuerpo que consiste en todas las tupias T, tal que T aparece en A y no en B”.Como el resultado es una relación nueva se cumple la propiedad de cierre.

Para que la diferencia se pueda efectuar se tienen que cumplir las siguientes condiciones:

  - Las relaciones A y B tienen que tener igual grado (La misma cantidad de atributos).
  - Todos los atributos de las relaciones A y B tienen que tener el mismo nombre.
  - Todos los atributos de las relaciones A y B tienen que tener el mismo dominio.

  Ejemplo:

Relación A

ID	DNI	NOMBRE
1	30270067	Juan
5	20270067	Beto
10	10270067	Master

Relación B

ID	DNI	NOMBRE
1	30270067	Juan
8	50270067	Dante
10	10270067	Master

Relación resultado de A DIFERENCIA B

ID	DNI	NOMBRE
5	20270067	Beto

Relación resultado de B DIFERENCIA A

ID	DNI	NOMBRE
8	50270067	Dante

Propiedades:

- La diferencia no es asociativa.
- La diferencia no es conmutativa.

- El grado (cantidad de atributos en la cabecera) de la relación resultado es igual al grado de las relaciones A y B.

$$G_R = G_A = G_B$$

- La cardinalidad (cantidad de tuplas de una relación) de la relación resultado esta entre 0 y la cardinalidad de la primer relación de las 2 relaciones a utilizar.

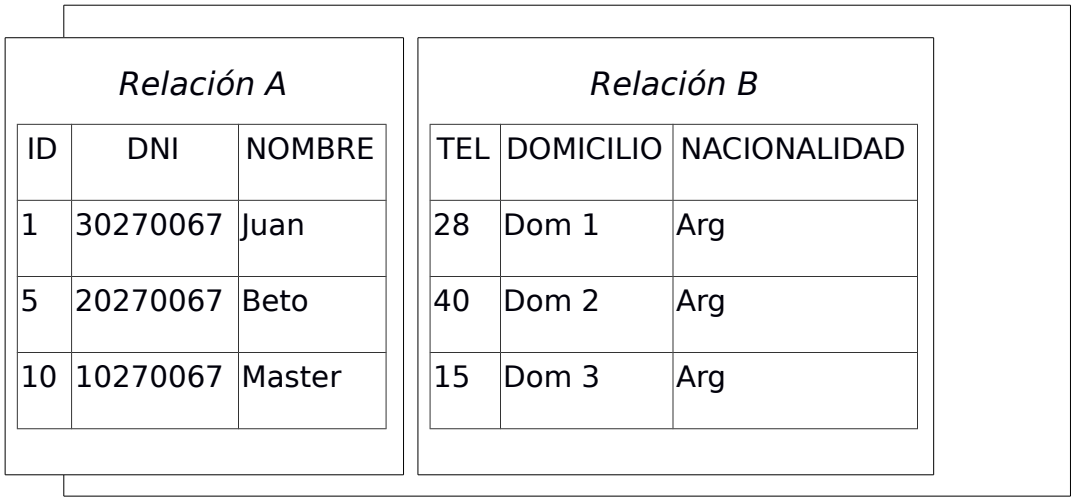
$$0 \leqslant C_{Resultado} \leqslant C_{De\,la\,primer-relacion}$$

- **Producto Cartesiano (Extendido):** Regresa una relación que contiene todas las tuplas posibles que son una combinación de dos tuplas, una de cada una de dos relaciones especificadas O MEJOR DICHO, “Definimos al producto cartesiano (relacional) de dos relaciones A y B (A PRODUCTO B) (donde A y B no tienen nombres de atributo comunes en sus cabeceras) como una relación con un encabezado que es la unión de los encabezados de A y B y con un cuerpo que consiste en el conjunto de todas las tuplas T, tal que T es la unión de una tupla que aparece en A y una tupla que aparece en B”. Como el resultado es una relación nueva se cumple la propiedad de cierre.

  Para que el producto cartesiano se pueda efectuar se tienen que cumplir las siguientes condiciones:

  - Las relaciones A y B no tienen que tener atributos con el mismo nombre en sus cabeceras. Si esto ocurre se puede utilizar el operador RENOMBRAR en alguna de las 2 relaciones antes de utilizar el producto cartesiano.

Ejemplo:



Relación A

ID	DNI	NOMBRE
1	30270067	Juan
5	20270067	Beto
10	10270067	Master

Relación B

TEL	DOMICILIO	NACIONALIDAD
28	Dom 1	Arg
40	Dom 2	Arg
15	Dom 3	Arg

Relación resultado de A PRODUCTO B

ID	DNI	NOMBRE	TEL	DOMICILIO	NACIONALIDAD
1	30270067	Juan	28	Dom 1	Arg
1	30270067	Juan	40	Dom 2	Arg
1	30270067	Juan	15	Dom 3	Arg
5	20270067	Beto	28	Dom 1	Arg
5	20270067	Beto	40	Dom 2	Arg
5	20270067	Beto	15	Dom 3	Arg
10	10270067	Master	28	Dom 1	Arg
10	10270067	Master	40	Dom 2	Arg
10	10270067	Master	15	Dom 3	Arg

Propiedades:

- El producto cartesiano es asociativo.

A PRODUCTO (B PRODUCTO C) = (A PRODUCTO B) PRODUCTO C

- El producto cartesiano es conmutativo.

A PRODUCTO B = B PRODUCTO A

- El grado (cantidad de atributos en la cabecera) de la relación resultado es igual a la suma de los grados de las relaciones A y B.

$$G_R = G_A + G_B$$

- La cardinalidad (cantidad de tuplas de una relación) de la relación resultado es el producto de las cardinalidades de las 2 relaciones a utilizar.

$$C_{Resultado} = C_A * C_B$$

- **Proyección:** Si la relación A tiene los atributos X, Y, ... , Z (y posiblemente otros) entonces la proyección de la relación A sobre X, Y, ... , Z da como resultado una relación con las siguientes características:

  - Un encabezado derivado del encabezado de A al quitar todos los atributos no mencionados en el conjunto {X,Y,...,Z}.

  - El cuerpo de la relación resultado consistente en todas las tuplas {X:x, Y:y, ... , Z:z} tal que una tupla que aparece en la relación A con el valor x para X, el valor y para Y, ... , y el valor z para Z.

  Por lo tanto, el operador de proyección produce en efecto un subconjunto "vertical" de una relación dada; es decir, ese subconjunto obtenido quitar todos los atributos no mencionados en la lista de atributos especificada y después elimina las sub-tuplas duplicadas. Como el resultado es una relación nueva se cumple la propiedad de cierre.

  Para que la proyección se pueda efectuar se tienen que cumplir las siguientes condiciones:

  - Ningún atributo puede ser mencionado más de una vez en la lista con comas de nombre de atributo.

  - Los atributos mencionados en la lista deben existir en la cabecera de la relación A.

  Existen 2 tipos de proyecciones especiales que son:

- Si la lista de nombres de atributo separados con comas menciona todos los atributos de la relación A, a la proyección se la llama “una proyección identidad”.
- Si la lista de nombres de atributo separados con comas no menciona a ninguno de los atributos de la relación A, a la proyección se la llama “una proyección de carácter nulo”.

Ejemplo:

Relación A

ID	DNI	NOMBRE
1	30270067	Juan
5	20270067	Beto
10	10270067	Master

Relación resultado de A PROYECCION {DNI, NOMBRE}

DNI	NOMBRE
30270067	Juan
20270067	Beto
10270067	Master

Propiedades:

- La proyección es una operación unaria (Se aplica solamente a una relación).

- El grado (cantidad de atributos en la cabecera) de la relación resultado esta entre 0 (proyección nula) y el grado de la relación de A (proyección identidad).

$$0 \leqslant G_{Resultado} \leqslant G_A$$

- La cardinalidad (cantidad de tuplas de una relación) de la relación resultado esta entre 0 (proyección nula) y la cardinalidad de la relación A (proyección identidad).

$$0 \leqslant C_{Resultado} \leqslant C_A$$

- **Restricción:** Si se tiene la relación A con atributos X ,Y … , Z (y quizás otros) y sea O un operador, tal que la condición C esté bien definida y dé como resultado un valor de verdad (verdadero o falso) (Restriccion booleana) para valores particulares de X o Y o cualquier otro atributo (uno solo) entonces la restricción de la relación A sobre los atributos X o Y o cualquier otro (en ese orden) da como resultado una relacion con las siguientes caracteristicas:

  - El resultado es una relación con el mismo encabezado que A y con un cuerpo que consiste en todas las tuplas t de A tal que la condición C dé como resultado verdadero para esa tupla t.

  Los operadores que se pueden aplicar dentro de las condiciones pueden ser:

Nombre	**Símbolo**
Igual	“=”
Mayor	“>”
Menor	“<”
Mayor o igual	“>=”
Menor o igual	“<=”
Distinto	“<>”

Operador lógico Y	AND
Operador lógico O	OR
Operador lógico NO	NOT

Es importante destacar que los operadores del álgebra relacional no generan nueva información en las relaciones, pero las operaciones que están compuestas por estos operadores si generan nueva información en una nueva relación.

Ejemplo:

Relación A

ID	DNI	NOMBRE
1	30270067	Juan
5	20270067	Beto
10	10270067	Master

Relación resultado de A RESTRICCION DNI = 20270067

DNI	NOMBRE
20270067	Beto

También es posible agrupar diferentes condiciones en una misma restricción separándolas por una coma.

La comparación se puede realizar con números escalares o se pueden comparar atributos de la misma relación entre si.

Propiedades:

- La restricción es una operación unaria (Se aplica solamente a una relación).
- El grado (cantidad de atributos en la cabecera) de la relación resultado es igual y el grado de la relación de A.

$$G_{Resultado} = G_A$$

- La cardinalidad (cantidad de tuplas de una relación) de la relación esta entre 0 y la cardinalidad de la relación A.
- $0 \leqslant C_{Resultado} \leqslant C_A$

- **Reunión:** SI una relación A tiene los atributos X ,Y ,... ,Z en su cabezera y un cuerpo que consiste en el conjunto de todas las tuplas X:x ,Y:y ,... ,Z:z, tal que una tupla aparece en A con el valor x para X y el valor y para Y (Atributo comun), mientras que una tupla aparece en una relacion B con el valor y para Y (Atributo comun de gual valor que en la relacion A) y el valor z para Z la reunin da como resultado una relacion con la siguientes caracteristicas:
  - La cabezera combina todos los atributos de las relaciones A y B y el atributo en comun aparece una sola vez.
  - Las tuplas de esta relacion son la combinacion de tuplas de A y B donde el atributo en comun tenia igual valor.

Para que la reunion se pueda llevar a cabo se tiene que cumplir las siguientes condiciones:

- Tienen que existir uno o varios atributos en común entre las relaciones A y B. Los mismos tienen que tener igual nombre en ambas relaciones he igual dominio (O sea que tienen que tener el mismo significado).

Ejemplo:

Relación A

ID	DNI	NOMBRE
1	30270067	Juan
5	20270067	Beto
10	10270067	Master

Relación B

ID	DOMICILIO	NACIONALIDAD
1	Dom 1	Arg
6	Dom 2	Arg
10	Dom 3	Arg

Relación resultado de A REUNION B

ID	DNI	NOMBRE	DOMICILIO	NACIONALIDAD
1	30270067	Juan	Dom 1	Arg
10	10270067	Master	Dom 3	Arg

Propiedades:

- La reunión es asociativa.

  A REUNON (B REUNION C) = (A REUNION B) REUNION C

- La reunion es conmutativa.

  A REUNION B = B REUNION A

- El grado (cantidad de atributos en la cabecera) de la relación resultado es menor a la suma de los grados de las relaciones A y B.

$$G_R < G_A + G_B$$

- La cardinalidad (cantidad de tuplas de una relación) de la relación resultado esta entre 0 y el producto de las cardinalidades de las 2 relaciones a utilizar.

$$0 \leqslant C_{Resultado} \leqslant C_A * C_B$$

- **División:**Si se tienen 2 relaciones A y B las cuales tienen un atributo en común, la division da como resultado una relacion con las siguientes caracteristicas:
  - La relacion resultado se compone de todas las tuplas de la relacion A (Dividendo) que coincidan con toda la relacion B (Divisor) en su conjunto (que coincidan con todas sus tuplas).
  - La cabezera de la relacion resultado esta compuesta por todos los atributos de la relacion A menos el atributo en comun.

Para que la division se pueda llevar a cabo se tiene que cumplir las siguientes condiciones:

- La relacion A, que es el dividendo, tiene que tener una cabezera con grado igual a 2.
- La relacion B, que es el divisor, tiene que tener una cabezera con grado igual a 1.
- Las relaciones A y B tienen que tener un atributo en comun.

Ejemplo:

Relación A

COLOR	DNI
Blanco	30270067
Rojo	20270067
Verde	30270067

Relación B

COLOR
Blanco
Verde

Relación resultado de A DIVISION B

DNI
30270069

Propiedades:

- El grado (cantidad de atributos en la cabecera) de la relación resultado es igual a uno.
- La cardinalidad (cantidad de tuplas de una relación) de la relación esta entre 0 y la cardinalidad de la relación A.

$$0 \leqslant C_{Resultado} \leqslant C_A$$

Operaciones del AR: Las operaciones del AR nos permiten generar nueva informacion, a diferencia de los operadores que no generan nueva informacion, a partir de una relacion determinada. El resultado de estas operaciones es una nueva relacion con diferente informacion que la relacion original.

Las operaciones del AR son las siguientes:

Nombre
Extender
Renombrar
Sumarizar

Definición de las operaciones:

- **Extender:** Si se tiene una relación A con un encabezado determinado, esta operación dara como resultado un encabezado igual al de A con un nuevo atributo z y con un cuerpo que consiste en todas las tuplas de A con un valor para el nuevo atributo z, el cual se calcula evaluando una expresión sobre esa tupla en la relacion A original. Esto quiere decir que extender toma una

relación y regresa otra que es idéntica a la primera, con excepción de que incluye un atributo adicional cuyos valores se obtienen mediante la evaluación de cierta expresión de cálculo especificada.

Para que esta operación se pueda realizar se tienen que tener en cuenta los siguientes puntos:

- La relación A no debe tener un atributo de nombre z y la exprecion no debe hacer referencia a z.
- Si se utiliza un operador de totales en la exprecion la relacion resultado siempre tendra una unica tupla con el resultado que genere el operador de totales.
- En las expresiones solo se pueden utilizar atributos de la relación. No se pueden utilizar números escalares.

Los operadores de totales que se pueden aplicar dentro de las expreciones pueden ser:

Nombre	**Descripcion**
Contar	Cuenta cuantas tuplas existen de un atributo determinado.
Sumar	Suma los valores de las tuplas de un atributo determinado.
Mayor	Obtiene el mayor valor de un atributo determinado.
Menor	Obtiene el menor valor de un atributo determinado.
Promedio	Obtiene un promedio con los valores de las tuplas de un atributo determinado.

Ejemplo:

Relación A

ID	DNI	NOMBRE
1	30270067	Juan
5	20270067	Beto

10	10270067	Master

Resultado de EXTENDER Relacion A AGREGAR MAYOR (ID) COMO MAX

MAX
10

Propiedades:

- El operador extender es unario (Se aplica solamente a una relación).
- El grado (cantidad de atributos en la cabecera) de la relación resultado es igual al grado de la relación de A mas 1.

$$G_{Resultado} = G_A + 1$$

- La cardinalidad (cantidad de tuplas de una relación) de la relación resultado puede tener 2 resulados:

$$Si\ en\ la\ expresion\ se\ utilizo\ un\ operador\ de\ totales: C_{Resultado} = 1$$

$$Si\ en\ la\ expresion\ no\ se\ utilizo\ un\ operador\ de\ totales: C_{Resultado} = C_A$$

- **Renombrar:** Este operador se utiliza para cambiar el nombre de alguno de los atributos que estén dentro de una cabecera de una relación.

Para que esta operación se pueda realizar se tienen que tener en cuenta los siguientes puntos:

- El nuevo nombre que tendra el atributo a renombrar no puede ser igual al nombre de otro atributo difrerente dentro de la cabecera de la relacion.

Ejemplo:

Relacion A

ID	DNI	NOMBRE
1	30270067	Juan
5	20270067	Beto
10	10270067	Master

Resultado de Relacion A RENOMBRAR DNI COMO DOCUMENTO

ID	DOCUMENTO	NOMBRE
1	30270067	Juan
5	20270067	Beto
10	10270067	Master

Propiedades:

- El operador renombrar es unario (Se aplica solamente a una relación).
- El grado (cantidad de atributos en la cabecera) de la relación resultado es igual al grado de la relación A original

$$G_{Resultado} = G_A$$

- La cardinalidad (cantidad de tuplas de una relación) de la relación resultado es igual a la cardinalidad de la relacion A original:

$$C_{Resultado} = C_A$$

- **Sumarizar:** Este operador toma el cuerpo de una relacion A y genera un subconjunto de tuplas mediante algun criterio de agrupamiento y luego se le aplica una expresion para generar una nuevo atributo a la cabecera que contiene el resultado de esa expresión.

Para que esta operación se pueda realizar se tienen que tener en cuenta los siguientes puntos:

  - El criterio de agrupamiento es un atributo o varios atributos (separados por coma) de la cabezera.
  - En la expresión de la operación solo pueden ir operadores de totales.

Ejemplo:

Relación A

ID	DNI	NOMBRE	NOTA
1	30270067	Juan	4
1	30270067	Juan	8
1	30270067	Juan	10

Relacion resultado de SUMARIZAR Relacion A POR {ID} AGREGAR SUMA(NOTA) COMO TOTALNOTA

ID	TOTALNOTA
1	22

Propiedades:

- El operador sumarizar es unario (Se aplica solamente a una relación).
- El grado (cantidad de atributos en la cabecera) de la relación resultado es igual a la cantidad de atributos que hay en el criterio de agrupamiento mas uno.
- La cardinalidad (cantidad de tuplas de una relación) de la relación resultado esta entre cero y la cardinalidad de la relacion A original:

$$0 \leqslant C_{Resultado} \leqslant C_A$$

Tabla de operadores y operaciones y su uso:

Nombre	Uso
UNION	*Relacion UNION Relacion*
INTERSECCION	*Relacion INTERSECCION Relacion*
DIFERENCIA	*Relacion DIFERENCIA Relacion*
PRODUCTO CARTECIANO	*Relacion PRODUCTO Relacion*
PROYECCION	*Relacion PROYECCION {Atributo 1, Atributo 2, etc}*
RESTRICCION	Relacion RESTRICCION Atributo + operación, Atributo + operación, etc.
REUNION	*Relacion REUNION Relacion*
DIVISION	*Relacion DIVISION Relacion*
EXTENDER	*EXTENDER Relacion AGREGAR MAYOR (Atributo) COMO*

	“Nuevo nombre”
RENOMBRAR	*Relacion RENOMBRAR Atributo COMO “Nuevo nombre”*
SUMARIZAR	*SUMARIZAR Relacion POR {Atributo} AGREGAR SUMA(Atributo) COMO “Nuevo nombre”*

Lenguaje SQL

El lenguaje SQL (Structured Query Language) es un lenguaje estandar definido por ANSI (American National Standards Institute) he ISO (International Standards Organization) para la comunicación con las bases de datos relacionales. Este lenguaje esta basado en el algebra relacional definido por Edgar F. Codd cuando trabajaba para IBM. Existen varias actualizaciones del estandar y la ultima se realizo en el 2003.

NOTA: Hay que tener mucho cuidado cuando se migra una BD de un determinado proveedor a otro porque, si bien es un estandar, cada uno de estos lo implementa de diferente forma.

Diferencias con el algebra relacional

Cuando se trabaja con el lenguaje SQL hay que tener en cuenta 2 diferencias importantes que tiene con el algebra relacional. Estas son:

1. Cambian las definiciones de algunos conceptos:

Álgebra Relacional	SQL
Relación	Tabla
Tupla	Fila/Registro
Atributo	Columna

2. Se permiten tener filas duplicadas a diferencia del algebra relacional que no permitia tuplas (filas/registro) duplicadas.

Tipos de sentencias SQL

1. **Sentencias DDL (Data Definition Language):** Un lenguaje de definición de datos (Data Definition Language), 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 definición de las estructuras que almacenarán los datos así como de los procedimientos o funciones que permitan consultarlos. Estas tareas se utilizan para crear/eliminar distintos tipos de objetos en la BD como pueden ser: Tablas, vistas (es una consulta que se presenta como una tabla (virtual) a partir de un conjunto de tablas en una base de datos relacional), Procedimientos (Programas que se ejecutan directamente en el motor de la Base de datos), Roles, Etc.

En SQL, las sentencias que se utilizan para realizar estas tareas son:

- **Sentencia CREATE:** Se utiliza para crear una nueva tabla en la base de datos.

Para poder utilizar esta sentencia se tiene que tener en cuenta ciertos requisitos:

Requisito	Es opcional?
Se debe definir un nombre para la tabla	NO
Se debe especificar las columnas con su tipo de datos correspondiente. Tambien se pueden agregar diferentes restricciones (constraints) opcionales a cada una de las columnas, estas son: NOT NULL: La columna no puede aceptar datos de tipo NULL. DEFAULT: Define un valor por defecto para esa columna cuando se crea una nueva fila.	NO
Se pueden especificar restricciones (constraints) a la tabla, estas son: Unique: Clave candidata de la tabla. Primary Key: Clave primaria de la tabla. Una clave primaria por definición también es una clave candidata.	SI

Foreign Key: Clave externa a otra tabla. Check: Chequeo de integridad sobre las filas de la tabla.	

Ejemplo:

```
CREATE TABLE Empleado(
     codEmpleado    INT NOT NULL,
     nombre         VARCHAR(30) NOT NULL,
     fechaNacimiento DATE NOT NULL,
     fechaIngreso   DATE NOT NULL,
     codDepto       INT,
     CONSTRAINT pk_empleados
          PRIMARY KEY(codEmpleado),
     CONSTRAINT fk_depto
          FOREIGN KEY(codDepto) REFERENCES
          Departamento(codDepto)
CONSTRAINT u_nombre
UNIQUE (nombre)
          CONSTRAINT ck_f_ing_f_na
               CHECK  fechaNacimiento  < fechaIngreso
)
```

- **Sentencia DROP:** Se utiliza para eliminar una tabla y todos sus datos en la base de datos.

Para poder utilizar esta sentencia se tiene que tener en cuenta ciertas restricciones:

  - La clave primaria de la tabla a eliminar no puede ser clave foránea de otra tabla.

Ejemplo:

```
DROP TABLE Empleado
```

- **Sentencia ALTER TABLE:** Se utiliza para modificar la estructura de una tabla que fue previamente creada.

Para poder utilizar esta sentencia se tiene que tener en cuenta ciertas restricciones:

  - Si la clave primaria de la tabla es una clave foranea para otra tabla no se puede eliminar esa columna.
  - Si la clave primaria de la tabla es una clave foranea para otra tabla no se puede eliminar esa constraint.
  - Si se agrega una columna nueva y la misma no puede tener datos NULL se debe crear si o si con un dato default.
  - No se puede eliminar una columna que este relacionada con una constraint.
  - Cuando se modifica el tipo de dato de una columna se debe tener en cuenta que el nuevo tipo pueda contener al tipo anterior, por ej: Si la columna es de tipo INT actualmente, el nuevo tipo puede ser un BIGINT (Contiene a INT) pero no un SMALLINT (No contiene a INT).

Ejemplos:

  - Agregar una columna nueva:

```
ALTER TABLE Empleado ADD COLUMN fechaBaja INT NULL
```

  - Para agregar una columna nueva con valor default:

```
ALTER TABLE Empleado ADD COLUMN salario INT NOT NULL DEFAULT 100
```

  - Para modificar el tipo de una columna:

```
ALTER TABLE Empleado MODIFY COLUMN fechaBaja DATE NULL
```

  - Para eliminar una columna:

```
ALTER TABLE Empleado DROP COLUMN fechaBaja
```

- Para agregar una constraint:

```
ALTER TABLE Empleado ADD CONSTRAINT ck_fechas CHECK
(fechaIngreso > fechaNacimiento)
```

- Para eliminar una constraint:

```
ALTER TABLE Empleado DROP CONSTRAINT ck_fechas
```

2. **Sentencias DML (Data Manupulation Language):** El Lenguaje de Manipulación de Datos (Data Manipulation Language) es un lenguaje proporcionado por los sistemas gestores de bases de datos que permite a los usuarios de la misma llevar a cabo las tareas de consulta o modificación de los datos contenidos en las bases de datos.

En SQL, las sentencias que se utilizan para realizar estas tareas son:

- **Sentencia INSERT INTO:** Permite crear una nueva fila dentro de una tabla. En esta sentencia poner el nombre de las columnas donde se quieren cargar los nuevos datos es opcional

Ejemplos:

- Sintaxis de la sentencia:

```
INSERT INTO table [(column [, column …])] VALUES (value [,
value...])
```

NOTA: Todo lo que esta entre corchetes es opcional

- Insertar una fila en una tabla – Version 1:

```
INSERT INTO Jugador VALUES (1321,'Martín Gómez','1-1-1988')
```

NOTA: Si no se espesifican las columnas de una tabla se tienen que poner valores para todas las columnas.

- Insertar una fila en una tabla – Version 2:

```
INSERT INTO Jugador(codJugador,nombre) VALUES (1321,'Martín
Gómez')
```

NOTA: Se deben colocar valores únicamente para las columnas especificadas en la sentencia, para las no especificadas, se pondrá un valor NULL o DEFAULT si esta definido para esa columna.

- **Sentencia UPDATE:** Permite actualizar el contenido de una fila dentro de una tabla. Hay que tener en cuenta que se pueden actualizar varias filas en simultaneo, por eso es recomendable usar esta sentencia con una clausula WHERE (esto es opcional) para restringir la actualizacion de todas las columnas de las filas solamente a las que cumplan una condicion determinada.

Ejemplos:

  - Sintaxis de esta sentencia:

```
UPDATE table SET column = value [, column = value, …] [WHERE condition]
```

NOTA: Todo lo que esta entre corchetes es opcional

  - Cambiar de departamento a los empleados:

```
UPDATE Empleado SET codDepto = 10 WHERE codDepto = 1
```

  - Aumentarle el sueldo a todos los empleados:

```
UPDATE Empleado SET sueldo = 1.5 * sueldo
```

- **Sentencia DELETE:** Permite eliminar filas dentro de una tabla. Hay que tener cuidado con esta sentencia porque si no se le especifica una restriccion con la clausula WHERE (esta clausula es opcional) puede eliminar todas las filas de una tabla. Hay que tener en cuenta que si se eliminan todas las filas, la tabla seguirá existiendo (no se elimina) pero sin contener ningún dato en ella (sin filas).

Ejemplos:

  - Sintaxis de esta sentencia:

```
DELETE [FROM] table [WHERE condition]
```

NOTA: Todo lo que esta entre corchetes es opcional

  - Eliminar los departamentos de Finanzas:

```
DELETE FROM Departamento WHERE nombre = ‘Finanzas’
```

- Eliminar todos los departamentos:

```
DELETE FROM Departamento
```

- **Sentencia SELECT:**Permite recuperar los datos guardados en las tablas dentro de una base de datos. Es una de las sentencias mas usadas y que permite mas variantes.

Ejemplos:

- Sintaxis de esta sentencia:

```
SELECT [DISTINCT] select_list

FROM table [table alias] [,...]

[WHERE condition]

[GROUP BY column_list]

[HAVING condition]

[ORDER BY column_name [ASC/DESC]]
```

NOTA: Todo lo que esta entre corchetes es opcional.

- Obtener los nombres de los empleados del departamento 10:

```
SELECT nombre FROM Empleado WHERE codDepto=10
```

NOTA: La clausula WHERE siente efecto sobre las columnas de toda la tabla. En estos casos se dice que el grupo donde se aplica la clausula es toda la tabla.

- Obtener todos los campos de los empleados del departamento 10:

```
SELECT * FROM Empleado WHERE codDepto=10
```

NOTA: Para devolver todas las columnas de las filas se utiliza el comodin * para no tener que escribir el nombre de todas las columnas una por una.

- Obtener el sueldo de los empleados en base a un cálculo (50$ x hora trabajada):

```
SELECT cantHorasTrabajadas * 50 AS sueldo FROM Empleado
```

NOTA: Utilizando la clausula AS se puede dar un nombre específico a la columna resultado o tambien se puede utilizar para columnas calculadas (como en el ejemplo) o para renombrar una columna existente con un nombre diferente.

- Obtener los códigos de departamento que tienen empleados (sin repetidos):

```
SELECT DISTINCT codDepto FROM Empleado
```

NOTA: La clausula DISTINCT se utiliza para eliminar las filas repetidas

- Obtener los códigos de empleados del departamento 10 ordenados en forma ascendente:

```
SELECT codEmpleado FROM Empleado WHERE codDepto = 10 ORDER BY codEmpleado ASC
```

NOTA: Es posible ordenar el resultado en forma descendentemente utilisando el modificador DESC (en lugar de ASC) de la clausula ORDER BY. También se puede utilizar la clausula ORDEY BY con mas de una columna para obtener un ordenamiento por múltiples columnas.

- Obtener todos los nombres de empleados del departamento de RRHH vinculando informacion de 2 tablas diferentes – JOIN Version 1:

```
SELECT eNombre FROM Empleado INNER JOIN Departamento ON eCodDepto = dCodDepto WHERE dNombre=‘RRHH’
```

NOTA: Hay que tener en cuenta que cuando se quiere obtener info. De 2 tablas diferentes los nombres de las columnas no pueden coinsidir porque si no el compilador no sabe a cual nos estamos refiriendo.

- Obtener todos los nombres de empleados del departamento de RRHH vinculando información de 2 tablas diferentes – JOIN Version 2:

```
SELECT eNombre FROM Empleado, Departamento WHERE eCodDepto = dCodDepto AND dNombre=‘RRHH’
```

NOTA: Hay que tener en cuenta que cuando se quiere obtener info. de 2 tablas diferentes los nombres de las columnas no pueden coinsidir porque si no el compilador no sabe a cual nos estamos refiriendo. El modificador AND de la clausula WHERE nos permite concatenar mas de una condicion, hay que tener en cuenta que funciona como el operador logico Y que solo trae resultados cuando las 2 condiciones son verdaderas. Tambien se puede utilizar el operador logico OR que es el equivalente a un O.

- Obtener todos los nombres de empleados del departamento de RRHH vinculando información de 2 tablas diferentes – JOIN Version 3:

```
SELECT Empleado.nombre FROM  Empleado, Departamento WHERE
Empleado.codDepto = Departamento.codDepto AND
Departamento.nombre='RRHH'
```

NOTA: En este ejemplo como las 2 tablas tienen el mismo nombre de columna antes del nombre hay que anteponer el nombre de la tabla y despues un punto para que el compilador pueda diferenciar con que columnas esta trabajando. El modificador AND de la clausula WHERE nos permite concatenar mas de una condicion, hay que tener en cuenta que funciona como el operador logico Y que solo trae resultados cuando las 2 condiciones son verdaderas. Tambien se puede utilizar el operador logico OR que es el equivalente a un O.

- Obtener todos los nombres de empleados del departamento de RRHH vinculando información de 2 tablas diferentes – JOIN Version 4:

```
SELECT e.nombre FROM Empleado e, Departamento WHERE
e.codDepto = d.codDepto AND d.nombre='RRHH'
```

NOTA: En este ejemplo como las 2 tablas tienen el mismo nombre de columna en la clasula FROM despues del nombre de la tabla se deja un espacio y luego se coloca un nombre nuevo (en este caso las letras e y d), a esta forma de trabajar se la conoce como definir un alias al nombre de la tabla para que el compilador pueda diferenciar con que columnas esta trabajando. El modificador AND de la clausula WHERE nos permite concatenar mas de una condicion, hay que tener en cuenta que funciona como el operador logico Y que solo trae resultados cuando las 2

condiciones son verdaderas. Tambien se puede utilizar el operador logico OR que es el equivalente a un O.

- **Sentencia INSERT combinada con una sentencia SELECT:** Empleando esta combinación se pueden agregar múltiples filas a la vez.

Ejemplo:

  - Dar de alta en una nueva tabla a todos los gerentes:

```
INSERT INTO Gerente(codGerente, nombre, salario) SELECT
codEmpleado, nombre, salario FROM Empleado WHERE cargo =
‘GERENTE'
```

NOTA: La cantidad de columnas que debe mostrar el resultado de la sentencia SELECT tiene que coincidir con la cantidad de columnas especificadas en la sentencia INSERT.

- **Sentencia SELECT con valores de columnas que admiten NULL:** Suele ocurrir que en algunos casos no se dispone de un valor para asignar a una columna, por ejemplo: En la tabla de empleados puede haber ciertos de empleados sin un departamento asignado. En estos casos SQL provee un valor especial, el NULL. Este tipo de valor tiene ciertas particularidades que son:

  - Existen operadores especiales para controlar si un valor es nulo o no

  IS NULL (evalor nulos) o IS NOT NULL (no es valor nulo)

  - El resultado surge de una comparacin trinaria (3-valued logic) (true, false & null) Por ejemplo: “edad > 21” ¿ es true o false cuando edad es null? Poder determinar el resultado de esta pregunta se utilizan las siguientes tablas de verdad:

Tabla de resultados para operaciones que involucran a NULL

Operación a evaluar	**Resultado**
NULL > 0	IS NULL
NULL + 1	IS NULL

NULL = 0	IS NULL
NULL AND TRUE	IS NULL

Tabla de verdad trinaría para AND

AND	TRUE	FALSE	NULL
TRUE	**TRUE**	**FALSE**	**NULL**
FALSE	**FALSE**	**FALSE**	**FALSE**
NULL	**NULL**	**FALSE**	**NULL**

Tabla de verdad trinaria para OR

OR	TRUE	FALSE	NULL
TRUE	**TRUE**	**TRUE**	**TRUE**
FALSE	**TRUE**	**FALSE**	**NULL**
NULL	**TRUE**	**NULL**	**NULL**

De la primer tabla se resuelve que la pregunta “edad > 21” ¿ es true o false cuando edad es null? tiene como resultado el valor NULL.

Ejemplos:

Empleado

codEmp	nombre	codDepto
1	Pedro	1
2	Mario	NULL
3	José	1

Departamento

codDepto	nombre
1	Finanzas

- Suponiendo que puede haber empleados sin departamento, obtener los nombres de los empleados y su número de departamento descartando en el resultado los empleados que no tengan departamento asignado (Sean NULL):

```
SELECT e.nombre, e.codDepto FROM Empleado e, Departamento d
WHERE e.codDepto = d.codDepto
```

nombre	codDepto
Pedro	1
José	1

- Suponiendo que puede haber empleados sin departamento, obtener los nombres de los empleados y su número de departamento incluyendo en el resultado los empleados que no tengan departamento asignado (Sean null):

```
SELECT e.nombre, e.codDepto FROM Empleado e LEFT OUTER JOIN
Departamento d ON e.codDepto = d.codDepto
```

nombre	codDepto
Pedro	1
Mario	NULL
José	1

NOTA: Para obtener el resultado se utiliza la clausula LEFT OUTER JOIN que devuelve la pareja de todos los valores de la tabla izquierda con los valores de la tabla de la derecha correspondientes (sin devolver la tabla derecha), o retorna un valor nulo NULL en caso de no correspondencia.

- **Sentencia SELECT con la clausula GROUP BY:**La cláusula GROUP BY especifica el criterio (la columna o columnas) por el cual la tabla va a ser agrupada. Entre las columnas del SELECT solamente podrán aparecer las columnas del GROUP BY o funciones de totales que son:
  - COUNT: Cuenta la cantidad de filas que hay en un grupo determinado.
  - SUM: Suma los valores de una columna que es distinta a la del criterio de agrupamiento.
  - MAX: Obtiene el maximo de los valores de una columna que es distinta a la del criterio de agrupamiento.

    - MIN: Obtiene el minimo de los valores de una columna que es distinta a la del criterio de agrupamiento.
    - AVG: Obtiene el promedio de los valores de una columna que es distinta a la del criterio de agrupamiento.

Los grupos también se pueden filtrar usando la cláusula HAVING que funciona de la misma forma que la clausula WHERE pero en vez de aplicar a todas las filas de la tabla aplica solamente a las filas de los grupos.

Ejemplos:

    - Devolver los salarios máximos por departamento:

```
SELECT codDepto, MAX(salario) FROM Empleado GROUP BY codDepto
```

    - Devolver los salarios máximos por departamento, para departamentos con más de 10 empleados:

```
SELECT codDepto, MAX(salario) FROM Empleado GROUP BY codDepto
HAVING COUNT(*)>10
```

    - Devolver los salarios mínimo y máximo:

```
SELECT MIN(salario), MAX(salario) FROM Empleado
```

NOTA: Cuando no se especifica un criterio de agrupación, se considera a toda la tabla como un grupo.

- **Como es el orden de ejecucion conceptual de una query en SQL:** Cuando se están armando querys de SQL hay que tener en cuenta el orden (conceptual) en el cual el compilador va resolviendo cada una de las sentencias para llegar al resultado deseado. Este orden de ejecución conceptual es el siguiente:
    1. Se resuelven las tablas del FROM
    2. Se aplica la condición del WHERE
    3. Se agrupa con el criterio del GROUP BY
    4. Se filtran los grupos con la condición del HAVING
    5. Se resuelven las columnas a devolver

6. Se ordenan los resultados con el ORDER BY
7. Se eliminan los duplicados del DISTINCT

Ejemplo de una sentencia genérica:

```
SELECT [DISTINCT] select_list

FROM table [table alias] [,...]

[WHERE condition]

[GROUP BY column_list]

[HAVING condition]

[ORDER BY column_name [ASC/DESC]]
```

3. **Sentencias DCL (Data Control Language):** El Lenguaje de Control de Datos (Data Control Language) es un lenguaje proporcionado por el Sistema de Gestión de Base de Datos que permiten al administrador controlar el acceso a los datos contenidos en la Base de Datos.

En SQL, las sentencias que se utilizan para realizar estas tareas son:

- **Sentencia GRANT:** Esta sentencia se utiliza para otorgar diferentes permisos a los usuarios de una tabla determinada dentro de la base de datos. Estos permisos pueden ser de consulta (el usuario puede realizar un SELECT sobre la tabla), de creacion (el usuario puede realizar INSERT sobre la tabla), de modificacion (el usuario puede realizar ALTER TABLE o UPDATE sobre la tabla), etc.
- **Sentencia REVOKE:** Esta sentencia se utiliza para quitar los permisos a los usuarios en una tabla determinada dentro de la base de datos. Permite realizar la tarea inversa a la sentencia GRANT.