Fundamentos de Base de Datos

Contenido

Diseño Conceptual – MODELADO ER

Modelado Conceptual

Es la primera etapa en el diseño de una BD. Consiste en estudiar el problema real, especificar usando lenguaje de muy alto nivel y validar resultado.

Se concentra en definir el dominio del problema (estructura y restricciones de integridad).

En los modelos de datos conceptuales encontramos:

- **Conjuntos**
  Los elementos de interés aparecen agrupados o clasificados en conjuntos de acuerdo a sus características
- **Relaciones entre conjuntos**
  Conjuntos de parejas, ternas, cuaternas, de elementos de los conjuntos anteriores.
- **Restricciones de integridad**
  Condiciones que indican cuando un elemento o una pareja puede o no puede pertenecer a un conjunto o relación.

Se distinguen los siguientes términos en el modelado conceptual

- **Atributos**
  Característica de interés de un determinado elemento de la realidad
- **Cardinalidad**
  Cuantos elementos de un conjunto pueden estar relacionados con elemento de origen. La cardinalidad puede ser (N:1 o N:N)
- **Totalidad**
  Dada una relación entre dos conjuntos **A** y **B** se dice que es **Total** con respecto a A si todos los elementos de **A** deben aparecer en alguna pareja de la relación.

En la modelización conceptual, se construyen esquemas conceptuales de una realidad. Existen dos principios que NO podemos olvidar:

- **Principio del 100%**
  El esquema conceptual asociado a un problema debe representar todos sus aspectos
- **Principio de Conceptualización**
  El esquema conceptual no debe incluir ningún elemento asociado a la implementación del esquema, así como ningún elemento orientado a la performance futura de la BD.

Modelo Entidad-Relación (MER)

Los conceptos básicos del modelo entidad relación son la existencia de **entidades** (elementos de la realidad) y **relaciones** (asociaciones entre elementos).

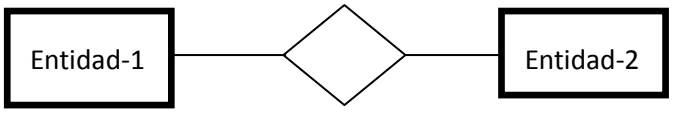
El MER tiene dos elementos básicos :

- Diagrama Entidad-Relación
- Restricciones no estructurales

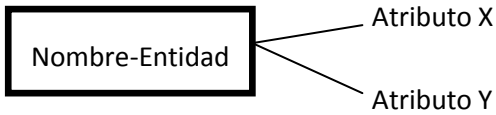
Los elementos que definen y se utilizan para la construcción del diagrama son:

- Entidades

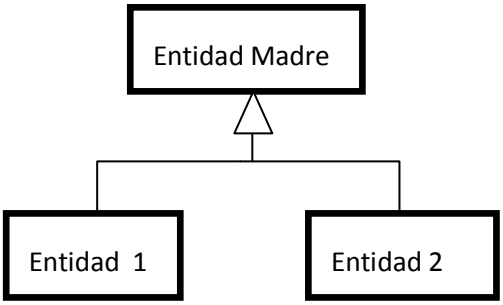
- Relaciones



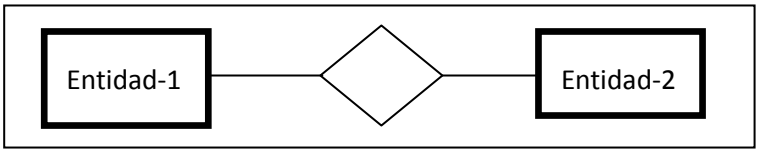
- Atributos



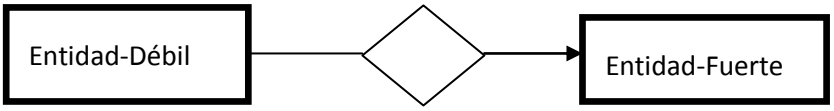
- Generalizaciones - Especializacion



- Agregaciones



- Entidades Débiles



Algoritmo de Modelización

1. Identificar elementos del problema
2. Identificar relaciones entre objetos
3. Representar propiedades de interés
4. Determinar restricciones

Una **ENTIDAD** es un elemento distinguible de nuestra realidad. Las entidades se agrupan en Conjuntos de Entidades o Tipos de Entidades.

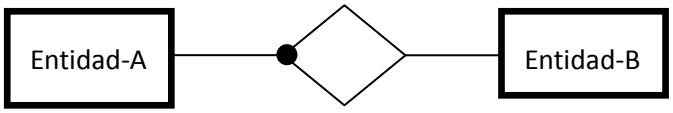
Un **ATRIBUTO** es una función tal que dado un elemento de un determinado conjunto de entidades, devuelve un valor de un determinado conjunto de valores. Los atributos pueden ser comunes, estructurados (compuestos) o multivalorados.

Restricciones

Se dice que un atributo es **DETERMINANTE** cuando no pueden existir dos entidades en el conjunto que tengan el mismo valor en ese atributo. Permite IDENTIFICAR a las entidades.

La **CARDINALIDAD** en las relaciones nos indica cuantos elementos de un conjunto pueden estar relacionados con un elemento del otro conjunto y viceversa. Se indica poniendo las cantidades en la relación.

La **TOTALIDAD** nos indica que todo elemento de un conjunto A debe aparecer en alguna pareja de la relación A-B. Se representa de la siguiente forma:



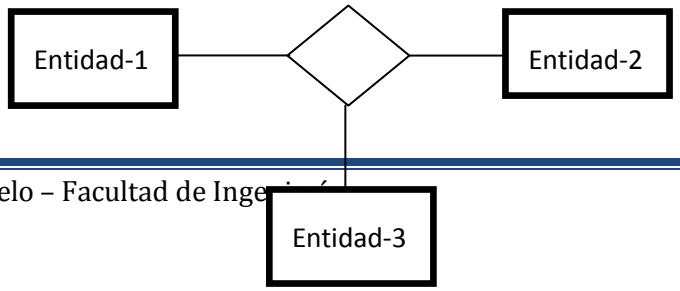
También existen relaciones sobre las relaciones **NO ESTRUCTURALES.** Estas se definen en lenguaje natural o con algebra relación de manera de poder añadir información al diagrama.

Otros elementos del MER

Se permite agregar **Atributos en Relaciones**, cuando el atributo no es específico de las entidades que forman la relación sino que surge de la propia relación.

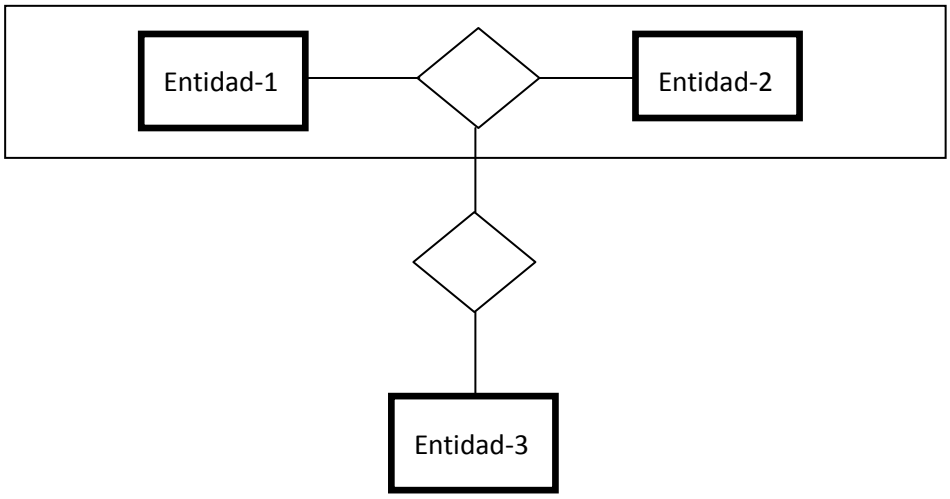
Se permiten usar **Autorelaciones**, pero es necesario utilizar roles y restricciones nos estructurales para evitar cicrularidad.

Se pueden crear relaciones multiples que contemplan más de dos entidades.



La relación triple anterior no permite que existan parejas (Entidad-1, Entidad-2) si no existe la relación con Entidad-3.

Si quisiéramos permitir que puedan existir esas parejas y a su vez si existe la pareja relacionarla con la Entidad-3, deberíamos aplicar el concepto de agregación.



Calidad de Esquemas y Correctitud

Para asegurar la calidad de los esquemas conceptuales se define un conjunto de propiedades que se deben chequear durante y al final del desarrollo.

Maximizar **Completitud y Correctitud**

Balancear **Minimalidad, Expresividad, Explicitud**

Un esquema es **COMPLETO** cuando representa todas las características relevantes del problema. Para ello, debemos chequear:

- Todos los conceptos del problema estén representados en alguna parte
- Todos los requerimientos sean realizables con el esquema
- Leer el resultado y compararlo con la descripción original.

Existen dos tipos de **CORRECTITUD**. La Sintáctica y la Semántica.

Un esquema es correcto sintácticamente cuando las distintas partes de este están construidas correctamente con respecto al lenguaje utilizado. Para ello debemos chequear:

- Existencia de cardinalidades en cada relación
- Existencia de atributos determinantes en cada entidad, si no existen verificar que sea entidad débil.
- Existencia de una y solo una relación y todas las entidades que intervienen en la misma dentro de cada agregación.

Un esquema es correcto semánticamente si cada elemento del problema se representa utilizando estructuras adecuadas. Para comprobarlo, chequear y responder para cada concepto del problema:

- Atributo, entidad o relación?
- Una sola categoría de entidades o más de una?
- Una relación binaria o múltiple?
- Cuál es el mecanismo de determinación de un conjunto?
- Las cardinalidades y totalidades tienen sentido?

Modelo y Álgebra Relacional

Modelo Relacional

Las estructuras consisten en TABLAS cuyas columnas corresponden a ATRIBUTOS de tipo atómico y las filas corresponden a registros de datos.

Las operaciones están fundamentalmente orientadas a manejo de TABLAS, como conjunto de registros.

Es un modelo de datos extremadamente simple y claro, que también ha resultado potente para la mayor parte de las aplicaciones de las BDs.

Conceptos generales

- **Dominio D**
  Conjunto de valores atómicos
- **Esquema Relación R (A1…An)**
  R es el nombre de la relación, A1…An son los atributos con dominio D1…Dn.
- **Relación r (R)**
  Es una instancia de un esquema R. Consiste en un conjunto de tuplas
- **Tupla**
  Elemento de un producto cartesiano de N dominios.

Características de Relaciones

Una relación es un conjunto de tuplas que NO ESTA ORDENADO y donde NO EXISTEN REPETIDOS. Los valores de los atributos en las tuplas, son valores atómicos (primera forma normal).

Restricciones de Integridad

- **Restricciones de Dominio** (de tipo)
- **Superclave y Clave**
  Dado R(A1…An) se dice que X C {A1…An} es superclave en un esquema R, si no puede existir ninguna r(R) tal que tenga dos tuplas con valores iguales de X.
  Una clave es una superclave minimal (no contiene propiamente una superclave).
- **Integridad Referencial**
  Dado R, un conjunto de atributos X es una FK (Foreign Key) de R si:
  - Los atributos de X coinciden en dominio con los de una clave Y de S.
  - Los valores de X en tuplas de r(R) corresponden a valores de Y en s(S).
    (Existe una RI Referencial entre R y S)

Las restricciones de integridad surgen de la observación de la realidad, se definen a nivel de esquema relación, y son violadas por instancias.

Operaciones

- **INSERT (a,b,c) into R**
  Incluye la tupla (a,b,c) en la relación r. Debe cumplir las RI.
- **DELETE from R where A = 'a'**
  Borra de las tuplas de r cuyo valor para A es 'a'. Puede generar violaciones RI.
- **UPDATE R set A='a1' where B = 'b1'**
  Modifica las tuplas cuyo valor de r es B = b1, colocando a1 como valor de A.

Algebra Relacional

A partir de un conjunto de operadores para consultar BD relacionales, define un conjunto de operaciones estándar en BD.

Los operadores reciben relaciones y devuelve relaciones. Las operaciones comunes para tuplas son la unión, diferencia y el producto cartesiano. Para BD relacionales se suman la selección, proyección y el join.

La sintaxis define que símbolos se utilizan para cada operador y que parámetros recibe. Y la semántica define cual es el esquema resultado, cual es la instancia del resultado y que condiciones deben cumplir para que se pueda aplicar el operador.

Selección - $\sigma_\theta(r)$

Permite obtener las tuplas que cumplen una cierta condición. Dada una relación R y una condición Θ, la selección $\sigma_\theta(R)$ da como resultado otra relación con mismo esquema que R y con una instancia del conjunto de tuplas que cumplen con Θ.

Proyección - $\pi_\alpha(r)$

Permite obtener las tuplas con un cierto conjunto de atributos. Devuelve una relación con distinto esquema. Donde α es la lista de atributos que conforman el esquema devuelto.

Unión – R U S

Obtener la unión de 2 relaciones (R y S) tomadas de un conjunto de tuplas. Las relaciones R y S DEBEN tener el mismo esquema.

Diferencia – R – S

Obtener la diferencia de dos relaciones (R y S) tomadas como conjunto de tuplas. Las relaciones R y S DEBEN tener el mismo esquema.

Producto Cartesiano – R x S

Obtener el producto cartesiano de dos relaciones. El resultado es otra relación cuyo esquema es (E1 + E2) y las tuplas son generadas por todas las combinaciones posibles de las tuplas de R con las de S.

Hasta el momento se mostraron los operadores básicos. Los siguiente se pueden derivar de aplicar los anteriores.

JOIN – R |x|$_{\Theta}$ S

Combina tuplas de dos relaciones a través de una condición Θ sobre los atributos. Es exactamente lo mismo que hacer un producto cartesiano entre R y S y luego sobre el resultado hacer una selección con Θ.

Una variante es el **JOIN NATURAL (R*S)** donde la condicion es de igualdad entre los atributos de igual nombre.

DIVISION - R % S

La relación da como resultado otra relación con esquema (A1…An) y su contenido son las tuplas tomadas a partir de R tales que su valor está asociado a TODOS los valores de S.

Importante: Si bien se exige que la solución este relacionada con todos los valores de S, no se exige que lo este solamente con esos valores.

Cálculo Relacional

Es un lenguaje o familia de lenguajes de consultas sobre modelos relacionales basadas en formulas lógicas de primer orden para definir conjuntos. Una consulta es una especificación por comprensión de un conjunto de tuplas.

Existen dos sublenguajes en el calculo relacional:

- **TUPLAS**:
  El universo está formado por tuplas
- **DOMINIOS:**
  El universo está formado por valores individuales

Cálculo de Tuplas

Una expresión **CRT** está compuesta por :

- **Variable** tupla que caracteriza el resultado. Se especifica la estructura de dicha variable.
- **Condición** que caracteriza las tuplas resultado en términos de lógica de primer orden.
- **"Cuidados"** para escribir formulas que realmente describan los conjuntos que nos interesan.

$$\{< t_1 A_1, \dots, t_n A_n > \mid \varphi(t_1 \dots t_n) \}$$

Donde el universo utilizado es el conjunto de todos los posibles. Es decir, es un conjunto infinito que contiene tuplas de cualquier aridad, tuplas con cualquier dominio en cualquier posición.

La formula $\varphi(t_1 \dots t_n)$ es una fórmula lógica de primer orden donde los predicados corresponden a relaciones o comparaciones entre atributos, las variables son tuplas y no existen funciones del usuario.

Las fórmulas a utilizar pueden ser de:

$t_1 < op > t_2 \mid op = \{=, <, >, \neq\}$

$P_i(x_i) \mid P_i \; es \; tabla$

$\alpha < CONN > \beta \mid \alpha, \beta \; formulas$

$\sim\alpha \mid \alpha \; es \; formula$

$\exists x_i \partial \mid \partial \; es \; formula$

$\nabla x_i \partial \mid \partial \; es \; formula$

En el cálculo relacional, puede darse que se obtengan **formulas inseguras**, estas permiten resultados infinitos en consultas CRT.

Llamemos **Dominio de Expresión CRT** al conjunto de todos los valores que aparecen como valores constantes en la expresión o bien que existen en cualquiera de las relaciones a las que se hace referencia en la expresión.

Fórmulas Seguras

Una expresión CRT es **SEGURA** si todos los valores de su resultado pertenecen al dominio de la expresión. Para chequear si una formula es segura:

- Pensar en la formula traducida con conectores binarios lógicos.
- Si **f** es tipo $\vartheta_1 \vee \ldots \vee \vartheta_n$ en cada ϑ_i deben aparecer TODAS las variables libres en un predicado NO NEGADO.
- Si **f** es tipo $\vartheta_1 \wedge \ldots \wedge \vartheta_n$ cada variable debe aparecer en al menos una ϑ_i en un predicado NO NEGADO.

Calculo de Dominios

Las formulas son del tipo { $t_1 \ldots t_n$ / α } donde t_i es o una variable o una constante. Si es variable, aparece libre en α.

La ventaja es que permite expresiones más compactas que las de CRT.

SQL

Es un lenguaje procedural que opera sobre conjunto de tuplas. Su poder de expresión incluye el Algebra Relacional, y la extiende.

Se identifican dos lenguajes dentro del SQL. El **DDL (Data Definition Language)** y el **DML (Data Manipulation Language)**. El primero permite crear, modificar y eliminar objetos de la base de datos. Mientras que el último permite crear, modificar, eliminar y recuperar datos de una base de datos.

La sentencia SELECT

Esta sentencia es utilizada para obtener tuplas de la base.

- A_i son nombres de atributos
- R_i son nombres de relaciones
- P es una condicion

```
SELECT A1, A2, … , An
FROM R1, R2, … , Rm
WHERE P
```

Order By

Ordenar las tuplas devueltas por uno o varios atributos. Dependiendo si se le agrega ASC o DESC luego del atributo es como realizará el ordenamiento.

Distinct

Esta clausular filtra los elementos repetidos.

Joins

Un join se puede realizar agregando en la condicion P, las condiciones del join. Es decir, estableciendo que atributos deben ser iguales entre sí para poder hacer el join. A su vez, se puede utilizar **NATURAL JOIN** que lo que hace automáticamente es filtrar por atributos de distintas tablas que tengan el mismo nombre.

Existen otros tipos de JOIN pero no los utilizamos con normalidad. Estos son:

- **LEFT JOIN**: se agrega, para cada tupla de T1 que no satisface la condición de JOIN con NINGUNA de T2, una fila con NULOS en las columnas de T2
- **RIGHT JOIN**: análogo a Left Join pero al revés.
- **FULL JOIN**: Unión de Left y Right

Renombrar Atributos

A veces es muy practico y otras necesario renombrar atributos para evitar ambigüedades en la consulta SQL. Se pueden renombrar tanto atributos como tablas.

Union

Si tengo dos consultas S y T, quiero hacer la unión de las dos. **S UNION T.** La unión elimina las tuplas repetidas.

Diferencia

Para utilizar u obtener una diferencia entre dos conjuntos de tuplas puede realizar la consulta **S EXCEPT T** o puedo utilizar el operador NOT IN cuando creo la condición

Funciones

- Count(atributo)
- Max (atributo)
- Min(atributo)

Todas trabajan sobre conjuntos de tuplas no sobre una tupla individual.

Consultas Anidadas

Puedo agregar una nueva consulta en la condicion de otra consulta utilizando los operadores IN y EXISTS.

Group By

Es una clausula mas que se agrega al final de la consulta. Si se utiliza la clausula, en las expresiones del SELECT solo puedo utilizar atributos que se encuentran dentro del group by, funciones de agregación sobre esos atributos, o expresiones aritméticas que utilicen los anteriores.

Having

Se pueden especificar condiciones sobre los grupos formados por Group By.

Diseño de Base de Datos Relacional

Pautas Informales

Existen 4 medidas informales para verificar la calidad de los diseños de base de datos relacionales.

Semántica de los Atributos

Diseñar un esquema de relación de modo que sea fácil de explicar su significado. No combinar atributos de varios tipos de entidades y tipos de vínculos en una sola relación.

Reducción de Valores Redundantes

El hecho de que existan valores redundantes implica que haya anomalías de inserción, eliminación o modificación.

Diseñar los esquemas de las relaciones de modo que no haya anomalías. En el caso de que existan anomalías, señalarlas con claridad a fin de que los programas que actualicen la BD operen correctamente.

Reducción de Valores Nulos

La existencia de valores nulos en tuplas generan **desperdicio de espacio**, **dificultad para entender el significado**, **problemas al aplicar funciones agregadas**, **existencia de multiples interpretaciones.**

Hasta donde sea posible, evitar incluir en una relación atributos cuyos valores pueden ser nulos. Si no es posible, asegurarse de que se apliquen solo en casos excepcionales y no a la mayoría de las tuplas en una relación.

Inexistencia de Tuplas Erróneas

Diseñar los esquemas de modo que puedan reunirse por condicion de igualdad sobre atributos claves, para garantizar que no se formen tuplas erróneas.

Dependencias Funcionales

Una dependencia funcional $df : X \rightarrow Y$ entre dos conjuntos de atributos X e Y que son subconjuntos de R, especifica una restricción sobre las posibles tuplas que formarían una instancia de R.

Sean t_1 y t_2 tuplas de R tales que $t_1[X] = t_2[X] \rightarrow t_1[Y] = t_2[Y]$

Si X es una clave candidata de R, entonces $df : X \rightarrow Y$ para cualquier subconjunto de atributos Y de R.

Si $df : X \rightarrow Y$ en R, no implica que se deba cumplir $df : Y \rightarrow X$

Clausura de F – F+

Sea F el conjunto de dependencias funcionales especificadas sobre el esquema relación R. Llamaremos F+ como **Clausura de F** al conjunto de todas las dependencias funcionales que se cumplen en todas las instancias que satisfacen F.

Reglas de Inferencia

- **Reflexiva :** $si\ X \supset Y \ \rightarrow \{X \rightarrow Y\}$
- **De Aumento :** $\{X \ \rightarrow Y\} \ \rightarrow \ \{XZ \rightarrow YZ\}$
- **Transitiva :** $\{X \ \rightarrow Y,\ Y \ \rightarrow Z\} \ \rightarrow \ \{X \rightarrow Z\}$
- **Descomposición :** $\{X \ \rightarrow YZ\} \ \rightarrow \ \{X \rightarrow Y\}$
- **Unión :** $\{X \ \rightarrow Y,\ X \ \rightarrow Z\} \ \rightarrow \ \{X \rightarrow YZ\}$
- **Pseudo-transitiva :** $\{X \ \rightarrow Y,\ WY \ \rightarrow Z\} \ \rightarrow \ \{WX \rightarrow Z\}$

Definimos **X+** como el conjunto de atributos determinados funcionalmente por X.

El algoritmo para obtener este conjunto es el siguiente:

```
X+ := X
REPETIR
        viejo X+ := X+
        para cada df : Y → Z en F
                si  Y ⊇ X+ entonces X+ := X+ U Z
HASTA (viejo X+ == X+)
```

Dos conjuntos de dependencias funcionales E y F son **equivalentes** si y solo si se cumple que E+ = F+. Es decir, todas las dependencias funcionales en E se pueden inferir de F y E cubre a F.

Para determinar si F cubre a E:

Para cada $df : X \rightarrow Y$ perteneciente a E, calculamos X+ (F) y verificamos que X+ incluya los atributos en Y.

Conjunto Minimal

Un conjunto F de dependencias funcionales es Minimal si cumple:

- Toda $df : X \rightarrow Y$ en F tiene un solo atributo a la derecha
- No se puede reemplazar ninguna $df : X \rightarrow A$ Ɛ F por $df : Y \rightarrow A$ donde Y ⊃ X

- No podemos quitar dependencias funcionales de F y seguir teniendo un conjunto de dependencias funcionales equivalente.

Algoritmo de Cubrimiento Minimal

1. Hacer G := F
2. Reemplazar cada $df : X \rightarrow A_1A_2 \dots A_n$ en G por las n $dfs : X \rightarrow A_i$
3. Para cada $df : X \rightarrow A$ restante en G
    i. Para cada atributo B que sea un elemento de X
        1. Calcular (X-B)+ respecto a G
        2. Si (X-B)+ contiene a A, reemplazar $df : X \rightarrow A$ por $df : (X - B) \rightarrow A$
4. Para cada $df : X \rightarrow A$ en G
    a. Calcular X+ respecto a $\big(G - (X \rightarrow A)\big)$
    b. Si X+ contiene a A, eliminar $X \rightarrow A$ de G

Formas Normales

La normalización es un proceso durante el cual los esquemas relacionales insatisfactorios se descomponen repartiendo sus atributos en esquemas relacionales mas pequeños que poseen propiedades deseables.

En este proceso el esquema se somete a una serie de pruebas para "certificar" si pertenece o no a una cierta forma normal.

Una forma normal no garantiza, sin considerar otros factores un buen diseño de BD. Para ello se debe considerar la propiedad de **Join sin perdida y la preservación de dependencias.**

Claves y Superclaves

Una **superclave** es un conjunto de Atributos S $\subseteq$ R tal que no existen 2 tuplas t_1 y t_2 en ningún esquema R tal que $t_1[S] = t_2[S]$.

Una **clave** es una superclave que si se le quita alguno de sus atributos, deja de ser superclave.

Si una relación tiene mas de una clave, cada una es clave candidata. Una de ellas, elegida arbitrariamente es designada como clave primaria. El resto son secundarias.

Atributos y Dependencias

Un atributo del esquema relación R es **primo** si es miembro de alguna clave de R.

$df : X \rightarrow Y$ es **total** si la eliminación de cualquier atributo A de X hace que la dependencia funcional deje de ser válida. O sea, no tiene atributos redundantes a la izquierda.

$df : X \rightarrow Y$ es **parcial** si es posible eliminar un atributo A de X y la dependencia funcional sigue siendo válida.

$df : X \rightarrow Y$ es **transitiva** si existe un subconjunto de atributos Z que no sea un subconjunto de cualquier clave de R y se cumplen tanto $X \rightarrow Z$ como $Z \rightarrow Y$

Primera Forma Normal (1NF)

Los dominios de los atributos deben incluir solo valores atomicos (no pueden ser multivaluados ni compuestos).

Segunda Forma Normal (2NF)

Un esquema relacional R esta en 2NF si ningún atributo no primo A de R, depende parcialmente de cualquier clave de R.

Tercera Forma Normal (3NF)

Un esquema relacional R esta en 3NF si esta en 2NF y ningún atributo no primo de R depende transitivamente de una clave de R. Es decir R esta en 3NF si siempre que una dependencia funcional $df : X \rightarrow A$ se cumple en R o que **X es Superclave** o de lo contrario **A es atributo primo de R**.

Forma Normal Boyce-CODD (BCNF)

Un esquema relacional esta en BCNF si siempre que una $df : X \rightarrow A$ se cumple en R, **X es superclave de R.**

Algoritmos de Diseño

Descomposición de Relaciones y Joins Sin Perdida

Sea R = (A_1,…,A_n) el esquema relacional universal que contiene TODOS los atributos de la BD.

D = (R_1,…,R_m) se obtiene mediante el algoritmo de descomposición utilizando dependencias funcionales.

Proyección: Dado un conjunto de dependencias funcionales de F sobre R, la proyección de F sobre R_i , $\pi_{R_i}(F)$ donde R_i es un subconjunto de R, es el conjunto de $dfs : X \rightarrow Y$ en F+ tal que los atributos en **X U Y** estén todos contenidos en R_i.

Una descomposición D de R preserva dependencias respecto a F si cumple:

$$\left(\left(\pi_{R_1}(F)\right), \left(\pi_{R_2}(F)\right), \ldots, \left(\pi_{R_m}(F)\right)\right) + = F +$$

Una descomposición D = (R_1,…,R_m) de R tiene la propiedad de **JSP** respecto al conjunto de dependencias funcionales F sobre R, si por cada estado de relación r de R que satisfaga F, se cumple $\left(\left(\pi_{R_1}(r)\right), \left(\pi_{R_2}(r)\right), \ldots, \left(\pi_{R_m}(r)\right)\right) = r$.

Decimos que D = (R_1,R_2) de R tiene JSP respecto a F sobre R si y solo si:

- si $(R_1 \cap R_2) \rightarrow (R_1 - R_2)\ \epsilon\ F +$
- o $(R_1 \cap R_2) \rightarrow (R_2 - R_1)\ \epsilon\ F +$

Descomposición en 3NF con pres de dependencias

1. Encontrar un cubrimiento Minimal G para F
2. Para cada miembro izquierdo X de una dependencia funcional de G
   a. Crear esquema relacional {X U A_1 U…U A_m} en D, donde X → A_i son las únicas dependencias con X como miembro izquierdo.
3. Colocar cualquier atributo restante en un solo er para asegurar la prop de preservación de dependencias.

Test Join Sin Perdidas (Test JSP)

1. Crear matriz S con fila *i* para cada relación R_i en D y columna j por cada atributo A_j en R.
2. S (i, j) = B_{ij}
3. Para cada fila i
    a. Para cada columna j
        i. Si R_i incluye A_j entonces S(i,j) = a_j
4. Repetir hasta que no se modifique S
    a. Para cada df X → Y en F
        i. Igualar los simbolos de los atributos de Y, para aquellas filas que coinciden en los atributos X
5. **Si una fila tiene todos sus simbolos "a", D es con JSP.**

Descomposición en BCNF con JSP

1. Hacer D := { R }
2. Mientras haya un esquema relacional Q en D que no este en BCNF
    a. Escoger un er Q
    b. Encontrar una df X→ *Y* en Q que viole BCNF
    c. Reemplazar Q en D por dos esquemas (Q – Y) y (X U Y)

Descomposición en 3NF con JSP y Pres. Dependencias

1. Encontrar cubrimiento Minimal de G para F
2. Para cada miembro X (izq) de una dependencia funcional que aparezca en G. Crear un esquema relacional l {X U A_1 U…U A_m} en D, donde X → A_i son las únicas dependencias en G con X como miembro izquierdo
3. Colocar cualquier atributo correspondiente en un solo esquema r.
4. Si ninguno de los er contiene una clave de R, crear un er adicional que contenga atributos que formen una clave.

Dependencias Multivaluadas y Cuarta Forma Normal

Si tenemos 2 o más atributos multivaluados **independientes** en el mismo esquema relacional, tendremos que repetir todos los valores de uno de los atributos con cada valor del otro atributo, para que las tuplas de la relación sigan siendo consistentes. Esta restricción se especifica con una **dependencia multivaluada**.

Una $dmv\ X \twoheadrightarrow Y$ especificada sobre el esquema relacional R, especifica la siguiente restricción sobre cualquier relación r de R. **Si existen dos tuplas t_1 y t_2 en r tales que $t_1(X)=t_2(X)$ entonces deberán existir 2 tuplas t_3 y t_4 en r con las siguientes propiedades:**

- $t_3(X) = t_4(X) = t_1(X) = t_2(X)$
- $t_3(Y) = t_1(Y)$ y $t_4(Y) = t_2(Y)$
- $t_3[R - (XY)] = t_2[R - (XY)]$ y $t_1[R - (XY)] = t_4[R - (XY)]$

Una dependencia multivaluada es **trivial** si se cumple que Y es un subconjunto de X, o si la unión de X con Y es R.

Si existen dependencias Multivaluadas **NO TRIVIALES** en una relación, tendremos valores redundantes en las tuplas.

Cuarta Forma Normal

Un er R esta en 4NF respecto a un conjunto de dependencias F si para cada dependencia multivaluada no trivial en F+, X es una superclave de R.

Descomposición con JSP

Sea D = (R_1,R_2) de R tiene JSP respecto a F sobre R si y solo si

- La dmv ($R_1 \cap R_2$) ->> (R_1 - R_2) está en F+ ó
- La dmv ($R_1 \cap R_2$) ->> (R_2 – R_1) está en F+

Algoritmo de Descomposición en 4NF con JSP

1. Hacer D := { R }
2. Mientras haya un esquema relacional Q en D que no esté en 4NF
    a. Comenzar
        i. Escoger un esquema relacional Q
        ii. Encontrar una dmv no trivial X->> Y que viole 4NF
        iii. Reemplazar Q en D por dos esquemas (Q-Y) y (X U Y)
    b. Fin

Estrategias de un DBMS

Procesamiento y Optimización de Consultas

Organización y Acceso a los datos

Que estructuras de datos suelen implementar los manejadores y como se utilizan

A nivel físico se cuenta con:

- **Discos**
  Conjuntos de Bloques de tamaño fijo que son direccionables a través de alguna forma de dirección por alguna instrucción específica (READ).
- **Particiones**
  Subconjunto continuo de los bloques de un disco
- **File System (FS)**
  Estructura de datos que facilita el acceso del S.O a los distintos bloques de la partición.
- **Archivos**
  Estructura lógica que se construye sobre el FS. Es la que usualmente utilizan los programas de aplicación.

Los DBMS re-implementan lo anterior mencionado a su gusto. Toman una partición del disco o a lo sumo un gran archivo sobre el FS y usan esa área como SU disco.

Re-implementan las estructuras de datos y sus algoritmos de manipulación, los algoritmos de sorting y los mecanismos de buffering y paginado.

Para evitar utilizar operaciones de acceso a disco (costosas) al utilizar archivos secuenciales, se usa otro tipo de estructura sobre el disco, llamada Índices.

Tipos de Índices

- Datos físicamente ordenados
  - Primarios
    Archivo ordenado por la clave primaria de la tabla. Con cada clave se guarda un puntero al bloque de disco que contiene el registro.
  - De agrupamiento (cluster)
    Es un archivo ordenado por atributo no clave. Se usan para agrupar los datos que tienen un mismo valor en un atributo.
- Datos físicamente no ordenados
  - Secundarios
    El puntero es a un registro o a un bloque.

Las estructuras típicas para los índices son el **HASH** que tiene un muy buen comportamiento en la inserción y en la recuperación por condiciones de igualdad, pero no funciona bien para condiciones con relaciones de orden. Y por otro lado están los **arboles B y B+.** Estos tienen buen comportamiento en recuperación tanto por condiciones de igualdad como de orden. Tiene un buen comportamiento en la inserción, pero tienen la desventaja que ocupan más espacio en disco.

La comparación entre estructuras se realiza en base a la cantidad de operaciones básicas que se utilizan para ejecutar una operación en la estructura (solo operaciones de I/O) y a su vez se considera que un **bloque de disco** puede almacenar varios nodos de una estructura dada, y que el acceso se hace habitualmente a través de buffers que leen simultáneamente más de un bloque.

Procesamiento de consultas

Diferentes algoritmos para implementar los operadores del algebra relacional

Proceso de Optimización

- **Generación del Algebra (Árbol Canónico)**

- **Generación de planes lógicos (Optimización Heurística)**
  Aplicación de determinadas estrategias y consultas al catalogo para tamaños de las relaciones para transformar el árbol original.
- **Generación de planes físicas (Optimización por Costos)**
  Asociar a cada operación de los planes lógicos, una o mas implementaciones. La implementación depende de las estructuras de datos disponibles.
- **Selección del Plan Final (Optimización por Costos)**
  Evaluación de los planes físicos generados, en base a las cantidades de operaciones de I/O que realiza cada algoritmo.

Optimización de Consultas

Estrategias que utiliza un manejador para modificar las consultas y para elegir los algoritmos a ejecutar.

Optimización Heurística

Basada en equivalencia de la expresión del algebra y ciertas estrategias básicas para limitar el tamaño de los resultados.

Se basa en aplicar equivalencias de los operadores del algebra de forma que las selecciones se apliquen lo antes posible, y que las ramas izquierdas de los joins, sean más chicas que las derechas.

Reglas:

1. Cambiar las selecciones conjuntivas por una “cascada” de selecciones simples
2. Mover las selecciones lo más abajo posible en el árbol.
3. Poner a la izquierda de los productos las hojas que generen menos tuplas
4. Cambiar secuencias de selecciones y productos por joins.
5. Mover las proyecciones lo más abajo posible en el árbol, agregando las proyecciones que sean necesarias.

Optimización por Costos

Basada en estimaciones y datos del catalogo que permiten seleccionar un mejor plan de proceso.

El plan físico asocia a cada operación que aparece en un plan lógico, una implementación. Como se pueden considerar diferentes implementaciones en cada operador, un mismo plan lógico puede originar distintos planes físicos.

Es necesario estimar el costo de los diferentes planes, y elegir el de costo mínimo.

Para evaluar el costo es necesario considerar ciertos parámetros que tienen influencia en el cálculo de la cantidad de operaciones.

Implementación de los operadores

Es importante saber cuál es la estrategia de implementación. Si es **PIPELINED**, hay operadores que se ejecutan simultáneamente y pueden pasarse los resultados a medida que se generan. En cambio, si es **NO PIPELINED**, los operadores se ejecutan secuencialmente y es necesario grabar los resultados intermedios.

La estrategia puede depender de los operadores, nosotros asumimos que la selección y el join son NO PIPELINED, mientras que la proyección es PIPELINED.

En el calculo de los costos, se considera la lectura y la grabación en disco. Siempre se realizan las operaciones de a bloque.

Los costos de lectura dependen de la organización de los datos pero el costo de grabación siempre es el costo de grabar todo el resultado R. En los algoritmos, se considera la lectura, pero a ello se le debe agregar el costo de grabación.

Tipos de Implementación

- Selección con condición C
  Puede ser búsqueda lineal o binaria. La primera consiste en leer cada registro, y si

cumple la condición se pone en el resultado. La segunda, consiste en leer el bloque medio y en función de la condición leer el del medio de la primera o segunda mitad. La ultima forma, requiere que los datos estén ordenados.

- Selección con Indices
  Se puede realizar por Indice Primario o Cluster, donde los registros deben estar ordenados y consiste en acceder por el índice y recuperar los registros que cumplen la condición. Luego, se puede realizar con Hash donde los costos de lectura son constantes (1 o 2) pero requiere que sean condiciones de igualdad sobre los índices. O se puede implementar como Indices Secundarios con B+ y el costo de lectura es un moco màs elevado que las anteriores, pero no tiene restricciones.
- Joins con condición de igualdad en conjuntos de atributos
  - Loop Anidado por Registros
    Para cada registro de R, se accede a todos los bloques de S y se combina ese registro en R con todos los de S.
  - Loop Anidado por Bloques
    Para cada bloque de R combinar todos los registros de ese bloque con los de S, luego pasar al siguiente bloque de S.
  - Sort-Merge Join
    Recorrer R y S, en paralelo combinando los registros. Para este caso, se deben tener los registros ordenados. Si no es asi, se deben agregar los costos de ordenación.
  - Index Join (Single Loop)
    Recorrer R y acceder por el índice a S. Para esto, se debe tener un índice para S.

Control de Concurrencia

Coordinación de procesos concurrentes que operan sobre datos compartidos y que podrían interferir entre ellos.

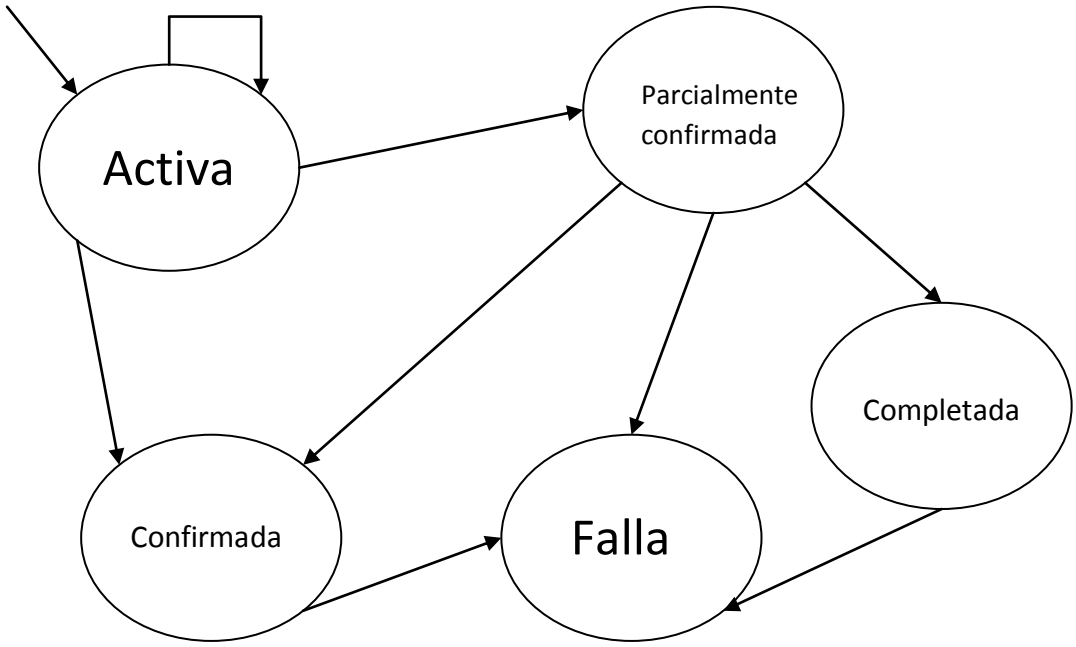
Transacciones

Operaciones complejas que se ven como una sola

A cada proceso que se ejecuta se le llama transacción si cumple las propiedades ACID

- **Atomicidad**
  Desde el punto de vista del estado, o se ejecuta o no se ejecuta
- **Consistencia**
  Siempre lleva a la base desde un estado consistente a otro
- **Aislamiento**
  Una transacción no debe interferir con otra
- **Durabilidad**
  Los cambios de una transacción confirmada deben ser permanentes en la base

Para que una transacción cumpla con ACID la transacción debe pasar o seguir el siguiente ciclo:



Concurrencia

Que debe considerar el manejador para que varias transacciones se ejecuten simultáneamente y produzcan resultados concretos en la base.

Para el control de la concurrencia se necesita el concepto de transacción como hemos definido anteriormente y a su vez las siguientes operaciones y conceptos.

- **Read – $r_i(x)$**
  La transacción i lee el ítem X de la base
- **Write – $w_i(x)$**
  La transacción i escribe el ítem X de la base
- **Commit - c_i**
  La transacción i confirma que todas sus modificaciones deben ser permanentes
- **Abort - a_i**
  La transacción i indica que ninguna de sus modificaciones deben ser permanentes.
- **ROLLBACK**
  Es la acción de recuperar el estado anterior de la base frente a un abort de una transacción.

Manejador de Transacciones

Recibe instrucciones desde los programas, las reordena sin cambiar el orden relativo de los read y write, agrega instrucciones si es necesario (ni read ni write), hace demorar algunas ejecuciones tratando de implementar ACID mientras pueda.

Definimos una **HISTORIA** como la ordenación de todas las operaciones que intervienen en las transacciones de un determinado conjunto de transacciones, siempre que estas aparezcan en el mismo orden que en la transacción.

En una historia se pueden encontrar **OPERACIONES EN CONFLICTO**, que cumplen a la vez

- Pertenecen a distintas transacciones.
- Acceden al mismo ítem.
- Una de ellas es un **write.**

Decimos que una historia es **COMPLETA** si tiene todas las operaciones de las transacciones involucradas y las operaciones en conflicto aparecen en el mismo orden.

T_1 lee de T_2 en H si $w_2(x)$ está antes de $r_1(x)$ y en el medio no hay otro $w_j(x)$ tal que c_j este en H y tampoco existe un a_2.

Serialización

Si las transacciones se ejecutaran siempre en forma serial, no habría concurrencia pero los datos siempre serían correctos. Si las historias son entrelazadas puede suceder que queden datos erróneos que no se puedan corregir o que si una transacción aborta otra también tenga que abortar.

Una historia es **SERIALIZABLE** si es equivalente a una historia serial con las mismas transacciones. La primera noción de equivalencia es la intuitiva, es decir dos historias son equivalentes si dejan la base en el mismo estado. También se puede resolver por equivalencia, si dos historias tienen las operaciones en conflicto en el mismo orden, también son equivalentes. Luego se puede definir un tercer criterio que es el de las vistas, si cada T_i lee de las mismas T_k en H y H´, son equivalentes.

Testeo de Seriabilidad por Conflictos

El método consiste en construir un grafo de Seriabilidad o precedencia de la siguiente forma:

1. Poner un nodo para cada transacción en la historia
2. Si $r_j(x)$ esta después de $w_i(x)$, entonces hay un arco de T_i a T_j
3. Si $w_j(x)$ esta después de $r_i(x)$, entonces hay un arco de T_i a T_j
4. Si $w_j(x)$ está después de $w_i(x)$, entonces hay un arco de T_i a T_j.

Siempre se pone un arco si hay una pareja de operaciones en conflicto, desde la primera transacción a la segunda.

Si el grafo es acíclico, la historia es serializable.

El manejador debe asegurar la construcción de historias serializables, recuperables y que no tengan abortos en cascada. La forma de realizar esto es demorar las operaciones en conflicto con otras anteriores hasta que se terminen. Puede ser con **LOCKING** o con **TIMESTAMPS**.

Una historia es **RECUPERABLE** si ninguna transacción confirma hasta que se confirmaron todas las transacciones desde las cuales se leyeron ítems. Es decir, los commits están en el mismo orden que las lecturas.

Una historia **EVITA ABORTOS EN CASCADA** si ninguna transacción lee de transacciones no confirmadas. Es decir, los commits tienen que estar antes de los reads siguientes.

Una historia es **ESTRICTA** si ninguna transacción lee o escribe hasta que todas las transacciones que escribieron ese ítem fueron confirmadas. Los commits deben estar antes que los reads o writes de las siguientes.

Técnicas de Locking

Consideramos dos operaciones nuevas en una transacción T_i: **$lock_i(x)$ y $unlock_i(x)$**. Cuando se ejecuta $lock_i(x)$ el DBMS hace que cualquier $lock_j(x)$ no termine hasta que T_i ejecute $unlock_i(x)$.

El lock puede ser **binario** como mencionamos recientemente, siendo fácil de implementar pero que tiene la desventaja de negar la lectura a otra transacción aunque no sea necesario. Y si no puede ser **Read/Write Lock**, aquí se deben considerar tres operaciones **$read\_lock_i(x)$, $write\_lock_i(x)$, $unlock_i(x)$**. Este tipo tiene la ventaja de que permite que varias operaciones

hagan un read_lock simultáneamente sobre el mismo ítem. En cambio, es un poco más complicado de implementar.

Reglas de uso de LOCKS – Protocolos de Locking

Si bien existen reglas para usar los distintos tipos de locks anteriores, estas por si solas no garantizan la Seriabilidad de la historia.

Entonces, mediante un protocolo de locking, se define un conjunto de reglas que sean más fuertes y si garanticen la Seriabilidad.

En el protocolo **2PL – Two Phase Locking** existen dos fases en una transacción. La primera la fase de crecimiento donde sea crean los locks y luego la fase de contracción donde se liberan los locks.

- **2PL BASICO**
  Solo exige que se cumplan las dos fases, es muy simple de implementar pero es susceptible a los deadlocks.

- **2PL CONSERVADOR**
  Exige que todos los locks se hagan antes del comienzo de la transacción. No pueden ocurrir deadlocks pero existe la predeclaración de todos los ítems que se van a leer o grabar.

- **2PL ESTRICTO**
  Exige que no se libere ningún write_lock hasta después de terminar la transacción. Garantiza historias estrictas, pero pueden ocurrir deadlocks.

- **2PL RIGUROSO**
  Exige que no libere ningún read_lock o write_lock hasta depues de terminar la transacción. Es mas simple de implementar que el estricto y pueden ocurrir deadlocks.

Un **DEADLOCK** es cuando dos o más transacciones esperan unas por otras. Para determinar si una historia tiene deadlocks, construir el grafo de espera (arco de T1 a T2 si T1 esta tratando de lockear un ítem que T2 tiene lockeado). Si el árbol tiene ciclos, entonces hay un deadlock.

Para solucionar el problema de los DEADLOCKS, existen protocolos de prevención, otros protocolos de detección y otros de TimeOut.

En los de detección La idea es mantener un grafo de espera y si hay deadlocks, seleccionar la víctima y que aborte. Es útil para transacciones cíclicas.

La idea del TimeOut es que si una transacción espera por demasiado tiempo, esta es abortada.

Los protocolos de prevención se basan en TIMESTAMPS (marcas de tiempo)