UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CONTADURÍA Y ADMINISTRACIÓN

APUNTES DE LA MATERIA DE BASES DE DATOS

L. I. CARLOS FRANCISCO MÉNDEZ CRUZ

{cmendezc@iingen.unam.mx}

1. MARCO TEÓRICO CONCEPTUAL

1.1. BASE DE DATOS

Tal vez el origen más remoto de las bases de datos esta en las pinturas rupestres.[1] Las primeras bibliotecas antiguas donde se guardaban rollos de pergaminos fueron ejemplos de la necesidad del hombre por almacenar su conocimiento para recuperarlo después.

Para entender el concepto de base de datos debemos aislarla del entorno computacional. Una base de datos es una colección de información almacenada y organizada de alguna manera con un fin determinado.[2]

¿Cuales podrían ser ejemplos de una base de datos?

Los tipos de datos que podríamos guardar en una base de datos son: matemáticos, estadísticos, financieros, históricos, artísticos, sobre personas y en general sobre cualquier cosa del universo conocido.

Pero situándonos en nuestra carrera podemos decir que una base de datos es:

"Es un conjunto de datos que es utilizado por los sistemas de aplicación de alguna empresa dada."[3]

Todavía podemos encontrar empresas que llevan sus registros en innumerables libros y hojas de papel, esta es una práctica antigua que ayudó al impulso de las computadoras. Descartando el origen militar de la tecnología computacional, la otra gran utilidad que se les dio a estos aparatos en sus inicios fue el almacenamiento de registros y datos de maneras que permitieran:

- Reducir papel.
- Mayor velocidad en recuperar datos (automático vs manual).
- Eliminar trabajo de archivo.
- Mejor actualización de la información.
- Control centralizado de los datos.

1.2. SISTEMA DE BASE DE DATOS

Con el uso de las computadoras para el almacenamiento de información relevante surgió el concepto de sistema de base de datos:

[1] ¿Por qué crees que se pueda decir esto?
[2] Almacenarla, hacer contabilidad con ella o simplemente tener un registro de la misma.
[3] C. J. Date, Introducción a los sistemas de bases de datos.

> Los sistemas de base de datos se diseñan para manejar grandes volúmenes de información, la manipulación de los datos involucra tanto la definición de estructuras para el almacenamiento de la información como la provisión de mecanismos para la manipulación de la información, además un sistema de base de datos debe de tener implementados mecanismos de seguridad que garanticen la integridad de la información, a pesar de caídas del sistema o intentos de accesos no autorizados.
>
> Un objetivo principal de un sistema de base de datos es proporcionar a los usuarios finales una visión abstracta de los datos, esto se logra escondiendo ciertos detalles de como se almacenan y mantienen los datos.[4]

Un sistema de base de datos es un "sistema computarizado para almacenar información y permitir a los usuarios recuperar y actualizar esa información."[5]

Podemos detectar 4 elementos que lo conforman:

Datos

Compartidos

Hardware

Software

Sistema operativo multiusuario

Servidor de bases de datos

Usuarios

- De sistemas
  - Programadores de aplicaciones
  - Programadores de bases de datos
- Finales
- DBA

Estos sistemas de bases de datos, a quienes podríamos llamarles ahora: sistemas manejadores de bases de datos, surgieron con objetivos más amplios:

[4] Microsoft, Manual de SQL Server, p. 44.
[5] C. J. Date, Op. Cit.

1.3. OBJETIVOS DE UN SISTEMA DE BASES DE DATOS

Los objetivos principales de un sistema de base de datos son disminuir los siguientes aspectos:

1. Redundancia e inconsistencia en los datos. Información repetida que aumenta el costo de almacenamiento y acceso a los datos. Falta de concordancia entre datos que se supone son iguales. Bajas vs pedidos, padres vs hijos.

2. Dificultad para tener acceso a los datos. Cubrir las necesidades de información del usuario o entidad, esto implica prevenir cualquier consulta o situación posible de ser solicitada.

3. Aislamiento de los datos. En las primeras bases de datos se utilizaban grupos de archivos que muchas veces eran de distinto tipo. Hoy en día aun sigue este problema por causa de los malos diseños de bases de datos.

4. Anomalías del acceso concurrente. Evitar inconsistencias por actualizaciones de usuarios que aceden al mismo tiempo a la base de datos.

5. Problemas de seguridad. La información que se guarda en una base de datos no debe ser vista con la misma profundidad por todos los usuarios de la misma. Existen niveles de usuarios y restricciones para consultar la información. También se requieren niveles de seguridad en contra de haking o craking.

6. Problemas de integridad. Los datos que ingresan a una base deben estar bien filtrados de manera que no se almacene información errónea o sin el formato adecuado. Para esto de implementan restricciones de integridad basadas en reglas de negocios integridad basadas en reglas de negocios.

EJERCICIO

Responda lo que se le pide de forma clara y concisa.

1. Defina y ejemplifique una base de datos (no piense en computadoras)

2. Pensando en la informática diga qué es una base de datos.

3. ¿Qué beneficios se obtienen con el uso de computadoras para almacenar bases de datos?

4. ¿Qué es un sistema manejador de bases de datos (DBMS)?

5. ¿Cuáles son los cuatro elementos que forman un DBMS?

6. Explique cada uno de los objetivos de los RDBMS. (ACLARAR LOS EN CLASE)

INTEGRIDAD vs CONSISTENCIA

INTEGRIDAD→ hacer que se cumplan las reglas del negocio (restricciones y políticas de la empresa)

EJEMPLOS

CONSISTENCIA → USUARIO

INTEGRIDAD → IMPLEMENTADOR

1.4. SISTEMA MANEJADOR DE BASE DE DATOS (DBMS)

El primer elemento: DATOS ¿Cómo se guardan y manipulan los datos?

Estructuras de almacenamiento basadas en un DISEÑO (ESQUEMAS)→DDL

Mecanismos de manipulación de información → DML

Procedimientos para la explotación de la información → QUERIES

Los sistemas manejadores de bases de datos se diseñan para manejar grandes volúmenes de información, la manipulación de los datos involucra tanto la definición de estructuras para el almacenamiento de la información (DDL), también llamadas esquemas, como la provisión de mecanismos para la manipulación de la información (DML), además un sistema de base de datos debe de tener implementados mecanismos de seguridad que garanticen la integridad de la información, a pesar de caídas del sistema o intentos de accesos no autorizados.

1.5. INICIO DE LAS BASES DE DATOS

CAMPO

Unidad mínima de almacenamiento.

REGISTRO

Un conjunto de campos relacionados.

ARCHIVO DE DATOS

Colección de registros almacenados siguiendo una estructura homogénea.

EJERCICIO

Responda lo que se le pide de forma clara y concisa.

1. Qué es una base de datos

2. Ejemplo de una base de datos real o física
3. Qué beneficios se obtienen con el uso de bases de datos en computadoras
4. Qué es un sistema manejador de bases de datos
5. Cuales son los elementos de un sistema manejador de bases de datos
6. Cuáles son los tres elementos para el manejo de los datos
7. Qué es el lenguaje DDL
8. Qué es el lenguaje DML
9. Cuales son los objetivos de un sistema manejador de bases de datos
10. Qué es consistencia
11. Qué es integridad
12. Ejemplo de una regla de negocio
13. Qué es campo
14. Qué es registro
15. Qué es archivo de datos

1.6 MODELO DE BASE DE DATOS

C. J. DATE

Es un método abstracto para organizar los elementos de datos y sus relaciones.

1.6.1. Modelo relacional

C. J. DATE

El modelo relacional se basa en una teoría abstracta de datos que esta basada en ciertos aspectos de las matemáticas (teoría de conjuntos y lógica de predicados), en el que el usuario percibe la información como tablas y nada más que tablas.

Los principios fueron establecidos por E. F. Codd en 1969 en su artículo “A Relational Model of Data for Large Shared Data Bank”

1.6.2. Relación

Es un término matemático para una tabla.

Que significan las relaciones:

- Dado un conjunto de n tipos o dominios Ti(i=1,2,...,n), que son no necesariamente todos distintos, r es una relación de esos tipos si consta de dos partes: encabezado y cuerpo, donde:

- El encabezado de r denota un cierto predicado o función valuada como verdadera.

- Cada fila en el cuerpo de r denota una cierta proposición verdadera, obtenida del predicado por medio de la sustitución de ciertos valores de argumento del tipo apropiado en los indicadores de posición o parámetros de ese predicado (como crear un ejemplo del predicado).

Predicado:

El empleado EMP# se llama NOEMP, trabaja en el departamento DEPTO# y gana un salario SALARIO

Proposición verdadera:

El empleado E1 se llama López, trabaja en el departamento D1 y gana el salario 40K

El empleado E2 se llama Cheng, trabaja en el departamento D1 y gana el salario 42K

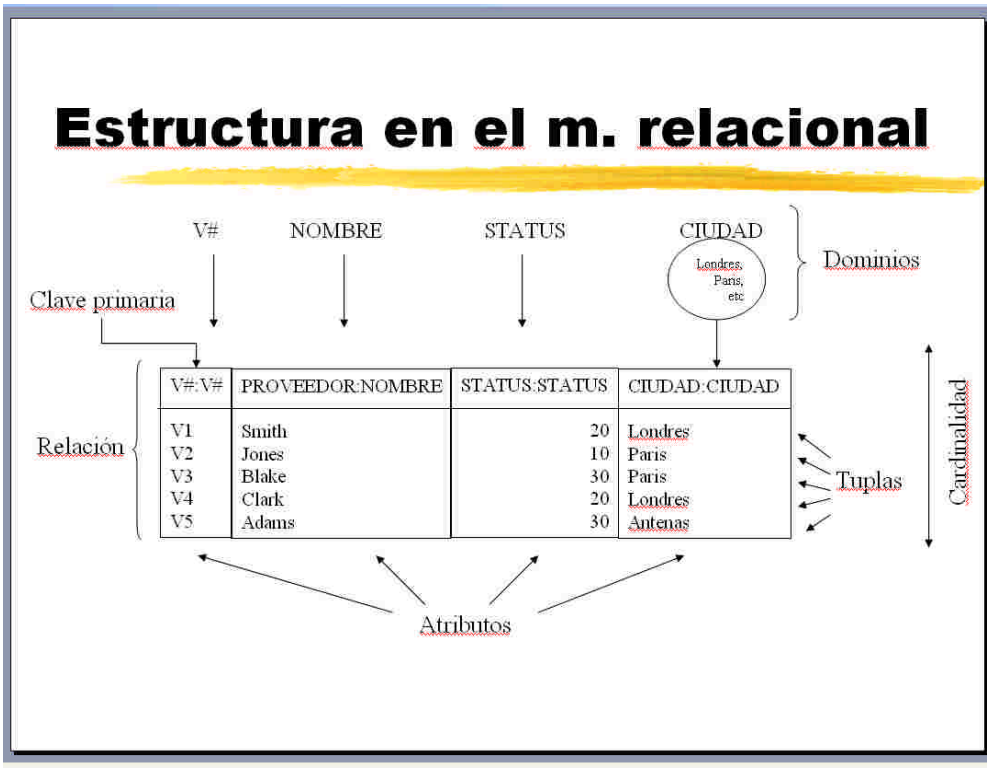
1.6.3. Encabezado

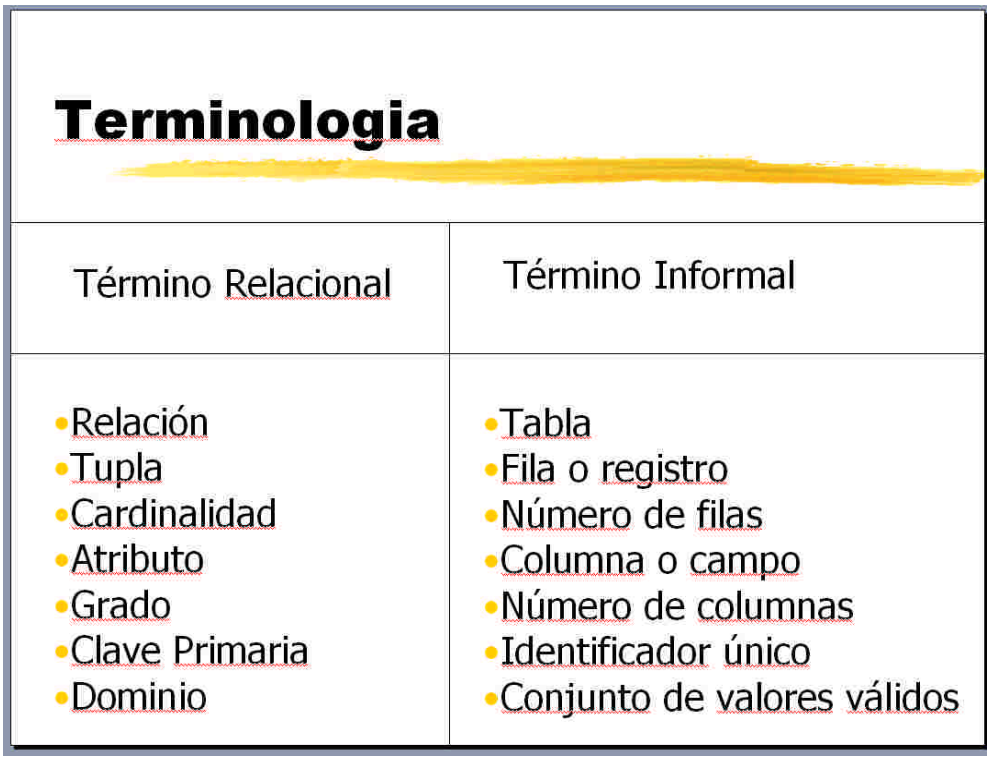
Conjunto de n atributos necesariamente distintos de la forma NombredeAtributo:NombredeTipo.

1.6.4. Cuerpo

Conjunto de m tuplas en donde cada una es un conjunto de componentes de la forma NombredeAtributo:ValordeAtributo

A los valores de m y n se les denomina cardinalidad y grado.





¿Cual es la diferencia entre una relación y una tabla?

1.6.5. Propiedades de las relaciones

- No existen tuplas duplicadas.

La teoría de conjuntos descarta los elementos duplicados, además de que no podemos afirmar dos veces el mismo hecho verdadero.

- Las tuplas están en desorden, de arriba hacia abajo.

La teoría de conjuntos nos dice que sus elementos están en desorden. Esto implica que no existan los conceptos de “tupla 1”, “siguiente tupla”, “cuarta tupla”; luego entonces nos referimos a una tupla por su proposición verdadera.

- Los atributos están en desorden, de izquierda a derecha.

Esto es por que el encabezado también es un conjunto (de atributos). Tampoco hay conceptos de “siguiente atributo”, “primer atributo”, siempre se hace referencia a ellos por nombre, nunca por posición.

1.6.6. Creación de una relación en el modelo relacional

VAR V BASE RELATION

{ ISBN ISBN,

NOMBRE NOMBRE,

AUTOR NOMBRE

SINOPSIS TEXTO}

1.6.7. Creación de una tabla en SQL

CREATE TABLE [database.[owner].]table_name

(col_name col_properties,

col_name2 col_properties)

1.6.8. Borrado de una relación en el modelo relacional

DROP VAR <nombrederelación>

1.6.9. Borrado de una tabla en SQL

DROP TABLE <nombrederelación>

TRUNCATE TABLE <nombrederelación>

El SQL Server puede tener hasta 2 billones de tablas por cada base de datos y 250 columnas por tabla.

1.6.10. Actualización de variables en el Modelo relacional

INSERT INTO <relación> <expresiónrelacional>

DELETE <relación> [WHERE <expresi'on lógica>]

UPDATE <relación> [WHERE <expresión lógica>] <nombre de atributo:=expresión>

1.6.11. Actualización de variables en SQL

INSERT [INTO]

{table_name } [(column_list)]

{values_list | select_statement}

DELETE [FROM] {table_name}

[WHERE clause]

UPDATE {table_name}

SET {column_name = variable_name}

[WHERE clause]

1.6.12. Dominio

No es más que un tipo de datos, puede ser simple (char, integer) o definido por el usuario.

Pueden de ser de cualquier clase, desde números y cadenas hasta grabaciones de sonido, mapas o dibujos.

Esta compuesto por todos los valores posibles del tipo en cuestión, así el tipo (dominio) INTEGER se compone de todos los número enteros posibles y el tipo CIUDAD es el conjunto de todas las ciudades posibles.

Cada dominio tiene asociados distintos tipos de operadores que permiten su manipulación. Estos son =, =, *, / SUBSTRING(), &, etc., los cuales dependen de cada sistema de bases de datos. Los operadores validos para cada dominios se determinan por lo que representa en el modelo (semántica), no por su representación física.

Se cuenta también con la posibilidad de hacer conversiones de tipos de datos. De esta manera se pueden realizar operaciones entre dominios de diferentes tipos.

1.6.13. Creación de tipos en el Modelo relacional

TYPE ISBN POSSREP(CHAR)

TYPE NOMBRE POSSREP(CHAR)

TYPE NUMTITULO POSSREP(INTEGER)

1.6.14. Borrado de tipos en el Modelo relacional

DROP TYPE <nombredetipo>

1.6.15. Creación de tipos en SQL

EXEC **sp_addtype** type_name, 'type_properties'

1.6.16. Borrado de tipos en SQL

EXEC **sp_droptype** typename

1.6.17. Conversión de tipos en SQL

CONVERT (datatype(length), expression[,style])

1.7. ALGEBRA RELACIONAL

1.7.1. RESTRINGIR

Regresa una relación que contiene todas las tuplas de una relación especificada que satisfacen una condición especificada.

1.7.2. PROYECTAR

Regresa una relación que contiene todas las tuplas o subtuplas que quedan en una relación especificada después de quitar los atributos específicos.

1.7.3. PRODUCTO (UNRESTRICTED JOIN)

Regresa una relación que contiene todas la tuplas posibles que son una combinación de dos tuplas, una de cada una de dos relaciones específicas.

1.7.4 UNION

Regresa una relación que contiene todas las tuplas que aparecen en una o en las dos relaciones específicas.

Combines the results of two or more queries into a single results set consisting of all the rows belonging to all queries in the union.

Syntax

SELECT select_list [INTO clause]

[FROM clause]

[WHERE clause]

[GROUP BY clause]

[HAVING clause]

[UNION [ALL]

SELECT select_list

[FROM clause]

[WHERE clause]

[GROUP BY clause]

[HAVING clause]...]

[ORDER BY clause]

[COMPUTE clause]

INTERSECCION

Regresa una relación que contiene todas las tuplas que aparecen en las dos relaciones específicadas (en ambas, no en una u otra).

1.7.5. DIFERENCIA (OUTER JOIN)

Regresa una relación que contiene todas las tuplas que aparecen en la primera pero no en la segunda de las dos relaciones específicas.

*= incluye todos los renglones de la primera tabla

=* incluye todos los renglones de la segunda tabla

1.7.6. JUNTAR (NATURAL JOIN)

Regresa una relación que contiene todas las tuplas posibles que son una combinación de dos tuplas de cada una de dos relaciones especificadas, tales que las dos tuplas que contribuyen a cualquier combinación dada tengan un valor común para los atributos comunes de las dos relaciones.

1.7.7. DIVIDIR

Toma dos relaciones unarias y una relación binaria y regresa una relación que contiene todas las tuplas de una relación unaria que aparecen en la relación binaria y que a la vez coinciden con todas las tuplas de la otra relación unaria.

Syntax

SELECT [ALL | DISTINCT] select_list

[INTO [new_table_name]]

[FROM {table_name | view_name}[(optimizer_hints)]

[[, {table_name2 | view_name2}[(optimizer_hints)]

[..., {table_name16 | view_name16}[(optimizer_hints)]]]

[WHERE clause]

[GROUP BY clause]

[HAVING clause]

[ORDER BY clause]

[COMPUTE clause]

[FOR BROWSE]

1.7.8. Operadores entre relaciones

Los operadores disponibles para que el usuario manipule estas tablas son operadores que derivan tablas a partir de tablas. Los más importantes son restringuir(select where para registros), proyectar (select sin where para columnas) y juntar (join). El resultado de estas operaciones siempre es una tabla (propiedad de cierre).

SQL

SELECT [ALL|DISTINCT] select_list

[FROM {table_name|view_name}, {table_name2|view_name2}]

[WHERE clause]

[GROUP BY clause]

[HAVING clause]

[ORDER BY clause]

[COMPUTE clause]

1.7.9 Optimización

Este modelo permite optimizar la navegación entre los datos ante una petición del usuario. Esto es porque permite usar operaciones relacionales como restringir, proyectar y juntar. En estas nos preocupamos en que queremos y no en como.

1.8. VISTAS

Podemos pensarlas como sub-conjuntos de un conjunto.

Es una relación derivada y su valor en un momento dado esta determinado por el resultado de evaluar cierta expresión relacional en ese momento.

Es como una ventana hacia el interior de la relación, no es una copia independiente de ella y por tanto puede mostrar cualquier cambio hecho a la relación.

Las relaciones base existen realmente, las vistas no existen realmente sólo proporcionan distintas formas de ver a los datos reales.

SQL

CREATE VIEW [owner.] view_name

(col_name, col_name...)

AS

select_statement

1.9. TRIGGERS

Ejemplo de la implementación de un trigger en SQL Server

```
CREATE TABLE prestamo
(
f_pre datetime NOT NULL DEFAULT GETDATE(),
nousuario smallint NOT NULL,
nolibro smallint NOT NULL
)
CREATE TABLE prestamohistorico
(
f_pre datetime NOT NULL,
nousuario smallint NOT NULL,
nolibro smallint NOT NULL
```

```
)
CREATE TRIGGER trrevisaprestamos
ON prestamo
FOR INSERT
AS
        IF (SELECT COUNT(*)
                FROM prestamo, inserted
                WHERE prestamo.nousuario = inserted.nousuario) > 4
        BEGIN
                PRINT 'NO SE PUEDEN PRESTAR MAS DE CUATRO LIBROS A
LA VEZ'
                ROLLBACK TRANSACTION
        END
CREATE TRIGGER trhistorico
ON prestamo
FOR DELETE
AS
        INSERT INTO prestamohistorico
                SELECT deleted.f_pre, deleted.nousuario, deleted.nolibro
                FROM deleted
```

1.10. NORMALIZACIÓN

1.10.1. CLAVE CANDIDATA

Sea K un conjunto de atributos de la relación R. Entonces K es una clave candidata de R si, y solamente si, posee las dos propiedades siguientes:

1. Unicidad: jamás, ningún valor de válido de R contiene dos tuplas distintas con el mismo valor de K.

Irreductibilidad: Ningún subconjunto propio de K tiene la propiedad de unicidad.

Una clave candidata es el mecanismo de direccionamiento en el nivel de tupla.

Cuando el clave candidata es un conjunto se la conoce como superclave y podemos definirla como un suconjunto SK de los atributos de R, tales que la dependencia funcional SK → A es verdadera para todos los atributos A de R.

1.10.2. DEPENDENCIAS FUNCIONALES (DF)

Es un vínculo muchos a uno que va de un conjunto de atributos a otro dentro de una determinada relación. Por ejemplo, en el caso de la relación de envíos VP existe una dependencia funcional del conjunto de atributos {V#, P#} al conjunto de atributos {CANT}. Lo que esto significa es que dentro de la relación VP:

1. Para cualquier valor dado del par de atributos V# y P#, sólo existe un valor correspondiente del atributo CANT, pero

2. Muchos valores distintos del par de atributos V# y P# pueden tener (en general) el mismo valor correspondiente del atributo CANT.

(ejemplo pag. 76)

Sea R una relación y sean X y Y subconjuntos cualesquiera del conjunto de atributos de R. Entonces decimos que Y es ***dependiente funcionalmente*** de X en símbolos,

X → Y

(lea "X determina funcionalmente a Y", o simplemente "X flecha Y") si y sólo si en todo valor válido posible de R, cada valor X está asociado precisamente con un valor de Y. En otras palabras, en todo valor válido posible de R, siempre que dos tuplas coincidan en su valor X, también coincidirán en su valor Y.

Por ejemplo:

{V#} → {CIUDAD}

{V#, P#} → {CANT}

{V#, P#} → {CIUDAD}

{V#, P#} → {CIUDAD, CANT}

{V#, P#} → {V#}

{V#, P#} → {V#, P#, CIUDAD, CANT}

{V#} → {CANT} NO

{CANT} → {V#} NO

¿POR QUÉ?

La parte izquierda de un DF se denomina ***determinante*** y la derecha ***dependiente***.

1.10.3. Dependencias triviales y no triviales

Una dependencia trivial se da si y solo si la parte derecha es un subconjunto de la parte izquierda.

Las no triviales son la que no son triviales. Estas son las que realmente importan en el proceso de normalización mientras que las otras no son tomadas en cuenta.

1.10.4. Dependencias transitivas

{V#, P#} → {CIUDAD, CANT}

implica a las siguientes dependencias:

{V#, P#} → CIUDAD

{V#, P#} → CANT

Supongamos que tenemos una relación R con tres atributos A, B y C, tales que las DF's A → B y B → C con válidas para R. Entonces es fácil ver que la DF A → C también es válida para R. Aquí la DF A → C es un ejemplo de DF transitiva; decimos que C depende de A transitivamente a través de B.

1.10.5. Cierre de un conjunto de dependencias

Axiomas de Armstrong

Al conjunto de todas las DF's implicadas por un conjunto dado S de DF's se le llama ***cierre*** de S y se escribe S+.

Reglas de inferencia para obtener nuevas DF's a partir de las ya dadas.

Sean A, B y C subconjuntos cualesquiera del conjunto de atributos de la relación dada R, entonces:

1. Reflexividad: Si B es subconjunto de A, entonces A → B.
2. Aumento: Si A → B, entonces AC → BC.
3. Transitividad: Si A → B y B → C, entonces A → C.
4. Autodeterminación: A → A.
5. Descomposición: Si A → BC, entonces A → B y A → C.
6. Unión: Si A → B y A → C, entonces A → BC.
7. Composición: Si A → B y C → D, entonces AC → BD.

Ejercicio:

A → BC

B → E

CD → EF

Resultado:

1. A → BC (dado)

2. A → C (1, descomposición)

3. AD → CD (2, aumento)

4. CD → EF (dado)

5. AD → EF (3 y 4, transitividad)

6. AD → F (5, descomposición)

1.10.6. Dependencias irreducibles

Un conjunto S de DF's es irreducible si y sólo si satisface las siguientes tres propiedades:

1. La parte derecha (el dependiente) de toda DF en S involucra sólo un atributo.

2. La parte izquierda (el determinante) de toda DF en S es a su vez irreducible a la izquierda, lo que significa que no es posible descartar ningún atributo del determinante sin cambiar el cierre

3. No es posible descartar de S ninguna DF sin cambiar el cierre S+ (perder información)

1.11. FORMAS NORMALES

1.11.1. Primera forma normal:

Una relación está en 1FN si y sólo si, en cada valor válido de esa relación, toda tupla contiene exactamente un valor para cada atributo.

1.11.2. Segunda forma normal:

Una relación está en 2FN si y sólo si está en 1FN y todo atributo que no sea clave es dependiente irreduciblemente de la clave primaria.

1.11.3. Tercera forma normal:

Una relación esta en 3FN si y sólo si está en 2FN y todos los atributos que no son clave son dependientes en forma no transitiva de la clave primaria.

1.11.4. Forma normal de Boyce/Codd:

Una relación está en FNBC si y sólo si toda DF no trivial, irreducible a la izquierda, tiene una clave candidata como su determinante.

1.12. DESCOMPOSICION SIN PERDIDA

El proceso de ***descomposición*** es en realidad un proceso de ***proyección*** y decimos que es ***sin pérdida*** si ***juntamos*** de nuevo las proyecciones y regresamos a la relación original.

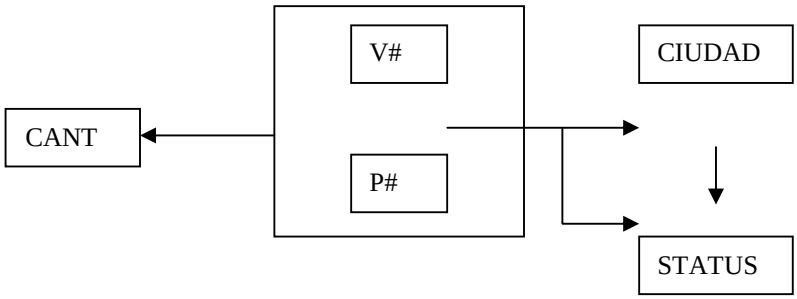
1.13. EJEMPLO DEL ANÁLISIS DE DEPENDENCIAS FUNCIONALES

PRIMERA {V#, STATUS, CIUDAD, P#, CANT}

PK {V#, P#}

STATUS es dependiente funcionalmente de CIUDAD. El significado de esta restricción es que el status de un proveedor se determina mediante la ubicación de ese proveedor; por ejemplo, todos los proveedores de Londres DEBEN tener un status de 20.

DIAGRAMA DE DEPENDENCIAS



PROBLEMAS DE

INSERT: No se puede registrar un proveedor hasta que se le suministre una parte.

DELETE: Si borro un proveedor en particular se borra la información de la ciudad.

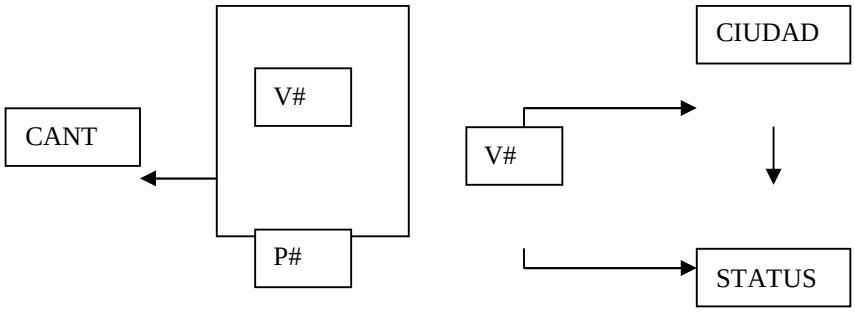
UPDATE: Si un proveedor cambia de ciudad tengo que actualizar todas las tuplas donde que refieran a ese proveedor.

REDUCCION

SEGUNDA {V#, STATUS, CIUDAD}

VP {V#, P#, CANT}

DIAGRAMA DE DEPENDENCIAS



VALIDACION

HACER EL JOIN DE LAS PROYECCIONES RESULTANTES DE LA REDUCCION, PARA VERIFICAR SI RESULTA LA RELACION ORIGINAL.

PROBLEMAS DE INSERT, DELETE y UPDATE

SEGUNDA FORMA NORMAL

Si y sólo si está en 1FN y todo atributo no PK es dependiente de ella.

REDUCCION

VC {V#, CIUDAD}

CS {CIUDAD, STATUS}

¿OPCION MALA?

VC {V#, CIUDAD}

VS {V#, STATUS}

VALIDACION

HACER EL JOIN DE LAS PROYECCIONES RESULTANTES DE LA REDUCCION, PARA VERIFICAR SI RESULTA LA RELACION ORIGINAL.

2. DISEÑO DE BASES DE DATOS

2.1. MODELO ENTIDAD-RELACIÓN

2.1.1. Modelos de datos

Es una representación de la realidad que contiene las características generales de algo que se va a realizar. Por lo general es de manera gráfica.

2.1.2. Modelo entidad-relación

Representa a la realidad a través de ENTIDADES, que son objetos que existen y que se distinguen de otros por sus características particulares como lo es el *nombre* o *el número de control* asignado al entrar a una institución educativa. Las entidades pueden ser de dos tipos:

Tangibles:

Son aquellos objetos físicos que podemos ver, sentir o tocar.

Intangibles:

Todos aquellos eventos u objetos conceptuales que no podemos ver, aun sabiendo que existen, por ejemplo: la entidad MATERIA, sabemos que existe, sin embargo, no la podemos visualizar o tocar.

Las características de las entidades se llaman ATRIBUTOS, por ejemplo *el nombre, dirección, teléfono, grado y grupo* son atributos de la entidad empleado.

A su vez una entidad se puede asociar o relacionar con más entidades a través de relaciones.

3. ADMINISTRACION DE BASES DE DATOS

3.1. Factores en la determinación de los requerimientos de espacio.

Las consideraciones a seguir para determinar el espacio requerido para una base de datos son:

1. Tamaño de la base de datos (cantidad de datos en las tablas)

Esto puede ser calculado determinando el total del número de renglones, el tamaño del renglón, el número de regalones que caben en una página (2K) y el número de páginas requeridas por cada tabla en la base de datos.

La fórmula para determinar el número de páginas de una tabla consiste en estimar el número de renglones en la tabla dividido por el número de renglones por página. Se puede determinar el número de renglones por página dividiendo 2016 (el tamaño de 2K menos 32 bytes de encabezado) por la longitud del renglón.

Para determinar la longitud del renglón se deben ver los tipos de datos en la definición de la tabla.

2. Tamaño del log de transacciones.

El tamaño del log de transacciones varia de acuerdo al número de actualizaciones y la frecuencia de respaldos del log. Como referencia inicial se deben reservar de un 10 a un 25% del tamaño reservado para la base de datos.

3. Número y tamaño de índices.

4. Proyección de crecimiento de la base de datos.

3.2. Acceso a la base de datos.

El esquema general de acceso a un RDBMS se basa en cuatro elementos:

Login ID: Nombre para cada usuario el cual es conocido a nivel del servidor.

Nombre de usuario (username): Nombre conocido en la base de datos y asignado a un login con el propósito de tener acceso a la misma.

Alias: Un nombre de usuario que en la base de datos es compartido por varios login ID's. Esto sirve para tratar a más de una persona como un mismo usuario en la base de datos dándole a todos ellos los mismos permisos.

Grupo: Colección de nombres de usuarios de una base de datos. Permite simplificar la administración para quienes comparten derechos de acceso en común (pe. un mismo departamento, grupo de trabajo, jerarquía)

3.3. Administración del acceso a la base de datos.

3.3.1. Fase de planeación:

1. Determinar las tareas que los usuarios van a realizar en la base de datos. Esto puede ser decidiendo quienes actualizarán, capturarán o consultarán los datos.

2. Agrupar de manera lógica a lo usuarios con tareas comunes. Estos grupos se basan en lo que harán los usuarios con la base.

3.3.2. Fase de implementación:

1. Crear grupos por cada base de datos usando nombres acordes a la organización de la empresa.

2. Crear un login ID por cada usuario.

3. Asignar una base de datos por default a cada login.

4. Para cada login ID crear un nombre de usuario en cada base a la que requiera acceso.

5. Asignar alias a los login ID si fuera necesario.

6. Asignar cada nombre de usuario a uno de los grupo determinados.

Cual es la diferencia entre un login ID y un nombre de usuario?

Cual es el propósito de usar un alias?

Si se crea un nuevo objeto en la base de datos a quien se asocia, al login ID o al nombre de usuario?

3.4. Asignación de permisos a usuarios.

SELECT Seleccionar datos de una tabla, vista o columna.

INSERT Insertar nuevos datos aq una tabla o vista.

UPDATE Actualizar datos existentes en una tabla, vista o columna.

DELETE Borrar datos de una tabla o vista.

EJECUTAR Ejecutar stored procedures.

DRI Referencia a una tabla sin tener derecho de SELECT en ella.

3.5. Importando y exportando datos.

Opción de la base de datos Select into/bulkcopy en true.

bcp basededatos.propietario.tabla {in | out} archivodedatos -c -t[terminadordecampo] -r[terminadorderenglon] -U[login] -S[servidor] -P[password]

4. BASES DE DATOS ORIENTADAS A OBJETOS

4.1. Introducción

Lectura del Prefacio (Hughes, 1991)

4.2. Sistemas de administración de bases de datos (DBMS6)

(Bertino y Martino, 1995)

Los DBMS surgieron para responder a las necesidades de información de las organizaciones. Se tratan de un conjunto de datos persistentes y de programas para acceder a ellos y actualizarlos.

La tecnología de bases de datos tiene más de treinta años. Primero surgieron sistemas administradores de archivos de tipo ISAM[7] que trabajaban con archivos separados. Luego vinieron sistemas que centralizaban los archivos en una colección llamada base de datos. Los primeros de éstos utilizaron el modelo jerárquico (sistemas IMS y 2000). A continuación surgieron los desarrollados por la CODASYL[8] como IDS, TOTAL, ADABAS e IDMC. La siguiente generación fue la de bases de datos relacionales. Estas utilizan lenguajes más accesibles y poderosos en la manipulación de datos como el SQL, QUEL y QBE.

[6] Por las siglas en inglés de Data Base Management System.
[7] Indexed Secuencial Access Mode.
[8] COnference of DAta SYstem Languages.

Los DBMS deben contar con un *modelo de datos*, es decir, estructuras lógicas para describir los datos, y operaciones para manipularlos (recuperación y actualización). Las operaciones sobre los datos se hacen por medio de tres lenguajes: un DDL para definir el esquema y la integridad, un DML para la actualización de los datos y un DCL para el manejo de las autorizaciones en la base de datos. Adicionalmente un DBMS incluye mecanismos de seguridad, acceso a los datos, recuperación, control de concurrencia y optimización de consultas.

4.3. Retos actuales de los DBMS

(Bertino y Martino, 1995)

Aunque los DBMS relacionales (RDBMS) son actualmente líderes del mercado y brindan las soluciones necesarias a las empresas comerciales, existen aplicaciones que necesitan funciones con las que no cuentan. Ejemplos de ellas son las CAD/CAM, CASE, CIM. Adicionalmente los sistemas multimedia como los geográficos y de medio ambiente, sistemas de gestión de imágenes y documentos, y los sistemas de apoyo a las decisiones necesitan de modelos de datos complejos difíciles de representar como tuplas de una tabla.

Estas aplicaciones necesitan manipular objetos y los modelos de datos deben permitirles expresar su comportamiento y las relaciones entre ellos. Los DBMS deben tomar en cuenta las siguientes operaciones:

Ser capaces de definir sus propios tipos de datos.

Manejar versiones de objetos y estados de evolución.

El tamaño de los datos puede ser muy grande.

La duración de las transacciones puede ser muy larga.

Recuperar rápidamente objetos complejos.

Ofrecer comunicación efectiva a los clientes del sistema, principalmente en desarrollos grupales.

Permitir cambios en el esquema de la base.

Manejar objetos completos y sus componentes.

Lenguajes de consulta de objetos y lenguajes computacionalmente complejos.

Mecanismos de seguridad basados en la noción de objeto.

Funciones para definir reglas deductivas y de integridad.

Tener la capacidad para comunicarse con las aplicaciones ya existentes y manipular sus datos.

4.4. Tendencias actuales en la tecnología de bases de datos

(Bertino y Martino, 1995)

Con miras a superar los retos, las bases de datos están tomando varias tendencias. En general se está auxiliando de los lenguajes de programación orientados a objetos, los lenguajes lógicos y la inteligencia artificial. Podemos determinar cuatro tendencias actuales:

- Sistemas relacionales extendidos

  Incluyen manejo de objetos y triggers.

- Sistemas de bases de datos orientadas a objetos

  Integran el paradigma de la orientación a objetos a la tecnología de bases de datos.

- Sistemas de bases de datos deductivas

  Unen a las bases de datos la programación lógica. Cuentan con mecanismos de inferencia, basados en reglas, para generar información adicional a partir de los datos almacenados en la base.

- Sistemas de bases de datos inteligentes

  Incorporan técnicas desarrolladas en el campo de la inteligencia artificial.

4.5. Sistemas de administración de bases de datos orientadas a objetos (OODBMS)

(Bertino y Martino, 1995)

Los sistemas de bases de datos orientadas a objetos parecen ser la tecnología más prometedora para los próximos años, aunque estos carecen de un modelo de datos común y de fundamentos formales, además de que sus comportamientos en seguridad y manejo de transacciones no están a la altura de los productos actuales. Hay organismos en pro de la estandarización como el OMG[9], la CAD Framework Initiative y el grupo de trabajo de ANSI.

Algo que apoya esta tendencia es que a pesar de que la ingeniería de software orientada a objetos requiere mucho tiempo de análisis, la mayoría de los proyectos de desarrollo son más cortos y requieren menos personas, además de que la cantidad de código es menor.

A pesar de esto, sería difícil para las empresas dejar de un día para otro los sistemas actuales, debido principalmente a la falta de personal calificado, al efecto sobre la continuidad de sus operaciones y a la ausencia de garantías en la reutilización de los datos.

[9] Object Management Group.

Para ver una discusión acerca de si los OODBMS son en realidad DBMS ver Date (2001: 845-847).

4.5.1. Antecedentes de los OODBMS

(Bertino y Martino, 1995)

Primera

Data de 1986 cuando el sistema G-Base fue lanzado por la compañía francesa Grápale. En 1987 Servio Corp introduce GemStone y en 1988 Ontologic promueve su Vbase, seguido de Statics por la empresa Simbolics. Estos sistemas estaban basados en lenguajes propios y plataformas independientes del mercado. Estos sistemas fueron considerados lenguajes orientados a objetos con persistencia.

Segunda

Se da con la salida al mercado de Ontos en 1989. Siguieron los productos Object Design, Objectivity y Versant Object Technology. Todos utilizaron una arquitectura cliente/servidor y una plataforma en C++, X Windows y UNIX.

Tercera

La generación comienza con Itasca, lanzado en agosto de 1990 por Microelectronics and Computer Corporation. Le siguieron O_2 producido por la compañía francesa Altair, y después Zeitgeist por Texas Instruments. Estos ya son sistemas administradores de bases de datos con características avanzadas, un DDL y DML orientados a objetos.

4.6. Sistemas de bases de datos orientadas a objetos (OODBS)

(Atkinson et. al., 1995)

4.6.1. Introducción

No obstante en la actualidad hay mucha atención hacia los OODBS, tanto en el terreno de desarrollo como en el teórico, no hay una definición estándar de lo que estos sistemas significan.

Existen tres problemas principales que impiden una definición generalizada:

1. La falta de un modelo de datos común entre los diferentes sistemas

   Los sistemas de bases de datos relacionales cuentan con especificaciones claras dadas por Codd, pero los orientados a objetos no tienen algo así. Se pueden encontrar muchos textos que describen diferentes modelos, pero no hay uno como estándar.

2. La carencia de fundamentos formales

El fundamento teórico de la programación orientada a objetos es escaso en comparación con otras áreas como la programación lógica. Además se carece de definiciones de diversos conceptos.

3. Una actividad experimental muy fuerte

Existe mucho trabajo experimental, la mayoría de los desarrollos son sistemas prototipo o comerciales, no hay trabajo de conceptualización y definición de estándares. El diseño de estos sistemas está orientado por las aplicaciones que los requieren y no por un modelo común.

El problema es similar al de las bases de datos relacionales a mitad de los setenta. La gente se dedicaba a desarrollar implementaciones en lugar de definir las especificaciones para luego hacer la tecnología que permitiera implementarlas.

Se espera que de los prototipos y desarrollos actuales de los OODBS surja un modelo. Aunque también se corre el riesgo de que alguno de estos se convierta en el estándar por su demanda en el mercado.

4.6.2. Definición

Un OODBS debe satisfacer dos criterios:

1. Debe ser un DBMS y

2. Debe ser un sistema orientado a objetos (consistente con los lenguajes de programación orientada a objetos).

El primer criterio puede llevarse a cinco características: persistencia, administración de almacenamiento secundario, concurrencia, recuperación y facilidad de consultas personalizadas. El segundo criterio corresponde a ocho características: objetos complejos, identidad de objetos, encapsulación, tipos y clases, herencia, overriding combinado late binding, extensibilidad y completa capacidad computacional (computational completeness).

4.6.3. Objetos complejos

Los objetos complejos son creados con objetos simples aplicándoles constructores. Los objetos simples son: enteros, caracteres, cadenas de bytes, booleans y de punto flotante. Los constructores de objetos complejos pueden ser por ejemplo: tuplas, sets, bags, listas y arreglos. El mínimo conjunto de constructores que debería tener el sistema son set, lista y tupla.

Sets

Son la manera natural de representar colecciones del mundo real.

Tuplas

Son la manera natural de representar propiedades de una entidad. Además, son importantes por la aceptación ganada con el modelo relacional.

Listas o arreglos

Son importantes porque capturan el orden, cosa que ocurre en el mundo real. Además ayudan a representar matrices y series de datos en el tiempo.

Los constructores de objeto deben ser ortogonales[10], es decir, cualquier constructor debería aplicar a cualquier objeto. Los constructores relacionales no son ortogonales porque el set sólo aplica a la tupla y la tupla sólo aplica a valores atómicos.

Para soportar objetos complejos se requieren operadores apropiados que propaguen transitivamente las operaciones a todos sus componentes. Por ejemplo, operaciones de recuperación o borrado de todo un objeto complejo, o una copia «a profundidad». También debe ser permitido al usuario del sistema crear sus propias operaciones. Esta capacidad del sistema requiere dos tipos de referencias: “es parte de” y “en general”.

4.6.4. Identidad de objetos (revisado otros textos)

La identidad de objetos ha existido desde hace mucho tiempo en los lenguajes de programación, pero en las bases de datos es más reciente. El objetivo es contar con objetos que tengan una existencia independiente de sus valores. Así, dos objetos pueden ser idénticos si son el mismo objeto o pueden ser iguales si tienen los mismos valores. Esto tiene dos implicaciones: compartir un objeto (object sharing) y realizar actualizaciones sobre un objeto (object updates).

Compartir un objeto

En un modelo basado en identidad, dos objetos pueden compartir un componente. Así, la representación gráfica de un objeto complejo es un grafo; en un sistema sin identidad de objetos la representación sería un árbol. Considere el ejemplo de que una persona tiene un nombre, edad y un conjunto de hijos. Por ejemplo, Peter y Susan tienen un hijo de 15 años llamado John. En la vida real hay dos posibles situaciones: Susan y Peter son padres de John (identidad) o hay dos niños del mismo nombre (igualdad). En un sistema sin identidad de objetos, Peter y Susan serían representados así:

(peter, 40, {(john, 15, {})})

(susan, 41, {(john, 15, {})})

No hay manera de expresar si Peter y Susan son padres del mismo niño (John). En un modelo basado en identidad de objetos, estas estructuras pueden compartir la parte común (john, 15, {}) o no, y así capturar cada situación.

[10] Adj. Que está en ángulo recto.

Actualizaciones sobre un objeto

Asumiendo que Peter y Susan son en realidad padres de un mismo niño John, todas las actualizaciones al hijo de Susan deben ser aplicadas al objeto John y por consiguiente al hijo de Peter. En un sistema basado en valores (sin identidad) los sub-objetos deben ser actualizados separadamente. La identidad de objetos es un principio poderoso de manipulación de datos que puede ser la base del manejo de sets, tuplas y de la manipulación recursividad de objetos complejos.

Soportar identidad de objetos implica ofrecer operaciones tales como asignación de objetos, copiado de objetos (a profundidad y de forma superficial) y comprobaciones para la identidad de objetos o su igualdad.

Por supuesto uno puede simular la identidad en un sistema basado en valores (p. e. el relacional) introduciendo identificadores de objetos. Pero este método pondría en el usuario la responsabilidad de asegurarse de la unicidad del identificador de objeto y mantener integridad referencial. Por el contrario, en Bertino y Martino (1995:15-16) encontramos que la manera de implementar la identidad de objetos es mediante un OID (Object Identifier) independiente de los valores de los atributos del objeto. También se mencionan algunas ventajas del uso de los OID en comparación con las llaves primarias. Entre ellas están: los OID son implementados por el sistema y son implementados a bajo nivel, lo que mejora el rendimiento.

Para Date (1991) los OID son innecesarios e indeseables a nivel del modelo, “debido a que en esencia sólo son apuntadores” (:847). Los OID están ocultos al usuario mientras que las claves (llaves) no lo están. El uso de estos object identifiers no elimina el uso de claves ya que son necesarias para entender la realidad que envuelve al sistema (:825).

4.6.5 Encapsulación

La encapsulación es necesaria por la necesidad de distinguir claramente la especificación y la implementación de una operación, y por la necesidad de modularidad. Esta última es necesaria para estructurar aplicaciones complejas diseñadas e implementadas por un equipo de programadores. Es también necesaria como herramienta de protección y autorización.

Hay dos visiones de la encapsulación: la del lenguaje de programación y la adaptación a la base de datos de esa primera visión.

La idea de encapsulación en lenguajes de programación viene de la abstracción de los tipos de datos. En esta visión, un objeto tiene una parte de interfaz y una parte de implementación. La primera es la especificación del conjunto de operaciones que pueden ser realizadas sobre el objeto. Ésta es su única parte visible. La parte de la implementación tiene una parte de datos y una parte de procedimientos. La de datos es la representación o estado del objeto y la procedural describe la implementación de cada operación.

Traducido a bases de datos: un objeto encapsula programas y datos. En el mundo de las bases de datos, no es claro si la parte estructural del tipo es o no parte de la interfaz, en los

lenguajes de programación la estructura de datos es claramente parte de la implementación y no de la interfaz.

Por ejemplo, en un sistema relacional, un empleado es representado por una tupla. Éste es rescatado usando un lenguaje relacional y después una aplicación permite actualizar el registro de manera que se le suba el sueldo o se la despida. Esta aplicación será escrita en un lenguaje de programación imperativo o de cuarta generación incluyendo instrucciones DML, para luego ser almacenada en archivos externos a la base de datos. Hay, por tanto, una marcada separación entre programas y datos, y entre el lenguaje de consultas y el lenguaje de programación de aplicaciones.

En un sistema orientado a objetos, definimos al empleado como un objeto que tiene una parte de datos (probablemente muy similar al registro que definiríamos en el sistema relacional) y una parte de operaciones, la cual consiste en operaciones de «aumento» y «despido» que accedan a los datos del empleado. Cuando se almacena un conjunto de empleados, datos y operaciones son almacenados en la base.

De esta manera, hay un sólo modelo para datos y operaciones. Ninguna operación, fuera de las especificadas en la interfaz, puede ser ejecutada. Esta restricción es para operaciones de consulta y actualización.

La encapsulación brinda una forma de “independencia lógica de los datos”, es decir, podemos cambiar la implementación de un tipo sin cambiar los programas que lo usan. Las aplicaciones estarían protegidas ante los cambios en la implementación en las capas bajas del sistema.

La adecuada encapsulación se obtiene cuando sólo las operaciones son visibles y los datos y la implementación de esas operaciones están escondidas en el objeto.

No obstante lo anterior, hay casos en que la encapsulación no es necesaria o puede ser violada bajo ciertas condiciones (p. e. en queries a la medida).

4.6.6. Tipos y clases

Hay dos categorías de sistemas orientados a objetos, aquellos que soportan la noción de clase y aquellos que soportan la noción de tipo. Entre los primeros están: Smalltalk, Gemstone, Vision, Orion, Flavors, G-Base, Lore y la mayoría de los sistemas derivados de Lips. En la segunda encontramos a C++, Simula, Trellis/Owl, Vbase y O_2.

> Un tipo resume los rasgos comunes de un conjunto de objetos con las mismas características. Éste corresponde con la noción de tipo de dato abstracto y tiene dos partes: la interfaz y la implementación. Sólo la interfaz es la parte visible al usuario del tipo, la implementación del objeto es vista sólo por el diseñador del tipo. La interfaz consiste de una lista de operaciones junto con el tipo de los parámetros de entrada y salida.

La implementación del tipo consiste en una parte de datos y una de operaciones. En la parte de datos se describe la estructura interna del dato del objeto. Dependiendo del poder del sistema, la estructura de esta parte de datos puede ser más o menos compleja. La parte de

operación consiste de los procedimientos que implementan las operaciones de la parte de la interfaz.

En los lenguajes de programación, los tipos son herramientas para incrementar la productividad de los programadores, asegurando la exactitud de los programas. Forzando al usuario a declarar los tipos de las variables y expresiones que manipulará, el sistema razona acerca de la exactitud de los programas basado en esta información sobre los tipos. Si el tipo definido por el sistema es diseñado cuidadosamente, el sistema puede hacer la revisión del tipo en tiempo de compilación, de otra manera alguno de éstos tendrá que ser aplazado del tiempo de compilación. Así, estos tipos son principalmente utilizados en tiempo de compilación para revisar la exactitud del programa. En general, en sistemas basados en tipos, un tipo no es una primera clase y tiene un estatus especial y no puede ser modificado en tiempo de ejecución.

La noción de clase es diferente a la de tipo. Su especificación es la misma que la del tipo, pero es más una noción de tiempo de ejecución. Ésta contiene dos aspectos: una «fábrica» de objetos y un «almacén» de objetos. La fábrica puede ser usada para crear nuevos objetos mediante la operación *new* en la clase o a través de clonar algún objeto prototipo representativo de la clase. El almacén significa que todo lo atado a la clase es su extensión, por ejemplo, los objetos instanciados de la clase. El usuario puede manipular el almacén aplicando operaciones a todos los elementos de la clase. La clase no es utilizada para revisar la exactitud de un programa sino para manipular y crear objetos. En la mayoría de los sistemas que utilizan el mecanismo de la clase, las clases son primeras clases y pueden manipularse en tiempo de ejecución (actualizarse o pasarse como parámetros).

Es necesario que el sistema deba ofrecer alguna forma de estructurar datos, sean éstos clases o tipos. Así la noción clásica de esquema de base de datos será reemplazada por el conjunto de clases o el conjunto de tipos. Tampoco es necesario que el sistema mantenga automáticamente a toda la extensión de un tipo (p. e. el conjunto de objetos de un tipo dado en la base de datos). Considere el tipo *rectángulo*, que es usado por muchas bases de datos y muchos usuarios. No tiene sentido pedir que el sistema mantenga todo el conjunto de rectángulos o que realice operaciones sobre todos ellos. Es más realista que el usuario manipule su propio rectángulo. Por el contrario en el caso del tipo *empleado*, es preferible que el sistema mantenga automáticamente toda la extensión empleado.

4.6.7. Jerarquía de clases o tipos

La herencia tiene dos ventajas: es una herramienta poderosa de modelado ya que brinda una descripción precisa del mundo y ayudan a factorizar implementaciones y especificaciones compartidas en aplicaciones.

Asumamos que tenemos empleados y estudiantes. Cada empleado tiene un nombre, edad mayor de dieciocho años y un salario, puede morir, casarse y ser remunerado. Cada estudiante tiene edad, nombre y un conjunto de grados; puede morir, casarse y tiene su cálculo de GPA.

En un sistema relacional, el diseñador de bases de datos definiría una relación *empleado*, una *estudiante*, escribiría el código para las operaciones *morir*, *casarse* y

remunerar para la relación empleado y *morir*, *casarse* y *cálculo GPA* para la relación estudiante; en total escribe seis programas.

En un sistema orientado a objetos, usando adecuadamente la herencia, nosotros reconoceríamos que *empleado* y *estudiante* son personas y tienen algo en común y algo diferente. Introduciríamos un tipo *persona* con los atributos *nombre, edad* y las operaciones *morir* y *casarse.* Entonces se declara *empleado* como un tipo especial de *persona*, el cual especifica una operación especial de *remunerar.* De forma similar se declara que el *estudiante* es un tipo especial de *persona* con el atributo *conjunto-de-grados* y la operación especial *cálculo GPA*. Obtenemos por tanto una mejor estructura y una descripción más concisa del esquema (factorizamos especificaciones) y escribiremos sólo cuatro programas (factorizamos en la implementación). La herencia ayuda a reutilizar código ya que cada programa está al nivel en el cual un numero grande de objetos puede compartirlo.

Existen al menos cuatro tipos de herencia:

Substitución

Decimos que un tipo t hereda de un tipo t' si podemos realizar más operaciones sobre objetos de tipo t que sobre objetos de tipo t'. Así, en cualquier lugar donde podamos tener un objeto de tipo t' podemos sustituirlo por uno de tipo t. Este tipo de herencia está basada en el comportamiento y no en los valores.

Inclusión

Corresponde a la noción de clasificación. Establece que t es subtipo de t' si cada objeto de tipo t es también un objeto de tipo t'. Este tipo de herencia está basado en estructura y no en operaciones. Un ejemplo es un tipo *cuadrado* con métodos *get* y *set(size)*, y un tipo *cuadrado-lleno* con métodos *get, set(size)* y *llenar(color)*.

Restricción

Es un subcaso de herencia de inclusión. Un tipo t es un subtipo de un tipo t' si éste consiste de todos los objetos de tipo t los cuales satisfacen una restricción dada. Un ejemplo es que un adolescente es una subclase de persona. No tiene más atributos u operaciones que una persona, pero tiene restricciones específicas como la restricción de la edad entre trece y diecinueve.

Especialización

Un tipo t es subtipo de un tipo t' si objetos de tipo t son objetos de tipo t' tal que contienen más información específica. Ejemplos de esto son personas y empleados donde la información sobre empleados es la de las personas más campos adicionales.

Varios grados de estos cuatro tipos de herencia están disponibles en los sistemas existentes por lo que en general no se prescribe un tipo determinado de herencia.

4.6.8. Overriding, overloading y late binding

Hay casos en los que se desea tener el mismo nombre utilizado por diferentes operaciones. Por ejemplo, la operación desplegar: ésta toma un objeto como entrada y lo despliega en la pantalla. Dependiendo del tipo de objeto, nosotros queremos usar diferentes mecanismos de despliegue. Si el objeto es una pintura, queremos que aparezca en la pantalla. Si el objeto es una persona, queremos de alguna manera una tupla impresa. Finalmente, si el objeto es un grafo, queremos su representación gráfica. Considere ahora el problema de desplegar un set, el tipo al que pertenece es desconocido en tiempo de compilación.

En una aplicación con un sistema convencional, tendríamos tres operaciones: desplegar-persona, desplegar-bitmap, y desplegar-grafo. El programador cuestionará el tipo de objeto en el set y usará la operación correspondiente. Esto forza al programador, a tener todos los tipos posibles de objetos en el set, a asociar la operación de despliegue asociada y a usarla correctamente.

```
for x in X do
  begin
    case of type(x)
      person: display(x);
      bitmap: display-bitmap(x);
      graph: display-graph(x);
    end
  end
```

En un sistema orientado a objetos, definimos la operación desplegar al nivel de tipo de objeto (el tipo más general en el sistema). De esta manera, desplegar tiene un sólo nombre y puede ser usado indiferentemente en grafos, personas y pinturas. De cualquier modo, *redefinimos* la implementación de la operación por cada uno de los tipos de acuerdo al tipo a imprimir (la redefinición es llamada *overriding*). Este resultado en un sólo nombre (*desplegar*) denotando tres diferentes programas es llamado *overloading*. Para desplegar el conjunto de elementos, simplemente aplicamos las operaciones de despliegue para cada uno de ellos y dejar al sistema seleccionar la apropiada implementación en tiempo de ejecución.

```
for x in X do  display(x)
```

Aquí, ganamos una ventaja diferente: el tipo de implementaciones escribe siempre el mismo número de programas. Pero el programador de aplicaciones no tiene que preocuparse

sobre los tres diferentes programas. Adicionalmente, el código es más simple que la instrucción *case*. Finalmente, el código es más fácil de mantener cuando un nuevo tipo es introducido, el programa de despliegue seguirá trabajando sin modificaciones (entendiendo que redefinimos el método *desplegar* para el nuevo tipo)

Con el fin de dar esta nueva funcionalidad, el sistema no puede atar nombres de operaciones a programas en tiempo de compilación. Entonces, los nombres de las operaciones deben ser resueltos (traducidos en direcciones de programa) en tiempo de ejecución. Esta traducción tardía es llamada late binding. Note que, no obstante el late binding hace la revisión del tipo más difícil (y en algunos caso imposible), éste no la excluye completamente.

4.6.9. Completa capacidad computacional (Computational completeness)

Desde el punto de vista del lenguaje de programación, esta propiedad es obvia: simplemente significa que uno puede expresar cualquier función computable, usando el DML del sistema de base de datos. Desde el punto de vista de la base de datos, esto es una novedad ya que el SQL, por ejemplo, no está completo.

No se propone que los diseñadores de bases de datos orientadas a objetos diseñen nuevos lenguajes de programación: la completa capacidad computacional puede ser introducida mediante una conexión razonable a lenguajes de programación existentes. La mayoría de los sistemas verdaderamente usan un lenguaje de programación ya existente (Orion, Iris, Vbase, O_2)

Note que esto es diferente de ser “resource complete”, por ejemplo, ser capaz de acceder a todos los recursos del sistema (pantalla, comunicación remota) desde el lenguaje. Entonces, el sistema, no obstante ser de capacidad computacional completa, tal vez no sería capaz de expresar una aplicación en su totalidad. Esto es, en pero, más poderoso que un sistema de base de datos el cual sólo almacena y recupere datos y realice computación simple sobre valores atómicos.

4.7. Software

4.7.1. Comparación entre software

Una página que compara cuatro OODBS se encuentra en http://galaxy.uci.agh.edu.pl/~vahe/products.htm.

4.7.2. Algunos OODBMS

Objectivity/DB

(http://www.objectivity.com/)

Objectivity/DB es una base de datos multi-hilos (multi-threaded) que maneja datos complejos. Cuenta con manejo de transacciones y seriación para asegurar la consistencia de los datos. Permite programación de interfaces en Java, C++, Smalltalk y SQL. Cuenta con un

soporte completo para Java, incluyendo manejo de binding de acuerdo al estándar ODMG 2.0 y al “single process model”.

Incluye Objectivity/C++, que soporta el estándar ANSI de C++ y la versión persistente de Standard Template Library, que permite crear, almacenar y recuperar elementos de la base de datos. También tiene Objectivity/SQL++, que cuenta con una interfaz para el estándar SQL3.

Este sistema de base de datos cuenta con tolerancia a fallas y replicación de datos. Está disponible para casi todas las plataformas y brinda varios frameworks para desarrolladores.

O2

Objectstore

Gemstone

Versant

Orion

OTGen

PJama

ThorUp

Bibliografía

Atkinson, Malcom. Bancilhon, François. DeWitt, David. Dittrich, Klaus. Maier, David y Zdonik, Stanley. *The Object-Oriented Database System Manifesto.* En http://www-2.cs.cmu.edu/People/clamen/OODBMS/Manifesto/htManifesto/Manifesto.html,1989. Visitada el 4 de febrero de 2004.

Bertino, Elisa y Martino, Lorenzo. *Sistemas de bases de datos orientadas a objetos. Conceptos y arquitectura*. USA: Addison-Wesley, 1995. 278 pp. QA76.9D3 B4818

Date, C. J. *Sistemas de Bases de Datos*. 7ª México: Pearson, 2001

De Miguel, Adoración, et. al. *Diseño de bases de datos relacionales***.** España, Alfaomega.-Rama, 2001.

Elmasri, Ramez. *Fundamentos de sistemas de bases de datos*. México: Pearson Educación, Addison-Wesley, 2002.

Hughes, J. G. *Object-Oriented databases*. Inglaterra: Prentice Hall, 1991. 280 pp. QA76.9D3 H84. Preface

Johnson, James L. *Bases de datos. Modelos , lenguajes, diseño*. México: Oxford, 1997. 1028 pp. Inclusión lógica p. 298

Mendelzon, Ale. *Introducción a las bases de datos relacionales.* Argentina: Pearson, 2000.

Scheer, Thad y Smith, Theresa. *Accelerating Your Object-Oriented development*. (White Paper) http://www.objectivity.com/DevCentral/Products/TechDocs/Whitepapers/Accelerating/AcceleratingOODB.html. Visitada el 6 de marzo de 2004. [Trata del problema de implementar persistencia de objetos en bases de datos relacionales.]

Wade, Andrew E. *Hitting the Relational Wall* (White Paper) http://www.objectivity.com/DevCentral/Products/TechDocs/Whitepapers/WallFiles/WallPprIntro.html. Visitada el 6 de marzo de 2004. [Dura crítica a los RDBMS.]