Preparación Base de Datos CENEVAL

Juan Carlos Lavariega Jarquín
Centro de Investigación en Informática
CETEC 6to PISO Torre Norte.
lavariega@itesm.mx

CONTENIDO

- Base de Datos Distribuidas
  - Conceptos Generales
  - Fragmentación
  - Optimización de Consultas
  - Manejo de Transacciones Distribuidas
- Bases de Datos Orientadas a Objetos
  - Conceptos Generales
  - Arquitecturas
  - Estándar ODMG
    - ODL (Object Definition Language)
    - OQL (Object Query Language)

Bases de Datos Distribuidas

- Una base de datos distribuida es una colección de datos que pertenecen lógicamente a un mismo sistema pero que están (los datos) dispersos en los diferentes sitios de una red computacional.
- Cada sitio en la red es autónomo en sus capacidades de procesamiento y es capaz de realizar operaciones locales
- Cada sitio participa en la ejecución de al menos una aplicación global, la cual requiere acceder datos en diferentes sitios usando un subsistema de comunicación.

Bases de Datos Distribuidas

Centralizado	Distribuido
Control centralizado: un solo DBA	Control jerárquico: DBA global y DBA local
Independencia de Datos: Organización de los datos es transparente para el programador	Transparencia en la Distribución: Localización de los datos es un aspecto adicional de independencia de datos
Reducción de redundancia: una sola copia de datos que se comparta	Replicación de Datos: copias múltiples de datos que incrementa la localidad y la disponibilidad de datos
Estructuras físicas complejas para accesos eficientes	No hay estructuras inter-sitios. Uso de optimización global para reducir transferencia de datos
Seguridad	Problemas de seguridad intrínsecos

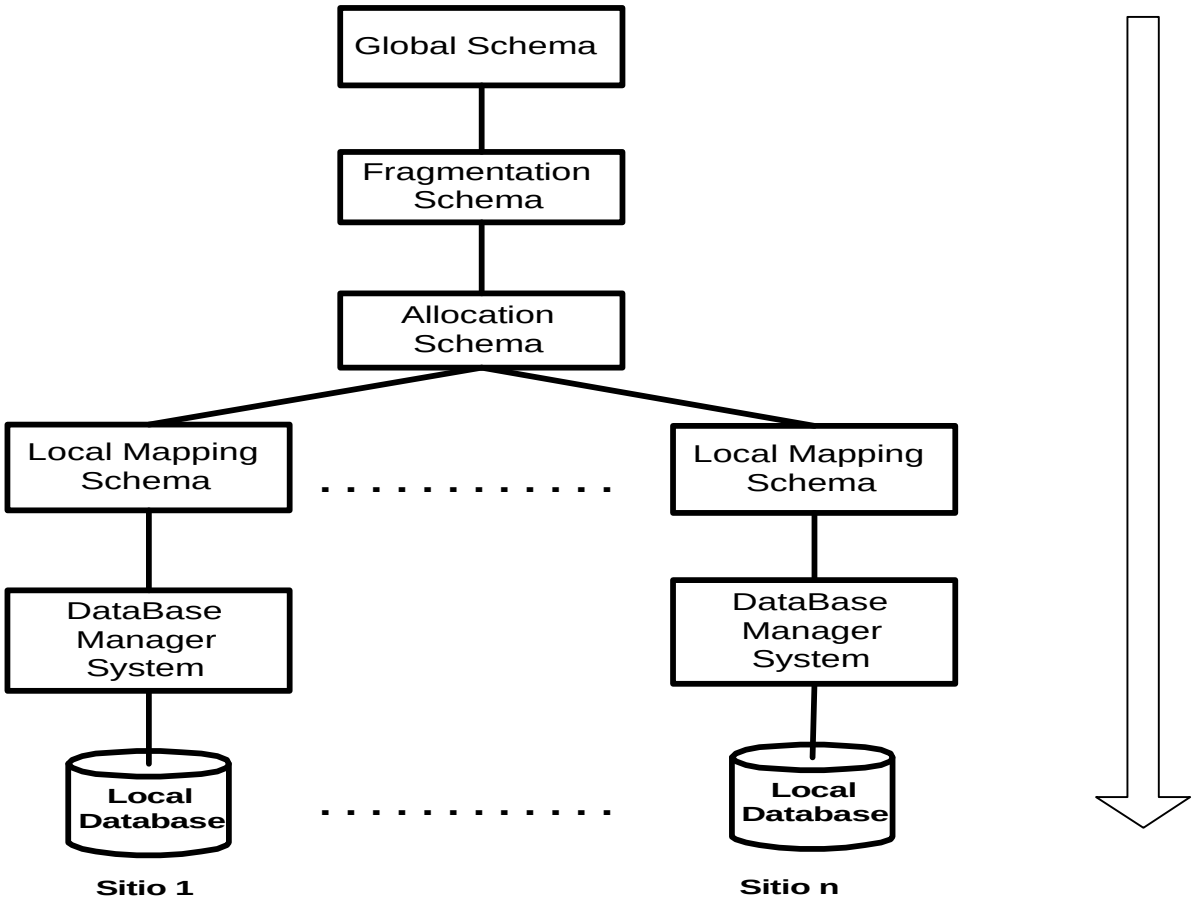
Bases de Datos Distribuidas

Motivación

- Conexión de bases de datos existentes.
- Crecimiento Incremental.
- Reducción de overhead en comunicación.
- Consideraciones de rendimiento (performance).
- Disponibilidad y Confiabilidad.
- Reducción de Costo

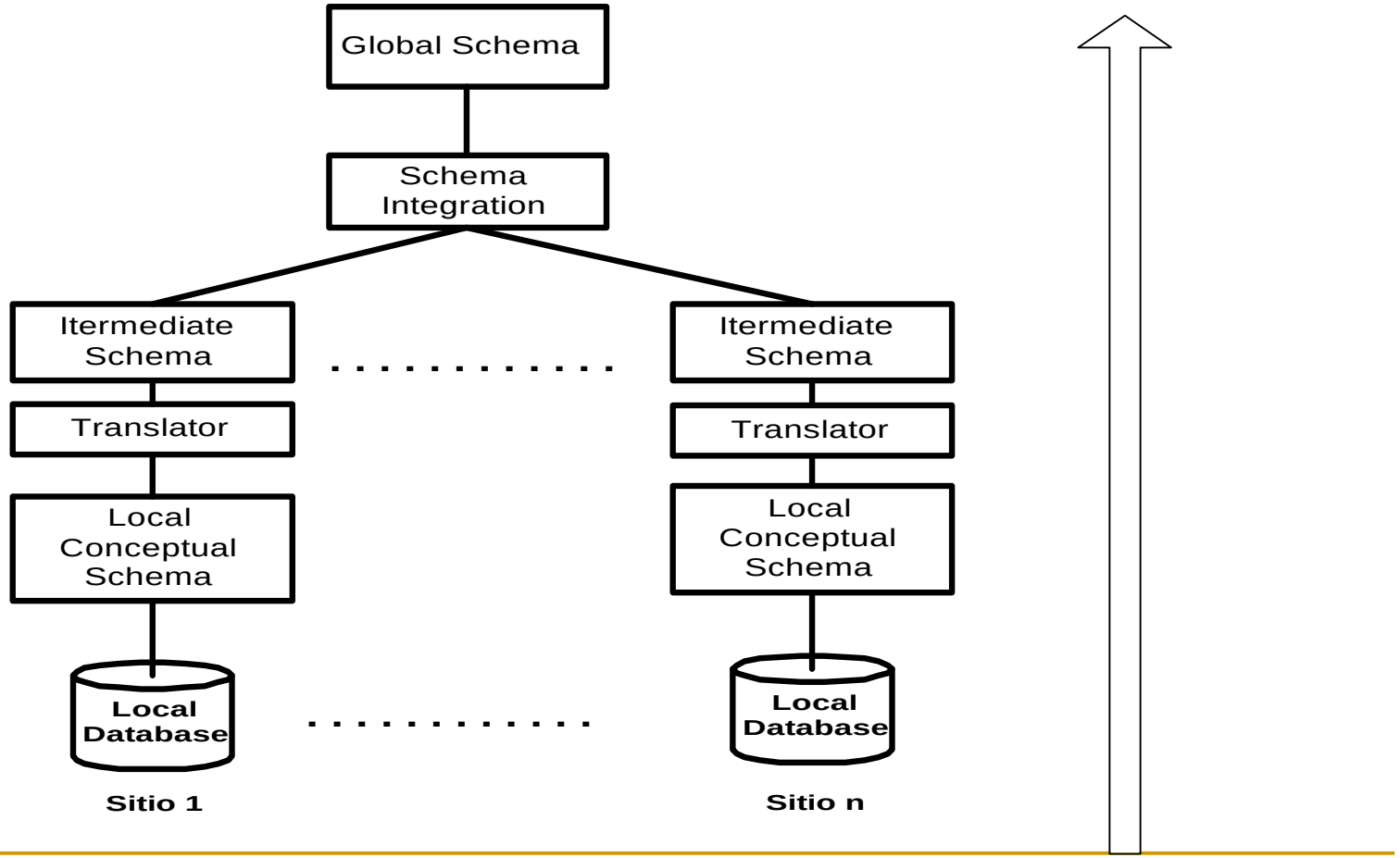
Bases de Datos Distribuidas-Arquitecturas

- Integración lógica por medio de diseño top-down (DistDB)



Bases de Datos Distribuidas-Arquitecturas

- Integración lógica por medio de bottom-up (Multidatabase)



Bases de Datos Distribuidas-Arquitecturas

- Esquema Global: Define todos los datos que están incluidos en la bd distribuida tal como si la BD no fuera distribuida. Consiste de una definición de relaciones globales.
- Esquema fragmentado: Traducción entre relaciones globales y fragmentos. (Una relación global puede consistir de varios *fragmentos* pero un fragmento está asociado con sólo una relación global)
- Esquema de Alojamiento: Define el sitio (o sitios) en el cual un fragmento está localizado.
- Esquema de Mapeo Local: Traduce los fragmentos locales a los objetos que son manejados por el SMBD local

Bases de Datos Distribuidas-Arquitecturas

- Separación entre fragmentación y localización.
  - Transparencia de Fragmentación. Usuario no debe saber que la información esta fragmentada
  - Transparencia de Localización. Usuario no debe saber donde se localizan los fragmentos
- Control explícito de redundancia.
- Independencia de BD locales.

Bases de Datos Distribuidas- Fragmentación

- Un *fragmento* es una porción lógica de relaciones globales. Los fragmentos están físicamente localizados en uno o más sitios de la red.

- Un fragmento está definido por una expresión del álgebra relacional que toma relaciones globales como operandos y produce un fragmento

- Condiciones para definir fragmentos:
  - Completitud
  - Reconstrucción
  - Disyunción

Bases de Datos Distribuidas-Criterios

- Completitud: Si una relación R esta descompuesta en fragmentos R_1, R_2, ...R_n, cada elemento que puede encontrarse en R también se encuentra en 1 o más R_i's,

  1<= i <= n (No hay pérdida de información)

- Reconstrucción: Si una relación R está descompuesta en fragmentos R_1, R_2, ...R_n, debe ser posible definir un operador relacional ∇ tal que

$$R = \nabla R_i \quad \forall R_i \in F_R$$

(Preservación de dependencias)

Bases de Datos Distribuidas-Criterios

- Disyunción: Si una relación R está horizontalmente descompuesta en fragmentos R_1, R_2, ...R_n, y un elemento d_i está incluido en un fragmento R_j, entonces d_i no existe en ningún otro fragmento R_k

  (Fragmentación horizontal no tiene mismo conjunto de tuplas)

- Si la relación R está verticalmente descompuesta esta regla excluye os atributos que forman la llave primaria

(i.e., atributos de la llave primaria aparecen en todos los fragmentos)

Bases de Datos Distribuidas Fragmentación Horizontal (HF)

- Parte tuplas de una relación global en subconjuntos
- Definidos por una operación de selección, llamada calificación, sobre una relación global

EJEMPLO

Considere la relación global equipos de béisbol

EQUIPO(NomEquipo, Liga, Localidad, Entrenador)

Esta relación global puede ser fragmentada horizontalmente basándose en el valor del atributo Liga:

$EQUIPO_A = \sigma_{liga=americana}\ EQUIPO$

$EQUIPO_N = \sigma_{liga=nacional}\ EQUIPO$

BD Distribuidas Fragmentación Horizontal Derivada (DHF)

- Fragmentación que se deriva de la fragmentación horizontal de otra relación

Ejemplo: Considere la relación global de jugadores de béisbol

JUGADOR(RFC, NombreJ, NombreE, Posición, Contrato, Salario)

Esta fragmentación global puede también ser fragmentada horizontalmente basada en la liga en la cual el jugador participa. La liga sin embargo no es un atributo de jugador.

$$\text{Jugador}_A = \text{JUGADOR SJ}_{\text{NombreE = NomEquipo}} \text{EQUIPO}_A$$

$$\text{Jugador}_N = \text{JUGADOR SJ}_{\text{NombreE = NomEquipo}} \text{EQUIPO}_N$$

Bases de Datos Distribuidas Fragmentación Vertical (VF)

- Fragmenta una relación global a través de la proyección de atributos.

Ejemplo: Considere la relación global de jugadores de béisbol

JUGADOR(RFC, NombreJ, NombreE, Posición, Contrato, Salario)

Esta relación pude ser fragmentada verticalmente de la siguiente forma

$Jugador_1 = \pi_{RFC, NombreJ, NombreE, Posición}\ JUGADOR$

$Jugador_2 = \pi_{RFC, Contrato, Salario}\ JUGADOR$

La operación de reconstrucción es:

$JUGADOR = Jugador_1$ **join** $Jugador_2$

Note que esta fragmentación no puede ser disjunta dado que la llave de la relación global debe aparecer en los fragmentos para efectos de reconstrucción.

Bases de Datos Distribuidas Fragmentación Mixta

- Generada a través de la aplicación recursiva de operadores del álgebra relacional en los fragmentos

Ejemplo: Considere la relación global de jugadores de béisbol

JUGADOR(RFC, NombreJ, NombreE, Posición, Contrato, Salario)

Después de la fragmentación vertical en

$Jugador_1 = \pi_{RFC, NombreJ, NombreE, Posición} JUGADOR$

$Jugador_2 = \pi_{RFC, Contrato, Salario} JUGADOR$

$Jugador_1$ puede tener una fragmentación horizontal derivada basada en la liga en la que juega el jugador

$Jugador_{1.A} = Jugador_1 \ SJ \ EQUIPO_A$ SJ= SemiJoin

$Jugador_{1.N} = Jugador_1 \ SJ \ EQUIPO_N$

Alojamiento de Fragmentos

Objetivo: Minimizar el numero de accesos remotos que son realizados por las aplicaciones distribuidas

NO REDUNDANTE: Un fragmento es alojado en exactamente un sitio. Técnica de “best fit” usando alguna métrica, alojar el fragmento en el sitio que tiene la mejor métrica

REDUNDANTE. Opción más compleja debido a la replicación de fragmentos y a la selección de los sitios para acceder los fragmentos.

“All benefical Sites”: Alojar el fragmento a cada sitio donde el beneficio de alojar una copia a ese sitio es más alto que su costo

“Additional Replication” : empezar con “best fit” y agregar una replicación hasta que no ya nos benéfico asignar fragmentos

Optimización de Consultas Distribuidas

- La optimización de consultas para bases de datos distribuidas consiste de técnicas de optimización centralizadas y optimización basada en fragmentos.

- Heurísticas de Optimización:
  - Realizar selecciones tan pronto como sea posible
  - Elegir un orden en las operaciones de join que reduce el tamaño de relaciones intermedias (aplicar asociatividad y conmutatividad)
  - Introducir proyecciones, si es necesario, y ejecutar las proyecciones tan pronto como sea posible
  - identificar sub-expresiones comunes.

Relaciones Calificadas

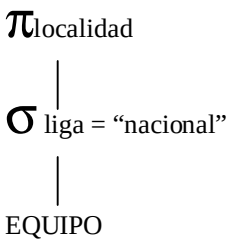
- Los fragmentos son relaciones “calificadas”. Una relación extendida por un calificador, denotado como [R:q_R], donde R denota el nombre del fragmento y q_R es un predicado que describe el contenido del fragmento.

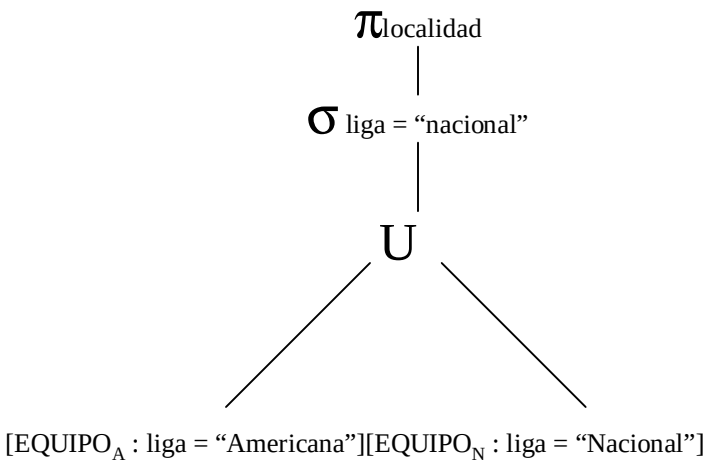
- Considere el ejemplo de la fragmentación horizontal de equipos de béisbol.
- La relación global EQUIPO cosiste de la unión de dos relaciones calificadas:
  - [EQUIPO $_A$: Liga=‘Americana’]
  - [EQUIPO $_N$: Liga=‘Nacional’]

Expresiones Canónicas

- Dada una expresión del álgebra relacional sobre un esquema global, su expresión canónica se obtiene sustituyendo en ella las expresiones que definen la reconstrucción de la relación global a partir de sus fragmentos.
  - Encontrar las locaciones de los equipos de la Liga Nacional:

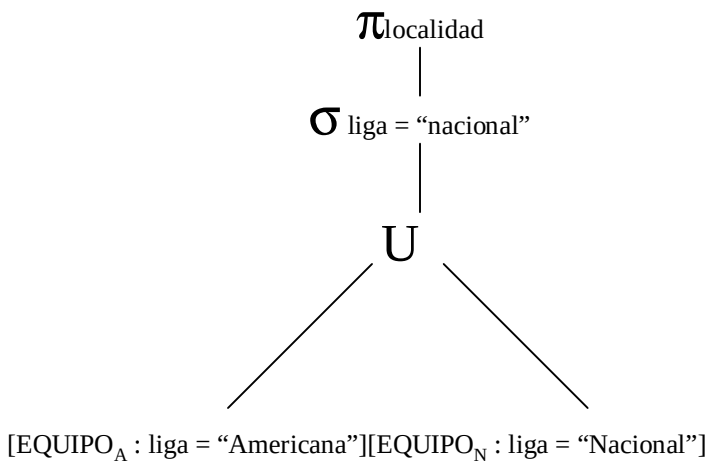
Árbol de Consulta





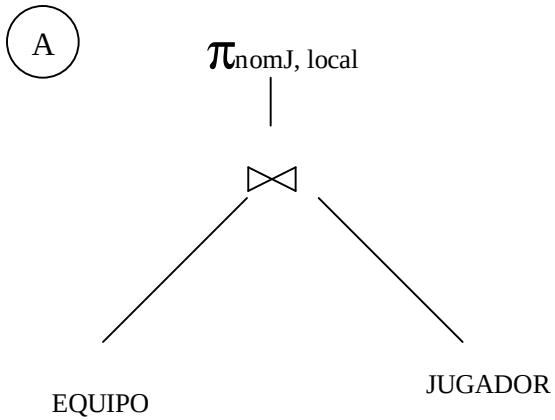
Optimización de Selección

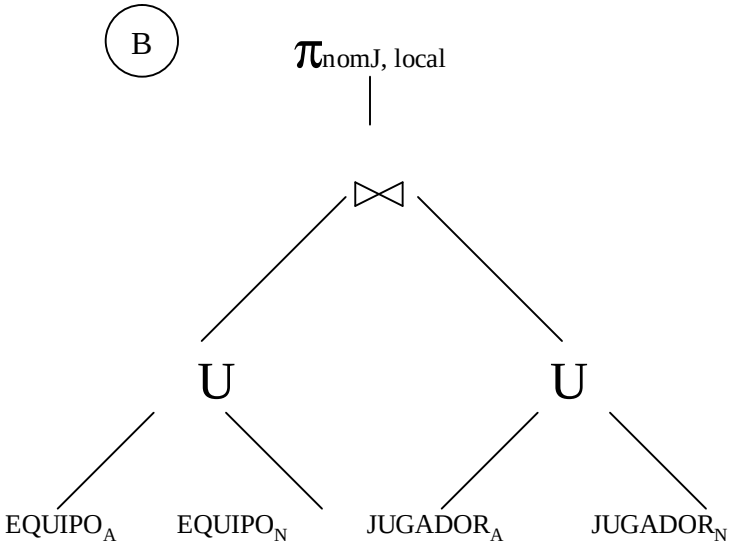
- Mover selecciones hacia abajo del árbol tan “profundo” como sea posible. Aplicar el álgebra de relaciones calificadas, substituyendo el resultado con las relaciones vacías si la calificación del resultado es contradictorio.



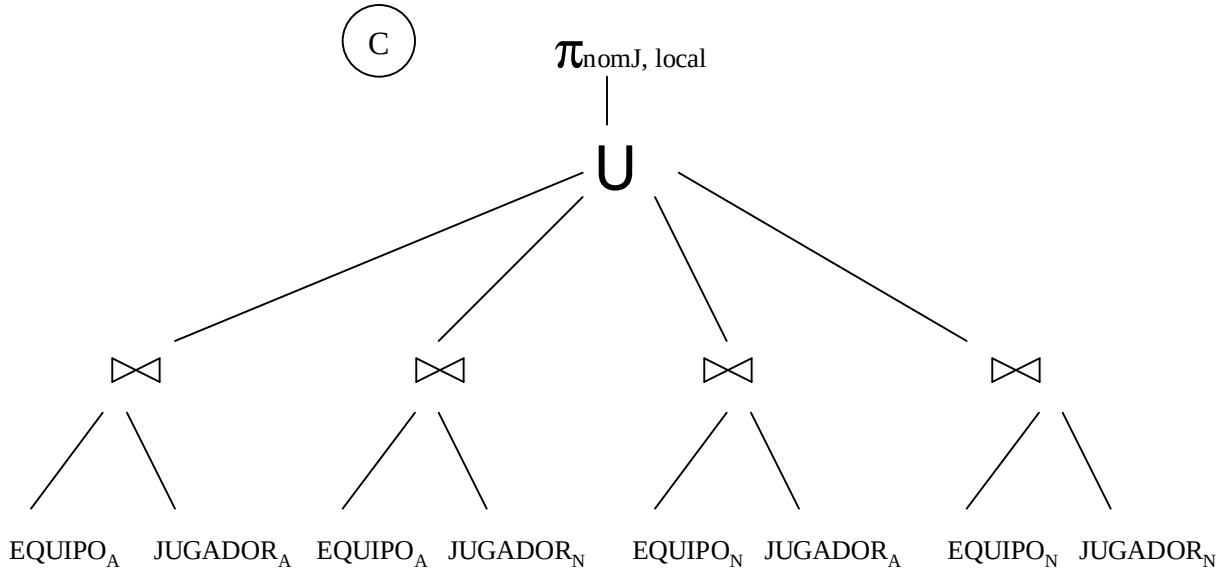
Optimización Distribuida de Join

- Ejecutar el join entre fragmentos y la unión de resultados. Conmutar uniones, representando colecciones de fragmentos, con las operaciones de join que serán distribuidas





Optimización Distribuida de Join

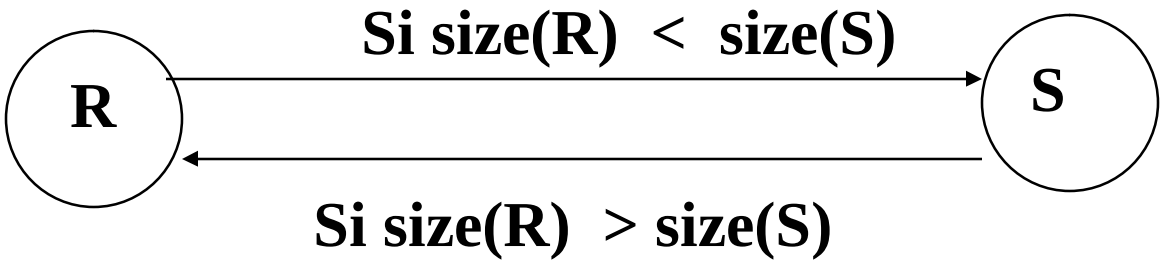


Estrategias para optimización de consultas.

- 1. Enviar todos los fragmentos, en paralelo, al sitio que requiere el resultado y ejecutar operaciones localmente.
- 2. Aplicar reductores de fragmentos (π y σ) y entonces enviar los fragmentos reducidos, en paralelo, al sitio requiriendo el resultado y ejecutar operaciones localmente
- 3. Utilizar álgebra de relaciones calificadas para optimizar la consulta
- 4 Usar semijoins para evaluar la consulta

Ordenamiento de Join

Considere dos relaciones únicamente:



El considerar múltiples relaciones incrementa el grado de dificultad al haber muchas alternativas

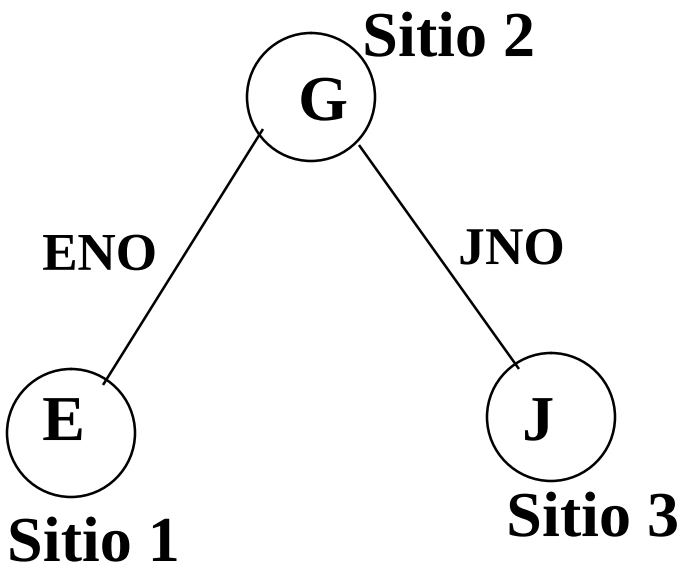
Se debe calcular el costo de todas las alternativas y seleccionar la mejor.
para esto es necesario calcular el tamaño de relaciones intermedias

Otra estrategia es usar heurísticas

Ordenamiento de Join- Ejemplo

Considere

$$J \bowtie_{JNO} G \bowtie_{ENO} E$$



Ordenamiento de Join- Ejemplo

Alternativas de Ejecución

1. E → Sitio2
   Sitio2 calcula E'=E join G
   E'→Sitio3
   Sitio3 calcula E' join J

2. G → Sitio1
   Sitio1 calcula E'=E join G
   E'→Sitio3
   Sitio3 calcula E' join J

3. G → Sitio3
   Sitio3 calcula G'=G join J
   G'→Sitio1
   Sitio1 calcula G' join E

4. J → Sitio2
   Sitio2 calcula J'=J join G
   J'→Sitio1
   Sitio1 calcula J' join E

5. E → Sitio2
   J → Sitio2
   Sitio2 calcula E join J join G

Join vía Semijoin $R \bowtie_{A=B} S \equiv (R \text{ SJ }_{A=B} S) \bowtie_{A=B} S$

- La operación de semijoin provee un mecanismo para la optimización de evaluación de joins en un ambiente distribuido
- Asuma que una relación **R** existe en un sitio *r* y una relación **S** existe en un sitio *s* y se requiere calcular $R \bowtie S$

  El procedimiento consistiría de

  - enviar $\pi_B(S)$ al sitio *r*
  - calcular semijoin en el sitio *r*
  - enviar el resultado de semijoin al sitio *s*
  - calcular join en el sitio *s*

Únicamente las tuplas de R que contribuyen al join son transmitidas

- Dado que Join es conmutativo:

$R \bowtie_{A=B} S \equiv (R \text{ SJ }_{A=B} S) \bowtie_{A=B} S \equiv S \bowtie_{A=B} (R \text{ SJ }_{A=B} S)$

$S \bowtie_{A=B} R \equiv (S \text{ SJ }_{A=B} R) \bowtie_{A=B} R \equiv R \bowtie_{A=B} (S \text{ SJ }_{A=B} R)$

Transacciones Distribuidas

Propiedades de Acidez de una Transacción

- Atomicidad (Atomicity): Todas o ninguna de las operaciones de la transacción son ejecutadas
- Consistencia (Consistency): Cualquier ejecución de la transacción toma la base de datos de un estado consistente a otro, incluyendo ejecuciones seriales o concurrentes
- Aislamiento (Isolation): Una transacción incompleta no puede revelar sus resultados a otras transacciones antes de finalizar (commitment)
- Durabilidad (Durability): Garantiza que los resultados de una transacción (committed transaction) no se pierda.

Para soportar atomicidad de una transacción distribuida, se asume que cada sitio tiene un manejador de transacciones locales (LTM) capaz de implementar transacciones locales

Recuperación de Errores en Transacciones Distribuidas

- Cada agente participante en una transacción distribuida envía primitivas a su LTM.
- (local_begin, local_commit, local_abort). Un agente que ha enviado un local_begin a su LTM es llamado una subtransacción.

- LTM debe tener capacidades de
  - Asegurar la atomicidad de una subtransacción
  - Escribir los registros de la bitácora (log records) en almacenamiento estable en beneficio de el Manejador de transacciones distribuidas (DTM)

- Condiciones de atomicidad de una transacción distribuida
  - En cada sitio, o se ejecutan todas las acciones o no se ejecuta ninguna
  - TODOS los sitios deben tomar la misma decisión con respecto a commit/abort de una transacción

- Manejador de Transacciones Distribuidas se debe asegurar que todos los manejadores locales toman la misma decisión.

Componentes de Protocolos Distribuidos de Confiabilidad

- Protocolos de Commit
  - ¿Cómo ejecutar comandos de commit para transacciones distribuidas?
  - ¿Cómo asegurar atomicidad y durabilidad?
- Protocolos de Recuperación
  - Cuando una falla ocurre, ¿cómo reaccionan los sitios en donde ocurre la falla?
  - Independiente: un sitio que ha fallado puede determinar el resultado de una transacción sin haber obtenido información remota.
- Protocolos de Terminación
  - Si ocurre una falla, ¿cómo reacciona los sitios que continúan siendo operacionales?
  - No-bloqueo: la ocurrencia de fallas no debe forzar al resto de los sitios a esperar hasta que la falla sea reparada para terminar la transacción.

Protocolo de Commit de Dos Fases

- Coordinador: Agente que es responsable de la decisión final
- Participantes: todos los agentes que deben tomar la misma acción

- 2PC: Determinar una decisión única (commit/abort) para todos los participantes.
  - Fase 1: Llegar a una decisión común
  - Fase 2: Implantar esa decisión

Protocolo de Commit de Dos Fases (Fase 1)

Coordinador (C)	Participante (P)
log PREPARE send PREPARE msg to P's activate timeout	

	wait for PREPARE msg
	if commit ok
	  write all log records
	  log READY
	  send READY msg to C
	else
	  log ABORT
	  send ABORT msg to C

wait for answer/timeout
if abort or timeout
  decision := abort
else
  decision := commit	

Protocolo de Commit de Dos Fases (Fase 2)

Coordinador (C)	Participante (P)
if decision = abort log GLOBAL_ABORT send ABORT to Ps else log GLOBAL_COMMIT send COMMIT to Ps	wait for decision msg log ABORT/COMMIT send ACK to C execute ABORT/COMMIT
wait for ACK from all Ps log COMPLETE	

Comentarios acerca del algoritmo 2PC

2PC es resistente a todas las fallas donde no haya habido pérdida de información en la bitácora.

Cada participante tiene capacidad unilateral de abortar. Cada sitio está autorizado a abortar su subtransacción hasta el momento en que hay respondido READY al mensaje de PREPARE.

Una subtransacción puede ser bloqueada (al fallar el coordinador o la red). Una subtransacción retiene todos sus recursos hasta que recibe la decisión final a ejecutar debido a que debe ser capaz de “commit/abort” eventualmente.

Base de Datos Distribuidas Bloqueo

- Una transacción está bien-formada si adquiere los “locks” apropiados sobre los datos antes de accesar los datos.
- Una transacción está en un esquema de lock de 2 fases (2-phase-locked) , si no requiere nuevos locks después de haber liberado algún lock
- Si las transacciones distribuidas están bien-formadas y siguen un esquema 2-phase-locked, entonces este es un mecanismo correcto que preserva la seriabilidad en bases de datos distribuidas.
- Para garantizar aislamiento, una transacción también debe retener todos sus locks exclusivos hasta el momento de “commit”

Consideraciones:

- Múltiples copias de datos
- Deadlock

Deadlock

- Deadlock- esperar por un evento que no ocurrirá
- Condiciones necesarias para que exista un deadlock
  - Exclusión mutua: control exclusivo de recursos
  - Espera: los recursos son retenidos mientras se espera por recursos adicionales
  - No preemption: recursos son mantenidos hasta que se usan en la terminación de la transacción.
  - Espera circular: cada transacción retiene uno o más recursos requeridos por la siguiente transacción en una cadena circular.

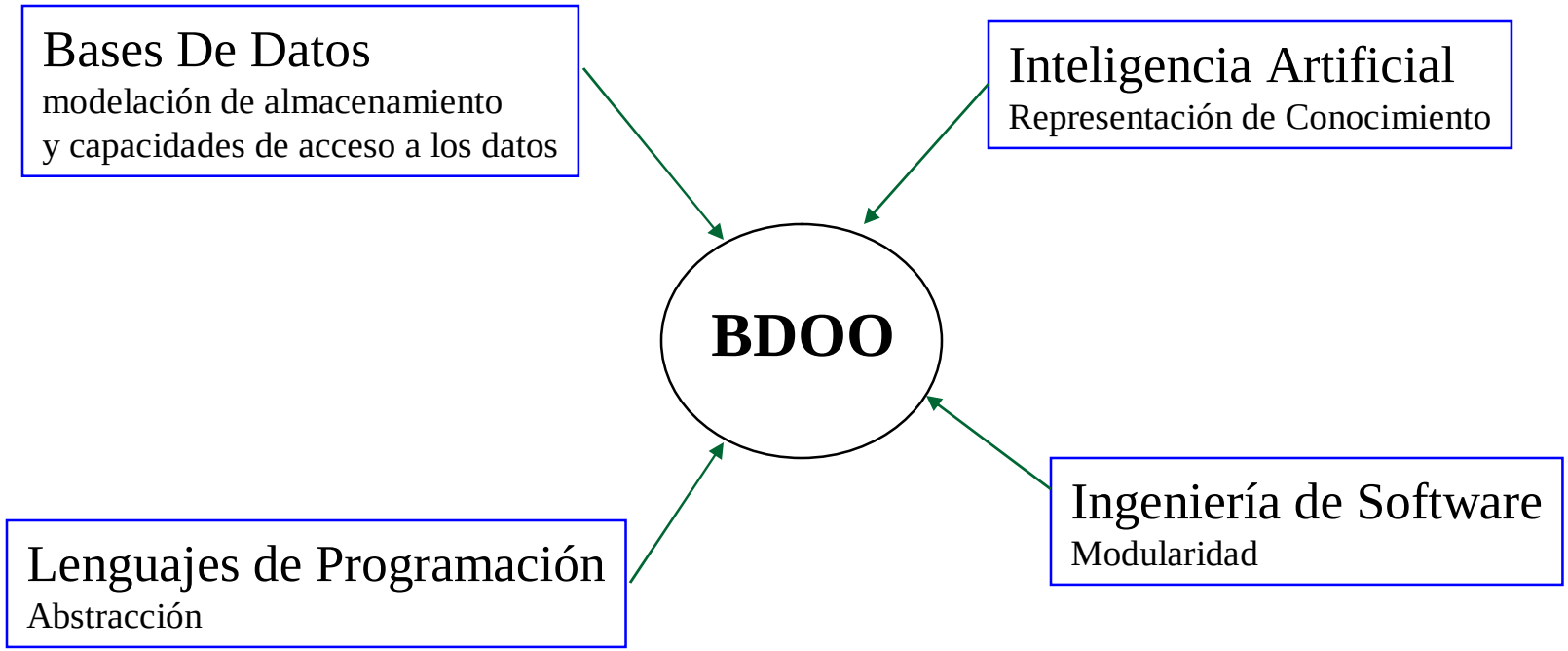
Estrategias para evitar deadlock

- Timeout: Abortar una transacción después de haber estado en un estado de espera por un intervalo específico de tiempo.
  - Pude ser que se aborten transacciones que no están en deadlock
  - Difícil de determinar el intervalo apropiado para timeout
  - Efecto cascada debido a sobrecarga del sistema.

- Prevención de Deadlock: Una transacción no es permitida a entrar en estado de espera si está en riesgo de deadlock
- Detección Centralizada de deadlock. Un sitio determinado actúa como detector de deadlock, recibiendo información de los otros sitios en el sistema distribuido
- Detección distribuida de deadlock: cada sitio tiene la misma responsabilidad de determinar un deadlock global al intercambiar información sobre transacciones en espera.

Bases de Datos Orientadas a Objetos

Bases de Datos Orientadas a Objetos



Bases de Datos Orientadas a Objetos

Aplicaciones

Las bases de datos Orientadas a Objetos están dirigidas hacia aplicaciones *no tradicionales* tales como:

- Diseño Asistido por Computadora (CAD)
- Ingeniería de Software Asistido por Computadora (CASE)
- Ambientes de Ingeniería de Software (SEE)
- Sistemas de Información de Oficina (OIS)
- Cartografía (GIS)

Aplicaciones de BDOO

Nuevas aplicaciones presentan nuevas demandas en bases de datos tradicionales

- Grandes cantidades de datos
- Relaciones entre datos más complejas
- Transacciones de larga duración.
- Evolución.
- Versiones
- Estructuras de datos complejas (arreglos, listas, árboles, etc.)
- Tipos de Datos extensibles

Arquitecturas de Bases de Datos OO

- Sistemas de Bases de Datos Extendidos. Proveen funcionalidad orientada a objetos al extender sistemas de bases de datos existentes
  - Extensiones al modelo relacional

Postgres
Starburst
RM/T
Algres
GEM

  - Extensiones a modelos semánticos/funcionales

IRIS
PROBE
ADPLEX
SIM

Arquitecturas de Bases de Datos OO

- Lenguajes de Programación para Bases de Datos: Extensión de lenguajes de programación existentes con funciones propias de sistemas de bases de datos. Persistencia y tipos de datos se convierten en aspectos ortogonales.
  - O2
  - Objectstore
  - Objectivity/DB
  - VERSANT
  - Gemstone
  - ITASCA

Arquitecturas de Bases de Datos OO

- Administradores de Objetos: Extensiones a sistemas de archivos existentes. Soportan un modelo físico únicamente (un repositorio para objetos persistentes)
  - ObServer
  - LOOM Smalltalk-80
  - Mneme
  - POMS
  - Thor

- Generadores de Bases de Datos: Proveen herramientas que permiten a un diseñador el moldear los datos y la implantación a necesidades específicas de la aplicación.
  - Exodus
  - Genesis

Comparación con Lenguajes de Programación

BDOO tratan de superar los diferentes enfoques que han existido entre bases de datos y lenguajes de programación.

BD	**LP**
DML (declarativo)	LP (imperativos)
Estructuras de BD	No existen estructuras de BD
No completos computacionalmente	Computacionalmente completos
Almacenamiento Persistente de datos	Archivos

Características de BDOO

1) Identidad de Objetos: Los datos son vistos como objetos con identificadores de objetos internos (OID´s). Los objetos son referenciados y relacionados por OID´s no por valores externos.

2) Objetos Complejos: Los objetos pueden referenciar otros objetos para construir objetos complejos a partir de constructores tales como conjuntos, listas, tuplas y arreglos.

3) Persistencia: Los datos deben ser accesibles aún después de que su proceso creador ha terminado. La capacidad de persistencia debe ser ortogonal a los tipos de datos.

4) Encapsulación: Agrega comportamiento a los objetos a través del concepto de tipos de datos abstractos

Características de BDOO

5) Clases y Tipos: Los tipos definen la intesion de un objeto (i.e., la estructura de un objeto). Las clases definen la extensión de un objeto (i.e., el conjunto de objetos que tienen la misma representación interna).

6) Clases y Tipos de Herencia: Clases y tipos pueden ser formados en relaciones de superclase/subclase (supertipo/subtipo). Los objetos en las subclases heredan los atributos y operaciones de las superclases

7) Overriding, Late Binding y Overloading: Estos conceptos añaden flexibilidad a BDOO's (i.e., varios programas con el mismo nombre, determinación al tiempo de ejecución de las direcciones del programa, y redefinición de operaciones heredadas)

Overriding, Late Binding, Overloading

- Overloading: Usar el mismo nombre para métodos diferentes
- Overriding: Redefinir la implementación de un método
- Late Binding: Asociar los nombres de las operaciones a los métodos al tiempo de ejecución dependiendo del tipo de los parámetros enviados

Características de BDOO

8) Completitud Computacional: Las BDOO's están integradas con lenguajes de programación de alto nivel para lograr que sean computacionalmente completas. SQL no es computacionalmente completo.

9) Extensibilidad: Los usuarios deben de ser capaces de definir sus propios tipos de objetos.

10) Administracion del Almacenamiento: Una BDOO debe soportar indexamiento, agrupación, selección con rutas de acceso, administración de consultas.

11) Concurrencia y Recuperación: Una BDOO debe soportar las capacidades de bases de datos tradicionales para la compartición y protección de los datos, tales como concurrencia de transacciones y recuperación de fallas.

Características de BDOO

características adicionales:

- Facilidades de consultas
- Herencia múltiple
- Restricciones de integridad
- Verificación de tipos
- Vistas
- Distribución
- Diseño de transacciones
- Versiones
- Paradigmas de programación (OO, funcional, etc.)

Identidad de Objetos

- Los datos son modelados en términos de los objetos lógicos que existen en la aplicación
- Cada objeto tiene su identificador único interno (OID)
- La existencia de un objeto es independiente de sus valores. Un OID es inmutable. Los valores son mutables

- Un objeto está definido como una tripleta:

  o=(id#, tipo, rep) donde:

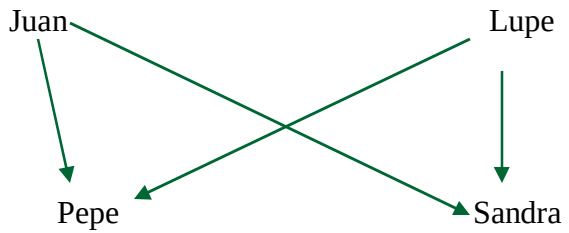
  id# es un OID generado por el sistema

  tipo es el tipo del objeto (puede incluir constructores como tupla, set, lista, array, bag)

  rep es el estado interno del objeto.

Ventajas de Identidad

- Los objetos se relacionan por OID´s y no por valores externos
- Los objetos pueden asumir diferentes roles y aun ser reconocidos como el mismo objeto
- Los objetos pueden ser compartidos



Identidad de Objetos vs Igualdad de Objetos

- Identidad de objetos compara objetos en términos de los identificadores de objetos
- Igualdad Superficial (shallow equality) compara los valores de los objetos a un nivel de profundidad
- Igualdad Profunda (deep equality) compara los valores de los objetos al más profundo nivel
- Dos objetos idénticos siempre son shallow-equal y deeep-equal
- Dos objetos que son shallow-equal siempre son deep-equal

Comparación de Objetos

- Supongamos un modelo de objetos que contiene
  - **átomos: objetos que se pueden imprimir**
  - **conjuntos: {O1, ...On}**
  - **Tuplas: [A1:O1,....An:On]**

- Decimos que
  - **los objetos simples son átomos**
  - **una instancia de una clase junto con los valores de sus atributos es una tupla**
  - **una propiedad multivaluada (o una clase) es un conjunto de objetos**

Identidad de Objetos

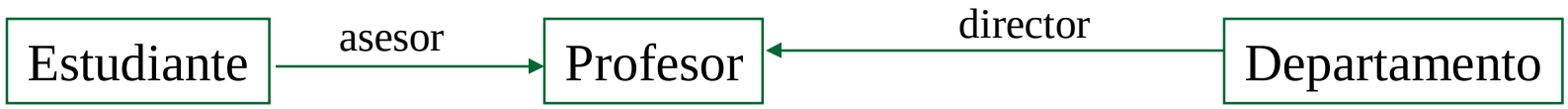
- Dos variables conteniendo identificadores de objetos son idénticos si contienen apuntadores a el mismo identificador.

- El predicado:

  identical (S.asesor, D.director),

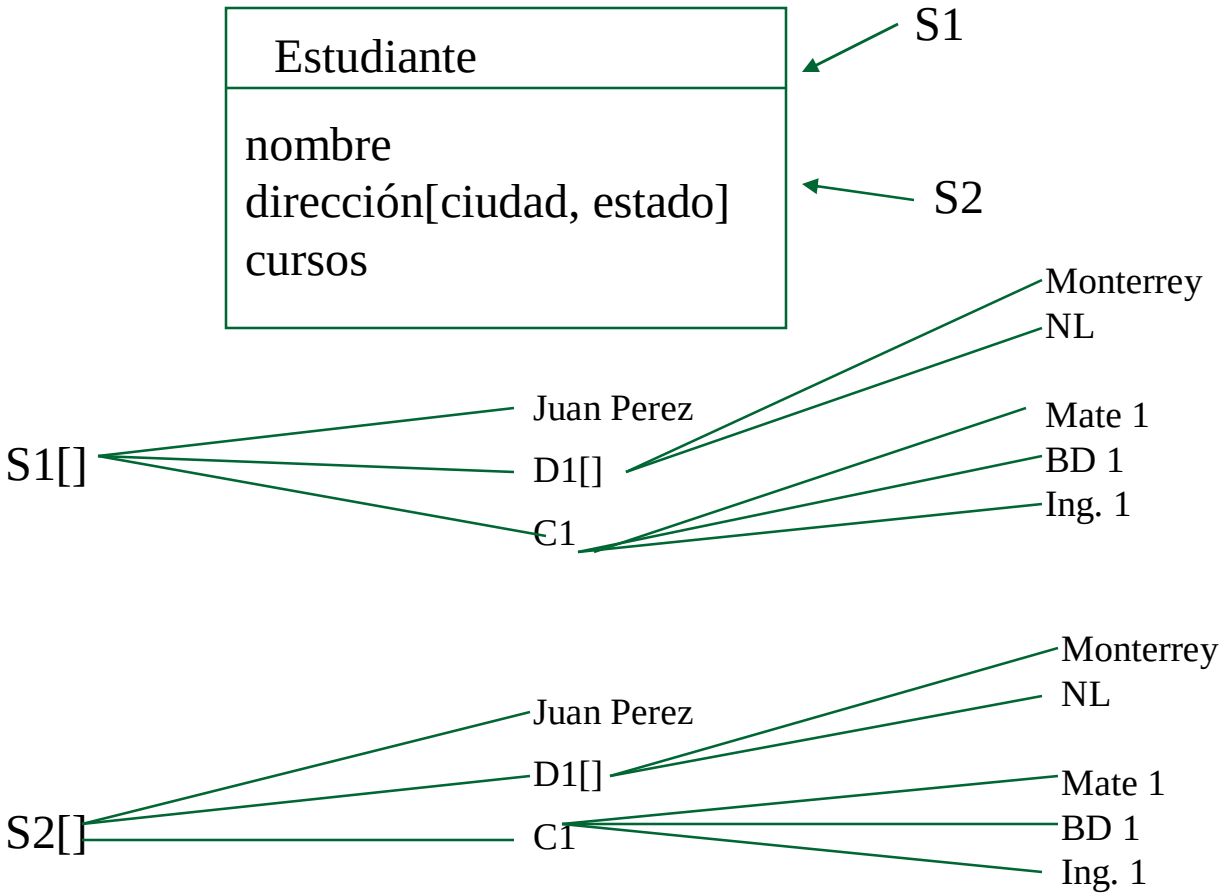
  evalúa a verdadero

  donde S representa un objeto estudiante, D representa un objeto departamento y el asesor de S es el mismo objeto que el director de D.



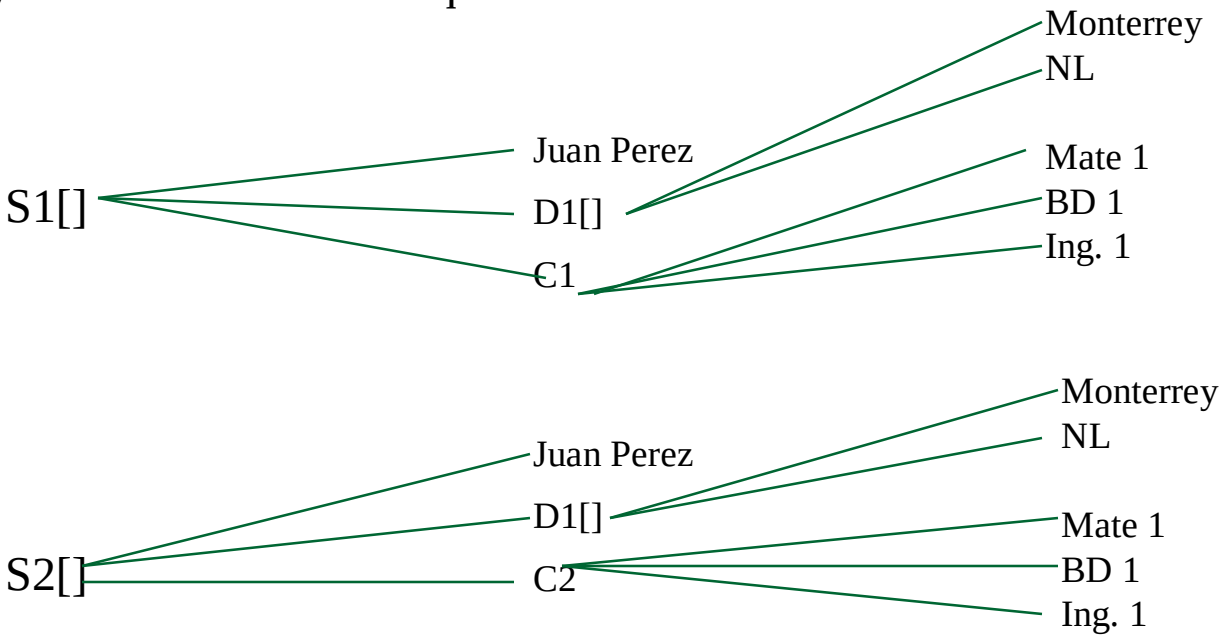
Igualdad Superficial (Shallow Equality)

S1 y S2 son shallow-equal si sus valores son idénticos



Igualdad Superficial (Shallow Equality)

S1 y S2 NO son shallow-equal.



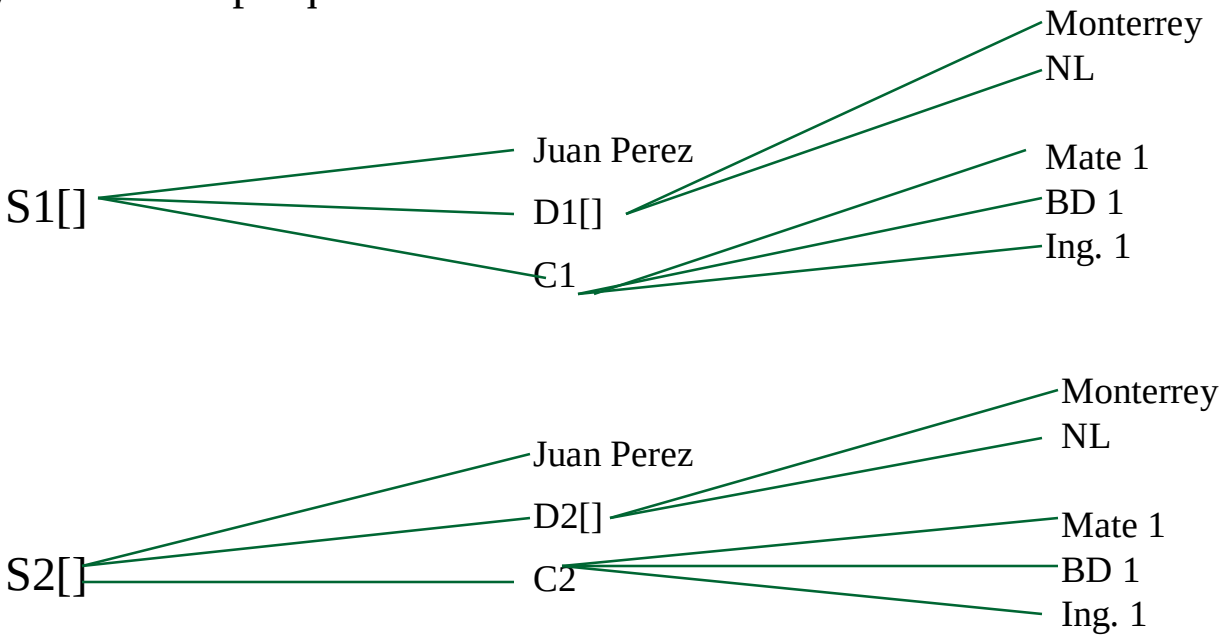
El valor del atributo cursos no es idéntico.

Igualdad Profunda (Deep Equality)

- Un objeto O1 es deep-equal a un objeto O2 si:
  - 1. Si O1 y O2 son atómicos, entonces O1 y O2 son deep-equal si sus valores son iguales
  - 2. Si O1 y O2 son conjuntos de objetos, entonces O1 y O2 son deep-equal si ambos tienen la misma cardinalidad Y los elementos de cada conjunto son uno-a-uno deep-equal
  - 3. Si O1 y 02 son tuplas de objetos, entonces O1 y O2 son deep-equal si sus valores para cada atributo son deep-equal.

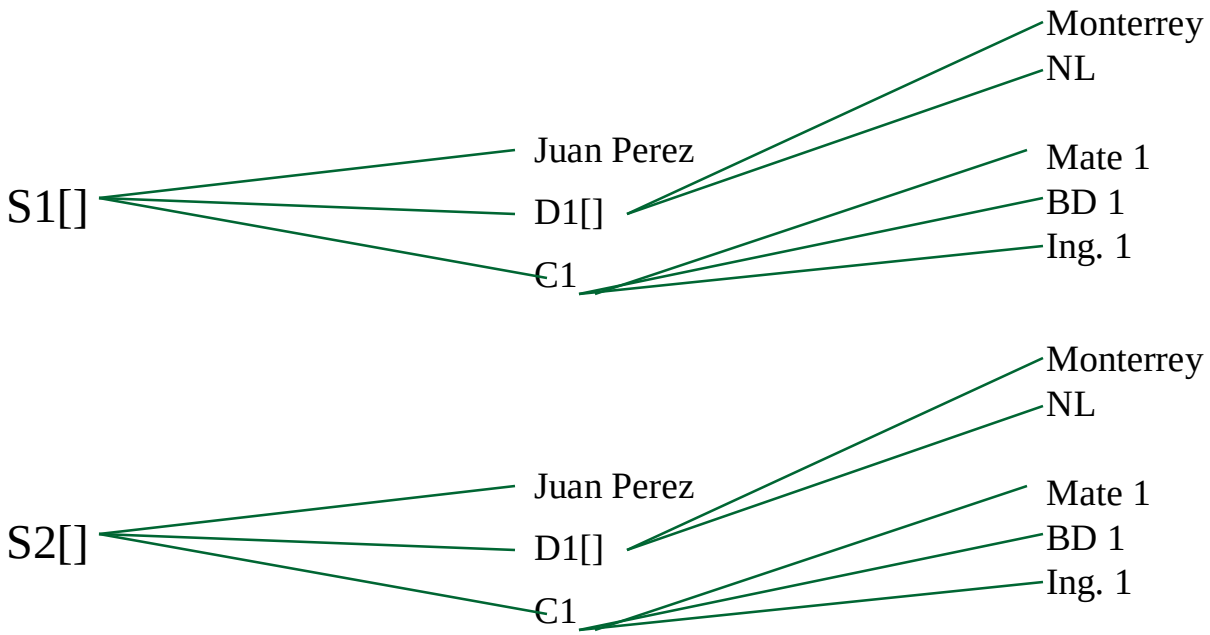
Igualdad Profunda (Deep Equality)

S1 y S2 son deep-equal.



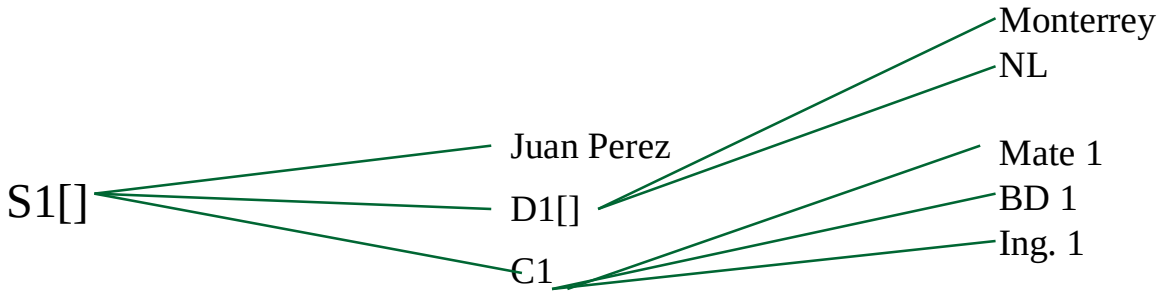
Copiado de Objetos

- Shallow-copy (O_1,O_2) genera un nuevo objeto O_2 tal que O_1 y O_2 no son idénticos pero O_1 y O_2 son shallow -equal
- Si X ← S1 entonces después de ejecutar shallow-copy(X,Y),
  Y ← S2 (S2 es un objeto nuevo)

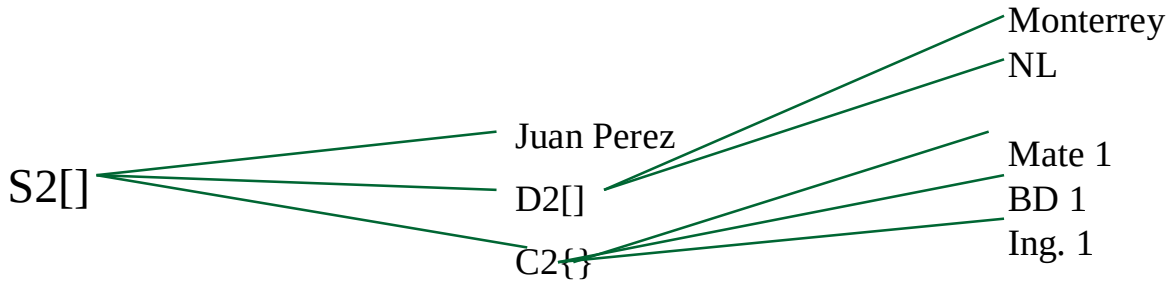


Copiado de Objetos

- Deep-copy (O_1,O_2) genera un nuevo objeto O_2. Para todos los atributos de o1 que son de tipo conjunto o tupla, O_1 y O_2 no son shallow-equal pero sí son deep-equal



- Si X ← S1, deep-copy(X,Y) crea un nuevo objeto S2 y lo asigna a Y



El Estándar ODMG

- Objetivo: Establecer estándares que permitan a los usuarios de BDOO escribir aplicaciones que sean portales y que puedan correr en más de un producto BDOO
- ODMG es soportado por los principales vendedores de BDOO y sus usuarios.
- Se busca que sean portables :
  - **El modelo de datos,**
  - **Los diferentes bindings a los lenguajes de programación**
  - **el lenguaje de manipulación de datos y**
  - **el lenguaje de consulta**
- Sin embargo puede haber diferencias en
  - **rendimiento**
  - **lenguajes soportados predefinidas**
  - **funcionalidad especifica (versiones)**
  - **herramientas de construcción**
  - **bibliotecas de funciones**

El Estándar ODMG

Los principales componentes del estándar incluyen (ODMG 3.0):

- Object Model: define los componentes básicos de un modelo de objetos
- Object Definition Language (ODL): define un lenguaje de definición de datos
- Object Query Language (OQL): define un lenguaje declarativo para la consulta y actualización de objetos. OQL es similar a una versión extendida de SQL
- C++ Binding: soporte de código C++ portable para la manipulación de objetos
- Smalltalk Binding: soporte de código Smalltalk portable para la manipulación de objetos
- Java Binding: soporte de código Java portable para la manipulación de objetos

El Estándar ODMG-ODL Características

- Los primitivos básicos de modelación de datos son objetos (*object*) y literales (*literal*). cada objeto se asume que tiene un identificador único. Literales no tienen identificadores.

- Los objetos se categorizan en tipos.
  - Un tipo (*type*) tiene una especificación y una o más implementaciones
  - Una interface es una especificación que define solo el comportamiento de un tipo de objetos
  - Una clase es una especificación que define el comportamiento y el estado de un tipo de objetos
  - Una implementación define estructuras de datos y/o métodos

El Estándar ODMG-ODL Características

- El comportamiento de los objetos está definido por un conjunto de operaciones
- El estado de los objetos está definido por propiedades
  - Las propiedades pueden ser atributos(*attribute*) o relaciones (*relationships*)
- Se definen dos formas de herencia: herencia de comportamiento y herencia de estado.
- Herencia de comportamiento: tipo/supertipo relación ISA
  - Herencia múltiple de comportamiento a través de *interface definition*
- Herencia de estado: relación EXTENDS.
  - EXTENDS es una relación de herencia sencilla entre dos clases donde la subclase hereda todas las propiedades y comportamiento de la superclase.

- *Extents* se refiere al conjunto de todas las instancias de un tipo en una base de datos en particular.

El Estándar ODMG-ODL Características

- La etiqueta *Key* se utiliza de manera similar al concepto de llaves en el modelo relacional. Pude ser simples o compuestas

- Métodos. Son operaciones definidas como funciones, donde los parámetros están calificados como parámetros de entrada y/o salida (*in , out, inout*)
  - Un método puede activar (*raise*) una excepción que debe ser evaluada y procesada
- Los objetos tienen un tiempo de vida
  - ***transient* (alojados solo en memoria)**
  - ***persistent* (almacenados en disco)**

El Estándar ODMG-OQL Características

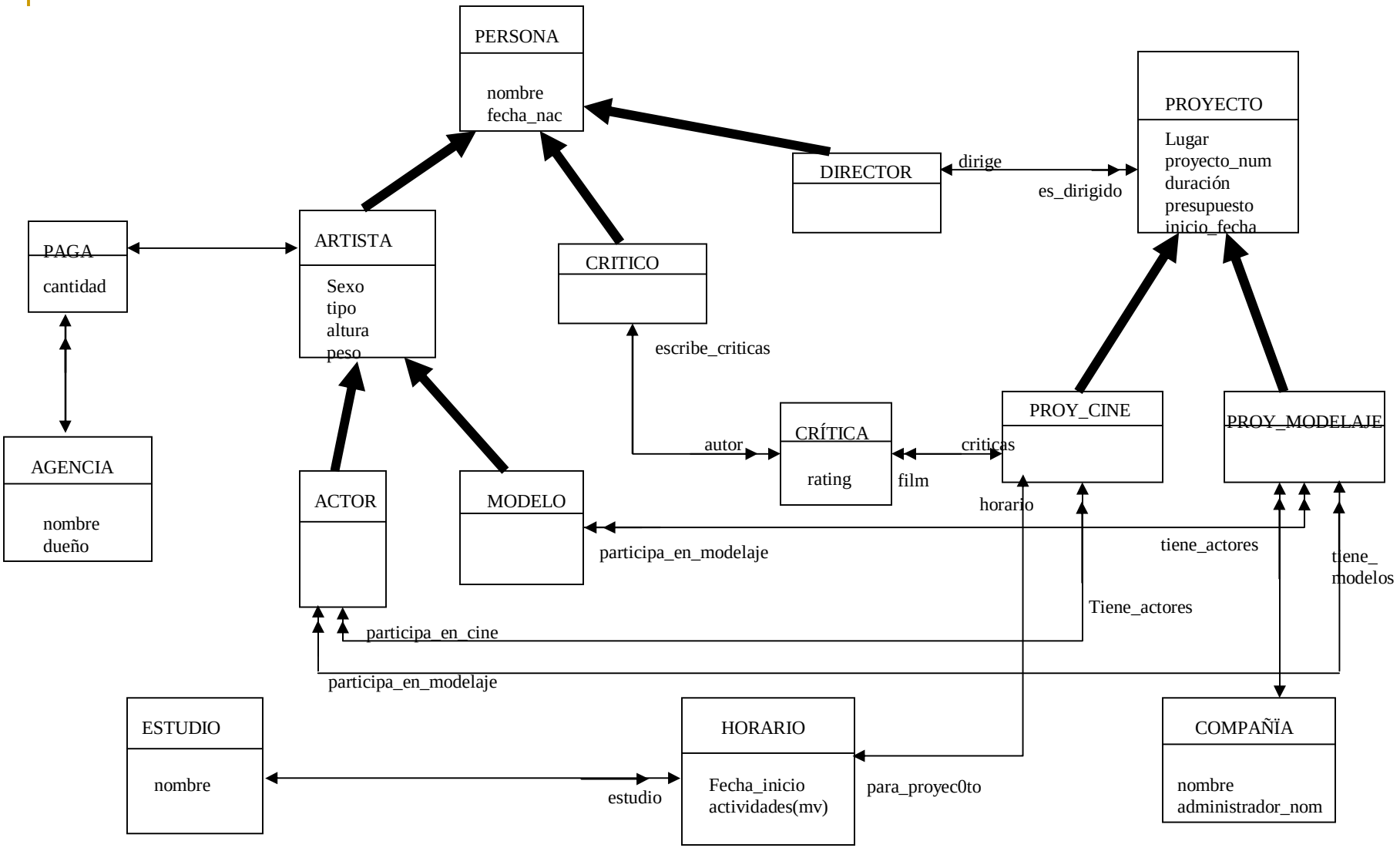
El Object Query Language (OQL) es el lenguaje de consultas asociado al ODL del estándar ODMG

- OQL no es computacionalmente completo, pero las consultas (queries) pueden invocar métodos y los métodos pueden incluir consultas
- OQL provee un acceso declarativo a objetos
- OQL está basado en el lenguaje de definición ODL
- OQL tiene una sintaxis abstracta
- La semántica formal de OQL puede definirse fácilmente (pero no se ha hecho!)
- La sintaxis de OQL puede ser extendida fácilmente para que el lenguaje se combine con lenguajes de programación como C++, Smalltalk y Java

El Estándar ODMG-OQL Características

- OQL provee constructores básicos para interactuar con conjuntos, estructuras y listas
- OQL no define sus operadores propios de actualización. Utiliza los operadores de actualización definidos en los objetos
- OQL puede ser fácilmente optimizado (pero no se ha hecho!)
- Expresiones en OQL producen como resultado átomos, estructuras, colecciones y literales. El tipo del resultado puede ser inferido de las expresiones en la consulta (query)

Esquema Actores-Proyectos



El Estándar ODMG-OQL Características

1. Obtiene el conjunto de fechas de nacimiento de personas llamadas “José”. El resultado es de tipo Set<string**>**

```
select distinct p.fecha_nac
from persona p
where p.nombre = “Joe”
```

2. Regresa el número de proyecto y el presupuesto de proyectos localizados en “Monterrey”. El tipo resultante es set<struct(pnúmero: string, presupuesto:real)>

```
select distinct struct(pnúmero:p.proyecto, presupuesto:p.presupuesto)
from proyecto p
where p.lugar  = “Monterrey”
```

El Estándar ODMG-OQL Características

3. Obtener el administrador de cada compañía y los proyectos de modelaje patrocinados por la compañía que tiene un presupuesto mayor a $1000,000 dls. El tipo resultante es Set<struct(nombre:string, proyectos:Bag<proyecto>)>

```
select distinct struct(nombre:c.administrador,
                       proyectos:(select p
                                  from c.patrocina p
                                  where p.presupuesto > $1000,000.00)
from Compañía c
```

Las consultas en OQL no necesariamente tienen que utilizar el formato **`select-from-where`**

4. Obtener el conjunto de todas las personas

   Persona

El Estándar ODMG-OQL Características

Las consultas también pueden regresar colecciones de objetos con identidad o objetos individuales con identidad

```
select p
from Persona p
where p.nombre =“José”
```

```
element (select p
         from Proyecto p
         where p.proyecto_num =“123”)
```

El estatuto define se usa para darle un nombre a una consulta.

Definir “PP” como el conjunto de todas las personas cuyo nombre es “José”/

```
define PP as select p
                 from Persona p
                 where p.nombre = “José”
```

Definir José como la persona cuyo nombre es “José”. (Se señala error si existe más de un José)

```
define Pepe as element (select p
                        from Persona p
                                where p.nombre = “José”)
```

El Estándar ODMG-OQL Características

- Construcción de expresiones

Un objeto puede crearse de la siguiete forma

```
Persona (nombre: "Gina", fechanac: "870823")
```

Para estructuras, si p_1, p_2,... p_n, son nombres de propiedades y e_1, e_2,... e_n son expresiones, entonces

`struct(`p_1`:`e_1`, `p_2`:`e_2`, `p_n`:`e_n`)`

es una expresión que define una estructura e_1, e_2,... e_n como valores para p_1, p_2,... p_n.

En general si e_1, e_2,... e_n son expresiones, entonces

`set(`e_1`, `e_2`,... `e_n`)` define un conjunto de n elementos

`list(`e_1`, `e_2`,... `e_n`)` define una lista de n elementos

`bag(`e_1`, `e_2`,... `e_n`)` define una bolsa de n elementos

`array(`e_1`, `e_2`,... `e_n`)` define un arreglo de n elementos

El Estándar ODMG-OQL Características

- Cuantificador Universal

  for all x in e_1:e_2

  evalúa a verdadero si todos los elementos en e_1 satisfacen e_2

- Cuantificador Existencial

  exists x in e_1:e_2

  evalúa a verdadero si al menos un elemento de e_1 satisfacen e_2

- Prueba de membresía

  José in Pepes

  evalúa a verdadero si el objeto José se encuentra en el conjunto de objetos Pepes

El Estándar ODMG-OQL Características

Expresiones de navegación (Path Expressions)

Para navegar de un objeto a otro se utilizan path expressions. En OQL, un punto o una flecha (->) se pueden usar para indicar navegación en una path expression

P.critica.proyecto.calendario_filme.fecha_inicio

P->critica->proyecto->calendario_filme->fecha_inicio

OQL no permite a las expresiones de navegación viajar a través de atributos múltiples. La consulta debe expresarse de tal forma que todas las variables sobre atributos múltiples están explicitamente definidos.

```
select p.lugar
from Director d
         d.dirige p
where d.nombre = “José”
```

El Estándar ODMG-OQL Características

OQL permite el uso de métodos en las consultas de la misma forma en que se especifican los atributos

(asumir un método edad para calcular edad tomando comoe entrada fecha de nacimiento)

```
select p.nombre
from Persona p
where p.edad() > 50
```

OQL tambiém permite al usuario declarar el tipo especifico de la clase de un objeto si el tipo no se puede inferir de la consulta.

```
select (Empleado) p.fecha_contrato
from Persona p
where p.sexo="M"
```

El Estándar ODMG-OQL Características

Extracción de elementos de colecciones.

OQL provee una notación explicita para extraer elementos de lita y arreglos. El primer elemento en una colección indexada se asume cero.

```
list(a,b,c) [1]        regresa el valor b
list(a,b,c,d)[1:3]     regresa  list(a,b,c)
```

Conversión de expresiones

OQL provee operaciones para convertir listas a conjuntos y desanidar colecciones de colecciones.

1. Transformar una lista a un conjunto

```
listtoset(list(1,2,3,2))        regresa set(1,2,3)
```

El Estándar ODMG-OQL Características

Conversión de expresiones

2. Desanidamiento de colecciones de colecciones

Asumir que e es una expresión del tipo $col_1<col_2<t>>$. El resultado de flatten(e) es:

- **Si col_2 es un set, entonces la unión de todos los elementos $col_2<t>$. Se hace y el resultado es un set. La misma regla aplica a bags**

```
flatten(list(set(1,2,3), set(3,4,5,6),set(7)))
     regresa set(1,2,3,4,5,6,7)
```

- **Si col_2 es una lista y col_1 es una lista, se hace la concatenación de $col_2<t>$ en el orden en que aparecen en col_1 y el resultado es $col_2<t>$. La misma regla aplica a arreglos**

```
flatten(list(list(1,2), list(1,2,3)))
regresa list(1,2,1,2,3)
```

El Estándar ODMG-OQL Características

Conversión de expresiones

- **Si col_2 es una lista o un arreglo y col_1 es un set o bag, las listas o arreglos se convierten a sets, la unión de sets es entonces calculada y el resultado es set<t>.**

```
Flatten(set(list(1,2), list(1,2,3)))
  regresa  set(1,2,3)
```