Bases de datos avanzadas

Universidad de Los Andes

Postgrado en Computación

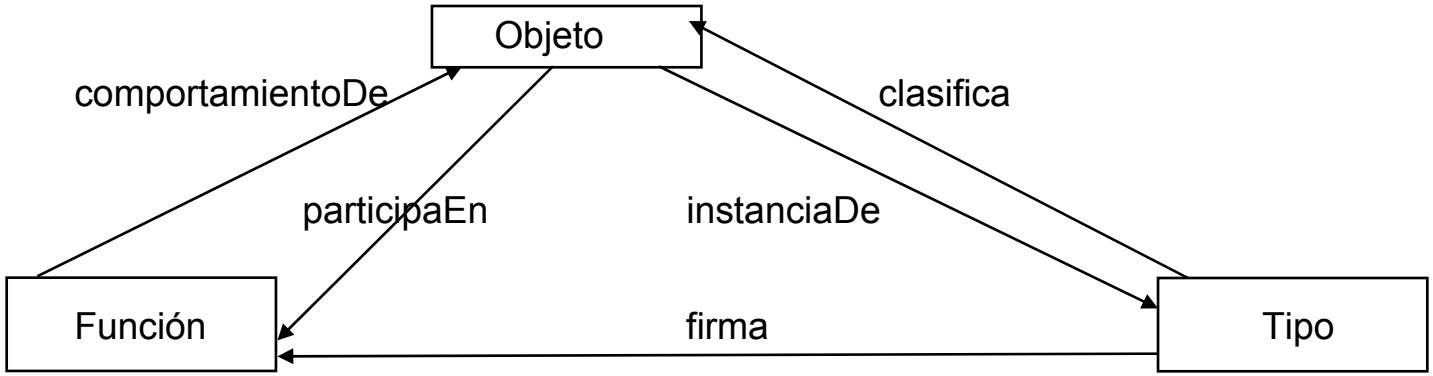
Prof. Isabel M. Besembel Carrera

Unidad I. Sesión 5. Diseño de BDOO.

UNIVERSIDAD DE LOS ANDES

Esquemas en BDOO

- OSQL- Modelo de objetos
- Conceptos básicos: objetos, tipos y funciones.



- Los objetos se pueden clasificar en:
  - ✓ Literales (enteros, cadenas, etc)
  - ✓ Agregados (listas, tupla, conjunto, bag) → Extensiones predefinidas
  - ✓ Subrrogados (contienen un OId) → Extensiones dinámicas

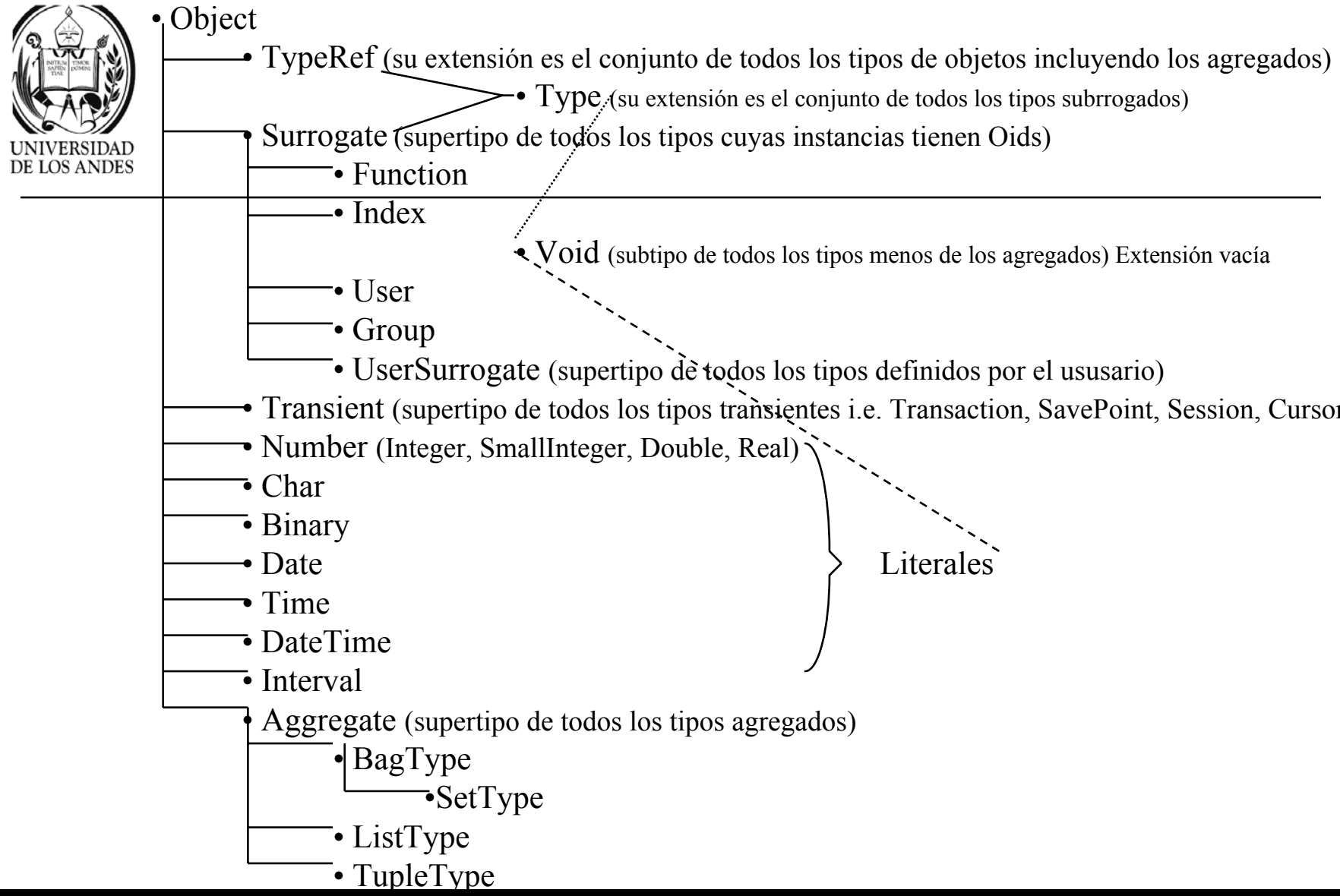
UNIVERSIDAD
DE LOS ANDES

Esquemas OSQL

- Tipos:
- Usados para clasificar los objetos en base a sus propiedades y comportamiento compartido.
- la extensión de un tipo es el conjunto de todas sus instancias
- Los tipos se relacionan en jerarquías supertipo/subtipo, con herencia múltiple.
- Los tipos definidos por el usuario deberán ser subtipos de UserSurrogate.
- El tipo Surrogate contiene objetos que pueden ser instancias de cualquier número de tipos, incluso de tipos no relacionados por la relación generalización/especialización es-un(a)

UNIVERSIDAD DE LOS ANDES

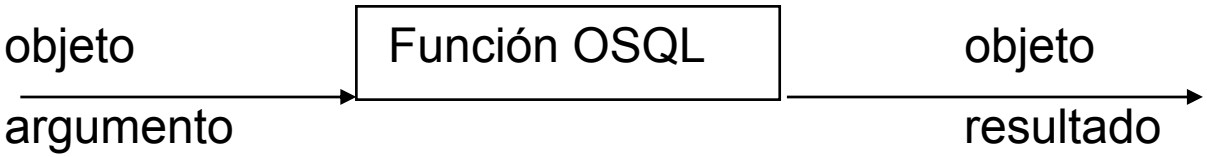
Jerarquía de tipos en OSQL



UNIVERSIDAD
DE LOS ANDES

Funciones OSQL

- Una función puede modelar atributos y relaciones almacenadas o derivadas y cálculos arbitrarios.



- Funciones definidas por el sistema:
- CreateType: crea un nuevo tipo definido por el usuario y opcionalmente crea una o más funciones que tienen ese tipo entre sus argumentos
- CreateFunction: crea una nueva función definida por el usuario
- ImplStored: implementa una o varias funciones como almacenadas (aquella cuya extensión se almacena en la BD)
- ImplOSQL: implementa una función como expresión OSQL, bien sea como expresión en un LP o como una consulta

UNIVERSIDAD
DE LOS ANDES

Funciones en OSQL

- ImplExternal: implementa una función como un programa en LPAN
- CreateObj: crea un nuevo objeto, una instancia del tipo definido por el usuario, opcionalmente inicia 1 o más funciones para el nuevo objeto.
- Para crear un nuevo tipo (OpenODB) se usa

```
create type Profesor subtype of Trabajador functions
(
        feConcOp        Date
        dedicación      Char        (var 10)
        categoría       Char        (var 10)
        salario         Real
);
```

- Ella será pasada a una llamada a la función CreateType como sigue

```
CreateType(‘Profesor’, Set(type Trabajador),
        List(Tuple(‘feConcOp’,type Date,
                ‘dedicación’,type Char (var 10),
                ‘categoría’, type Char (var 10),
                ‘salario’, type Real)))
```

- Se pueden definir funciones que involucren objetos del sistema como tipos y funciones,

```
Ejm:    create function ayuda(Type t) -> Char as stored;
        ayuda(Type Profesor):= ‘Descriptor de ayuda para el tipo Profesor’;
```

UNIVERSIDAD DE LOS ANDES

Expresiones OSQL

- Las constantes y los identificadores se definen de forma usual, pero además se soportan constantes agregadas como tuplas, conjuntos, bolsas y listas.
  - Ejm: Set(1,2,5-2).
- También se soportan constantes de tipos y funciones.
  - Ejm: function nombre, function nombre.Estudiante, type T.
- Para asignar valor a una función: nombre.Profesor(p1) := 'Carlos Mendez'
- El uso de variables, su alcance y asignaciones es similar al lenguaje C.
- Asignación de variables en OpenODB: i:=2
- Decisión: b := **if** salario(p1) >= 200000 **then** a **else** c;
- Bloque de expresiones:

```
begin
        declare Integer  i, j;
        i:=1;
        j:= 3;
        i + j ;
end
```

UNIVERSIDAD
DE LOS ANDES

Expresiones OSQL

- Lazos iterativos no regresan valor alguno (tipo void) y se pueden usar para iterar sobre los elementos de un tipo agregado.
  - Ejm: for p in Set(p1, p2, p3, p4) do cambioDedic(p) := 'exclusiva'
- Quote permite a las funciones evaluar sus propios argumentos, se puede usar para definir funciones que crean e implementan otras funciones.
- Las declaraciones solo se pueden usar dentro de un bloque de expresiones, el cual define su alcance
- Ejemplo:

```
create function filtro(Function f, SetType(Object) arg) -> SetType(Object) as osql
begin
    declare SetType(Object) r;
    declare Object e;
    for e in arg do          if ( f(e) )    then r := r + e;
                             endif;
    return r;
end;
```

Sintaxis OSQL

Sintaxis: |: escogencia, tupla: agregación, +: 1 o más, *: 0 o más)

Expr	:= Constant | Identifier | FuncAppl | Assign | Conditional | BlockExpr | ForLoop | WhileLoop | Quote | Select;
FuncAppl	:= func_ref : Expr arg : Expr;
Conditional	:= pred : Expr then : Expr else : Expr;
Assign	:= id : Identifier val : Expr
BlockExpr	:= declare : Declaration+ exprs : Expr+;
ForLoop	:= id : Identifier domain : Expr loop : Expr;
WhileLoop	:= pred : Expr loop : Expr;
Quote	:= Expr;
Declaration	:= t : TypeRef id : Identifier;

UNIVERSIDAD
DE LOS ANDES

ODL

- La especificación del esquema de la base de datos en ODMG se realiza con las instrucciones siguientes.
- Especificación instrucciones ODL:

<declaraciónInterfaz> ::= **interface** <id> [<especificaciónIdentificxador>]
<listaPropocisionesTipo>
[: <declaraciónPersistencia>]
{ [<cuerpoDeLaInterfaz>] };
<declaraciónPersistencia> ::= **persistent** | **transient**
<especificaciónHerencia> ::= <nombreAlcance> | <nombreAlcance>, <especificaciónHerencia>
<listaProposicionesTipo> ::= ([<especificaciónExtensión>] [<especificaciónDeClave>])
<especificaciónExtensión> ::= **extent** <id>
<especificaciónDeClave> ::= **key[s]** <listaDeClaves>
<listaDeClaves> ::= <clave> | <clave> , <listaDeClaves>
<clave> ::= <nombreProposición> | <listaDeProposiciones>
<listaDeProposiciones> ::= <nombreProposición> | <nombreProposición> , <listaDeProposiciones>
<nombreProposición> ::= <nombreAlcance>
<nombreAlcance> ::= <id> | ::<id> | <nombreAlcance>::<id>

UNIVERSIDAD
DE LOS ANDES

ODL

- **Propiedades de las instancias:** Las propiedades de un tipo de instancia son los atributos y las relaciones de dichas instancias. Estas propiedades se definen con las especificaciones de los atributos y de las relaciones.

  < cuerpoInterfaz > ::= <export>|<export> < cuerpoInterfaz >

  <export> ::= <declaTipo>; |<declaConstante>; |<declaExcepción>; |<declaAtributo>; |<declaRelación>; |<declaOperación>;

- **Atributos:**

  <declaAtributo> ::= **[readonly] attribute** <tipo><identificador> [[<positive_int_const>]]

  <tipo> ::= <especTipoBásico> |<tipoEstructurado> |<tipoEnumerado> |<colección><literal> |<colección><identificador>

  <colección> **::= Set | List | Bag | Array**

UNIVERSIDAD
DE LOS ANDES

ODL

- **Relaciones:**

  < declaRelación > ::= **relationship** <nombreRol><identificador> [**Inverse** < nombreRolInverso>]
  [{**order_by** <listaAtributos>}]

  < nombreRol > ::= < identificador >|<tipoColecciónRelación> < < identificador > >

  < nombreRolInverso > ::= < identificador > :: < identificador >

  < listaAtributos > ::= <nombre> | < nombre >,< listaAtributos >

- **Operaciones:**

  < declaOperación > ::= **[oneway]** <especifTipoOperación> < identificador > <declaParámetros>
  [<expreRaises>][<expreContexto>]

  < especifTipoOperación > ::= < especTipoBásico > **| void**

  < declaParámetros > ::= ([<listaDeclaParámetros >])

  < listaDeclaParámetros > ::= < declaParámetros > |< declaParámetros >,< listaDeclaParámetros >

  < declaParámetros > ::= <atributoParámetro>< especTipoBásico ><declarador>

  < atributoParámetro > ::= **in** | **out** | **inout**

  < expreRaises > ::= **raises** (<listaNombres>)

  < expreContexto > ::= **context**(<listaCadenasLiterales>)

  < listaNombres > ::= <nombre> |< nombre >, < listaNombres >

  < listaCadenasLiterales > ::= <cadenaLiteral> | < listaCadenasLiterales>, <listaCadenasLiterales>

Ejemplo ODL

```
interface  Paciente
   (extent pacientes
    key nroHist  ): persistent
   {attribute String  nroHist;
              String  nombre;
              String dirección;
              String teléfono;
              String referencia;
              String profesión;
              Date   fechaNac;
   relationship set<Cita> tieneCitas inverse Cita::esDe;
   relationship set<Ficha> tieneFichas inverse
      Ficha::pertenece;
   Integer edad( );
   Boolean nuevoPac( ) raises (Ya está);
   Boolean cambioTlf();
   Boolean cambioDir();
};
```

```
interface Cita
  (extent Citas
   keys (esDe, fechaCita, horaCita ):persistent
  { attribute Date fechaCita;
              Time horaCita;
              Boolean asistió;
   relationship Paciente esDe inverse Paciente::tieneCitas;
   Boolean darCita() raises(No es posible);
   Boolean cambiarCita() raises(No es posible);
   Boolean citasDeHoy();
   Boolean emergencia();
   Boolean cancelarCita() raises(No está la cita);     };
interface Tratamiento
 ( extent Tratamientos
   key (nroTra ) ):persistent
   { attribute String nroTra;
              String nomTra;
              String descTra;
              Real costo;
    relationship Set<Ficha> esDe inverse Ficha::tratadoCon;
    Boolean nuevoTra() raises (Ya está);
    Boolean cambiarTra();
    Boolean cambioCosto();    };
```

UNIVERSIDAD DE LOS ANDES

Manejo de vistas

- Usadas para:
  - ✓ especificar particiones lógicas de la extensión de las clases
  - ✓ Definir autorizaciones basadas en contenido
- Definición similar a las clases del esquema de una BDOO, pero incluyendo una consulta para **materializar** la extensión de la clase
- Propiedades:
  - ✓ Contiene un nombre, una lista de atributos con su tipo de dato, sus relaciones, sus métodos (nombre, parámetros y cuerpo) y consulta
  - ✓ La consulta puede ser un método, una consulta u otra vista, con operaciones de proyección y selección o métodos sobre 1 o más clases
  - ✓ Un atributo puede definirse igual o diferente al de la clase o puede ser un atributo nuevo

UNIVERSIDAD
DE LOS ANDES

Propiedades

- El tipo de dato de un atributo o método en la vista puede ser el mismo de la clase almacenada o una especialización/generalización
- Un atributo nuevamente definido es derivado de 1 o más atributos
- El método puede ser el mismo de la clase almacenada o se puede definir nuevamente
- Solo se almacena la definición de la vista y se materializa sólo cuando ella se invoca
- Puede ser utilizada como clase en una consulta
- Hay restricciones en el tipo de actualizaciones a través de vistas, en particular aquellas que involucran atributos derivados y productos

Ejemplo en UniSQL/X

```
Create Class Empleado
    ( ci: char(10), nombre: char(32), edad: integer, salario: money,
      dpto: char(24), comision: money );
Create View PagoEmp
    ( cedula, montoAPagar: money, dpto ) as
    select cedula, salario+comision, dpto
    from Empleado;
Create Class Vehiculo
    ( placa: char(6), color: tipoColor, fabricante: Empresa, año: integer )
    methods ( peso( ): integer );
Create View VehiculosViejos
    ( placa, fabricante, año ) methods ( peso ) as
    select placa, fabricante, año
    from Vehiculo
    where año < 1988 ;
```

UNIVERSIDAD DE LOS ANDES

Herencia

- Problemas de tener una jerarquía de herencia con clases y vistas:
  - ✓ Al ocurrir una cambio de esquema, el SGBDOO no puede cambiar automáticamente la especificación de la consulta en la vista
  - ✓ Viola la semántica de la jerarquía de herencia, pues la vista utiliza las clases y se da un solapamiento de extensiones de clases y de vistas
    - Una sola clase: no hay muchos problemas
    - Varias clases o una vista definida en varias clases: problemas de detección del solapamiento durante el procesamiento de la consulta
- Solución:
  - ✓ mantener dos jerarquías, la de herencia y la de derivación de vistas
  - ✓ pero la jerarquía de derivación de vistas no está definida en la orientación por objetos

UNIVERSIDAD DE LOS ANDES

Atributos anidados

- Dominio del atributo:
  - ✓ Tipo básico de datos: Integer, Real, Char, etc.
  - ✓ Tipo complejo: set, bag, etc.
  - ✓ Clase definida por el usuario
  - ✓ Vista? Conceptualmente su valor debería ser un OId
    - Problema: el OId es de una instancia de una clase o de una vista?
      - Si la vista está definida en 2 o más clases o vistas => **no hay un solo OId**
      - Si el OId es de una instancia de clase => **realmente el dominio no es la vista**
    - Solución:
      - Tener el valor del atributo cuyo dominio es una instancia materializada de la vista
      - Hay mucho overhead en la implementación
        - Generar OIds de las instancias materializadas y mantenerlas hasta el fin de la Ti
        - Si la vista es actualizable => crear una tabla hash con los OIds y sus instancias almacenadas

UNIVERSIDAD
DE LOS ANDES

Atributos anidados

Create class Compañía

(nombre: **char(20),** ubicación: **char(20),** gerentes: **set-of** Empleado, capital: **money)**

Create view GranCompañia(nombre, ubicación) **as**

Select nombre, ubicación **from** Compañía **where** capital > 100.000.000,00;

Create class Vehiculo

(placa: **char(6),** color: **char(20),** fabricante:GranCompañia, año: **integer)**

Methods (peso**( ): integer);**

- Si la vista involucra clases definidas por el usuario => caminos en las jerarquías de clase y de composición, i.e. Vehiculo.fabricante.ubicacion = "Cumaná"
- Si el dominio de un atributo A de la clase C es una vista V, los valores almacenados en A son referencias a la consulta de V, procesamiento:
  - ✓ Recuperación de la consulta de V
  - ✓ Ejecutarla para encontrar los Oids
  - ✓ Obtener las instancias almacenadas

Solución UniSQL/X

- Dominio de un atributo de una vista: otra vista o una clase
  - ✓ Si es otra vista => ésta no puede ser el producto de 2 o más clases
- Dominio de un atributo de una clase: no puede ser una vista

Create view NuevoVehiculo**(**placa, fabricante: GranCompañia, año**) as**

Select placa, fabricante, año **from** Vehiculo **where** año > 1991

- Dominio de un atributo de una vista (general): si el dominio de un atributo en una clase es un conjunto heterogéneo S de tipos Tj => el dominio de un atributo en una vista que usa esa clase puede ser S o un subconjunto de S.

Create class CompañiaVenezolana **as subclass of** Compañía;

Create class CompañiaExtranjera **as subclass of** Compañía;

Create class VehiculosImportados**(**fasbricante: CompañiaExtranjera**) as subclass of** Vehiculo;

Create class VehiculosEnsamblados**(**fabricante: CompañiaVenezolana**) as subclass of** Vehiculo;

Create view TodoVehiculo **(**placa, fabricante: Compañía, año**) methods (**peso**) as**

(select placa, fabricante, año **from** VehiculosImportados **) union (select** placa, fabricante, año **from** VehiculosEnsamblados **);**

Métodos y actualización de vistas

- Métodos en la vista: para efectuar cálculos sobre las instancias materializadas
  - ✓ Operan sobre instancias virtuales, no asumen persistencia de los Oids
  - ✓ UniSQL/X: Oids de las instancias virtuales no son generados, por ello los métodos deben ser escritos para operar sobre los valores de las instancias materializadas o sobre los Oids de las clases de la vista
- Actualización de vistas:
  - ✓ Mismas restricciones que en BDR para correspondencia entre atributos
  - ✓ Identidad del objeto: correspondencia 1:1 entre atributos de instancias de clase e instancias de la vista

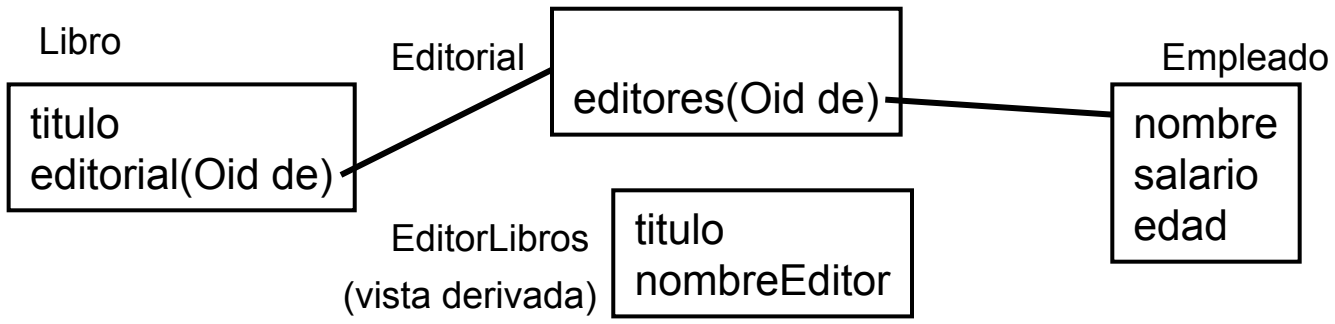
Create view EmpleadoSalario **(**salario**) as**

Select salario **from** Empleado**;**

UNIVERSIDAD DE LOS ANDES

Actualización a través de vistas

- Atributos anidados: atributo a ser actualizado no puede ser un atributo derivado
- Regla de materialización:
  - ✓ La actualización será reflejada en la vista cuando ella se materialice luego de la actualización
  - ✓ Eliminación de la raíz del atributo anidado
  - ✓ Inserciones: la vista debe contener todos los atributos en el **where**



UNIVERSIDAD
DE LOS ANDES

Cambios de esquema

- Cambios de esquema afectan severamente a las vistas definidas
  - ✓ Cambio sobre la clase o vista sobre la cual se definió la vista
    - Anexar un nuevo atributo
    - Anexar un nuevo método
    - Eliminar un atributo
    - Eliminar un método
    - Cambiar el dominio de un atributo
  - ✓ Cambio de la estructura de la jerarquía de clases o de vistas
    - Anexar una nueva clase o vista
    - Eliminar una clase o vista
    - Colocar una clase o vista S como nueva superclase de una clase o vista C
    - Eliminar una clase o vista S que haya sido superclase de una clase o vista C

UNIVERSIDAD DE LOS ANDES

Cambios de esquema

- Inserción o eliminación de clases y superclases se reduce a inserción o eliminación de atributos y métodos en una clase
- Si un atributo es eliminado de una clase => la definición de cualquier vista que lo referencie es inválida => la vista debe ser eliminada o modificada, así como todas sus subvistas
- Si un atributo nuevo es insertado => no afecta a las vistas a menos que incluya todos los atributos *
- El SGBDOO no puede hacer los cambios automáticamente
- Inserción o eliminación de métodos: similar a los atributos
- Eliminación de una instancia de la vista que se deriva del objeto anidado:
  - ✓ Eliminación de todas las instancias del objeto anidado (Libro, Editorial, Empleado)
  - ✓ Eliminación de la raíz de la jerarquía de composición (Libro).

UNIVERSIDAD DE LOS ANDES

Extendiendo el uso de las vistas

1. Integración de esquemas heterogéneos: definición de vistas extendida para contener una lista de consultas
2. Simulación de la evolución del esquema: una vista puede simular los cambios sin propagarlos a la BD. Tipos:
   - ✓ Cambio de definición de una clase: anexar/eliminar atributos o métodos y cambiar el dominio de un atributo
   - ✓ Cambio de la estructura de la jerarquía de clases: anexar/eliminar una clase, colocar/remover una clase como superclase de otra.

➢ Se puede definir una vista que elimine atributos, métodos o clases pero no para anexar

Create view Vehiculo **(**placa, fabricante, año **) as**

Select placa, fabricante, año **from** BD1.Vehiculo**,**

Select placa, fabricante, año **from** BD2.Vehiculo**;**

UNIVERSIDAD DE LOS ANDES

Seguridad en BDOO

- Un sistema de bases de datos permite a los usuarios compartir los datos y el mecanismo de autorización debe asegurar que los mismos son accedidos solamente por los usuarios autorizados
- Basada en:
  - ✓ Identificación y autorización de los usuarios: uso de códigos de acceso y palabras claves, exámenes, impresiones digitales, reconocimiento de voz, barrido de retina, etc.
  - ✓ Autorización: usar derechos de acceso dados: por terminal, por la operación que puede realizar o por la hora del día.
  - ✓ Uso de técnicas de cifrado: protección en BD distribuidas o con acceso por red o Internet.
  - ✓ Diferentes tipos de cuentas: especialmente la del ABD con permisos para: creación de cuentas, concesión y revocación de privilegios y asignación de los niveles de seguridad.
  - ✓ Manejo de la tabla de usuarios con código y contraseña, control de las operaciones efectuadas en cada sesión y registradas en la bitácora para facilitar la auditoría de la BD

UNIVERSIDAD
DE LOS ANDES

Seguridad y autorización

- La seguridad en BD involucra dos aspectos:
  - ✓ Políticas de seguridad: pautas de alto nivel para determinar cómo se controlan y deciden los accesos
  - ✓ Mecanismos de seguridad
- **Políticas de seguridad:** contienen todo lo relacionado con la seguridad de la información, seleccionada entre muchas alternativas, incluye:
  - o las necesidades de los usuarios,
  - o el ambiente de instalación,
  - o las regulaciones de la institución y
  - o las restricciones legales.
- **Mecanismos de seguridad:** conjunto de funciones usadas para implementar y asegurar dichas políticas, pueden implementarse en hardware o software.
- Las políticas son expresadas en un sistema discrecional que traslada esas políticas a reglas de acceso, denominado **autorización**.
- El control de acceso puede ser obligatorio y/o a discreción.

UNIVERSIDAD
DE LOS ANDES

Seguridad discrecional

- Modelo orientado al chequeo de políticas de control de acceso basadas en la identidad del usuario
- Usada para otorgar y revocar privilegios a nivel de objetos en un modo determinado (consulta o modificación)
- La autorización a discreción puede estar conectada a cada objeto o almacenada en alguna estructura de autorización separada.
- **Almacenamiento en estructura separada:** confronta problemas de implementación en la política denominada último privilegio.
- **Conexión de una lista de autorización a cada objeto:** permite que la lista sea heredada. Contiene los grupos de usuarios que tienen acceso. (Un individuo está representado por un grupo de un elemento)
- Un objeto hereda la autorización de sus superclases.
- Adicionalmente es conveniente restringuir o reescribir las autorizaciones heredadas, para poder asignar la autorización precisa a cada objeto según el usuario.

Autorización

- **Explícita:** lista o grupos de usuarios incluidos en cada objeto que poseen un determinado privilegio para acceder ese objeto
- **Implícita:** las que se pueden calcular a partir de las explícitas
- **Administrativas:** El usuario que crea un objeto es considerado su propietario y como tal es el único que puede eliminarlo
- **Positiva:** el usuario debe pertenecer a algún grupo en la lista de autorización positiva (auto+).
- **Negativa:** el usuario NO debe pertenecer a uno de los grupos en la lista de autorización negativa (auto-). Se usa para anular los efectos de la anterior.
- El uso de la auto+ junto con la auto- permite la implementación de políticas que asignan mayores privilegios conforme se desciende en la jerarquía de clases.
- La auto- permite restringir la herencia de autorizaciones y asignar a cada usuario la autorización precisa y deseada para un objeto.

Autorización fuerte y débil

- **Fuerte:** aquella que no puede ser reescrita por otra autorización.
- **Débil:** aquella que puede ser reescrita por una autorización fuerte.
- La combinación de autorizaciones positivas/negativas y fuertes/débiles permiten una representación más precisa y flexible de los diversos requerimientos de las autorizaciones.
  - ✓ Ejemplo: un usuario posee:
    - autorización positiva/débil para el privilegio *select* sobre un objeto y pertenece al grupo ***a*** que tiene una autorización negativa/fuerte para el mismo objeto
    - tiene una autorización positiva/fuerte para el privilegio *update* sobre ese objeto y también pertenece al grupo ***b*** que posee una autorización negativa/débil.
    - Conflicto de autorizaciones según el objeto, operación y objeto

UNIVERSIDAD DE LOS ANDES

Enfoques de gestión de autorizaciones positivas y negativas

- **Regla del más-específico:** Si un usuario tiene o no autorización para un objeto, ello toma precedencia sobre cualquier otra autorización relacionada con los grupos a los cuales pueda pertenecer.
  - ✓ Si el usuario A desea colocar disponible el objeto P al grupo G excepto el usuario B, A puede colocar G en la lista de autorización+ y a B en la -.
- Esta regla se implementa colocando las autorizaciones individuales fuertes y las de grupo débiles.
- **Negación-primero:** La lista de auto- toma precedencia sobre la +.
  - ✓ Si un usuario A niega explícitamente la autorización al usuario U para el objeto P, y un usuario B le da autorización a U para P, entonces U no está autorizado en P.
- Esta regla se implementa colocando las auto- fuertes y todas las auto+ débiles.
- Ejemplo:
  - ✓ El conflicto con el privilegio *select* es resuelto según la regla del más específico fijando la autorización del sujeto en positiva/fuerte y la del grupo ***a*** en negativa/débil.
  - ✓ La regla de la negación toma precedencia resuelve el conflicto para el privilegio *update* fijando la autorización del grupo ***b*** en negativa/fuerte y la del sujeto en positiva/débil

Enfoques de gestión de autorizaciones

- Grupos de usuarios: agrupamiento de usuarios con ciertas autorizaciones para colecciones de sujetos que comparten algunas características. Cada usuario es un grupo de 1 sujeto.
- Son estáticos: un miembro de un grupo siempre lleva su membresía de grupo
- Etiquetar cada autorización como fuerte o débil: Asignación de 4 listas de autorización a los objetos.
- Asumir la unión de las autorizaciones individuales y la de un grupo al cual pertenece. El grupo puede ser seleccionado por el usuario o por la aplicación al tiempo de invocación o predesignado.
- Asumir las autorizaciones individuales o aquellas de uno de los del grupo. La escogencia puede ser hecha por el usuario o por la aplicación al tiempo de invocación o predesignado.

UNIVERSIDAD
DE LOS ANDES

Roles

- Roles: Permiten definir grupos de usuarios con funciones similares, asignando a cada rol el conjunto de autorizaciones que permitan realizar sus funciones.
- Son dinámicos: los usuarios pueden actuar en un rol o no y pueden tener uno o varios roles.
- El control discrecional puede ser especificado en base a roles.
  - ✓ Ejm: el administrador del sistema, el de BD, el de seguridad, así como aquellos definidos por la aplicación, etc.
- Para especificar los roles de los usuarios se pueden utilizar los *dominios de protección (DP)*.
- Cada DP definido por el diseñador de una aplicación tendrá sus autorizaciones específicas.
  - ✓ Ejm: el DP denominado salario-empleado puede tener auto+ débil para la invocación del método incSalario por los usuarios que no sean el encargado de salarios, y puede tener auto- fuerte para la invocación de los métodos de incSalario y verSalario por el encargado de salarios.

UNIVERSIDAD DE LOS ANDES

Colaterales

- Reescritura de autorizaciones: cambio de un privilegio por otro, para evitar conflictos entre autorizaciones.
- Consistencia: son consistentes si y solo sí no hay autorizaciones en conflicto con respecto a cualquier sujeto.
- Políticas administrativas: regulan la especificación de autorizaciones.
  - ✓ Permiten a un sujeto conceder y revocar autorizaciones de acceso a otros sujetos.
  - ✓ Más comunes:
    - política centralizada: donde un usuario privilegiado puede especificar autorizaciones y
    - política de propiedad: donde cada usuario puede regular los accesos sobre los objetos que él crea

UNIVERSIDAD
DE LOS ANDES

Seguridad obligatoria

- Modelo basado en el chequeo de políticas que restringen el acceso a la información clasificada por parte del personal autorizado.
- Seguridad obligatoria sirve para imponer varios niveles de seguridad a los usuarios y a los objetos (seguridad multi-nivel)
- Cuando un sistema contiene diversas clasificaciones de la información que contiene y múltiples usuarios que no tienen los mismos permisos de acceso a dicha información.
- La clasificación de la información protegida refleja el daño potencial que se puede alcanzar con accesos no autorizados.
- La confiabilidad asignada al usuario refleja su credibilidad.
- La *clase de acceso* se aplica a los usuarios y a la información.
- La clase de acceso consiste de los niveles de sensibilidad (máximo secreto, secreto, confidencial, no clasificado) y de un conjunto de categorías.

UNIVERSIDAD DE LOS ANDES

Múltiples niveles de seguridad

- Para que un usuario tenga acceso garantizado a alguna información clasificada, él deberá tener una confiabilidad dada en los niveles de sensibilidad y en cada una de las categorías de las clases de acceso de la información.
- El conjunto de clases de acceso (*nivel de sensibilidad, conjunto de categorías)* está parcialmente ordenado y forma una retícula denominada la relación *dominación*.
  - ✓ Una clase de acceso A domina otra B si el nivel de sensibilidad de A es mayor o igual al de B y si la categoría de A incluye todas aquellas de B.

Control de acceso obligatorio

- Los datos clasificados deben ser protegidos de accesos directos por usuarios no autorizados y de su descubrimiento a través de medios indirectos (canales secretos de señalización e inferencia).
- Canales secretos: son canales que no han sido diseñados para usar el flujo de información, pero que pueden ser utilizados por algún software malicioso para señalar datos *altos* a usuarios *bajos*.
- **Inferencia:** cuando un usuario *bajo* puede inferir información *alta* basado en el comportamiento observable del sistema

UNIVERSIDAD DE LOS ANDES

Control de acceso obligatorio

- Requerimientos:
  - ✓ **Propiedad simple:** Un sujeto S no tiene permiso de leer datos de la clase C a menos que *clasificación(S) C.* Protege contra el descubrimiento no autorizado.
  - ✓ **Propiedad *:** Un sujeto S no tiene permiso de escribir datos de la clase C a menos que *clasificación(S) C.* Protege los datos bajos de la contaminación de los datos altos.
- Un sujeto es un proceso actuando a favor de un usuario.
- El proceso tiene una clase de acceso derivada de la confiabilidad del usuario.
- Un sujeto es una entidad activa que ejecuta métodos y envía mensajes

Reglas de acceso

- Cada objeto tiene una clase de acceso que se aplica a todo lo que él contiene.
- Dicha clase de acceso está restringida por la regla:
  - ✓ **Propiedad de jerarquía:** La clase de acceso de un objeto debe dominar las clases de acceso de sus superclases. Usada para permitir la herencia.
- Cada sujeto tiene un usuario asociado, derivado de un procedimiento de login o pasado desde otro sujeto que lo invocó. Cuando un objeto recibe un mensaje, se crea un sujeto que maneja el mensaje mediante la ejecución del método apropiado y el sujeto se destruye cuando el método termina. Al crearse el sujeto se le asigna una clase de acceso siguiendo la regla siguiente:
  - ✓ **Propiedad del nivel del sujeto:** La clase de acceso de un sujeto domina la clase de acceso del sujeto que lo invocó y la del objeto que recibe el mensaje. Usada para que el sujeto pueda leer el mensaje y las variables y métodos del objeto.
- El comportamiento de los sujetos se restringe en base a dos reglas:
  - ✓ **Propiedad de la localidad del objeto:** Un sujeto puede ejecutar métodos o leer variables del objeto o de cualquiera de sus superclases, y puede escribir variables en el objeto.

UNIVERSIDAD DE LOS ANDES

Reglas de acceso

- ✓ **Propiedad*:** Un sujeto puede escribir en un objeto solo si su clase de acceso es igual a la del objeto.

- La posibilidad de recibir la respuesta a un mensaje está restringida por:
  - ✓ **Propiedad de regreso de valores:** Un sujeto que invoca un mensaje puede recibir la respuesta si el sujeto que maneja su mensaje tiene una clase de acceso igual a la de él.
- Para protegerse contra los canales secretos, al crearse un objeto se le asigna una clase de acceso más alta que la del sujeto que lo crea.
  - ✓ **Propiedad de creación de objetos:** La clase de acceso del objeto creado domina la clase de acceso del sujeto que invocó la creación

UNIVERSIDAD
DE LOS ANDES

Existencia del objeto

- Cuando un sujeto alto crea un objeto, su existencia debe esconderse para los sujetos bajos y con ello no se permite el canal secreto.
- La existencia de un objeto puede ser inferida por la apariencia del Oid en el valor de una variable dentro de otro objeto.
- Como los Oids son únicos, ello puede conducir a un canal secreto a través de la creación de los mismos.
- Si los sujetos son libres de tratar los Oids o si el sistema asigna esos Oids secuencialmente, los sujetos altos pueden maliciosamente transmitir información a los bajos a través de la creación de objetos.
- El canal desaparece si los Oids se concatenan con la clase de acceso del sujeto que lo pide, o alternativamente usando un generador de números pseudoaleatorios para los Oids

UNIVERSIDAD
DE LOS ANDES

Polinstanciación

- **De entidad:** Cuando una persona con baja credibilidad asigna un identificador a una entidad del mundo real accesible para todas las personas de esa misma credibilidad y no sabe que ese identificador ya existe en otra entidad del mundo real, pero conocida solamente por las personas de alta credibilidad.
  - ✓ Como en BDOO se usan Oids este tipo de polinstanciación no se presenta.
- **Aparente:** Similar al anterior, pero usando nombres de objetos asignados por el usuario o valores para identificar las variables instancia.
  - ✓ Para evitarla se usan las restricciones de clasificación

UNIVERSIDAD DE LOS ANDES

Restricciones de clasificación

- Similar a las restricciones de integridad, las restricciones de clasificación restringen la clasificación que un objeto pueda tener.
- Ellas se implementan a través de métodos que son invocados automáticamente ante cualquier cambio en los objetos relevantes.
- Estos métodos son heredados.

UNIVERSIDAD
DE LOS ANDES

Seguridad según roles

- Modelo de control de acceso basado en roles (RBAC): discrecional y obligatorio.
- Permisos asociados a los roles, con usuarios miembros de los roles.
- Las políticas basadas en roles regulan los accesos de los usuarios a la información basándose en las actividades que estos ejecutan en el sistema.
- Los roles son creados para las diversas funciones de trabajo en una organización y los usuarios son asignados a los roles basándose en sus responsabilidades y calificaciones.

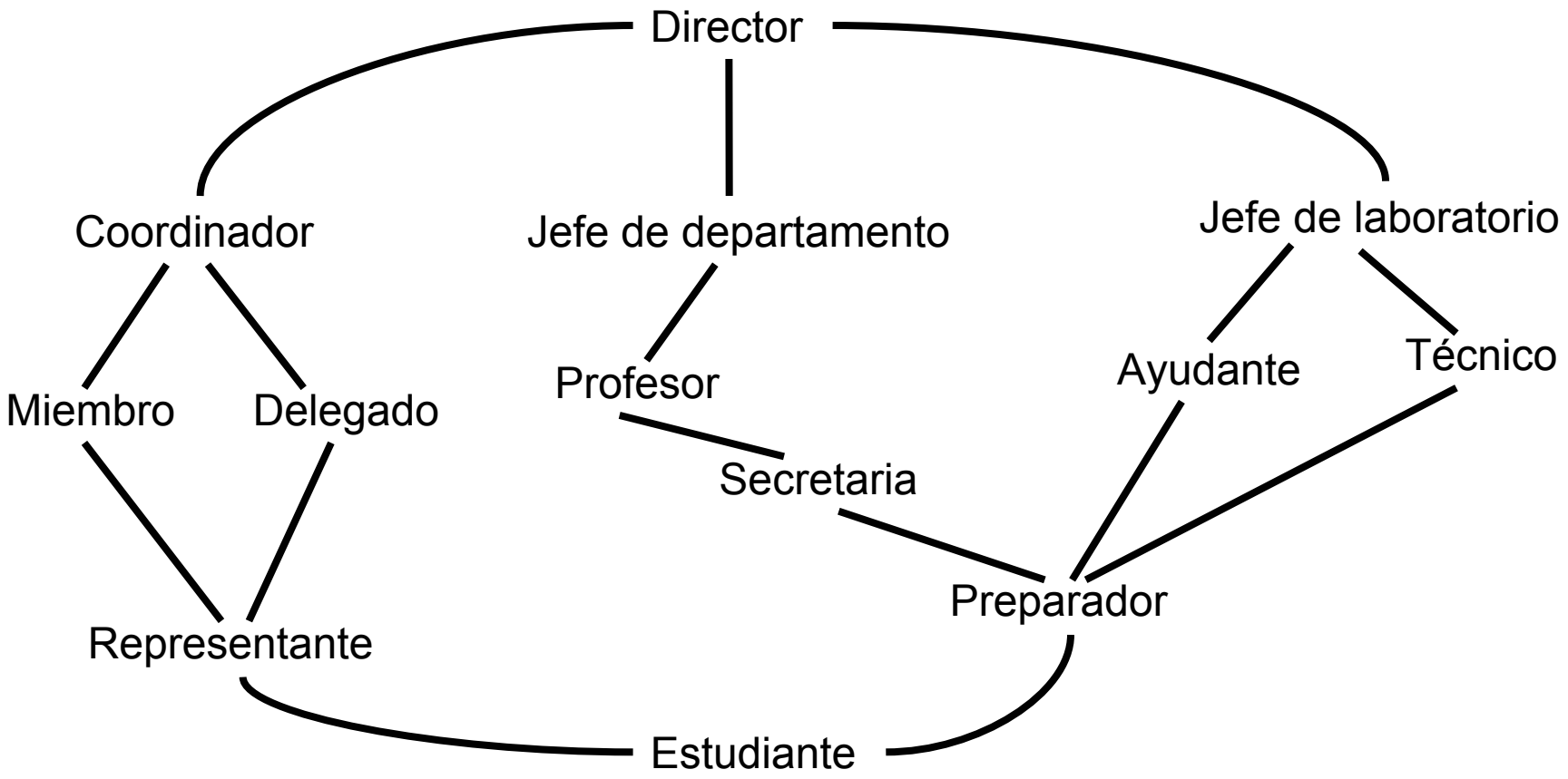
UNIVERSIDAD
DE LOS ANDES

Ventajas

- Administración de las autorizaciones: otorga una independencia lógica especificando las autorizaciones de los usuarios rompiendo esta tarea en dos partes:
  - ✓ una en la que se asignan usuarios a roles y
  - ✓ otra en la que se asignan los derechos de acceso sobre los objetos a los roles.
- Jerarquía de Roles: En muchas aplicaciones hay una jerarquía de roles natural basada en los principios de generalización y especificación (organigrama).
- Privilegio mínimo: los roles otorgados a un usuario contienen los mínimos privilegios requeridos para ejecutar una tarea particular.
- Separación de deberes: ningún usuario debe tener los privilegios suficientes que le permitan emplear mal el sistema.
- Clases de Objetos: clasificación de los objetos análoga a la clasificación de los usuarios de acuerdo a las actividades que estos ejecutan.
  - ✓ Los objetos pueden ser clasificados según su tipo o su área de aplicación.
  - ✓ Las autorizaciones de acceso de los roles se deben dar en base a las clases de objetos y **no** de objetos específicos

UNIVERSIDAD
DE LOS ANDES

Jerarquía de roles



UNIVERSIDAD DE LOS ANDES

Metodología de desarrollo

- Modelo de procesos Reloj
  - ✓ integración de los modelos de desarrollo de software denominados: Cascada, Modelo V y Modelo de Prototipos
  - ✓ Objetivo: servir de guía para la construcción de aplicaciones de pequeña y mediana complejidad
  - ✓ sigue el paradigma de la Orientación por Objetos propuesto por Bruegge y Dutoit y se basa en el estándar IEEE 1074, para el desarrollo del ciclo de vida

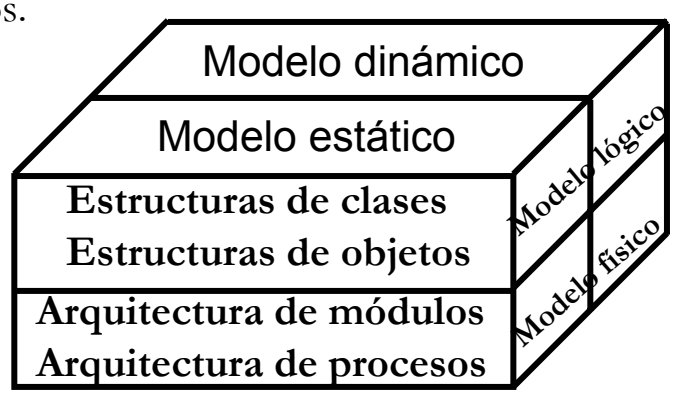
UML:

Modelo lógico estático: diagramas de clases y diagramas de objetos.

Modelo lógico dinámico: diagramas de interacción (de colaboración y de secuencia).

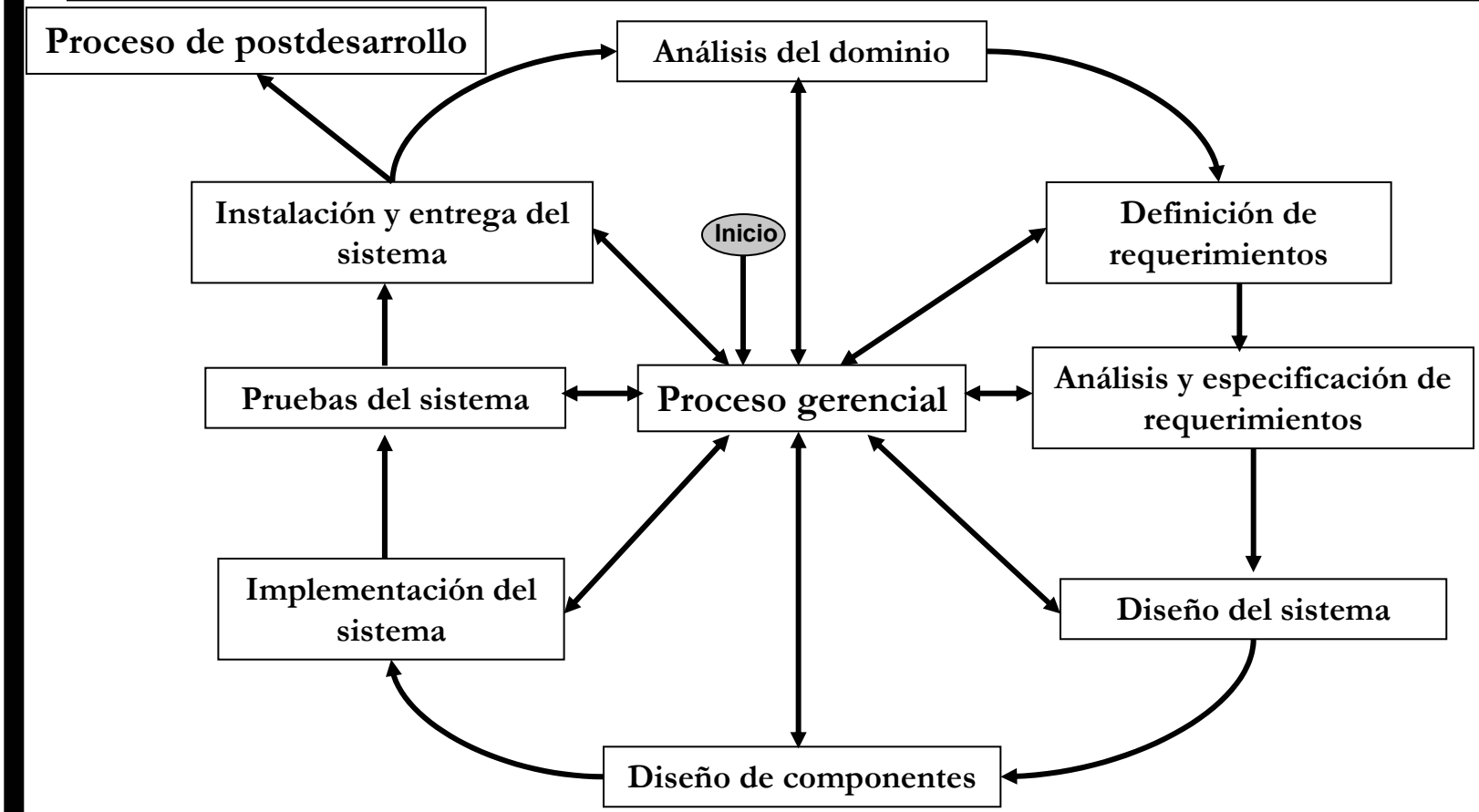
Modelo físico estático: diagramas de componentes y diagramas de despliegue.

Modelo físico dinámico: diagramas de actividades y diagramas de transición de estados.



UNIVERSIDAD DE LOS ANDES

Modelo de procesos Reloj



UNIVERSIDAD DE LOS ANDES

Modelo de procesos Reloj

- **Proceso gerencial:** actividad principal ya que ella dirige y controla la ejecución del proceso de desarrollo, haciendo alusión al motor de un reloj.
- **Análisis del dominio de la aplicación:** permite obtener un buen entendimiento del dominio de la aplicación, antes de iniciar las fases de definición y especificación de requerimientos.
- **Definición de requerimientos:** permite definir y describir los requerimientos del usuario que la aplicación debe satisfacer.
- **Análisis y especificación de requerimientos:** permite expresar los requerimientos del usuario de manera formal y técnica, para que esto sea entendido sin ambigüedad por los diseñadores del sistema. Se especifican los requerimientos funcionales y los de la interfaz de usuario, y la representación preliminar del diagrama de clases.

UNIVERSIDAD
DE LOS ANDES

Modelo de procesos Reloj

- **Diseño del sistema:** permite traducir los requerimientos en una solución. Esto incluye la arquitectura del sistema, diseño de los subsistemas, diseño de las clases, diseño implementable del sistema y diseño de la interfaz de usuario.
- **Diseño de componentes:** permite especificar en detalle cada uno de los componentes y de las conexiones identificadas en la arquitectura de la aplicación.
- **Implementación del sistema:** permite traducir las especificaciones de diseño en un producto concreto de software, probándose la integración de los componentes, si los hubiere. Consiste de la codificación del sistema, la creación de la base de datos, la prueba de los códigos programados y las pruebas de integración del sistema.

UNIVERSIDAD
DE LOS ANDES

Modelo de procesos Reloj

- **Pruebas del sistema:** permite probar funcionalmente el sistema implementado, obteniendo como resultado el sistema probado y listo para ser instalado. Esta consta de las pruebas funcionales, las de rendimiento y las de aceptación.
- **Instalación y entrega del sistema:** el sistema es instalado y probado en su entorno operacional, permitiendo que los usuarios y los operadores sean entrenados en su uso. Esta fase incluye la instalación y pruebas del mismo, así como el entrenamiento de los usuarios y operadores y la distribución de la documentación.

UNIVERSIDAD
DE LOS ANDES

Ciclo de vida de BD

- Definición del sistema: Alcance, usuarios y aplicaciones.
- Diseño: lógico y físico.
- Implantación: Esquemas conceptual, externo e interno, creación de la base de datos e implementación de las aplicaciones.
- Carga o conversión de los datos.
- Conversión de aplicaciones existentes.
- Prueba y validación.
- Operación.
- Supervisión y mantenimiento: puede haber crecimiento y expansión de contenido (datos + aplicaciones). Modificaciones y/o reorganizaciones.

UNIVERSIDAD DE LOS ANDES

Proceso de diseño de BD