estudios de postgrado en computación

# Bases de datos avanzadas

Universidad de Los Andes

Postgrado en Computación

Prof. Isabel M. Besembel Carrera

***Unidad I. Sesión 4. SGBDOO.***

UNIVERSIDAD DE LOS ANDES

# Introducción …

- Características de los SGBDOO {Atkinson, M. et al. *The Object-Oriented Database Manifesto*. Proceedings 1st. International Conference on Deductive and Object-Oriented Databases. Elsevier, 1989}
  - ✓ **Obligatorias:** persistencia, manejo de memoria secundaria, concurrencia, recuperación, consultas, objetos complejos, identidad del objeto, encapsulación, tipos o clases, herencia, reescritura combinada con encadenamiento tardío, extensibilidad y completitud computacional.
  - ✓ **Opcionales:** herencia múltiple, chequeo e inferencia de tipo, distribución, diseño de transacciones y versiones.
  - ✓ **Abiertas:** paradigma de programación, sistema de representación, sistema de tipos y uniformidad
- Objetos complejos: objetos que se construyen a partir de los simples aplicandoles los constructores

UNIVERSIDAD DE LOS ANDES

# Jasmine

- Manejo de objetos complejos y multimedia {http://www.cai.com/products/fdb/jasmine}
- Clases, métodos, instancias con identificador único y atributos transientes o persistentes
- Métodos almacenados con la clase, modificación dinámica
- Herencia simple y múltiple
- Encapsulamiento y polimorfismo
- Concurrencia, recuperación y facilidades de consultas ad-hoc
- Distribución: intranet, extranet e Internet
- Lenguajes: C++, Java, Jasmine Studio
- Procesamiento de consultas: con ODQL

UNIVERSIDAD DE LOS ANDES

# Poet

- Objetos complejos y multimedia {http://www.poet.com/products/documentation}
- Clases, métodos, instancias con identificador único y atributos transientes o persistentes
- Herencia simple y múltiple
- Encapsulamiento y polimorfismo
- Completitud computacional soportada por C++
- Concurrencia, transacciones anidadas, recuperación y facilidades de consultas ad-hoc
- Versionamiento
- Procesamiento de consultas: OQL o PoetQL
- Lenguajes: C++, Java, VisualBasic

UNIVERSIDAD DE LOS ANDES

# O2

- Un tipo de objeto se define usando los tipos atómicos del O2 y aplicando sus constructores de tipo. {Deux, O. *The story of O2*. IEEE transactions on knowledge and data engineering. Vol. 2. Nro. 1. Marzo, 1990}
- Tipos atómicos o simples: **boolean**, **integer**, **string**, **real**, **character** y **bit**.
- Constructores de tipos complejos: **tuple**, **set**, **unique set** y **list**.
- Un objeto tiene identidad única (oid) y un valor actual (estado).
- Un valor **no** tiene identidad. Los objetos tienen identidad.
- Los objetos son persistentes o transitorios. Los valores son transitorios a menos que formen parte de un objeto persistente.
- Las clases se definen anteponiendo la palabra **class** y los métodos con **method**
- La herencia de clases se define con la palabra **inherit**. Se puede cambiar el nombre de los atributos o métodos heredados con **rename** atrib **as** nuevoNombre

UNIVERSIDAD DE LOS ANDES

# Ejemplo O2

```
class Persona
    type tuple (        nss: string,
                        nombre: tuple (     nomPila: string,
                                            paterno: string,
                                            materno: string),
                        dirección: tuple (  número: integer,
                                            ubic: string,
                                            ciudad: string,
                                            estado: string,
                                            codPostal: string ),
                        fechaNac: Fecha,
                        sexo: character )
    method  edad: integer
end
type Fecha: tuple (     año: integer,
                        mes: integer,
                        dia: integer );
```

- Encapsulamiento, reescritura y encadenamiento tardío
- Tipos y clases
- Herencia simple y múltiple
- Extensibilidad soportada por los cambios de esquema
- Manejo de memoria secundaria y persistencia
- Concurrencia y recuperación en caso de fallas con puntos de chequeo
- Distribución soportada por Ethernet y TCP/IP. Cliente-servidor.
- Chequeo e inferencia de tipo
- Tipos son componentes de las clases


# Ejemplo O2

```
class Estudiante inherit Persona
    type tuple (        clase: string,
                        carreraEn: Departamento,
                        especialidadEn: Departamento,
                        inscritoEn: set ( Sección ),
                        boleta: set ( tuple (   nota: character,
                                                nnota: real,
                                                sección: Sección ) ) )
    method              promedioDeNotas: real,
                        cambiarClase: boolean,
                        cambiarCarrera ( nc: Departamento ): boolean
end
class EstudianteDePostgrado inherit Estudiante
    type tuple (        grados: set ( tuple (   colegio: string,
                                                grado: string,
                                                año: integer ) )
                        asesor: Profesor )
end
```

- Manejo de transacciones
- Sin versiones
- Manejador de mensajes
- Lenguajes: CO2 y BasicO2
- Manejo de cluster y de buffer
- Implementado en C bajo Unix (SunOS4.0)
- Manejo de espacio de trabajo
- Indexación basada en objetos complejos y herencia
- No permite cambios de la estructura de la clase. Una clase se puede eliminar si ella no tiene instancias y ninguna otra clase depende de ella

UNIVERSIDAD DE LOS ANDES

# Implementación de los métodos

**method body** edad: **integer in class** Persona

```
{   int a;
    Fecha f;
    f  = today():
    a = f->año – self ->fechaNac ->año;
    if ( f->mes < self ->fechaNac -> mes || ( ( f->mes == self->fechaNac->mes) && ( f->dia < self->fechaNac->dia ) )
            - - a;
    return a;
}
```

UNIVERSIDAD DE LOS ANDES

# Consultas en O2

**select tuple (** nombre: e.nombre.nomPila, apellido: e.nombre.paterno **)**

**from** e **in** Estudiante

**where** e.carreraEn.nombreD = “Ciencias de la computación”

- Declaración de la raíz persistente:

  **name** Estudiantes: **set ( Estudiante )**

- **Una consulta regresa un objeto o un valor**
- **El lenguaje de consulta es un subconjunto del lenguaje de programación, funcional y con cálculo de predicados de primer orden**

UNIVERSIDAD DE LOS ANDES

# ObjectStore

- **Usa las declaraciones de C++ extendido con elementos adicionales para bases de datos.** **{Lamb et al., 1991}.**
- **Constructores de tipos: os_Set**, **os_Bag**, **os_List**
- Tipos básicos, los mismos de C: char, int, long int, unsigned int, float, double, char*
- Declaración de relaciones: tipo atributo **inverse_member** tipo**::**atributo;


# Ejemplo ObjectStore

```
class Persona                                          struct Fecha { int año, mes, dia };
{     char    nss[10];
      struct { char*      nomPila, paterno, materno: } nombre;
      struct { int        número;
      char*   ubic, ciudad, estado, codPostal } dirección;
      Fecha   fechaNac;
      char    sexo;
public:
void          Persona();                               // Constructor vacío
nombre        getNombre();
dirección     getDirección();
Fecha         getFechaNac();
char          getSexo();
int           edad();
};
```

UNIVERSIDAD DE LOS ANDES

# Ejemplo ObjectStore

```
class Estudiante: public Persona
{   char*                    clase;
    Departamento *carreraEn;
    Departamento *especialidadEn;
    os_Set<Sección*>  inscritoEn;
    os_Set<Boleta*>    boleta;
    public:
    void  Estudiante();
    float promedioDeNotas();
    int   cambiarClase();
    int   cambiarCarrera ( Departamento *nd );
}
```

```
struct Boleta { char          nota;
                float         nnota;
                Sección *sección;
              }
struct Grado { char* colegio;
               char* grado;
               int   año;
             }
```


# Ejemplo ObjectStore

```
class EstudianteDePostgrado: public Estudiante
{    os_Set<Grado*>   grados;
     Profesor                    asesor;     }
extern database *bdUniversidad;
class Profesor: public Persona
{    float               salario;
     char*               rango;
     char*               oficinaP;
     os_Set<Departamento*>       perteneceA inverse_member Departamento::miembros;
     Departamento *preside inverse_member Departamento::director;
     os_Set<Estudiante*>         asesora inverse_member Estudiante::asesoradoPor;
     public:
     Profesor(char s[10]) {nss = new(bdUniversidad) char[10]; strcpy(nss, s); };
     void    ascenso();
}
```

UNIVERSIDAD DE LOS ANDES

# Implementación de los métodos

- Manipulación de los objetos en C++.
- Se incluye la instrucción: **foreach**(c, colección)

```
main( )
{   database bdUniversidad = database::open("/bases/univ");
    transaction::begin();
    os_Set<Departamento*> &departamentos =
    os_Set<Departamento*>::create(bdUniversidad);
    departamentos.insert(d);
    transaction::commit();
}
```

UNIVERSIDAD DE LOS ANDES

# ORION

- Objetos complejos, compuestos y multimedia. {W. Kim. Introduction to Object-Oriented Databases. The MIT Press, 1991}.
- Identificador del objeto: compuesto del identificador de la clase y de la instancia, eventualmente uno del lugar
- Encapsulación de datos y comportamiento
- Extensiones de clases
- Herencia simple y múltiple
- Reescritura y encadenamiento tardío
- Complejidad computacional soportada por C
- Extensibilidad que soporta cambios de esquema
- Persistencia y recolector de basura por alcanzabilidad
- Agrupamiento simple por clases y manejo de buffers

UNIVERSIDAD DE LOS ANDES

# ORION

Clase

NombreDeLaClase
Atributos
Superclases
Subclases
Métodos

Atributo

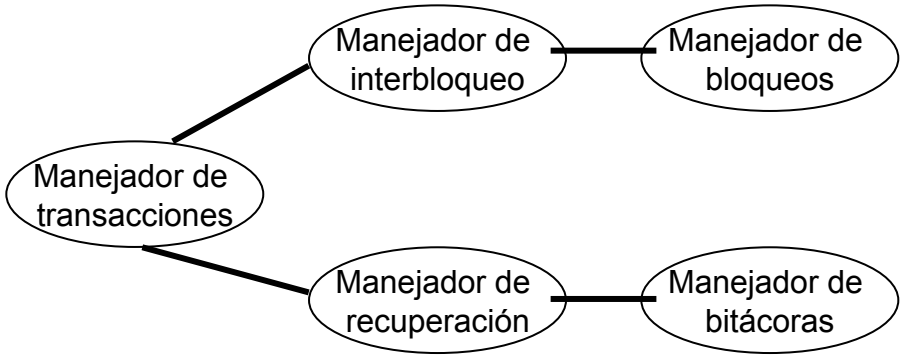
Clase
Dominio
HeredadaDe

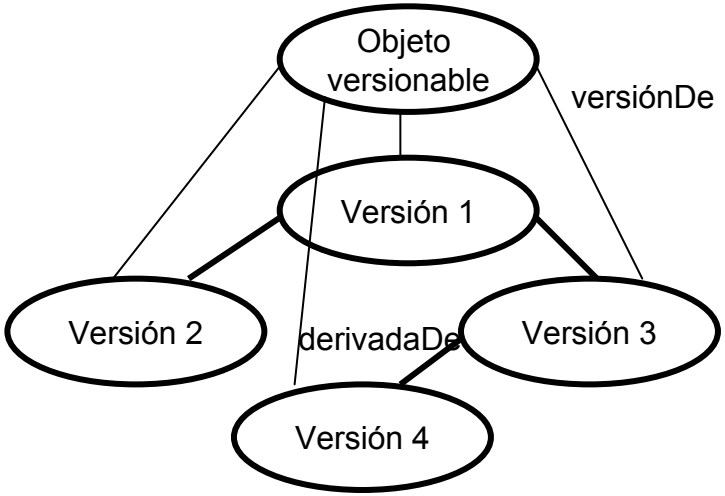
Método

Clase
NombreDelMétodo
Código
HeredadoDe

Clases del sistema

subsistema de transacciones



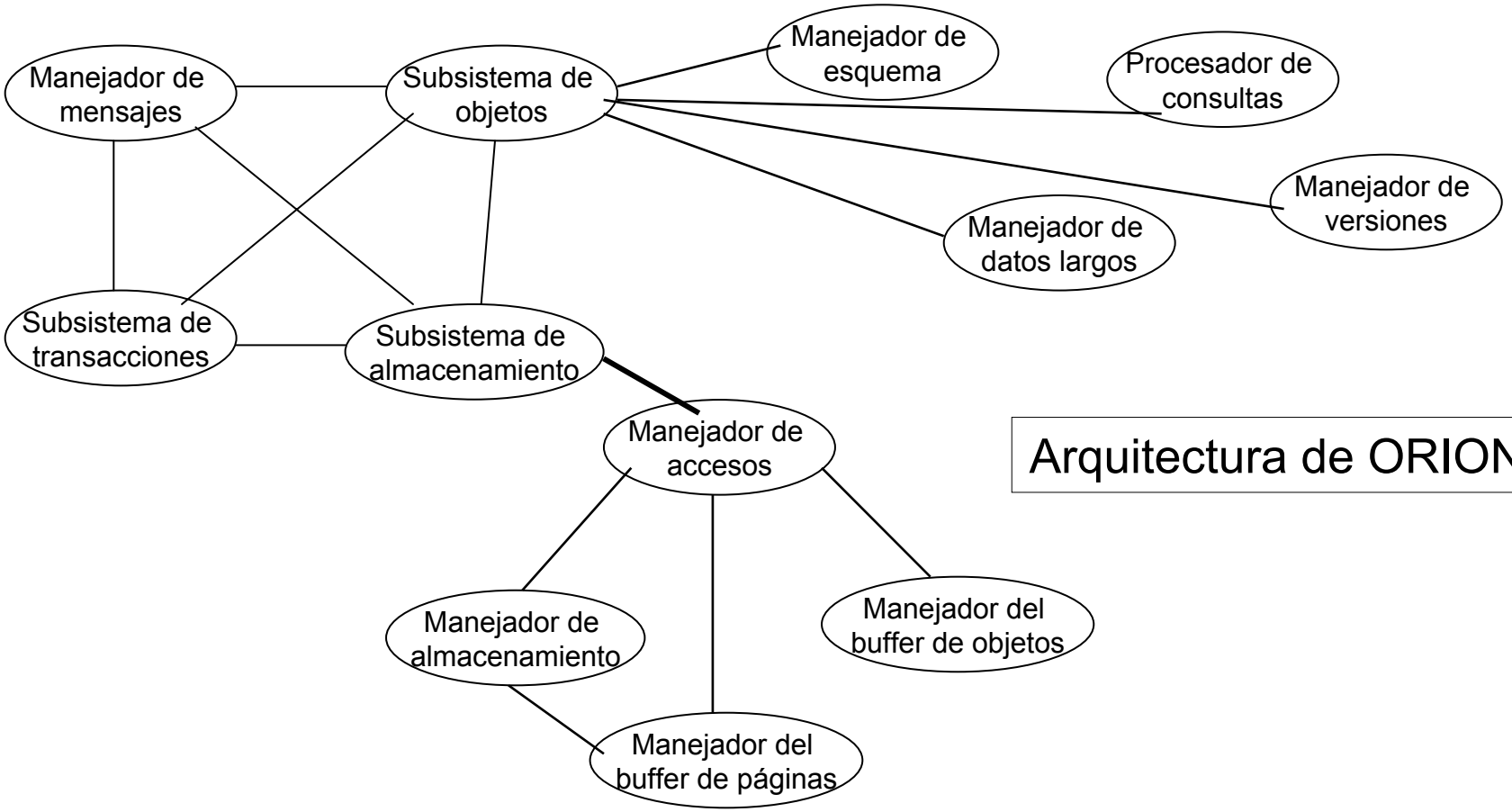
UNIVERSIDAD DE LOS ANDES

# ORION

- Manejo de espacio en disco e indexación asociativa con árboles_B+ sobre la jerarquía de clases
- Manejo de concurrencia y recuperación sólo para fallas no catastróficas
- Distribución solo en la versión 2. TCP/IP.
- Transacciones seriales con dos tablas: de bloqueos y de transacciones bloqueadas
- Versiones: objetos versionados transientes y versiones de trabajo
- Métodos son objetos que se comunican por pase de mensajes
- Lenguaje de definición de datos: Lisp Statice extensión de LISP
- Modificación dinámica del esquema
- Manejo de consultas similar a los SGBDR (algoritmos de lazos anidados)

UNIVERSIDAD DE LOS ANDES

# ORION




Arquitectura de ORION

UNIVERSIDAD DE LOS ANDES

# Versiones ORION

- Versiones transientes:
  - ✓ Puede ser actualizada por el usuario que la creó
  - ✓ Puede ser eliminada por el usuario que la creó
  - ✓ Una nueva versión transiente puede ser derivada de otra, pasando esta última a versión de trabajo
- Versiones de trabajo:
  - ✓ Se considera estable y no puede ser actualizada
  - ✓ Puede ser eliminada por su propietario
  - ✓ Una versión transiente puede ser derivada de una versión de trabajo
  - ✓ Una versión transiente puede ser promovida a versión de trabajo explícitamente por el usuario o implícitamente por el sistema

UNIVERSIDAD DE LOS ANDES

# Consultas ORION

- Simple:

  consultaSimple ::= **select** Lista **from** Rango **where** Cualificación

  **select** :V **from** Vehiculo :V **where** color = ‘azul’ **and** ensambladora ubicación = ‘Valencia’

  **select** :C **from** Compañia :C, :E **is-in** :C divisiones staff, :D **is** :E conduce **where** :E residencia = ‘Puerto Cabello’ **and** :D color = ‘azul’ **and** :D ensambladora ubicación = ‘Valencia’

- Recursivas:

  **select** :X **from** Dispositivo :X, :Y **in** Dispositivo :Y **where** :Y tipo = ‘bus’ **and** :Y nombre = ‘FR-34’ **and** Y **is-in** :X ( **recurse** conexiones componentes * )

- Incluyen métodos:

  **select** :E nombre **from** Empleado :E, Divisiones :D **where** :E **exists** disponibilidad (:D ubicación) ensambladora nombre = ‘Chevrolet’ **and** :D funcion = ‘Fábrica’

UNIVERSIDAD DE LOS ANDES

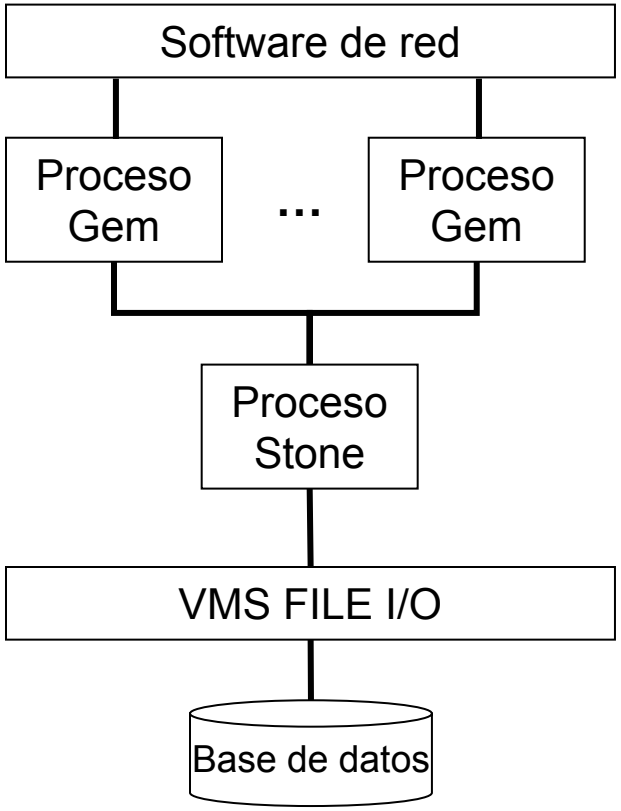
# GemStone

- Objetos complejos, identidad de objetos, encapsulación, tipos y clases como Smalltalk-80. {Maier, D. y Stein, J. *Development and Implementation of an Object-Oriented Database Management System*. Readings in Object-Oriented Database Management Systems, Zdonik, S. and Maier, D. (Eds.) Morgan Kaufman, 1990}.
- Herencia simple
- Reescritura, encadenamiento tardío, completitud computacional y extensibilidad soportada por Smalltalk-80
- Persistencia por alcanzabilidad desde una raíz persistente. Recolector de basura.
- Manejo de memoria secundaria por Stone
- Concurrencia optimista y pesimista
- Recuperación en caso de fallas completo
- Chequeo e inferencia de tipos realizada fuera de Smalltalk-80

UNIVERSIDAD DE LOS ANDES

# Arquitectura GemStone

- Procesos Gem
  - ✓ Compilación y ejecución de código
  - ✓ Control de sesión y autenticación de usuarios
- Proceso Stone
  - ✓ Manejo de memoria secundaria
  - ✓ Control de concurrencia y transacciones
  - ✓ Autorización a nivel de clases y recuperación
  - ✓ Acceso asociativo



UNIVERSIDAD DE LOS ANDES

# Consultas GemStone

- Caminos:
  - ✓ El valor de la expresión de camino $A.L_1.L_2.,,,.L_n$ se define como:
    - Si n = 0 => el valor de la expresión de camino es el valor A
    - Si n > 0 => el valor de la expresión de camino es el valor de $L_n$ dentro de $A.L_1.L_2.,,,.L_{n-1}$ si esto está definido y $L_n$ es una variable de instancia en $A.L_1.L_2.,,,.L_{n-1}$, de lo contrario el valor de la expresión de camino está indefinido
  - ✓ Prefijos de caminos: $A.L_1.L_2.,,,.L_{n-1}$
- Consultas:

  Empleado select:

  { e | e.nombre.apellido = 'Perez' & e.salario > e.trabajaEn.gerente.salario }

  Donde e es una instancia de Empleado

UNIVERSIDAD DE LOS ANDES

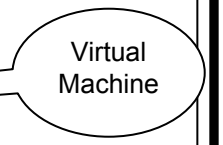
# GemStone

- Sin distribución y sin versiones
- Transacciones seriales y espacio de trabajo por sesión de usuario
- Pase de mensajes
- Lenguajes: OPAL y Alltalk (extensión de Smalltalk) computacionalmente completos
- Soporta conjuntos heterogéneos
- Sin optimizador de consultas
- Manejo de cluster bajo responsabilidad del administrador de BD
- Indexación con árboles_B+ sobre colecciones de objetos
- Lenguajes de implementación: Smalltalk-80 y C
- Modificación dinámica del esquema

UNIVERSIDAD DE LOS ANDES

# FastObjects

- Modelo de objetos: ODMG 3.0 con navegación entre objetos {http://www.poet.com/products}
- Control de concurrencia completo con mecanismos de bloqueo y acceso a nivel de objetos.
- Marco de trabajo con observación y notificación según cambios
- Almacenamiento de tipos de datos desconocidos
- Transporte de objetos vía VM o en los límites de la transacción
- Interfaces de programación: ODMG 3.0 Java API, Java, C++ API
- Procesamiento de consultas: OQL, JDOQL, PtQL, optimizador de consultas avanzado e indexación de texto completo e índices compuestos
- Integración con herramientas UML (Rose y Together J)



UNIVERSIDAD DE LOS ANDES

# FastObjects

- Manejo de autorización a nivel de clases, cifrado de la BD y de las comunicaciones cliente-servidor
- BD sombra, respaldo en línea completo e incremental
- Manejo de fallas inmediato a nivel del servidor y recuperación automática
- Soporte de XML, J2EE, ODBC, SQL Object Factory (modelo relacional)
- Servidor multi-hilos y balance de carga
- Memoria caché activa e indexación de colecciones
- BD en memoria
- Versionamiento al vuelo
- Independencia de los lenguajes de programación y del sistema operativo