



# BASES DE DATOS AVANZADAS

## Tema 3, parte a

### *Modelo de Objetos*

### *Objeto-Relacional*

*Univ. Cantabria – Fac. de Ciencias*

*Francisco Ruiz*



## Agradecimientos

- Este material ha sido preparado con la colaboración de:
  - Coral Calero (Univ. de Castilla-La Mancha)



## Objetivos

- Presentar las diferentes propuestas **objetuales** en la tecnología de bases de datos.
- Conocer los principales **estándares** en este tópico: **SQL:2003** y **ODMG 3.0**.
- Manipular un **SGBD objeto-relacional** mediante el estándar SQL:2003.



## Contenido

- Tercera Generación de BD
  - Debilidades de los SGBD Relacionales
  - Modelos de Objetos
  - Tipos de SGBD con Objetos
- Bases de Datos Objeto-Relacionales
  - Características
  - Aspectos de Objetos en SQL
    - Identidad de Objetos
    - Tipos de Datos en SQL:2003
    - Objetos Grandes
    - Tipos Definidos por el Usuario
    - Tipos Construidos
    - Módulos y Rutinas
    - Métodos
    - Jerarquías de Tablas y Vistas
    - Resumen
    - Ejemplo
  - SGBD Objeto-Relacionales
    - ORACLE
    - Comparación
- Bases de Datos Orientadas a Objetos Puras
  - Características
  - ODMG 3.0
  - SGBD Orientados a Objetos
  - SGBD Objeto-Relacionales vs Orientados a Objetos
- Manejo de Objetos
  - Identidad de objetos
  - Persistencia



## Bibliografía

- Básica
  - Piattini et al. (2006): Tecnología y Diseño de Bases de Datos.
    - Cap. 20.
  - Connolly y Begg (2005): Sistemas de Bases de Datos.
    - Caps. 26 y 28.
- Complementaria
  - Elmasri y Navathe (2007): Fundamentos de Sistemas de Bases de Datos.
    - Caps. 20 y 22.



## Tercera Generación de BD Debilidades de los SGBD Relacionales

- **Representación pobre de las entidades del "mundo real"**: el proceso de normalización generalmente lleva a la creación de relaciones que no corresponden con entidades del "mundo real". La fragmentación de una entidad del mundo del "mundo real", en varias relaciones, es ineficiente llevando a muchos "joins" en el procesamiento de consultas.
- **Sobrecarga semántica**: un solo constructor, la relación, para representar tanto los datos como las relaciones entre ellos.
- Soporte pobre de las restricciones de integridad y de negocio.
- **Estructura de datos que no permite heterogeneidad**: el modelo relacional asume homogeneidad horizontal y vertical. Cada tupla tiene los mismos atributos. Los valores en una columna deben pertenecer al mismo dominio. La intersección de fila y columna debe ser un valor atómico.
- **Operaciones restringidas**: operaciones de conjuntos y orientadas a tuplas, operaciones que provee SQL. Pero SQL no permite definir nuevas operaciones.
- **Manejo difícil de consultas recursivas**: Es difícil manejar consultas sobre relaciones (relationships) que una relación (relation) tiene consigo misma (directa o indirectamente)



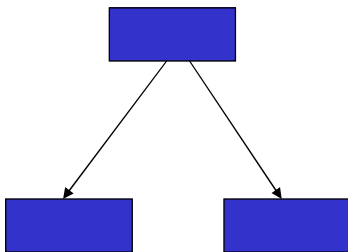
## Debilidades de los SGBD Relacionales

- **Desajuste de impedancia** (en el SQL embebido): debido a que se mezclan diferentes paradigmas de programación. SQL es un lenguaje declarativo que maneja filas de datos, mientras que un lenguaje como C es un lenguaje procedural que permite trabajar con una fila o a la vez. SQL provee tipos de datos que no tienen los lenguajes tradicionales (Interval, Date).
- **Mal soporte de las transacciones de larga duración**: que suelen más comunes para objetos complejos.
- **Evolución de esquemas difícil**: los administradores deben intervenir en el cambio de la estructura de la BD, y los programas que acceden a tales estructuras deben ser modificadas para reflejar estos cambios de estructura.
- **Pocas facilidades para navegar por los datos**: acceso basado en el movimiento entre registros individuales.



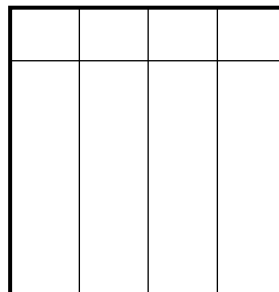
## Tercera Generación de BD

1ª GENERACIÓN



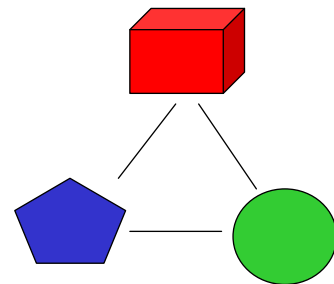
**EN RED**

2ª GENERACIÓN



**RELACIONAL**

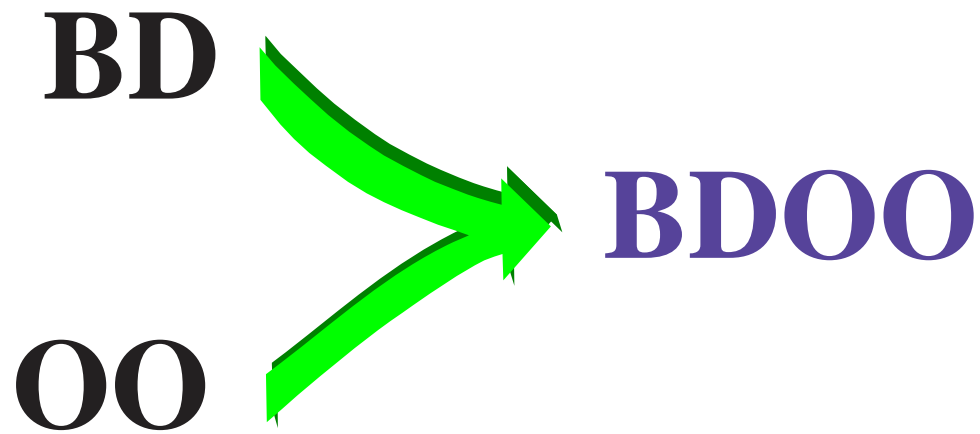
3ª GENERACIÓN



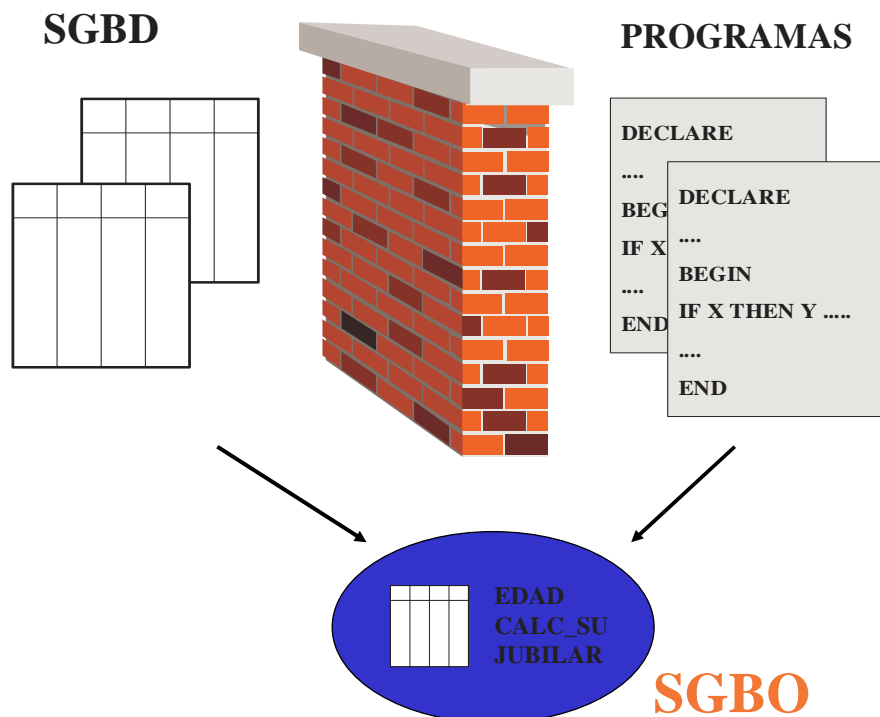
**ORIENTADA  
A OBJETOS**



## Tercera Generación de BD



## Tercera Generación de BD





## Tercera Generación de BD

KENT (1990)

<b>APLICACIONES</b>	<b>Código de aplicación</b>	
<b>SGBD</b>	<b>Estructuras de datos</b>	<b>DBD</b>

Modelo conceptual  
↓  
Modelo de datos de aplicación



## Tercera Generación de BD

KENT (1990)

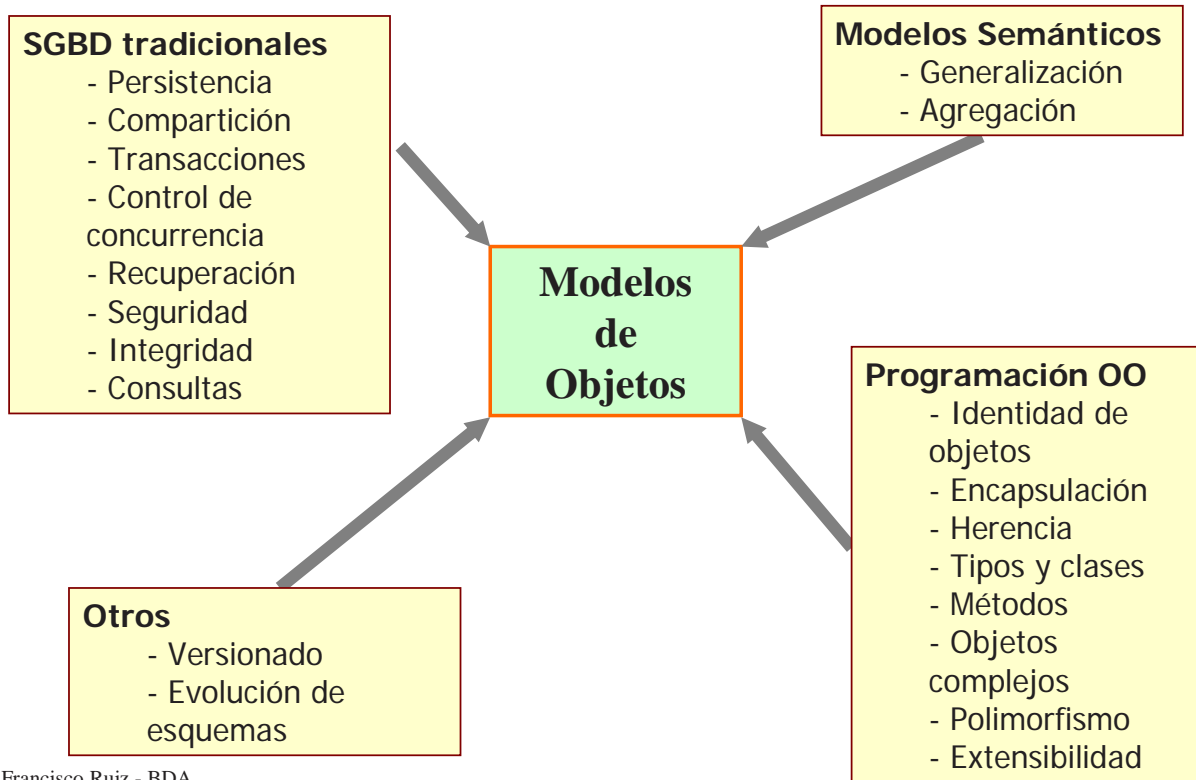
<b>APLICACIONES</b>	<b>Código de aplicación</b>	
<b>SGBO</b>	<b>Operaciones de datos</b>	<b>DBD</b>
	<b>Estructuras de datos</b>	

Modelo conceptual  
Modelo de datos de aplicación



## Tercera Generación de BD

# Modelos de Objetos – orígenes



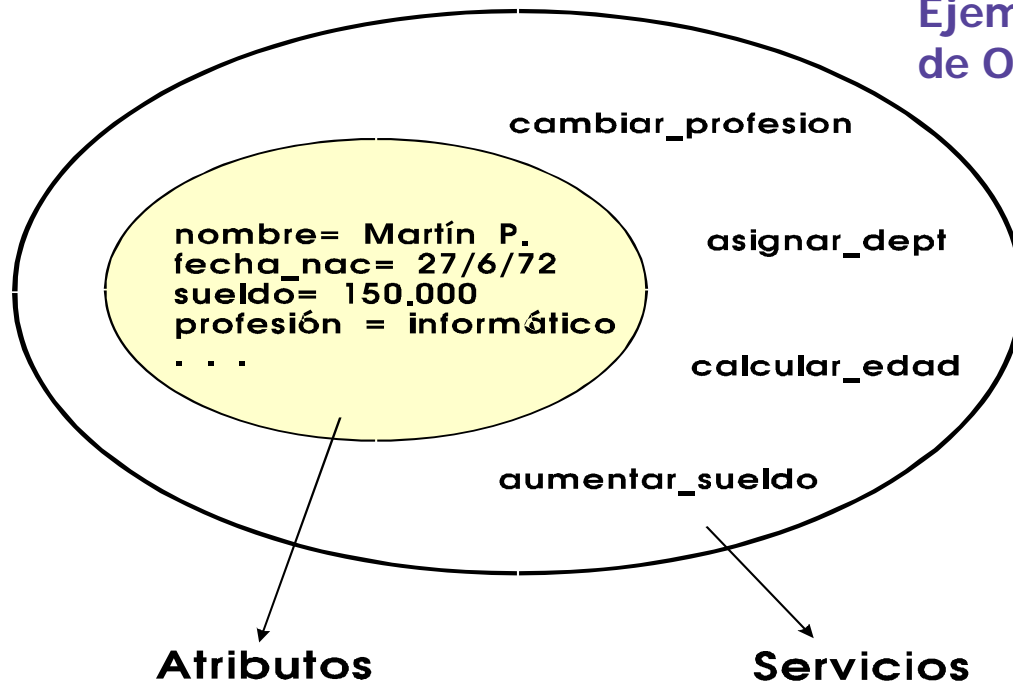
## Tercera Generación de BD

# Modelos de Objetos – conceptos básicos

- **Sistema Orientado a Objetos (OO):**
  - **Conjunto de objetos** que se comunican entre sí mediante **mensajes**.
- **Objeto:**
  - Conceptualmente, es una entidad percibida en el sistema.
  - A nivel de implementación, un objeto se corresponde con un encapsulamiento de un **conjunto de operaciones** (servicios) que pueden ser invocadas externamente y de un **estado** que recuerda el efecto de los servicios.
  - Un objeto se describe por sus propiedades, también llamadas **atributos** -estructura del objeto- y por los **servicios** que puede proporcionar -comportamiento del objeto-.
  - El estado de un objeto viene determinado por los **valores** que toman sus atributos, valores que siempre han de cumplir las **restricciones** impuestas sobre ellos.



## Ejemplo de Objeto



## • Abstracción

- Proceso de identificar los aspectos esenciales e ignorar el resto.
- Implica centrarnos en qué es y qué hace un objeto antes de pensar en cómo implementarlo.
- Tiene dos aspectos fundamentales:
  - **Encapsulación**: un objeto contiene la estructura de datos y el conjunto de operaciones que pueden ser usadas para manipularla.
  - **Ocultamiento**: Los aspectos internos de un objeto están ocultos al exterior. Provee una forma de **independencia de datos**.
- Estos mecanismos suponen una manera de realizar **modularización**: un objeto es una "caja negra" que puede ser construida y modificada independientemente del resto del sistema, respetando que el interfaz externo no cambie.



- **Objeto**

- Entidad identificable de forma unívoca que contiene los atributos que describen el estado de un objeto del “mundo real” y las acciones asociadas con él.
- Los **atributos** describen el **estado** de un objeto. Pueden ser **simples** o complejos. Los atributos **complejos** pueden contener colecciones o referencias (relaciones entre objetos).
- Los **métodos** definen el **comportamiento** del objeto.



- **Características de los Objetos:**

- Poseer un **estado** que puede cambiar.
- Tener una **identidad única**.
  - **Identificador de Objeto (OID)**: Característica especial que lo identifica unívocamente.
- Soportar **interrelaciones** con otros objetos.
- Poseer un **comportamiento**.
- Poder recibir y enviar **mensajes**.

- **Tipo de Objeto:**

- Clasificación de objetos, que define la estructura y el comportamiento de objetos que comparten características comunes.

- **Clase:**

- Implementación de un tipo de objeto. Los objetos son los **"ejemplares"** o **instancias** de las clases.



## Modelos de Objetos – interacciones entre objetos

- **Estáticas:**
  - **Generalización:** los tipos y las clases se organizan en **jerarquías** o retículos de super/subtipos (**super/subclases**) que presentan **herencia** y que corresponden al concepto "**es-un**", en las cuales los subtipos (o subclases) heredan los servicios o atributos de sus ancestros.
  - **Agregación:** permite construir **objetos compuestos** o complejos, corresponde a la noción "**parte-de**".
- **Dinámicas:**
  - **Mensajes:** sirven para que los objetos soliciten la prestación de servicios y entreguen, en su caso, los **resultados** que se obtienen cuando se lleva a cabo un cierto servicio.
  - Un mensaje es una petición desde un objeto (remitente) a otro objeto (receptor) para que el segundo ejecute alguno de sus métodos.



## Modelos de Objetos – polimorfismo y tipos genéricos

- **Polimorfismo:** capacidad de que un mensaje sea interpretado de maneras distintas, según el objeto que lo recibe. Existen dos tipos:
  - **De subclase:** cuando un servicio definido en una clase se redefine en alguna de sus subclases manteniendo el mismo nombre. Entonces un mensaje enviado a un objeto que pertenece a una cierta clase de la jerarquía puede invocar cualquiera de estos servicios, según sea la clase a la que pertenezca el objeto que lo recibe.
  - **De sobrecarga:** utilizando el mismo nombre para servicios distintos, no situados en una jerarquía de generalización.
  - El polimorfismo necesita que la **vinculación** (*binding*) entre el nombre de un servicio y su implementación se establezca en tiempo de ejecución (vinculación dinámica o tardía).
- **Tipo Genérico:** un tipo/clase parametrizable, es decir, una plantilla para construir tipos.



## ARQUITECTURA DE SGBO

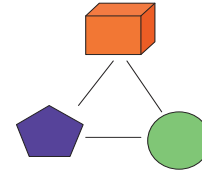
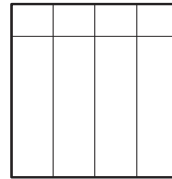
FRIEDMAN (1993)

**Lenguajes**

**SQL**

**OQL**

**MODELO DE DATOS**



**Maquinaria de Implementación**

**INDEXACIÓN**

**GESTIÓN DE MEMORIA**

**AGRUPAMIENTO**

**E/S**



## ARQUITECTURA DE SGBO

FRIEDMAN (1993)

**MODELO OO**

**SGBDR CON FRONTAL OO**

**SGBO PURO**

**MODELO RELACIONAL**

**SGBDR PURO**

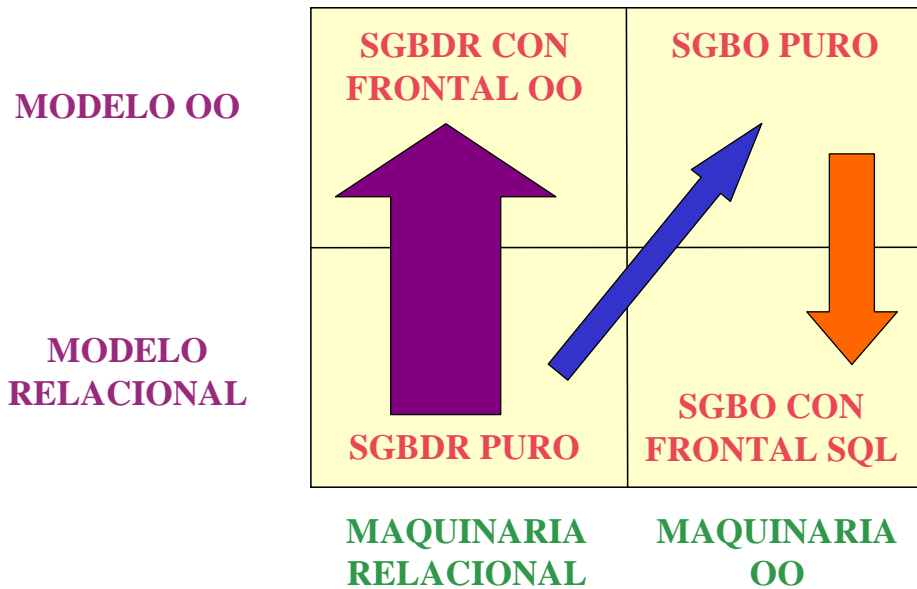
**SGBO CON FRONTAL SQL**

**MAQUINARIA RELACIONAL**

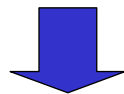
**MAQUINARIA OO**



### EVOLUCIÓN EN EL SECTOR DE LOS SGBD



**SGBDR “EXTENDIDOS”**  
**TERCERA GENERACIÓN**  
**OBJETO-RELACIONAL**

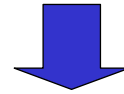


**ORACLE, IBM DB2, MS SQL Server,**  
**INFORMIX, SYBASE, CA, ...**

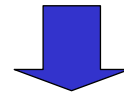


**SQL (1999+2003)**

**SGBD ORIENTADOS A**  
**OBJETOS “PUROS”**  
**SGB DE OBJETOS**



**ObjectStore, O2, ONTOS,**  
**VERSANT, POET, GEMSTONE, ...**



**ODMG 3.0**

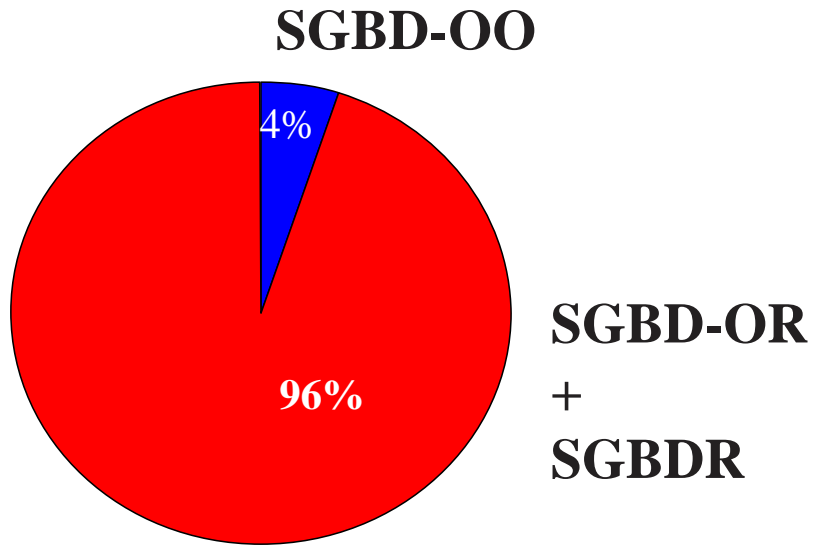


# Tercera Generación de BD Tipos de SGBD con Objetos

## MERCADO SGBD en 2003

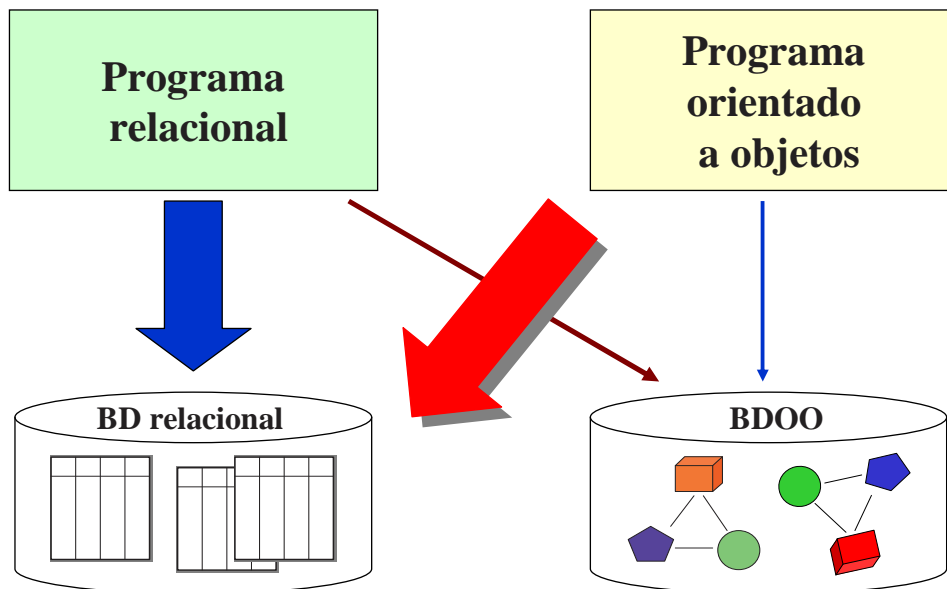
14200 millones \$

ORACLE	39'8%
IBM	31'3%
Microsoft	12'1%



# Tercera Generación de BD Tipos de SGBD con Objetos

## TENDENCIA A LA INTEGRACIÓN





## MANIFIESTO DE LOS SGBD 3ª GENERACIÓN

STONEBRAKER et al. (1990)

**1 principio:** *“Además de los servicios tradicionales de gestión de datos, los SGBD-3G proporcionarán gestión de objetos y reglas más ricos”*

- 1.1 Un SGBD-3G debe tener un rico sistema de tipos
- 1.2 La herencia es aconsejable
- 1.3 Las funciones (procedimientos y métodos) y el encapsulamiento son aconsejables
- 1.4 Los OID's para los registros deberían asignarse por el SGBD sólo si no se dispone de una clave primaria
- 1.5 Las reglas se convertirán en una característica primordial de los sistemas futuros



## MANIFIESTO DE LOS SGBD 3ª GENERACIÓN

STONEBRAKER et al. (1990)

**2 principio:** *“Los SGBD-3G deben subsumir a los SGBD-2G”*

- 2.1 Tener un lenguaje de acceso declarativo y de alto nivel
- 2.2 Dos formas de especificar colecciones: enumeración de sus miembros o mediante un lenguaje de consultas
- 2.3 Las vistas deben ser actualizables
- 2.4 Los indicadores de resultado no deben aparecer en los datos



## MANIFIESTO DE LOS SGBD 3ª GENERACIÓN

STONEBRAKER et al. (1990)

**3 principio:** *“Los SGBD-3G deben estar abiertos a otros subsistemas”*

- 3.1 Los SGBD-3G deben ser accesibles desde múltiples lenguajes de alto nivel
- 3.2 Soportar la persistencia de variables
- 3.3 El SQL es una forma **intergaláctica** de expresión de datos
- 3.4 Las consultas y las respuestas resultantes deben ser el nivel más bajo de comunicación entre un cliente y un servidor



- **Ventajas:**
  - Resuelven muchas de las debilidades de BD relacionales ya conocidas.
  - Reutilización y compartición.
  - Mejora significativa de la productividad.
  - Preservar el significativo cuerpo de conocimientos y experiencia alcanzado con las BD relacionales.
- **Inconvenientes:**
  - Mayor complejidad.
  - Mayores costes.
  - Los partidarios del modelo relacional creen que la simplicidad y pureza de éste se ha perdido.
  - Todo el esfuerzo asociado con la extensión objeto-relacional puede que sólo sea útil, al final, para un porcentaje muy pequeño de las aplicaciones.
  - Los puristas de la OO siguen rechazando cualquier idea de extensión (evolución).
  - Ahora el SQL se ha vuelto extremadamente complejo (SQL-2003 ocupa más de 2000 páginas).



## Aspectos de Objetos en SQL

- Recordemos las partes del estándar SQL actual que incluyen aspectos directamente relacionados con la extensión objeto-relacional:
  - 1- **Framework**: Introducción con el marco de trabajo conceptual general.
  - 2- **Foundation**: Define nuevos tipos de datos y constructores, tipos definidos por el usuario, rutinas definidas por el usuario, etc., incluyendo la sintaxis detallada en SQL.
  - 4- **Persistent Stored Modules (SQL/PSM)**: Documenta los procedimientos almacenados, tanto los expresados en SQL como los externos en otros lenguajes. Aporta la completitud computacional necesaria para escribir rutinas (estructuras de control, etc.).
  - 11- **Information and Definition Schemas (SQL/Schemata)**: Establece las estructuras y contenido del DEFINITION\_SCHEMA, es decir, los metadatos internos.



## Aspectos de Objetos en SQL

- Añadidos objetuales en SQL:1999 y SQL-2003
  - Identidad de objetos (**Identity column** type)
  - Objetos grandes (**Large Objects**, LOBs)
    - Binarios (**BLOB**)
    - Carácter (**CLOB**)
  - Tipos Definidos por el Usuario (**User-Defined Types**, UDTs)
    - Tipos distintos (**Distinct types**)
    - Tipos estructurados (**Structured types**)
  - Constructores de tipos
    - Tipos de filas (**Row types**)
    - Tipos por referencia (**Reference types**)
  - Tipos colección (**Collections**)
    - Vectores (**Arrays**)
    - Multiconjuntos (**Multisets**)
  - Funciones y procedimientos (y métodos) definidos por el usuario
    - **Functions, Procedures**
  - Jerarquías de tablas y de vistas
    - **Sub/Supertables, Sub/Superviews**



## Aspectos de Objetos en SQL

- Los principales **beneficios de la Extensión Objeto-Relacional** son:
  - **Extensibilidad**: Capacidad de ampliar el sistema de tipos para dar soporte a las nuevas necesidades de las aplicaciones.
    - Nuevos tipos de datos que representen mejor el dominio de la aplicación
    - Nuevas operaciones para soportar el comportamiento de los tipos
  - **Poder de expresión**: Necesidad de soportar objetos y relaciones complejas.
  - **Reusabilidad**: Capacidad de compartir librerías de tipos existentes.
  - **Integración**: de los modelos relacional y objetos en un solo lenguaje.
  - Nuevas **consultas más potentes**: recursivas, multimedia, etc.



## Aspectos de Objetos en SQL

### Identidad de Objetos

- Una tabla base puede incluir una **columna identidad**.
  - Desempeñan una funcionalidad similar al **OID**.
  - El tipo de la columna identidad debe ser un **numérico exacto** con escala empezando en 0 (integer, etc.).
  - Cada columna identidad tiene los siguientes valores:
    - Valor inicial, Incremento, Valor máximo, Valor mínimo, y Opción de ciclo.
  - Al añadir una fila a una tabla, el valor de la columna identidad es **generado** siempre por el sistema (**GENERATED ALWAYS**) o sólo en caso de que el usuario no lo facilite (**GENERATED BY DEFAULT**).

```
CREATE TABLE piezas (  
    num_pieza INTEGER  
    GENERATED ALWAYS AS IDENTITY (  
        START WITH 1  
        INCREMENT BY 1  
        MINVALUE 1  
        MAXVALUE 10000  
        NO CYCLE),  
    descripcion VARCHAR (100),  
    cantidad INTEGER )
```



## Clasificación de los Tipos de Datos en SQL-2003

- **Predefinido** (predefined)
  - Numéricos, Carácteres, Booleano, DateTime, Intervalos, XML
- **Construido** (constructed)
  - **Atómico**
    - **Tipo Referencia** (Reference type)
  - **Compuesto**
    - **Colecciones**
      - Vector (array), Multiconjunto (multiset)
    - **Tipo Fila** (row type)
- **Definido por el Usuario** (user-defined)
  - **Distinto** (distinct)
  - **Estructurado** (structured)



## Objetos Grandes

- **Large Object (LOB)**
  - Tipos de datos predefinidos de longitud variable con gran capacidad de almacenamiento (hasta gigabytes).
  - Dos tipos LOB:
    - **Binarios (BLOB):** BINARY LARGE OBJECT (audio, imagen, vídeo)
    - **Carácter (CLOB):** CHARACTER LARGE OBJECT (texto)
  - Se mantienen directamente en la BD, no en ficheros externos.
  - El tamaño máximo se puede indicar en la definición (Kb, Mb o Gb).

```
CREATE TABLE libro (  
    título    VARCHAR (200),  
    id_libro  INTEGER,  
    resumen  CLOB (32K),  
    texto_lib CLOB (20M),  
    película BLOB (2G)  
);
```



## Objetos Grandes

- Los tipos **LOB** pueden ser insertados, actualizados y recuperados como cualquier otro tipo. Para ello es necesario definir buffers de tamaño suficiente.
- Los **LOCATORS** facilitan la manipulación de los LOBs grandes (GB) por las aplicaciones.
- Los LOBs están **excluidos** de algunas operaciones y restricciones:
  - GREATER THAN, LESS THAN
  - PRIMARY KEY, FOREIGN KEY, UNIQUE
  - GROUP BY, ORDER BY
  - UNION, INTERSECT, EXCEPT
  - JOINS (como columnas del Join)
- Algunas **operaciones soportadas** por los LOBs:
  - Recuperar un valor (o parte de el)
  - Reemplazar un valor
  - Predicado LIKE
  - Concatenación
  - Y algunas **funciones**: SUBSTRING, POSITION, LENGH, TRIM



## Tipos Definidos por el Usuario

- **User-Defined Type (UDT)**
  - Junto con los métodos que tienen asociados, aportan la **funcionalidad de las clases** de objetos en programación OO.
    - Entonces, *¿para que siguen siendo necesarias las tablas?*
    - Para dar persistencia a los UDTs.
  - Son de dos tipos:
    - **Distintos (distinct)**
    - **Estructurados (structured)**.
  - Una **definición de un UDT** consta de:
    - Una o más **definiciones de atributos**.
    - Cero o más **declaraciones de métodos** (rutinas).
    - Cero o más **declaraciones de operadores**.
  - Los **métodos** más significativos son:
    - **Observador** (observer), para retornar el valor actual.
    - **Mutador** (mutator), para asignar valor. **SET()**
    - **Constructor** (constructor), para crear nuevas instancias. **GET()**



### • Tipos Distintos (DISTINCT)

- Son UDTs basados en algún tipo predefinido.
- Se basan en el **renombrado de tipos**:
  - Comparten los mismos valores y la misma representación que el tipo predefinido.
  - Tienen distinto comportamiento.
  - El tipo distinto no es comparable con el tipo fuente.
- Ejemplo:

```
CREATE TYPE dni AS VARCHAR(9) FINAL;
```
- Operaciones definidas sobre los tipos distintos:
  - Operadores de comparación
  - Operadores de CAST (conversión de tipo)
  - Métodos y funciones
  - No existe herencia entre tipos distintos



**SQL-92 no es fuertemente tipado**



```
CREATE TABLE Sala (  
  IdSala CHAR (10),  
  LongSala LONG,  
  AnchoSala INTEGER,  
  AreaSala INTEGER,  
  PerimSala INTEGER));
```

```
UPDATE Sala  
SET AreaSala=LongSala;
```

!!!Misma representación (INTEGER)  
PERO distinto COMPORTAMIENTO!!!



### SQL:1999/2003 **Sí** es fuertemente tipado

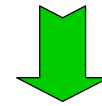
```
CREATE TYPE tiposala
  AS CHAR(10) FINAL;

CREATE TYPE tipometros
  AS INTEGER FINAL;

CREATE TYPE tipometroscuad
  AS INTEGER FINAL;

CREATE TABLE Sala (
  IdSala      tiposala,
  LongSala    tipometros,
  AnchoSala   tipometros,
  AreaSala    tipometroscuad,
  PerimSala   tipometros);
```

```
UPDATE Sala
SET AreaSala=LongSala;
```



Error

```
UPDATE Sala
SET AnchoSala=LongSala;
```



Correcto



### • Tipos Estructurados (Structured types)

- Son UDTs con nombre, formados por una **lista de atributos**.
- Cada atributo tiene un tipo de dato y un valor por defecto.
- Cada valor de un tipo estructurado contiene los valores de sus atributos.
- Los valores de los atributos están encapsulados => sólo son accesibles invocando su función observador (**observer**).

### • Herencia:

- Un tipo estructurado puede **heredar** (ser definido a partir de) otro tipo estructurado.
  - **Subtipo** (subtype)
  - **Supertipo** (supertype)
- Un subtipo hereda todos los atributos de su supertipo y puede tener atributos adicionales propios.
- **Sustitución**: un subtipo puede aparecer en cualquier lugar donde alguno de sus supertipos está permitido.



- **Tipos Estructurados no instanciables**

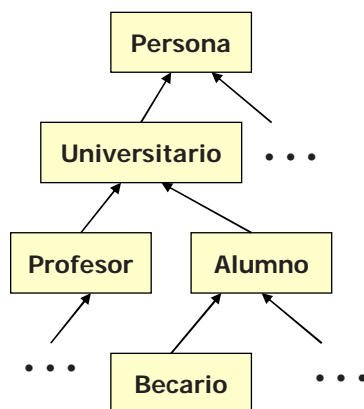
- Son supertipos que no pueden admitir instancias salvo a través de alguno de sus subtipos (no admiten el método constructor).
- Aportan la funcionalidad de las **clases abstractas**.

```
CREATE TYPE person AS
  (name VARCHAR (30),
   address address,
   sex CHAR (1))
NOT INSTANTIABLE NOT FINAL
```



- **Generalización de Tipos**

- Es posible crear jerarquías de tipos de múltiples niveles.



```
CREATE TYPE persona ... NOT FINAL
CREATE TYPE universitario UNDER persona ... NOT FINAL
CREATE TYPE profesor UNDER universitario ... NOT FINAL
CREATE TYPE alumno UNDER universitario ... NOT FINAL
CREATE TYPE becario UNDER alumno ... FINAL
```

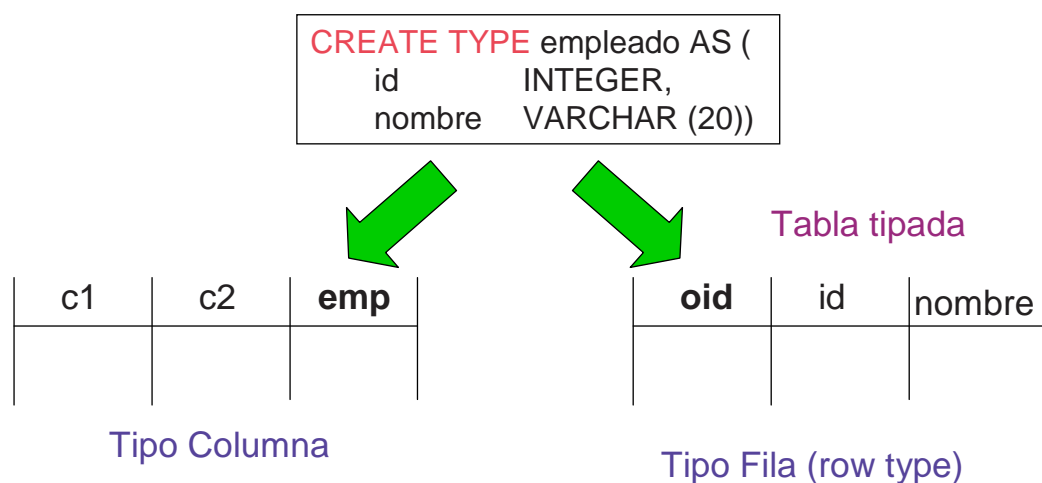


## Tipos Definidos por el Usuario - estructurados

- Los **Tipos Estructurados** se pueden usar como:
  - **Tipos de columnas:**
    - Permiten modelar atributos de las entidades
    - Ej: dirección, coordenadas, fecha, ...
    - Permiten mejorar el soporte para objetos multimedia (SQL/MM)
  - **Tipos de fila (Row types):**
    - Permiten modelar entidades con relaciones y comportamiento
    - Ej: empleado, departamento, alumno...
    - Permiten mejorar el soporte para objetos de negocio
    - Una tabla declarada con un tipo estructurado como su tipo fila se llama **Tabla Tipada (Typed Table)**.
      - Las columnas de una tabla tipada corresponden en nombre y tipo a los atributos del tipo estructurado declarado como su tipo fila.



## Tipos Definidos por el Usuario - estructurados



```
CREATE TABLE departamento (
  cdep VARCHAR(3) PRIMARY KEY,
  director empleado,
  ...
);
```



## Tipos Definidos por el Usuario - estructurados

Se pueden usar en cualquier sitio donde se pueda usar un **tipo predefinido**:

- Atributos de otros tipos estructurados
- Parámetros de funciones, métodos y procedimientos
- Variables SQL
- Dominios o columnas



```
CREATE TYPE dirección AS (
  calle      CHAR (30),
  ciudad    CHAR (20),
  provincia  CHAR (2),
  codigoP    CHAR (5));

CREATE TYPE bitmap AS BLOB FINAL;

CREATE TYPE finca AS (
  propietario REF (persona),
  precio      dinero,
  habitaciones INTEGER,
  tamaño      DECIMAL (8,2),
  ubicación   dirección,
  imagen      bitmap) NOT FINAL
```

También se pueden usar para definir **tablas y vistas (tipadas)**



```
CREATE TABLE propiedades OF finca
```



## Tipos Definidos por el Usuario - estructurados

### • Inserción

```
BEGIN
  DECLARE u libro_udt;           → Declaración de variable
  SET u libro();                 → Constructor
  SET u = u.titulo ('La Perla');
  SET u = u.precio_compra(10.00);   Funciones mutador
  SET u = u.precio_venta(20.00);
  INSERT INTO libro VALUES (u,'111'); → Inserción típica
END;
```

- ¿Por qué no usar directamente una asignación así?
  - u.title = 'La Perla';
  - Porque se viola el encapsulamiento (sólo los métodos pueden acceder a los atributos).



- **Creación e Inserción**

```
INSERT INTO libro VALUES  
(NEW libro_udt('Orca', 12.50, 14.50 ), '222');
```

- NEW invoca al método constructor e implícitamente a los métodos mutadores.

- **Selección**

```
SELECT libro_columna, codigo  
FROM libro;
```

```
SELECT codigo, libro_columna.titulo()  
FROM libro  
WHERE libro_columna.precio_venta() > 20;
```

Funciones observador



- Las **tablas tipadas** o referenciables están basadas en UDTs.

- No se les pueden agregar atributos adicionales

- Ejemplo:

```
CREATE TABLE libro OF libro_udt;
```

- No se requieren las funciones mutador ni observador:

```
INSERT INTO libro VALUES('King Kong en China',30,40);
```

```
SELECT * FROM book;
```

```
SELECT titulo, beneficio() AS beneficio  
FROM libro WHERE titulo LIKE '%día%';
```



## Aspectos de Objetos en SQL

# Tipos Construidos - filas

### • Tipo Fila (row type)

- Secuencia de uno o más campos (**fields**).
- **Campo**: Un par (nombre de campo, tipo de dato).
- Se utiliza la nomenclatura de punto.
- Son el tipo asociado a una tabla tipada.

```
CREATE TABLE empleado (  
  id_emp INTEGER,  
  nombre ROW (  
    n_de_pila VARCHAR(30),  
    apellido VARCHAR(30) ),  
  dirección ROW (  
    calle VARCHAR(50),  
    ciudad VARCHAR(30),  
    provincia CHAR(2),  
    salario REAL );  
  
SELECT E.nombre.apellido  
FROM empleado E;
```



## Aspectos de Objetos en SQL

# Tipos Construidos - referencia

### • Tipo Referencia (reference type)

- Sus valores referencian (apuntan) a una fila de una tabla tipada (similar a punteros en C).
- El tipo fila de la tabla referenciada se llama **tipo referenciado**.
- Los tipos referencia se pueden utilizar donde los demás tipos.

#### Tres Maneras de Asignar Valores:

- **Generados por el usuario** REF USING <tipo predefinido>  
CREATE TYPE propiedades as (.....)  
NOT FINAL **REF USING INTEGER**
- **Generados por el sistema** REF IS [<columna>] SYSTEM GENERATED  
**REF IS SYSTEM GENERATED**
- **Derivados de un conjunto de atributos** REF (<atributos>)  
CREATE TYPE persona as (nss INTEGER, nombre CHAR(20), ...)  
NOT FINAL **REF (nss)**



```
CREATE TYPE finca AS (  
    propietario REF (persona),  
    precio dinero,  
    habitaciones INTEGER,  
    tamaño DECIMAL (8,2),  
    ubicación dirección,  
    imagen bitmap) NOT FINAL  
  
CREATE TABLE propiedades OF finca  
(REF IS oid USER GENERATED)
```

En una **tabla tipada** se añade **una columna más** (columna auto-referenciada) para definir el valor REF de la fila (**OID**)



- Tipos Referencia vs Integridad Referencial
  - Las referencias **no tienen la misma semántica** que la restricción de integridad referencial.
  - La integridad referencial implica una **dependencia de inclusión** y las referencias no.
  - La integridad referencial no soporta la noción de **tipado fuerte**: se comprueba que el valor es el mismo pero no se obliga a que ambos tipos de datos sean exactamente el mismo.



## Aspectos de Objetos en SQL

### Tipos Construidos - referencia

#### Utilidad de los Tipos Referencia

#### Combinaciones sin necesidad de usar JOIN

```
SELECT f.precio, f.propietario->nombre
FROM finca f
WHERE ciudad="Madrid"
```

#### Llamada a métodos

```
SELECT e.nombre, e.departamento->num_empleados()
FROM empleado e
```

#### Obtención de datos a los que se referencia

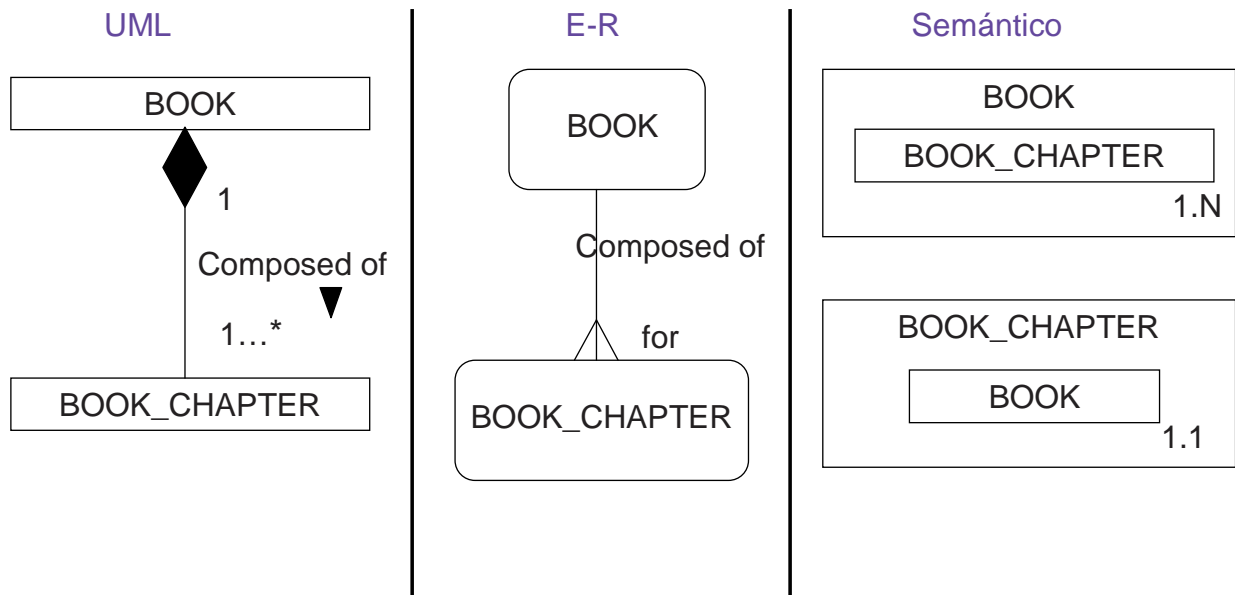
```
SELECT e.nombre, Deref(e.departamento)
FROM empleado e
WHERE e.nombre="J. Pérez"
```



## Aspectos de Objetos en SQL

### Tipos Construidos - referencia

#### • Ejemplo – Libro vs Capítulo





- **Ejemplo** – Libro vs Capítulo

- Tabla Libro

```
CREATE TABLE libro OF libro_udt;
```

- Tabla Capítulo

```
CREATE TABLE capitulo (  
  libro_ref REF(libro_udt) NOT NULL  
  SCOPE libro REFERENCES ARE CHECKED  
  ON DELETE CASCADE,  
  resumen VARCHAR(100),  
  fcontenido VARCHAR(2000),  
  nro_capitulo INTEGER  
);
```



- **Ejemplo** – Libro vs Capítulo

- Añadir Capítulo

```
INSERT INTO capitulo SELECT mi_ref, 'Historia de Cantabria',  
  '.. y los romanos ocuparon el paso ...', 1  
FROM libro WHERE titulo = 'Historia Universal Extraña';
```

- Listar el título de cada libro y el contenido de sus capítulos ordenados por número de capítulo

```
SELECT libro_ref → titulo, contenido  
FROM capitulo  
ORDER BY libro_ref → titulo, nro_capitulo;
```



- **Ejemplo** – Libro vs Capítulo

- Listar el resumen del primer capítulo de cada libro, su título y su beneficio, ordenados por título

```
SELECT resumen AS resumen, (libro_ref → titulo) AS titulo,  
       (libro_ref → beneficio()) AS beneficio  
FROM capitulo  
WHERE nro_capitulo = 1  
ORDER BY titulo;
```



- **Colecciones (collections)**

- Cero o más elementos de un tipo especificado, conocido como **tipo elemento**.
- Permiten representar **columnas multivaluadas**, es decir, que en cada celda de la tabla puede haber más de un valor (violando la primera forma normal).
- También permiten generar **tablas anidadas**, es decir, tablas donde alguna columna contiene otra tabla.

- Hay dos clases:

- **Vector (Array)**
  - Colección **ordenada** de valores no necesariamente diferentes, cuyos elementos se referencian por su **posición ordinal**.
- **Multiconjunto (Multiset)**
  - Colección **no ordenada** de valores no necesariamente distintos.



## Aspectos de Objetos en SQL

# Tipos Construidos - colecciones

### • Vectores (Arrays)

#### ■ Características:

- Diferencia entre la longitud máxima y la actual (como el CHAR VARYING)
- Pueden definirse sobre cualquier tipo (excepto array).
- Pueden definirse arrays en “cualquier sitio”.

#### ■ Operaciones:

- Acceso a los elementos por un número ordinal
- Cardinalidad (nº de elementos)
- Comparación
- Constructores
- Asignación
- Concatenación ( || )
- Conversión explícita (CAST)
- Facilidades de selección declarativa sobre arrays (por contenido o por posición, transformación implícita de un array en una tabla...)



## Aspectos de Objetos en SQL

# Tipos Construidos - colecciones

### Ejemplo Vectores

```
CREATE TABLE libro (  
  título      VARCHAR (200),  
  id_libro    INTEGER,  
  autores     VARCHAR (15) ARRAY [20],  
  resumen     CLOB (32K),  
  texto_libro CLOB (20M),  
  película    BLOB (2G));
```

```
INSERT INTO libro (título, id_libro, autores)  
VALUES (“A guide to the SQL Standard”, 15, ['Date', 'Darwen'])
```

```
SELECT id, autores[1] AS nombre  
FROM libro
```



## Aspectos de Objetos en SQL

# Tipos Construidos - colecciones

### • Multiconjuntos (Multisets)

#### ■ Características:

- Colección de elementos homogéneos, sin orden, y que permite elementos repetidos.
- Permiten definir columnas que contengan colecciones de elementos (= > no cumplen con la 1FN).
- Se pueden definir por enumeración o por una expresión de consulta:
  - `MULTISET [1, 2, 3, 4]`
  - `MULTISET (SELECT nombre FROM provincias)`

#### ■ Operaciones:

- Eliminar duplicados (`SET`)
- Unión (`UNION`)
- Intersección (`INTERSECT`)
- Diferencia (`EXCEPT`)
- Convertir en tabla virtual (`UNNEST`)
- Extraer un elemento (`ELEMENT`)



## Aspectos de Objetos en SQL

# Tipos Construidos - colecciones

### Ejemplo

## Multiconjuntos

```
CREATE TABLE Película (  
  título          VARCHAR (200),  
  id_película    INTEGER,  
  actores        VARCHAR (15) MULTISET,  
  resumen        CLOB (32K),  
  pelicula       BLOB (2G));
```

```
INSERT INTO Película (título, id_película, actores)  
VALUES ("Top Gun", 3, Multiset ['Tom Cruise', 'Kelly McGillis', 'Val Kilmer']);
```



## Aspectos de Objetos en SQL

# Módulos y Rutinas

### • Módulos (Modules)

- Contienen una o varias **rutinas (routines)**.
- Los que forman parte de un esquema SQL se llaman **Módulos de Servidor SQL (SQL-server module)**.
- Las rutinas pueden estar escritas en SQL o no, y ser invocadas desde SQL o desde otro lenguaje (cuatro posibilidades).
- Las **rutinas invocables desde SQL** son de tres clases (**SQL-invoked routine**):
  - **Funciones SQL (SQL functions)**.
    - **Métodos (methods)**: funciones definidas para un UDT específico.
  - **Procedimientos SQL (SQL procedures)**.
  - Funciones y procedimientos externos (no escritos en SQL).
- Un "SQL-server module" sólo puede incluir "SQL-invoked routines".
- **Sobrecarga**: dos o más rutinas con el mismo nombre.
  - Se ejecuta la rutina cuya signatura encaje mejor con los argumentos de la invocación.



## Aspectos de Objetos en SQL

# Métodos

### • Método (Method)

- Una **función SQL** asociada a un UDT.
- La **signatura** se define en el esquema dentro de la definición del UDT (**CREATE TYPE ... METHOD ...**).
- El **cuerpo** se define aparte, bien en el esquema directamente o en un módulo (**CREATE METHOD ... BEGIN ... END**).

```
CREATE TYPE empleado AS (  
  id          INTEGER,  
  nombre     VARCHAR (20)  
  salario_base DECIMAL (9,2)  
  bono       DECIMAL (9,2)  
  INSTANTIABLE NOT FINAL  
  METHOD salario() RETURNS DECIMAL (9,2);  
  
  CREATE METHOD salario(...) FOR empleado  
  BEGIN  
    ..... RETURN (coste – gastossociales);  
  END;
```

Se **invocan** utilizando la notación "punto":

```
SELECT e.salario()  
FROM empleado e
```



## Aspectos de Objetos en SQL

# Métodos

- Un método puede ser:
  - **Original (original)**, si es heredado sin más del supertipo)
  - **Sobrecargado (overriding)**, si se define en el subtipo con el mismo nombre que un método heredado de algún supertipo).
    - La asignación se hace mediante **vinculación dinámica**.

```
CREATE TYPE empleado AS (  
    id          INTEGER,  
    nombre     VARCHAR (20),  
    salario_base DECIMAL (9,2),  
    bono       DECIMAL (9,2))  
INSTANTIABLE NOT FINAL  
METHOD salario() RETURNS DECIMAL (9,2);  
  
CREATE TYPE manager UNDER empleado AS (  
    stock_option INTEGER)  
INSTANTIABLE NOT FINAL  
OVERRIDING METHOD salario() RETURNS DECIMAL (9,2),  
METHOD original1() RETURNS INTEGER
```



## Aspectos de Objetos en SQL

# Métodos

- **Métodos vs resto de Funciones**
  - Los métodos están asociados a **un solo UDT**; el resto de funciones funciones no.
  - El UDT de un método es el tipo del **argumento distinguido** para el método (el primero, que no se declara); el resto de funciones no tiene argumentos distinguidos.
  - La **vinculación** o asignación de una entre varias funciones cuando existen varias posibles a elegir en una invocación, es diferente:
    - **Métodos: dinámica** (en tiempo de **ejecución**).
    - **Resto de funciones: estática** (en tiempo de **compilación**).
  - Un método debe ser almacenado en el **mismo esquema** en el cual se almacena la definición de su **UDT**; las funciones no están limitadas a un determinado esquema.
  - Ambos pueden ser escritos en SQL gracias a la completitud computacional que proporciona SQL/PSM.



## Aspectos de Objetos en SQL

# Jerarquías de Tablas y Vistas

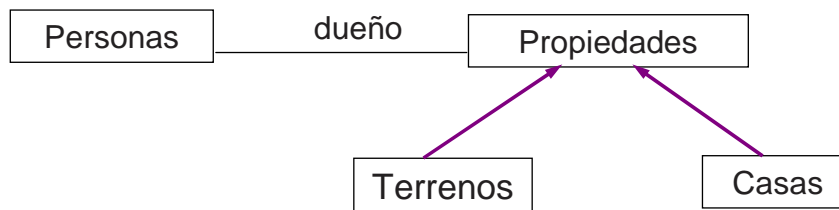
- Una tabla TB1 es **subtabla** de otra TB2 si
  - TB1 es una tabla tipada con tipo estructurado asociado TS1.
  - TB2 es una tabla tipada con tipo estructurado asociado TS2.
  - TS1 es un subtipo de TS2.
- Una tabla TB1 es **supertabla** de otra TB2 si
  - TB2 es subtabla de TB1.
- Las subtablas **heredan** las propiedades de las supertablas:
  - Atributos, restricciones, disparadores, métodos, referencias, ...
- Las **subvistas** (subviews) y **supervistas** (subviews) se definen de forma equivalente.



## Aspectos de Objetos en SQL

# Jerarquías de Tablas y Vistas

## Ejemplo



```
CREATE TYPE tpersona AS .... NOT FINAL
CREATE TYPE tpropiedad AS (dueño REF (tpersona), .... NOT FINAL
CREATE TYPE tterreno UNDER tpropiedad AS ....FINAL
CREATE TYPE tcasa UNDER tpropiedad AS .... FINAL

CREATE TABLE Personas OF tpersona (...)
CREATE TABLE Propiedades OF tpropiedad
CREATE TABLE Terrenos OF tterreno UNDER Propiedades
CREATE TABLE Casas OF tcasa UNDER Propiedades
```

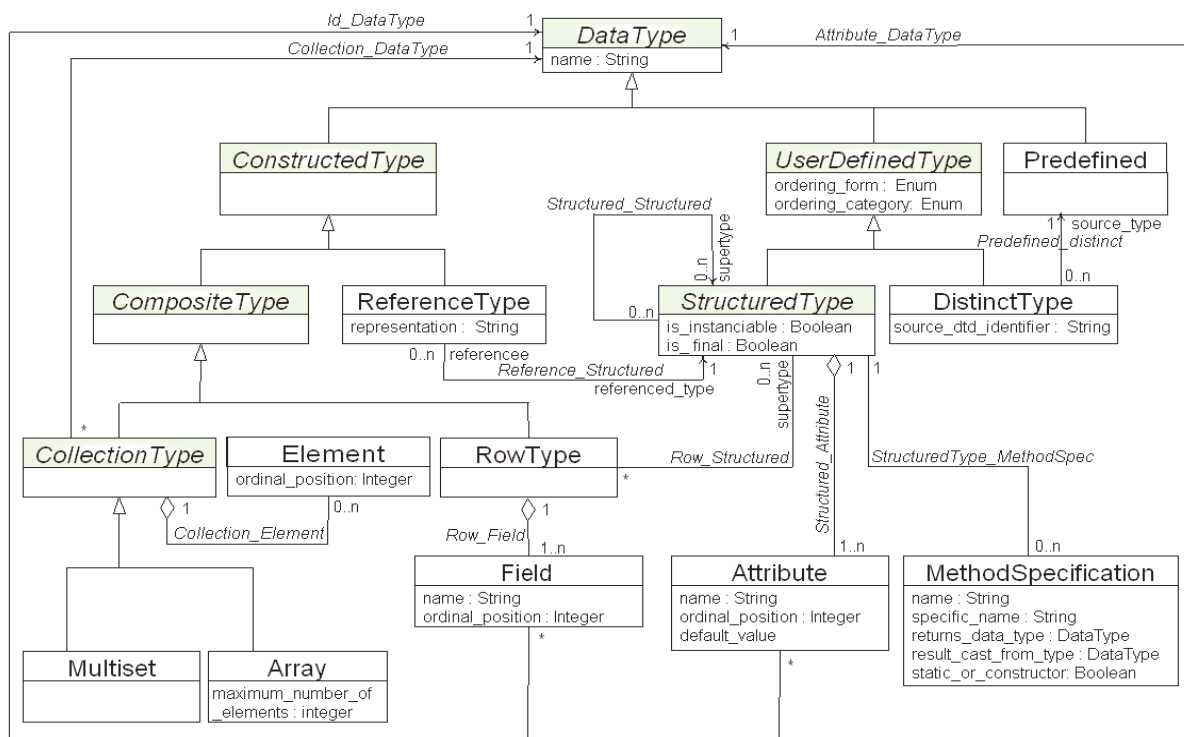


## Aspectos de Objetos en SQL Jerarquías de Tablas y Vistas

- **Consultas con jerarquías** de tablas/vistas:
  - Las consultas sobre la supertabla, devuelven también las filas de las subtablas  
`SELECT * FROM Propiedades WHERE ....`
  - Pero se puede restringir el resultado a sólo las filas de la supertabla  
`SELECT * FROM ONLY (Propiedades) WHERE ....`

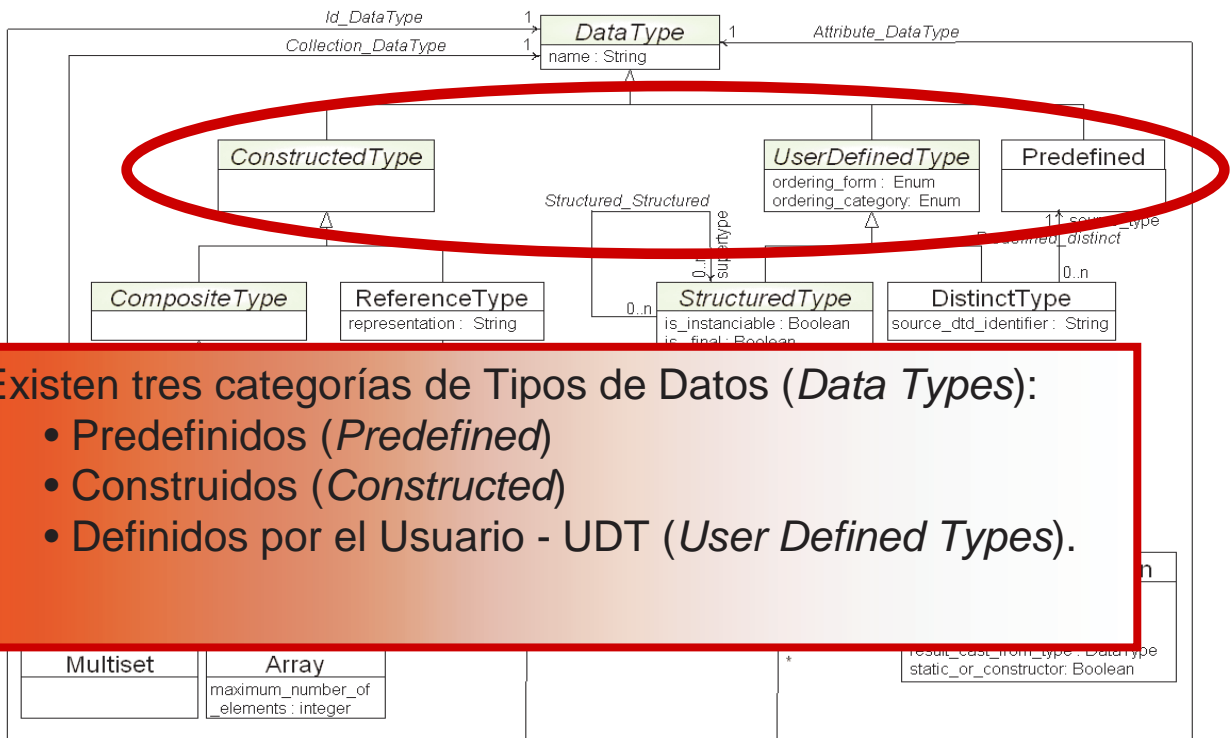


## Aspectos de Objetos en SQL Resumen – Tipos de Datos





## Aspectos de Objetos en SQL Resumen – Tipos de Datos

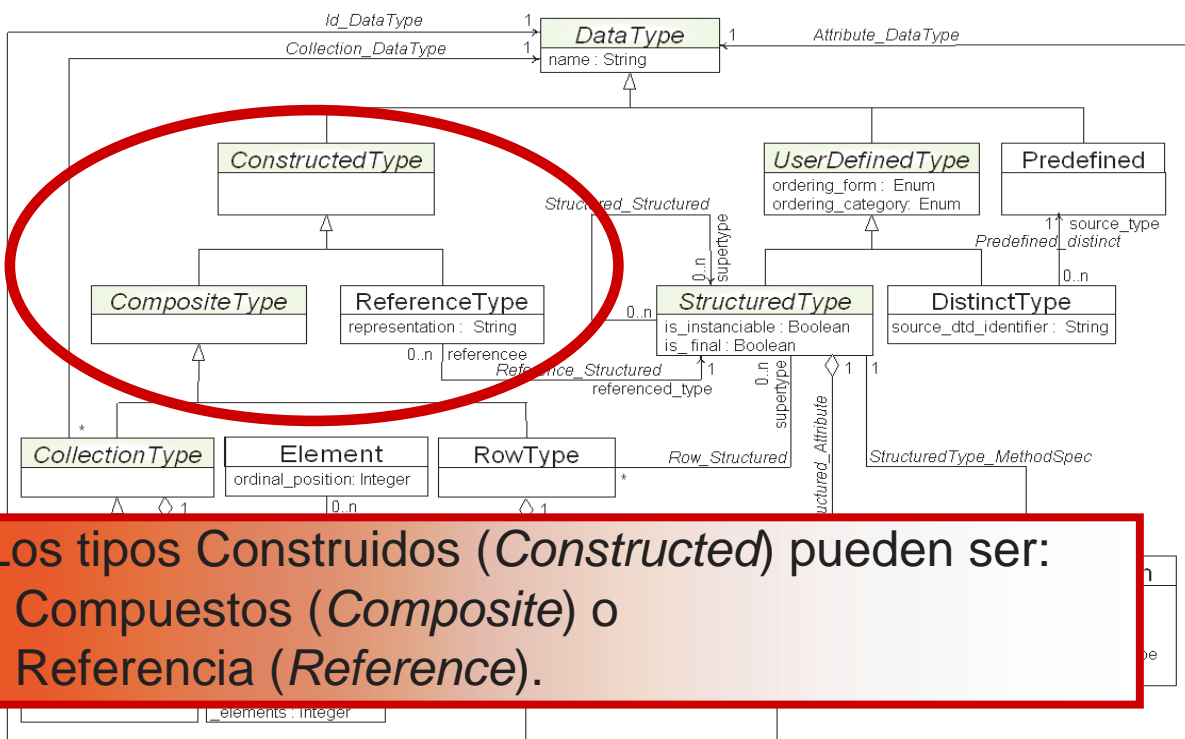


Existen tres categorías de Tipos de Datos (*Data Types*):

- Predefinidos (*Predefined*)
- Construidos (*Constructed*)
- Definidos por el Usuario - UDT (*User Defined Types*).



## Aspectos de Objetos en SQL Resumen – Tipos de Datos

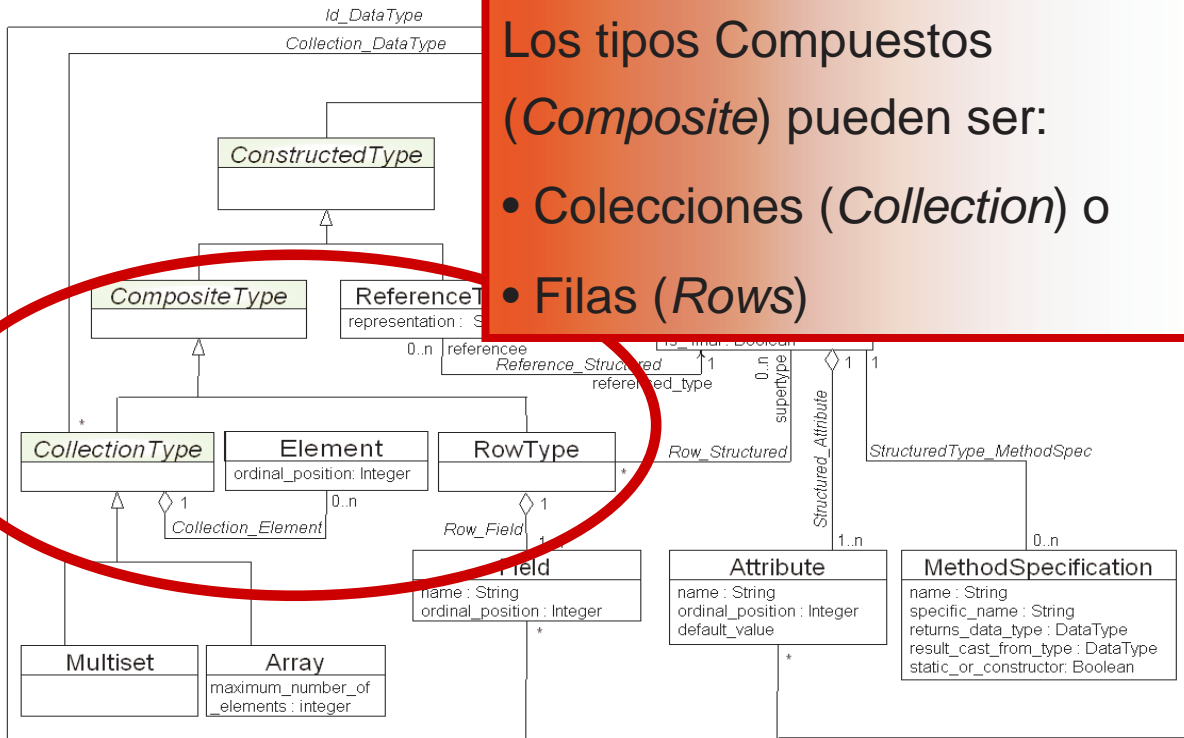


Los tipos Construidos (*Constructed*) pueden ser:

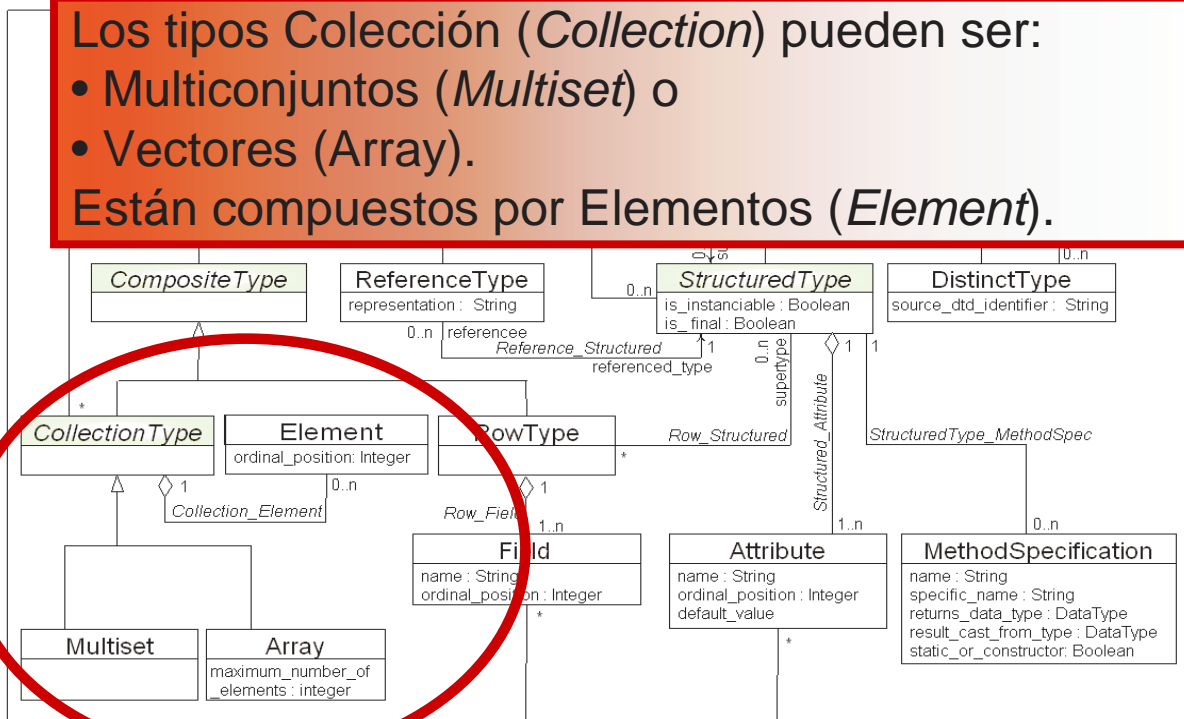
- Compuestos (*Composite*) o
- Referencia (*Reference*).



## Aspectos de Objetos en SQL Resumen – Tipos de Datos

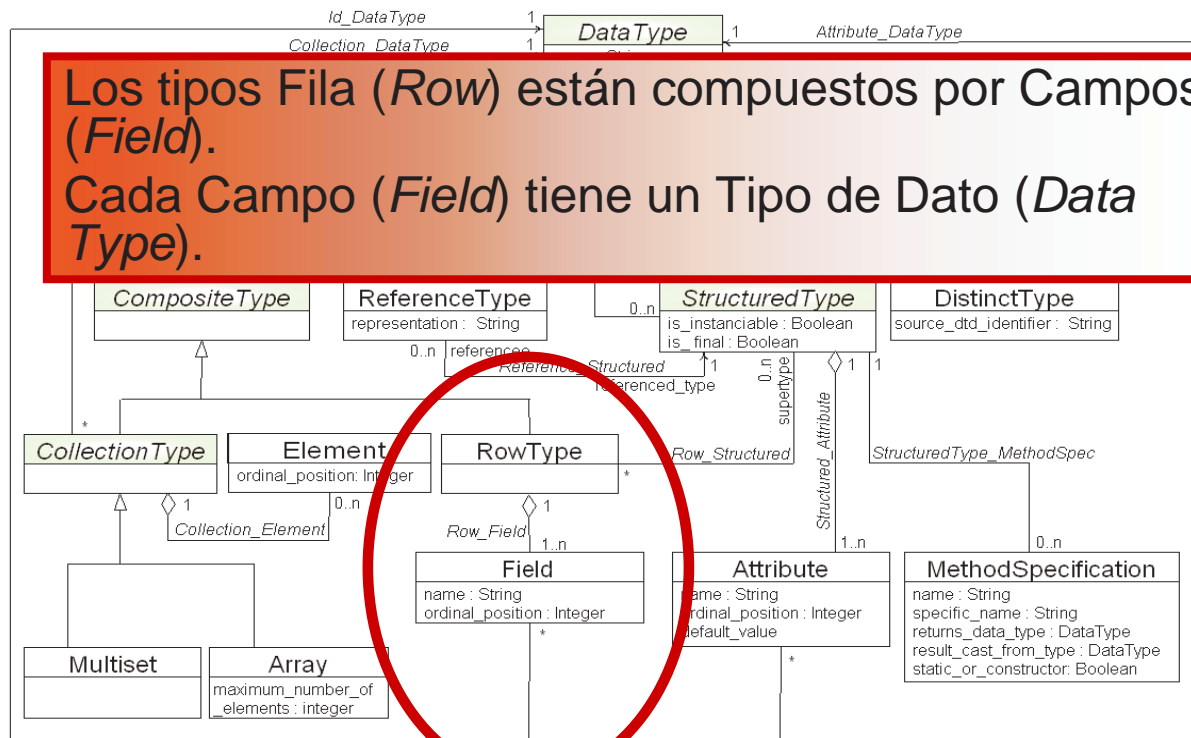


## Aspectos de Objetos en SQL Resumen – Tipos de Datos





## Aspectos de Objetos en SQL Resumen – Tipos de Datos

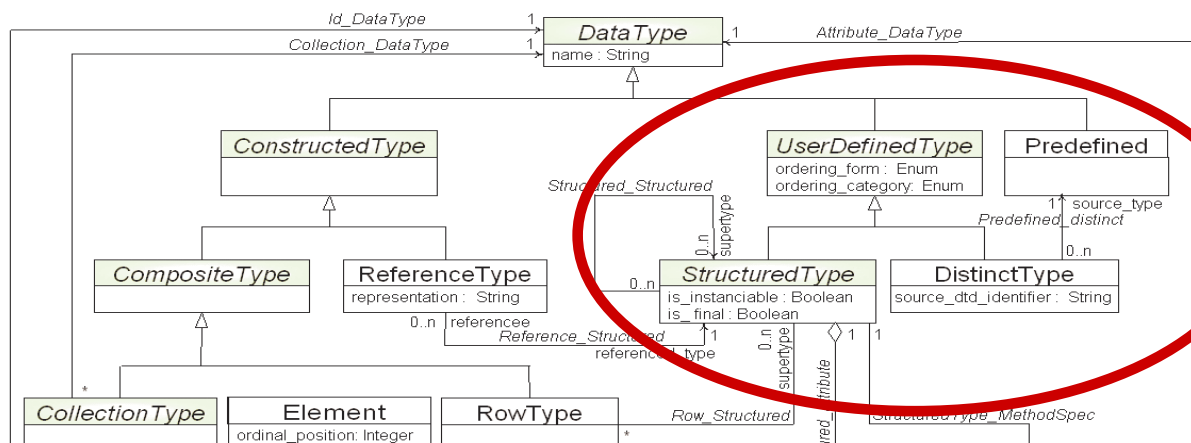


Francisco Ruiz - BDA

3a.77



## Aspectos de Objetos en SQL Resumen – Tipos de Datos



- Distintos (*Distinct*), que están definidos sobre un tipo de dato Predefinido (*Predefined*).
- Estructurados (*Structured*), que se corresponden con las clases en notación OO.

Francisco Ruiz - BDA

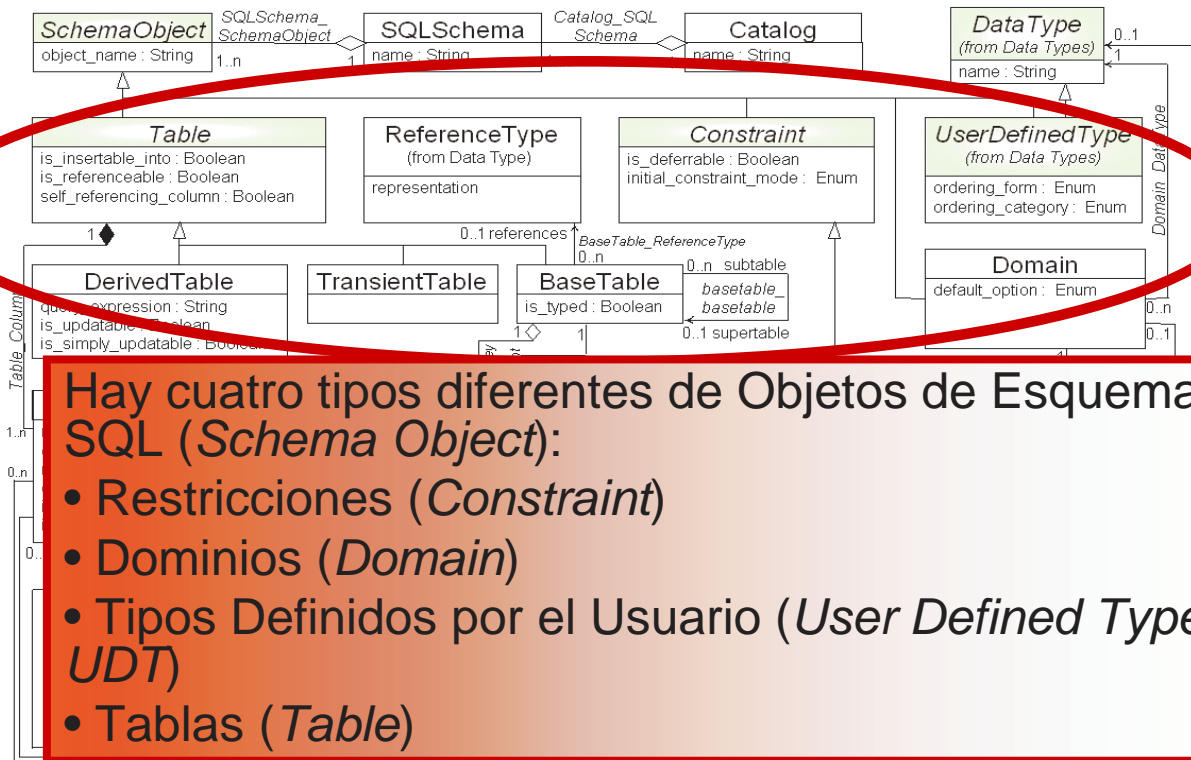
3a.78







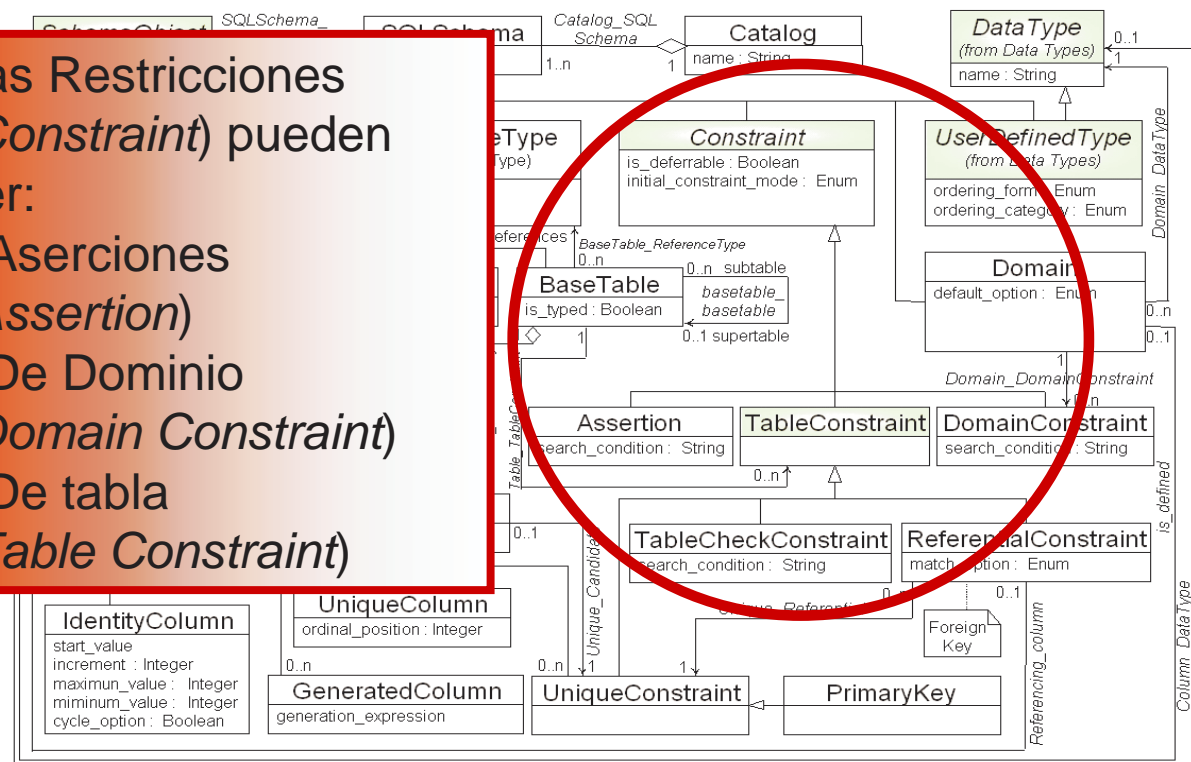
## Aspectos de Objetos en SQL Resumen – Objetos del Esquema



## Aspectos de Objetos en SQL Resumen – Objetos del Esquema

Las Restricciones (*Constraint*) pueden ser:

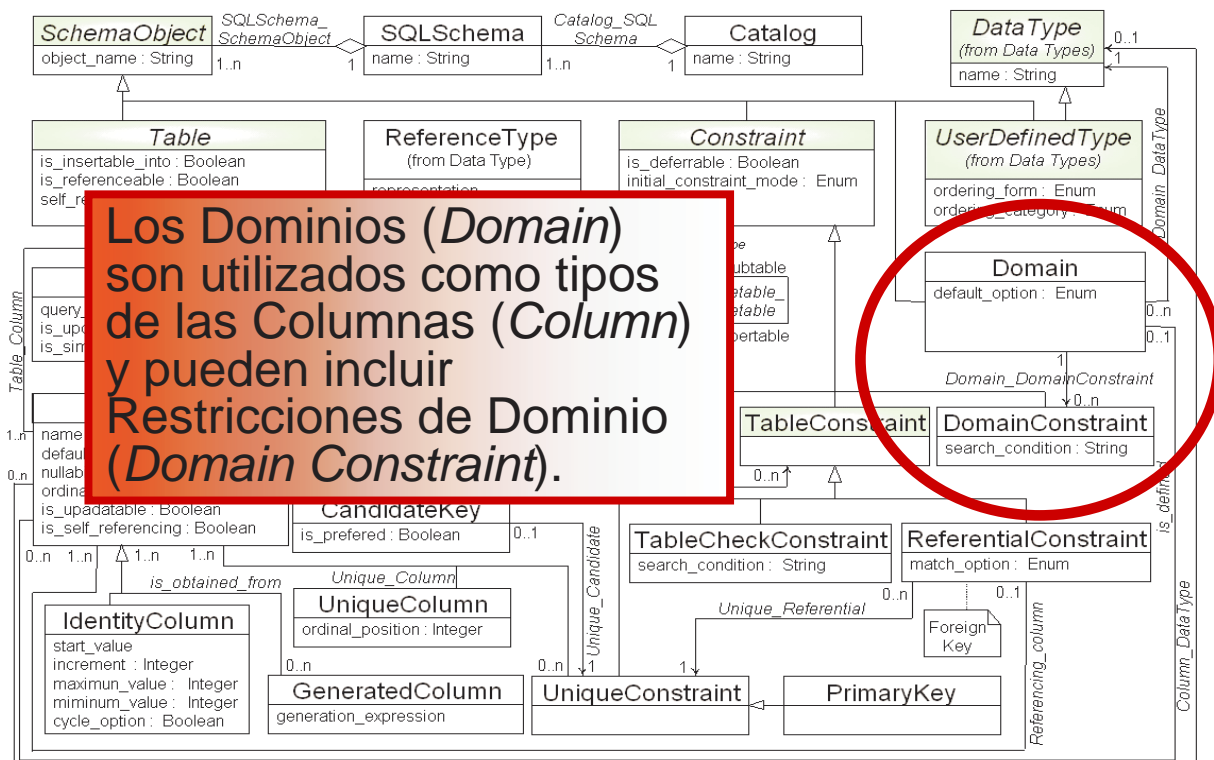
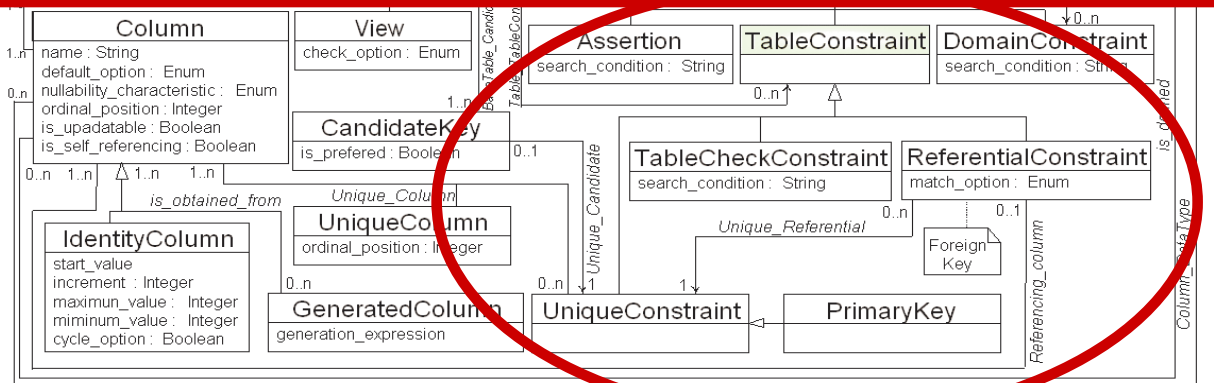
- Aserciones (*Assertion*)
- De Dominio (*Domain Constraint*)
- De tabla (*Table Constraint*)





Las Restricciones de Tabla (*Table Constraint*) pueden ser de:

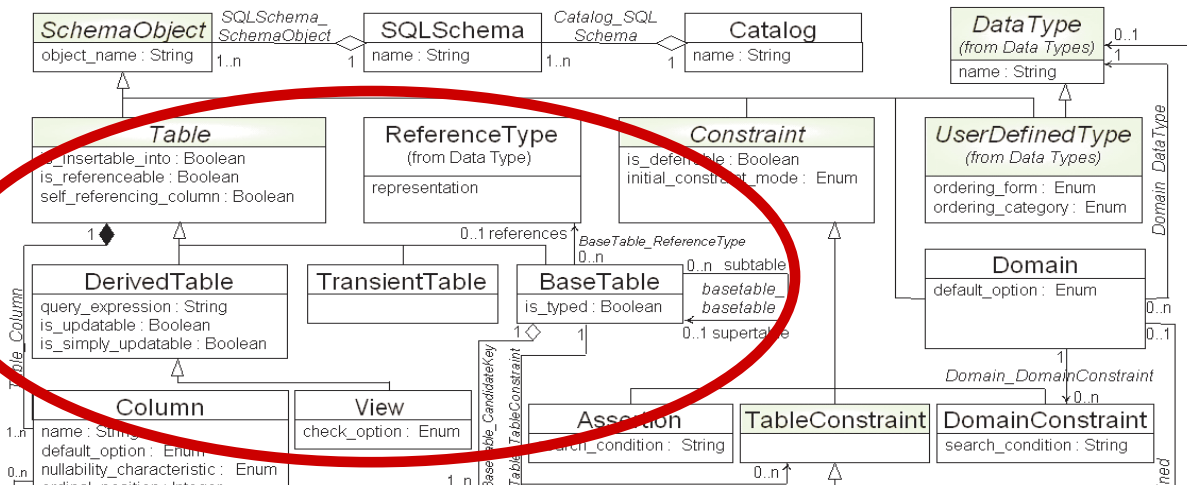
- Unicidad (*Unique Constraint*)
- Claves Primaria (*Primary Key*)
- Tipo Check (*Table Check Constraint*)
- Referenciales (*Referential Constraint*), para representar las claves ajenas



Los Dominios (*Domain*) son utilizados como tipos de las Columnas (*Column*) y pueden incluir Restricciones de Dominio (*Domain Constraint*).



## Aspectos de Objetos en SQL Resumen – Objetos del Esquema



Las Tablas (*Table*) pueden ser:

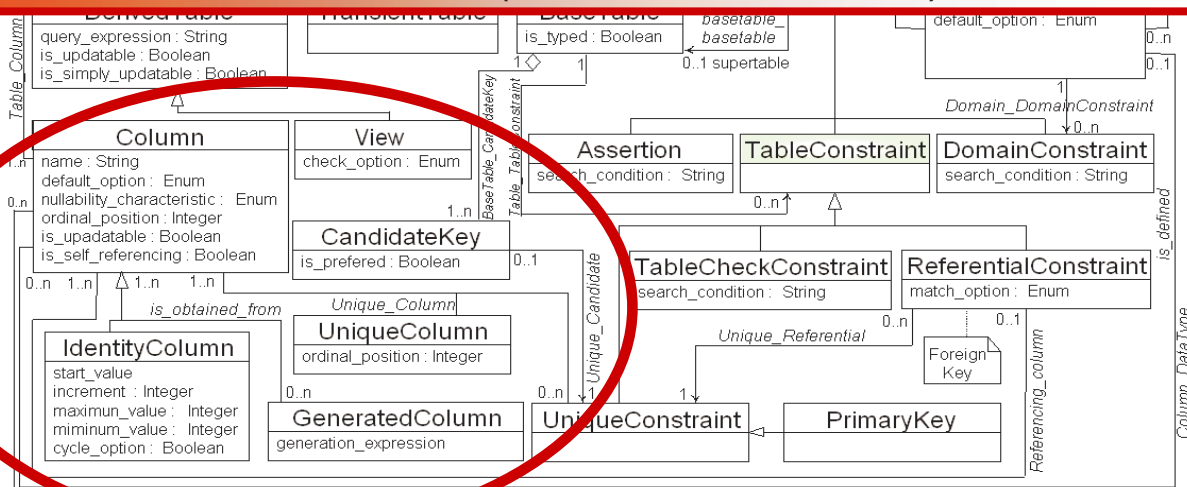
- Derivadas (*Derived*), dentro de las cuales se incluyen las Vistas (*View*)
- Transeuntes o temporales (*Transient*)
- Base (*Base*)



## Aspectos de Objetos en SQL Resumen – Objetos del Esquema

Las Tablas (*Table*) están compuestas por Columnas (*Column*), que pueden ser definidas como:

- Columnas Identidad (*Identity Column*)
- Columnas Generadas (*Generated Column*)

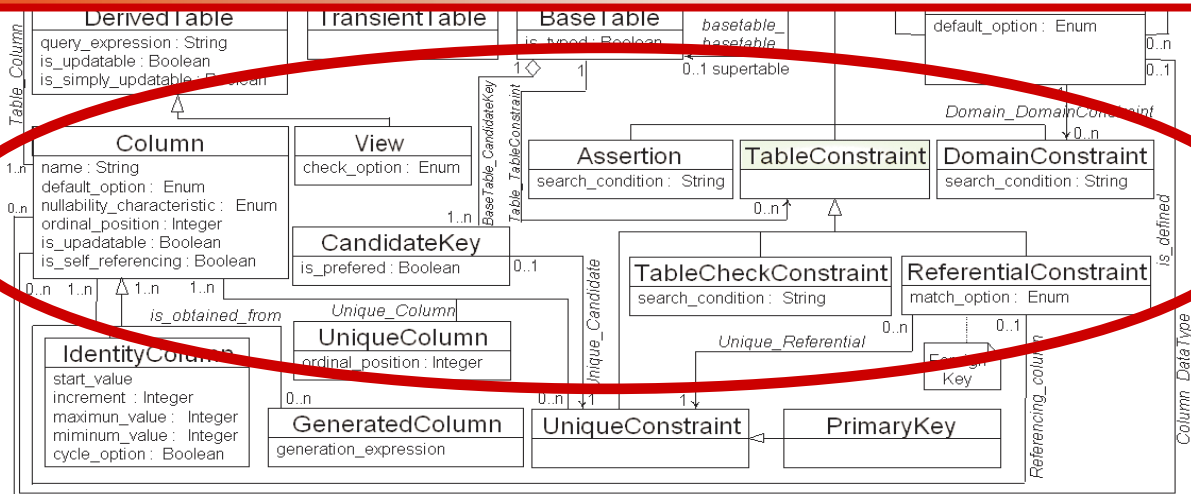




## Aspectos de Objetos en SQL Resumen – Objetos del Esquema

Las Columnas (*Column*) pueden ser definidas sobre un Dominio (*Domain*) y pueden tener Restricciones:

- Referenciales (*Referential Constraint*)
- De unicidad (*Unique Constraint*).

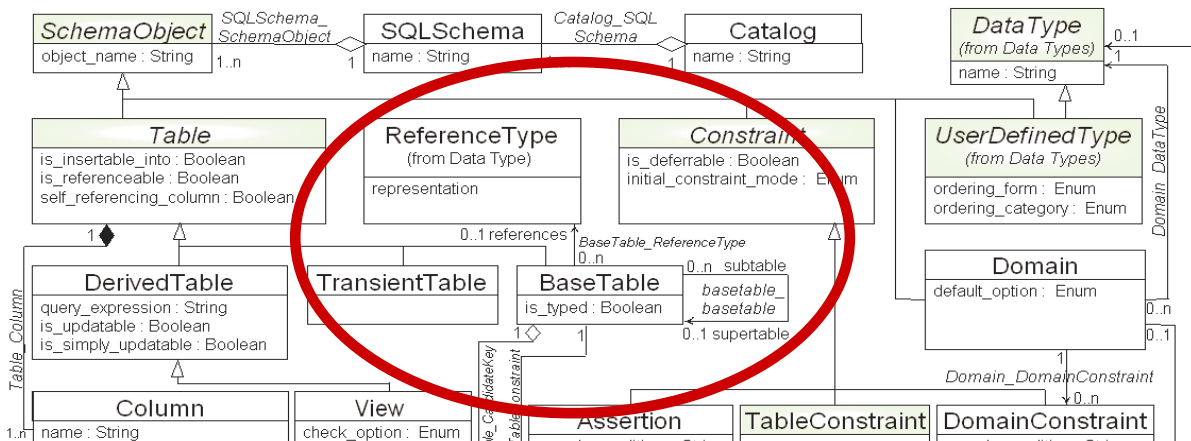


FRANCISCO KUIZ - BDA

3a.89



## Aspectos de Objetos en SQL Resumen – Objetos del Esquema



Las Tablas Base (*Base Table*) pueden también ser parte de una jerarquía de herencia. Además pueden estar definidas sobre un Tipo de Dato (*Data Type*) mediante un Tipo Referencia (*Reference Type*). También pueden tener Claves Candidatas (*Candidate Key*).

FRANCISCO KUIZ - BDA

3a.90



## Aspectos de Objetos en SQL

### Ejemplo

- El esquema está compuesto por cuatro tablas
  - **Customers**, con seis columnas simples (una de ellas es una columna identidad y otra una columna generada) y una columna compleja.
  - **Music\_Distributors**, que es tipada.
  - **Movie\_Stars**, con seis columnas simples.
  - **Movies**, con tres columnas simples y una compleja (con un tipo estructurado asociado).
- También incluye la definición de claves primarias, restricciones de unicidad, claves ajenas, restricciones de tabla, tipos distintos, tipos colección, tipos fila, etc.
- Igualmente hay restricciones, columnas generadas e identidad, una aserción y una definición de vista.

Francisco Ruiz - BDA

3a.91



## Aspectos de Objetos en SQL

### Ejemplo

```
CREATE SCHEMA video_and_music
AUTHORIZATION m_s_enterprises
DEFAULT CHARACTER SET "Latin_1"

CREATE DOMAIN price DECIMAL (7,2)
CHECK (VALUE IS NOT 0);

CREATE DISTINCT TYPE money AS DECIMAL
(9,2);

CREATE TYPE movie AS(
  movie_id INTEGER,
  title CHARACTER VARYING (100),
  languages MULTiset ['English', 'French',
'Spanish',
'Portuguese', 'Italian'],
  genre CHARACTER VARYING (20) ARRAY [10],
  run_time INTEGER)
INSTANTIABLE
NOT FINAL
METHOD length_interval ()
RETURNS INTERVAL HOUR (2) TO MINUTE

CREATE INSTANCE METHOD length_interval ()
RETURNS INTERVAL HOUR (2) TO MINUTE FOR
MOVIE
RETURN CAST (CAST (SELF.run_time AS
INTERVAL (4))
AS INTERVAL HOUR (2) TO MINUTE);

CREATE TYPE music_distrib AS (
distributor_id CHARACTER (15),
distributor_name CHARACTER (25));

CREATE TYPE address AS(
street CHARACTER VARYING (35),
city CHARACTER VARYING (40),
country character (3));

CREATE TYPE US_address UNDER address AS(
state CHARACTER (2),
zip ROW ( Basic INTEGER, Plus4 SMALLINT))

METHOD zipcode () RETURNS CHARACTER
VARYING (10);
```

```
CREATE INSTANCE METHOD zipcode ()
RETURNS CHARACTER VARYING (10)
FOR US_address
BEGIN
IF SELF.zip.plus4 IS NULL
THEN RETURN CAST (SELF.zip.basic AS
CHARACTER VARYING (5));
ELSE RETURN CAST (SELF.zip.basic AS
CHARACTER VARYING (5))
|| '-' || CAST (SELF.zip.basic AS
CHARACTER VARYING (4))
ENDIF;
END;

CREATE TABLE movies (
stock_number+r CHARACTER(10)
CONSTRAINT movies_stock_number_not_null NOT NULL,
movie_spec movie,
our_tape_cost price,
tapes_in_stock INTEGER
CONSTRAINT movies_primary_key
PRIMARY KEY (stock_number));

CREATE TABLE movie_stars (
movie_title CHARACTER (30)
CONSTRAINT movies_stars_movie_title_not_null NOT NULL,
movie_year_released DATE,
movie_number CHARACTER (10),
actor_last_name CHARACTER (35)
CONSTRAINT movies_stars_actor_last_name_not_null
NOT NULL,
actor_first_name CHARACTER (25)
CONSTRAINT movies_stars_actor_first_name_not_null
NOT NULL,
actor_middle_name CHARACTER (25),
CONSTRAINT movies_stars_unique
UNIQUE (movie_title, actor_last_name, actor_first_name,
actor_middle_name)
NOT DEFERRABLE,
CONSTRAINT movies_stars_fk_movie
FOREIGN KEY (movie_number)
REFERENCES movies (stock_number)
ON DELETE CASCADE
ON UPDATE CASCADE);
```

```
CREATE TABLE music_distributors OF
music_distributors (
REF IS dist_ref SYSTEM GENERATED,
distributor_id WITH OPTIONS
CONSTRAINT
music_distributors_distributor_id_not_null
NOT NULL,
distributor_name WITH OPTIONS
CONSTRAINT
music_distributors_distributor_name_not_null
NOT NULL, );

CREATE TABLE customers(
nr_of_customer INTEGER GENERATED ALWAYS
AS IDENTITY (START WITH 1
INCREMENTED BY
1MINVALUE 1),
cust_last_name CHARACTER (35)
CONSTRAINT customers_cust_last_name_not_null
NOT NULL,
cust_first_name CHARACTER (35)
CONSTRAINT customers_cust_first_name_not_null
NOT NULL,
cust_complete_name GENERATED ALWAYS AS
(cust_first_name ||
cust_last_name),
cust_address US_address,
cust_current_charges money,
number_of_problems SMALLINT);

CREATE VIEW problem_customers (last, first) AS
SELECT cust_last_name, cust_first_name
FROM customers
WHERE number_of_problems >
0.8 * (SELECT MAX(number_of_problems)
FROM customers);

CREATE ASSERTION limit_total_movie_stock_value
CHECK (( SELECT COUNT(*)
FROM customers
WHERE number_of_problems > 5
AND cust_current_charges > 150,00
AND cust_current_charges < 1000,00)
<10);
```

Francisco Ruiz - BDA

3a.92

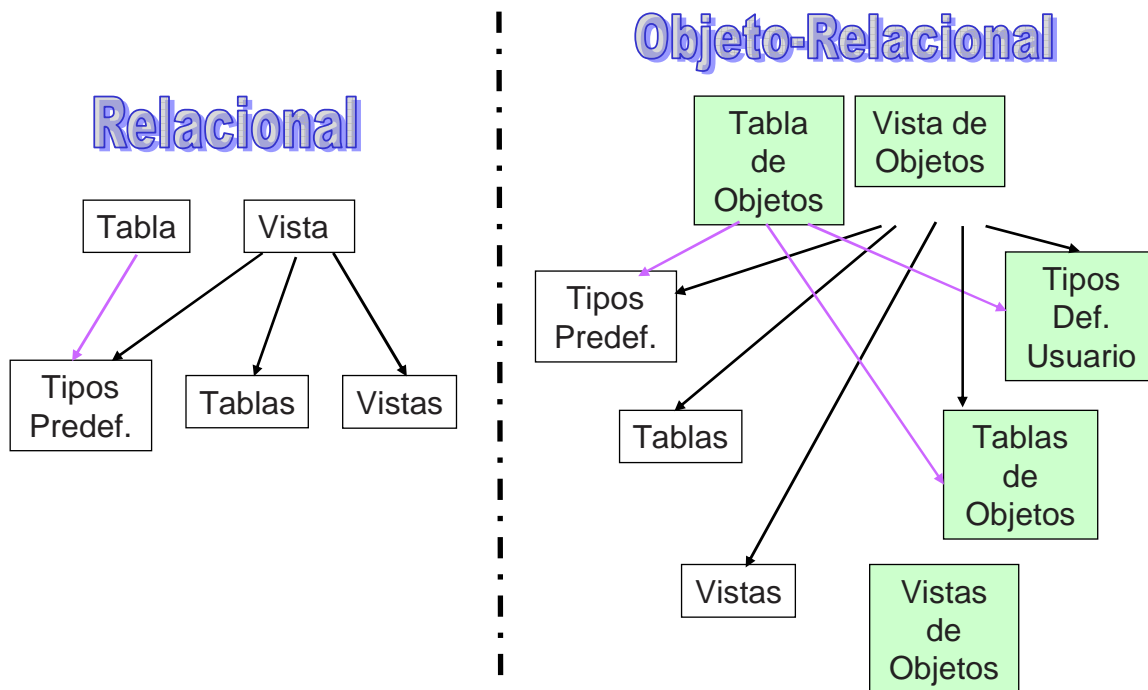


## SGBD Objeto-Relacionales

- Los sistemas objeto-relacionales están siendo **los más vendidos** en la actualidad
  - ¿Pero son también los más usados?
  - Es decir, ¿Se utilizan las extensiones objetuales?
- Algunos de los **productos más difundidos** son:
  - **DB2 Universal Database** de **IBM**
  - **Universal Server** de **INFORMIX**
  - **INGRES II** de **Computer Associates**
  - **ORACLE** de **Oracle Corporation** (desde la versión 8)
  - **SYBASE** de **SYBASE Inc.**



## SGBD Objeto-Relacionales ORACLE





- **Tipos Predefinidos:**
  - **Carácter** (CHAR, VARCHAR2, NCHAR, NVARCHAR2, LONG)
  - **Numéricos** (NUMBER, INTEGER, FLOAT)
  - **DATE**
  - **ROWID**
  - **LOB** (CLOB, BLOB, BFILE)
  - ....
- **Tipos Definidos por el Usuario:**
  - Tipos distintos (como en SQL)
  - Tipos estructurados
  - Tipos de Objeto
  - Tipo REF (referencia)
  - Tipos colección



## Tipos Estructurados

```
CREATE OR REPLACE TYPE dirección (  
  calle      VARCHAR2 (30),  
  ciudad    VARCHAR2 (20),  
  provincia  VARCHAR2 (2),  
  codigoP   VARCHAR2 (5));
```

```
CREATE OR REPLACE TABLE persona (  
  nombre    VARCHAR2(30),  
  vive_en   dirección,  
  foto      BLOB);
```



## Tipos Objeto y Tipos REF

```
CREATE OR REPLACE TYPE Empleado AS OBJECT
( DNI NUMBER,
  Nombre VARCHAR2(30),
  Fecha_nac DATE
  Pertenece_a REF Departamento,
  MEMBER FUNCTION Edad RETURN NUMBER)
```

```
CREATE TABLE Tabla_Empleado OF Empleado
```



## Tipos Colección: VARRAY (vectores SQL)

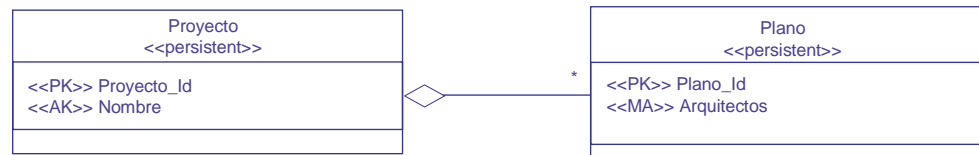
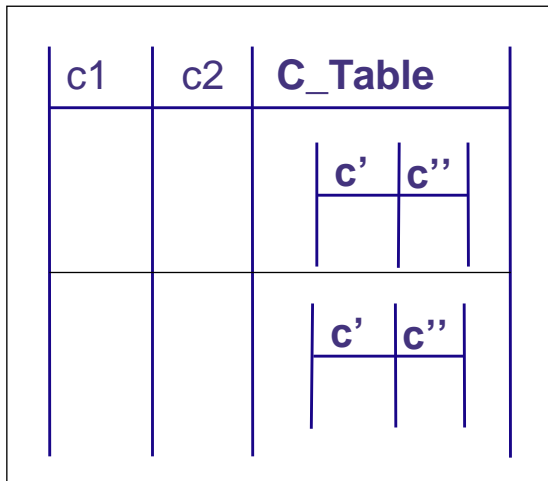
```
CREATE OR REPLACE TYPE telefono
AS VARRAY (3) of varchar (10);

CREATE TABLE Empleado
( DNI NUMBER,
  Nombre VARCHAR2(30),
  Teléfonos_contacto telefono);

INSERT INTO Empleado VALUES ('9876543', 'Pepe',
  telefono('914445566', '606445566', '934445566'));
```



## Tipos Colección: Nested Table (tablas anidadas)



## Tipos Colección: Nested Table (tablas anidadas)

```

CREATE TYPE Plano AS OBJECT
  (Plano_ID NUMBER,
  Numero_Figuras NUMBER,
  Arquitectos Tipo_Nombre);

CREATE TABLE T_Plano OF Plano (PRIMARY KEY (Plano_Id));

CREATE OR REPLACE TYPE Lista_Planos AS TABLE OF REF Plano;

CREATE TYPE Proyecto AS OBJECT
  (Proyecto_Id NUMBER,
  Nombre VARCHAR(30),
  Tiene_Plano Lista_Planos);

CREATE TABLE T_Proyecto OF Proyecto
  (PRIMARY KEY (Proyecto_ID),
  UNIQUE (Nombre))
  NESTED TABLE Tiene_Plano STORE AS Tabla_Planos;
  
```



## Tipos Colección: Nested Table (tablas anidadas)

```
INSERT INTO t_plano VALUES (1, 5,NULL);
INSERT INTO t_plano VALUES (2, 4,NULL);

DECLARE
  p1_ref REF plano;
  p2_ref REF plano;
BEGIN
  SELECT REF(t_plano) INTO p1_ref FROM t_plano WHERE plano_id=1;
  SELECT REF(t_plano) INTO p2_ref FROM t_plano WHERE plano_id=2;
  INSERT INTO t_proyecto VALUES (1,'MIDAS',Lista_Planos(p1_ref,p2_ref));
END;
```



## Tipos Colección: Nested Table vs. Varray

	VARRAY	NESTED TABLE
Tamaño máximo	Si	No
Borrado elementos ind.	No	Si
Almacenamiento datos	In-line	Out-of-line
Mantenimiento del orden	Si	No



## SGBD Objeto-Relacionales Comparación

### Modelos de Objetos de SQL, ORACLE e INFORMIX

Concepto	SQL	ORACLE	INFORMIX
Tipo objeto / clase	Tipo estructurado	Tipo objeto	Tipo ROW
Extensión	Tabla tipada	Tabla de tipo objeto	Tabla de tipo ROW
Objeto	Fila de tabla tipada	Fila de tabla objeto	Fila de tabla con tipo ROW
Herencia	De tablas y tipos estructurados	---	De tipos y tablas
OID	REF	REF	---
Colecciones	Array, Multiset	Varray, Nested Table	Set, List, Multiset
Estructura	Row, Tipo estructurado en columna	Tipo estructurado	Tipo ROW sin nombre
Método	Método	Método	No asociados
Polimorfismo	Sobrecarga de métodos	Sobrecarga de métodos	---



## SGBD Objeto-Relacionales Comparación

### Modelos de Objetos de SQL vs Java

JAVA	SQL
Clase	Tipo Definido por el Usuario (en particular, tipo estructurado)
Superclase	Supertipo
Subclase	Subtipo
Instancia	Valor
Objeto	Fila en una tabla de un tipo estructurado
Tipo primitivo	Tipo de dato predefinido
Método	Método (también función o procedimiento)
Constructor	Inicializador