



ACCESO A DATOS

UNIDAD 05 - BBDD DE OBJETOS Y OBJETO-RELACIONALES

INDICE

- 5.1 - Introducción.
- 5.2 - Características.
- 5.3 - El estándar ODMG.
- 5.4 - ODL y OQL.
- 5.5 - BD Matisse.

5.1 - Introducción



5.1 - INTRODUCCIÓN

- Con el auge de la programación OO se pusieron de manifiesto las dificultades para almacenar objetos complejos en BBDD-R (Desfase O-R).
- Como solución natural se plantearon las BBDD de objetos (DBO o ODB).



5.1 - INTRODUCCIÓN

- Se intentó repetir el éxito de las BBDD-R:
 - Modelo formal (modelo Atkinson 1989)
 - ❖ Propuesta de requisitos que debería cumplir un DBO.
 - Temprano desarrollo de estándares (ODMG 1991).
 - ❖ Desarrollo de estándares sobre los que basar las BDO.
- Tubo un arranque entusiasta pero los resultados a medio y largo plazo no fueron los esperados.



5.1 - INTRODUCCIÓN

- No llegó a desarrollarse ningún modelo formal ampliamente aceptado.
- ODMG desarrolló estándares como:
 - ODL para la definición de datos.
 - OQL para la consulta de datos.
 - *Language bindings* o vinculaciones con lenguajes orientados a objetos de propósito general ya existentes.



5.1 - INTRODUCCIÓN

- Última versión del estándar ODMG 3.0 publicada en 2000.
- ODMG se disolvió en 2001.
- En 2004 se cedieron los derechos para revisar la especificación ODMG 3.0 a OMG (Object Management Group).



5.1 - INTRODUCCIÓN

- Por otro lado, en SQL:99 se incluyeron tipos estructurados definidos por el usuario, entre ellos tipos de objetos.
- Estos tipos de objetos se han implementado en BBDD-R como Posgre-SQL y Oracle, considerándolas BDOR.



5.1 - INTRODUCCIÓN

- Las BDO siguen teniendo un uso muy limitado hoy día.
- Tenemos BDO de software libre y privativas como Matisse, Objectivity/DB y db4o.
- Oracle es una BDOR dominante en el segmento empresarial, el cual implementa los objetos en una capa de software sobre una base de datos relacional.

ORACLE®



5.1 - INTRODUCCIÓN

- Otros planteamientos alternativos para persistencia han tenido más éxito.
- La correspondencia objeto-relacional (ORM) ha tenido mucho éxito con productos como Hibernate (2001) y el estándar JPA para ORM integrante de Java EE desde Java EE 6 (2009).





5.2 - Características



5.2 - CARACTERÍSTICAS

- Una BDO puede trabajar directamente con objetos.
- El manifiesto Atkinson y otros (1989) propone una lista de características que cualquier SGBDO debería cumplir.
- <http://clamen.net/stewart/cs.cmu.edu/OODBMS/Manifesto/htManifesto/Manifesto.html>



5.2 - CARACTERÍSTICAS

- Resumen de características:
 - **Objetos complejos:** Debe ser posible almacenar objetos complejos, donde los atributos pueden ser referencias a otros objetos, colecciones desordenadas (conjuntos) y colecciones ordenadas (listas).
 - **Identidad de objetos:** Los objetos tienen una existencia independiente a su valor. Cada objeto tiene un id único. Se distingue entre:
 - Igualdad: dos objetos tienen el mismo valor.
 - Identidad: dos objetos son el mismo objeto.



5.2 - CARACTERÍSTICAS

- Resumen de características:
 - **Encapsulamiento:** Un objeto tiene una parte de interfaz y otra de implementación. La parte de implementación, a su vez, una parte de datos y otra de procedimientos o métodos.
 - **Tipos o clases:** Los tipos comprenden un conjunto de objetos con características comunes. La noción de clase es similar, pero es más un concepto de tiempo de ejecución. (Cuando nosotros hablemos de clase será extensible a tipo)
 - **Herencia:** Una clase se puede definir como subclase de otra (su superclase), y en ese caso hereda todos sus atributos y métodos.



5.2 - CARACTERÍSTICAS

- Resumen de características:
 - **Redefinición (overriding), sobrecarga (overloading) y vinculación dinámica (late binding):** Un método de una clase A se puede redefinir en subclases tuyas. En ese caso existe más de un método con igual nombre, esto es la sobrecarga de métodos. Si en un programa el método se ejecuta sobre un objeto declarado en la clase A, solo se sabe en tiempo de ejecución si el objeto es de esa clase o de alguna subclase suya. Si el método ha sido redefinido en alguna subclase, solo entonces y no antes, se debe decidir qué método ejecutar: el de la clase A o el de alguna subclase. En esto consiste la vinculación dinámica.



5.2 - CARACTERÍSTICAS

- Resumen de características:
 - **Completitud computacional:** Debe poder realizarse cualquier posible cálculo utilizando el DML de la BD (SQL no lo es). Este requisito se puede satisfacer mediante *bindings* o vinculaciones a Lenguajes OO, como por ejemplo Java.
 - **Extensibilidad:** Deben poder definirse nuevos tipos a partir de los ya existentes y usarse de igual forma.



5.2 - CARACTERÍSTICAS

- Resumen de características:
 - **Persistencia:** Todos los datos se deben mantener en la BD después de que la aplicación que los creó haya finalizado.
 - **Gestión del almacenamiento secundario:** Debe ser capaz de manejar grandes volúmenes de datos.
 - **Concurrencia:** Permitir uso concurrente por varios usuarios, proporcionando atomicidad y serializabilidad de una secuencia de operaciones.



5.2 - CARACTERÍSTICAS

- Resumen de características:
 - **Recuperación:** volver a un estado anterior coherente ante un fallo de hardware o software.
 - **Métodos de consulta sencillos:** Idealmente de alto nivel, eficientes e independientes de la aplicación, es decir, utilizables con cualquier posible BD.

5.3 - El estándar ODMG



5.3 - EL ESTÁNDAR ODMG

- Es un estándar para la persistencia de objetos en BD.
- La última versión es ODMG 3.0 del año 2000.
- En 1998 cambió el significado de sus siglas de Object Database Management Group a Object Data Management Group.



5.3 - EL ESTÁNDAR ODMG

- Características del estándar ODMG
 - https://www.service-architecture.com/articles/database/odmg_3_0.html
- Texto íntegro
 - https://cs.ulb.ac.be/public/_media/teaching/odmg.pdf
- Tabla soporte en plataforma Moodle.
 - <https://www.barryandassociates.com/odmg-odbms.pdf>



5.3 - EL ESTÁNDAR ODMG

- Los principales componentes son:
 - Modelo de objetos, basado en el modelo de objetos de OMG.
 - ODL o lenguaje de definición de objetos.
 - OQL o Lenguaje de consulta de objetos, inspirado en SQL-92
 - Language bindings o vinculaciones con los lenguajes C++, Smalltalk y Java.



5.3 - EL ESTÁNDAR ODMG

- ODMG 3.0 no incluye un DML, sino *language bindings* con lenguajes de propósito general (C++, Smalltalk y Java).
- Este planteamiento se conoce como **persistencia transparente**.



5.3 - EL ESTÁNDAR ODMG

- Desde el lenguaje de propósito general se crean, modifican y borran los objetos persistentes igual que los no persistentes.
- En el momento de confirmar los cambios realizados dentro de una transacción es cuando se graban los cambios de los objetos persistentes en la BD.
- Un *language binding* también proporciona mecanismos para realizar consultas de OQL y obtener los resultados como objetos persistentes.

5.4 - ODL y OQL



5.4 - ODL y OQL

- ODL es un lenguaje formal para especificación de objetos.
- Un SGBDO almacena objetos de tipos definidos en un esquema mediante ODL.



5.4 - ODL y OQL

- Se puede establecer una analogía con el modelo relacional de la siguiente forma.

Modelo de objetos de ODMG 3.0	Modelo Relacional
ODL: Lenguaje de definición de objetos	DDL: Lenguaje de definición de datos, subconjunto de SQL
Esquema de objetos	Esquema relacional
Clase	Tabla
Instancia de clase	Fila de tabla



5.4 - ODL y OQL

- ODL se basa en el modelo de OMG, cuyas principales características son:
 1. El estado de un objeto está definido por los valores de sus propiedades, las cuales pueden cambiar a lo largo del tiempo.
 2. Las propiedades pueden ser atributos o relaciones (1 a 1, 1 a muchos, o muchos a muchos).
 3. Un objeto tiene un UID que no cambia a lo largo del tiempo. Se distingue entre identidad (=UID) e igualdad (=valores de atributos).



5.4 - ODL y OQL

- ODL se basa en el modelo de OMG, cuyas principales características son:
 4. La conducta (*behavior*) de un objeto se define por el conjunto de operaciones que se pueden realizar sobre él.
 5. Un tipo (clase o interfaz) tiene una especificación externa y puede tener una o más implementaciones. La especificación externa puede ser:
 - ❖ Operaciones, que se pueden invocar en las instancias del tipo.
 - ❖ Propiedades, variables de estado consultables o modificables.
 - ❖ Excepciones, que pueden lanzar sus operaciones.



5.4 - ODL y OQL

CLASES E INTERFACES

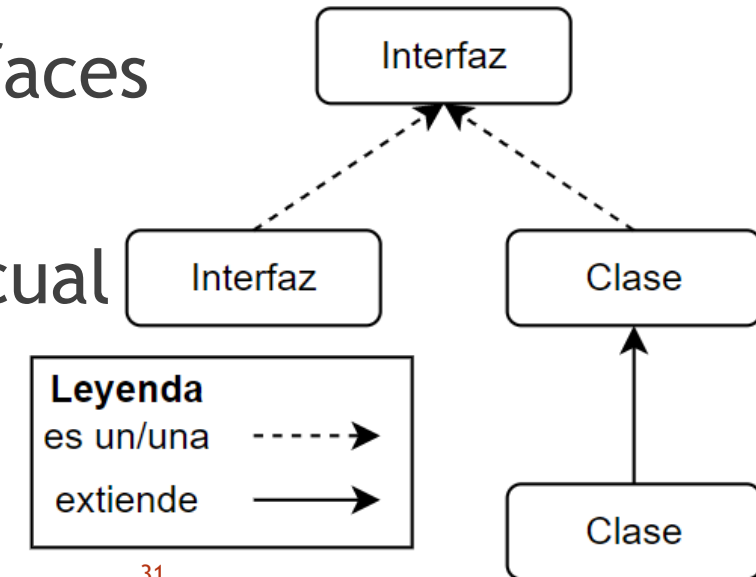
- Un tipo puede ser:
 - ❖ Una *interfaz* (*interface*) que se define por su conducta abstracta (sus operaciones).
 - ❖ Una *clase* (*class*) que se define por su conducta abstracta y su estado abstracto (sus atributos).
- La distinción entre interfaces y clases y su relación con los objetos, es similar a la que hay en Java.



5.4 - ODL y OQL

CLASES E INTERFACES

- No se pueden crear instancias de una interfaz, pero sí de una clase. Un objeto es una instancia de una clase.
- Clases e interfaces pueden heredar de interfaces (relaciones del tipo “es un/una”).
- Las clases pueden extender otras clases, la cual hereda las operaciones y propiedades de la *superclase*.





5.4 - ODL y OQL

CLASES E INTERFACES

- Para una clase se define una extensión con la palabra clave *extent*.
- Una extensión proporciona acceso a todas las instancias de una clase.
- A una extensión se le puede asociar una clave (*key*), donde no pueden haber dos objetos distintos dentro de la extensión con idéntico valor para la clave (restricción de integridad a cumplir por el SGBDO)



5.4 - ODL y OQL

RELACIONES

- Las relaciones son propiedades de las clases.
- Consisten en la asociación de cada instancia de una clase con una o más instancias de otras.
- ODMG 3.0 solo contempla relaciones binarias.
- Los tipos pueden ser e uno a uno (1-1), uno a muchos (1-N) y muchos a muchos (N-N).

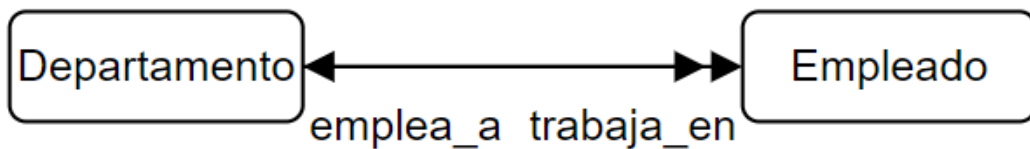


5.4 - ODL y OQL

RELACIONES

- Se definen en ambos extremos (las dos clases), como un *traversal path*.
- Ejemplo 1-N entre Departamento y Empleado

Relaciones y traversal path en ODL



```
class Departamento {  
    relationship set<Empleado> emplea_a  
    inverse Empleado::trabaja_en;  
};  
class Empleado {  
    relationship Departamento trabaja_en  
    inverse Departamento::emplea_a;  
};
```



5.4 - ODL y OQL

RELACIONES

- Según cardinalidad máxima de la relación en un *traversal path* se puede representar con un único objeto (cardinalidad 1) o una colección de objetos (N)
- En cardinalidad máxima muchos (N), se puede representar con una colección ordenada (*list*) o desordenada (*set*).



5.4 - ODL y OQL

RELACIONES

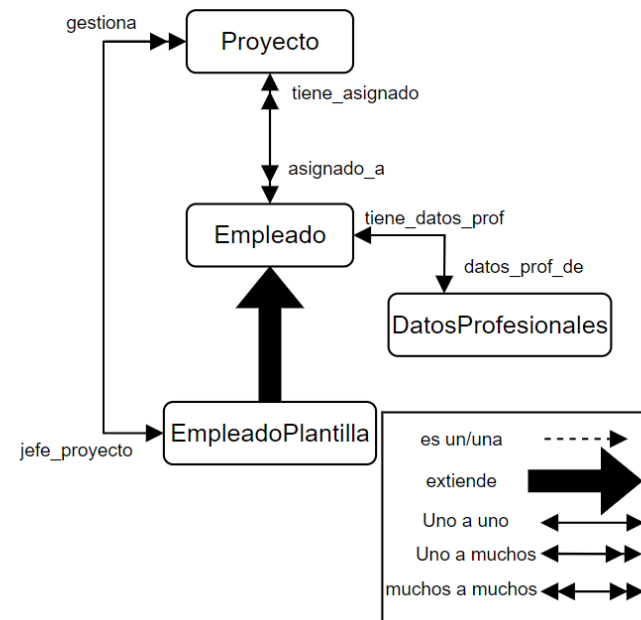
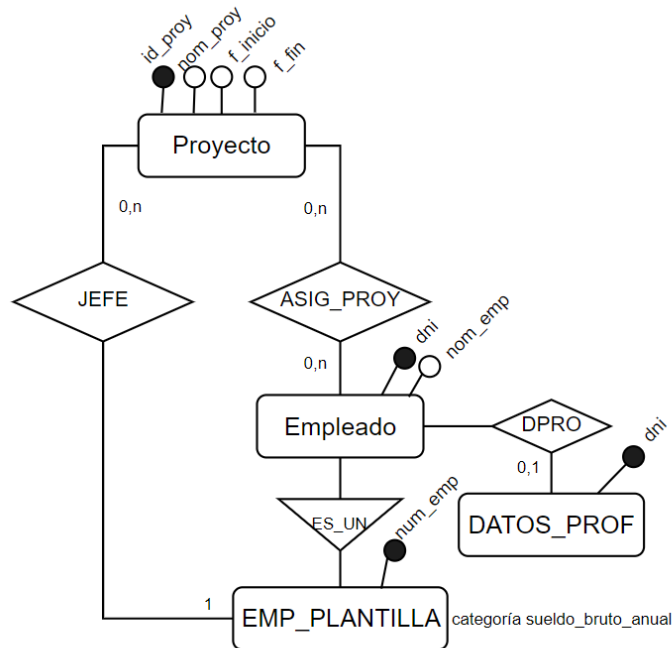
- Las operaciones se gestionan mediante operaciones públicas de las clases involucradas.
- Si en el *traversal path* la cardinalidad máxima es:
 - ❖ 1, operaciones de asignar y si mínima 0, borrar.
 - ❖ N, operaciones de añadir y borrar al conjunto.



5.4 - ODL y OQL

RELACIONES

- ODMG 3.0 tiene notación propia para diagramas de objetos que representan clases y relaciones entre ellas.





5.4 - ODL y OQL

ACTIVIDADES PROPUESTAS



- a) Crea un diagrama E-R, y el correspondiente diagrama de objetos, para representar las siguientes relaciones entre entidades:

Una publicación puede ser un libro o una revista. En cualquier caso, tiene un nombre de publicación (**nom_pub**), que en el caso del libro representa el título y en el de la revista, el nombre. Un libro tiene un autor, del que interesa su nombre (**nom_autor**) y su nacionalidad (que se indica en un campo como texto libre). Un libro tiene como identificador un ISBN (**isbn**), y una revista, un ISSN (**issn**). El diagrama E-R debe incluir las entidades PUBLICACIÓN, LIBRO, REVISTA Y AUTOR.



5.4 - ODL y OQL

OQL

- OQL es un lenguaje de consulta incluido en ODMG 3.0.
- Tiene una sintaxis inspirada en la sentencia **SELECT** de SQL pero adaptada a objetos.
- El *language binding* permite realizar consultas en OQL y recuperar resultados mediante un *iterador*.



5.4 - ODL y OQL

OQL

- OQL no incluye sentencias similares a INSERT, UPDATE o DELETE de SQL.
- Según el principio de persistencia transparente de ODMG 3.0, las operaciones de creación, borrado y modificación de objetos se realizan mediante el lenguaje de programación.
- El language binding garantiza que los cambios dentro de una transacción se reflejen en la BD al confirmarse.



5.4 - ODL y OQL

OQL

- Las consultas sobre BBDD se realizan utilizando *puntos de entrada*.
- Como punto de entrada puede servir una extensión (**extent**) de una clase (permite acceder a todas las instancias).



5.4 - ODL y OQL

OQL

- Ejemplo de definición de una extensión asociada y una clave.

```
class Proyecto(extent proyectos key id_proy)
```

```
class Empleado(extent empleados key dni)
```

```
class EmpleadoPlantilla(extent empleados_plantilla key numEmp)  
extends Empleado
```



5.4 - ODL y OQL

OQL

- La siguiente consulta obtiene los jefes de proyectos con fecha de inicio desde 2017.

```
SELECT p.jefe_proyecto  
FROM proyectos p  
WHERE p.f_inicio >= date '2017-01-01';
```

- Se utiliza como punto de entrada a la BD la extensión **proyectos** de la clase **Proyecto** con alias **p**.



5.4 - ODL y OQL

OQL

- En la consulta anterior el resultado es un objeto de tipo `bag<EmpleadoPlantilla>`.
- En un bag puede haber elementos repetidos.
- Se puede evitar usando `SELECT DISTINCT` y se obtendría un objeto del tipo `set<EmpleadoPlantilla>`
- Se puede acceder a la propiedad de un objeto añadiendo un punto y el nombre de la propiedad.

```
SELECT DISTINCT p.jefe_proyecto.num_emp  
FROM proyectos p  
WHERE p.f_inicio >= date '2017-01-01';
```



Recuperar el número de empleado de los jefes de proyecto: devuelve `set<String>`



5.4 - ODL y OQL

OQL

- Si se quisiera, además, el nombre de los jefes de proyecto, habría que recuperar ambas cosas en un tipo `struct(String, String)`.

```
SELECT DISTINCT struct(num: p.jefe_proyecto.num_emp, nom:  
p.jefe_proyecto.nom_emp)  
FROM proyectos p  
WHERE p.f_inicio >= date '2017-01-01';
```



Recuperar varias propiedades a través de un `struct<String, String>`



5.4 - ODL y OQL

CONSULTA Y MANIPULACIÓN CON JAVA BINDING

- OQL incluye cláusulas análogas a **GROUP BY**, **HAVING**, **ORDER BY** y operadores de agregación **min**, **max**, **count**, **sum**, **avg**.
- Una implementación del Java binding, conforme a ODMG 3.0, debe proporcionar una clase que implemente la interfaz **[org.odmg.Implementation](#)**.
- Esta tiene métodos para crear y recuperar BBDD (**[Database](#)**), transacciones (**[Transaction](#)**) y consultas OQL (**[OQLQuery](#)**).



5.4 - ODL y OQL

CONSULTA Y MANIPULACIÓN CON JAVA BINDING

- La manipulación de objetos persistentes se realiza según el principio de *persistencia transparente*
- No hay un lenguaje específico para la manipulación sino que se hace desde el lenguaje de programación y de la misma forma que los objetos no persistentes.



5.4 - ODL y OQL

CONSULTA Y MANIPULACIÓN CON JAVA BINDING

- El *language binding* garantiza que al hacer un *commit* se grabarán en la BBDD todos los cambios realizados sobre los objetos persistentes.
- También se grabarán los cambios realizados sobre objetos alcanzables siguiendo las referencias desde cualquier objeto persistente.
- Esto se conoce como *persistence by reachability* o *transitive persistence*.



5.4 - ODL y OQL

CONSULTA Y MANIPULACIÓN CON JAVA BINDING

- Las clase persistentes o *persistent-capable* se pueden generar mediante un preprocesador de ODL que genere su código a partir de definiciones ODL y posteriormente modificarlas.

5.5 - BD Matisse



Matisse
The Post-Relational SQL Database

5.5 - BD Matisse



Matisse
The Post-Relational SQL Database

- La BD Matisse es una de las más veteranas y con mejor soporte para ODMG 3.0.
- Es de pago pero tiene versión descargable tras registro.
- La empresa la denomina como posrelacional, pero lo que nos interesa es que se trata de una BDO.

5.5 - BD Matisse



Matisse
The Post-Relational SQL Database


- Soporta diversos aspectos del estándar ODMG 3.0:
- **ODL**: Conforme a ODMG 3.0 aunque con sintaxis distinta y algunas discrepancias.
- **OQL**: No implementa OQL de ODMG 3.0. Proporciona como alternativa un driver JDBC que permite recuperar colecciones de objetos mediante lo que llama SQL, una adaptación del SQL que también incluye sentencias **INSERT**, **UPDATE** y **DELETE**.
- **Language bindings**: a parte de los incluidos en ODMG 3.0 (C++, Smalltalk y Java) otros como C, C#, Eiffel, Perl, PHP, Python y Visual Basic.

5.5 - BD Matisse



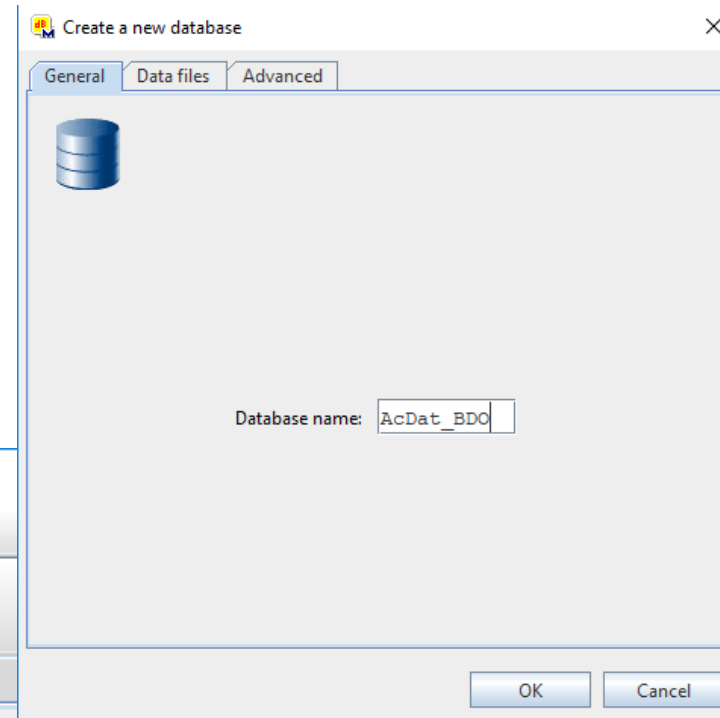
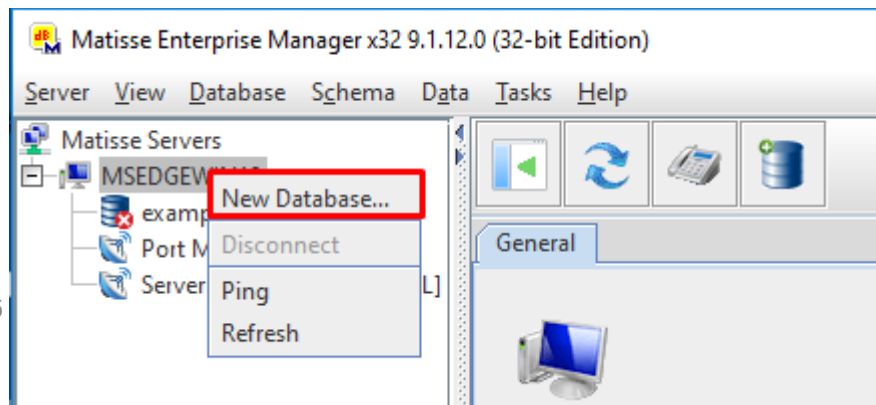
Matisse
The Post-Relational SQL Database

CREACIÓN DE UNA BD MEDIANTE ODL EN MATISSE

- Necesitaremos instalar el Enterprise Manager de Matisse.  matisse9112x32.exe

- Para crear la BD:

- ❖ Botón derecho, "New Database..."
- ❖ Nombre
- ❖ Seleccionada, clic sobre "Start"



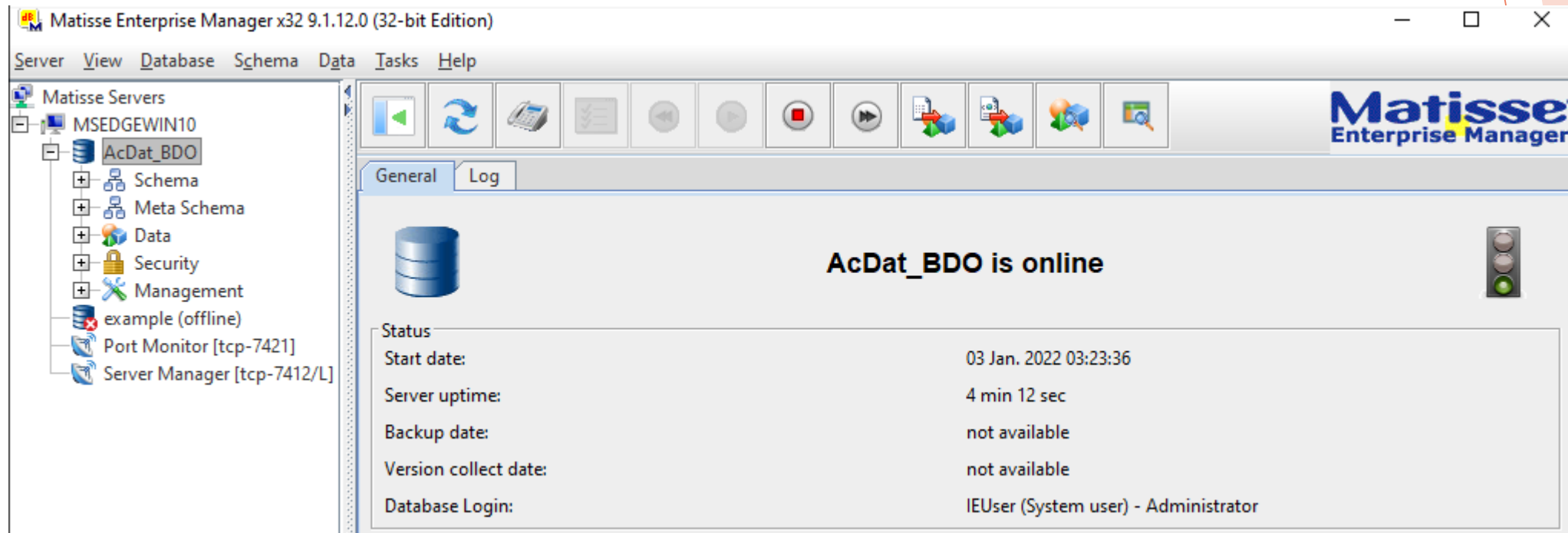
5.5 - BD Matisse



Matisse
The Post-Relational SQL Database

CREACIÓN DE UNA BD MEDIANTE ODL EN MATISSE

- BD arrancada





5.5 - BD Matisse

CREACIÓN DE UNA BD MEDIANTE ODL EN MATISSE

- Las sentencias ODL de Matisse para crear el esquema de objetos para ejemplo son las siguientes:

```
1 module gest_proyectos {
2
3
4   interface Proyecto: persistent
5   {
6     attribute String<32> nom_proy;
7     attribute Date f_inicio;
8     attribute Date Nullable f_fin;
9     relationship set<Empleado> tiene_asignado[0,-1]
10      inverse Empleado::asignado_a;
11     relationship EmpleadoPlantilla jefe_proyecto
12      inverse EmpleadoPlantilla::gestiona;
13   };
14
15   interface Empleado: persistent {
16     attribute String<9> dni;
17     attribute String<60> nom_emp;
18     relationship set<Proyecto> asignado_a[0,-1]
19      inverse Proyecto::tiene_asignado;
20     relationship DatosProfesionales tiene_datos_prof[0,1]
21      inverse DatosProfesionales::datos_prof_de;
22     mt_index Empleado_pk unique_key TRUE criteria { dni MT_ASCEND };
23     mt_index Empleado_i_nom_emp criteria { nom_emp MT_ASCEND };
24   };
```

```
25
26   interface DatosProfesionales: persistent {
27     attribute String<9> dni;
28     attribute String<2> categoria;
29     attribute Float sueldo_bruto_anual;
30     relationship Empleado datos_prof_de
31      inverse Empleado::tiene_datos_prof;
32     mt_index Empleado_DatosProf_pk unique_key TRUE criteria { dni MT_ASCEND };
33   };
34
35   interface EmpleadoPlantilla: Empleado: persistent {
36     attribute String<12> num_emp;
37     relationship set<Proyecto> gestiona
38      inverse Proyecto::jefe_proyecto;
39     mt_index EmpleadoPlantilla_i_dni unique_key TRUE criteria { dni MT_ASCEND };
40     mt_index EmpleadoPlantilla_i_nom_emp criteria { Empleado::nom_emp MT_ASCEND };
41   };
42
43   };
```



Matisse
The Post-Relational SQL Database

5.5 - BD Matisse

CREACIÓN DE UNA BD MEDIANTE ODL EN MATISSE

- Solo se admiten valores nulos con la opción **Nullable**.
- Por defecto la cardinalidad mínima en todos los extremos (*traversal path*) es 1 y no hay cardinalidad máxima si se define mediante colección (**set** o **list**)
- La cardinalidad se puede cambiar con **[mín, máx]**.
- Un valor -1 para **máx** indica que no hay cardinalidad máxima.

5.5 - BD Matisse



Matisse
The Post-Relational SQL Database

CREACIÓN DE UNA BD MEDIANTE ODL EN MATISSE

- Matisse proporciona un identificador único (OID) para cada objeto, que puede hacer las veces de clave primaria.
- Permite definir índices (`mt_index`) sobre un atributo o conjunto de atributos para acelerar las búsquedas.

5.5 - BD Matisse



Matisse
The Post-Relational SQL Database

CREACIÓN DE UNA BD MEDIANTE ODL EN MATISSE

- Los índices pueden ser únicos (**unique_key**) y servir como clave primaria (Ejemplo en la clase Empleado, sobre dni).

```
22      mt_index Empleado_pk unique_key TRUE criteria { dni MT_ASCEND };
```

- No se ha definido ningún índice único para Proyecto. En una BD relacional sí se hubiera necesitado como clave primaria.

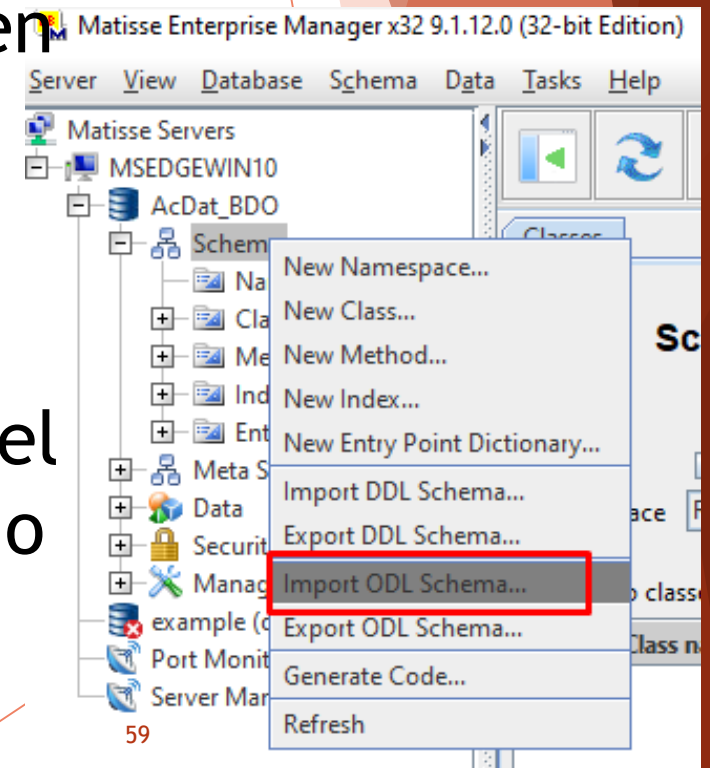
5.5 - BD Matisse



Matisse
The Post-Relational SQL Database

CREACIÓN DE UNA BD MEDIANTE ODL EN MATISSE

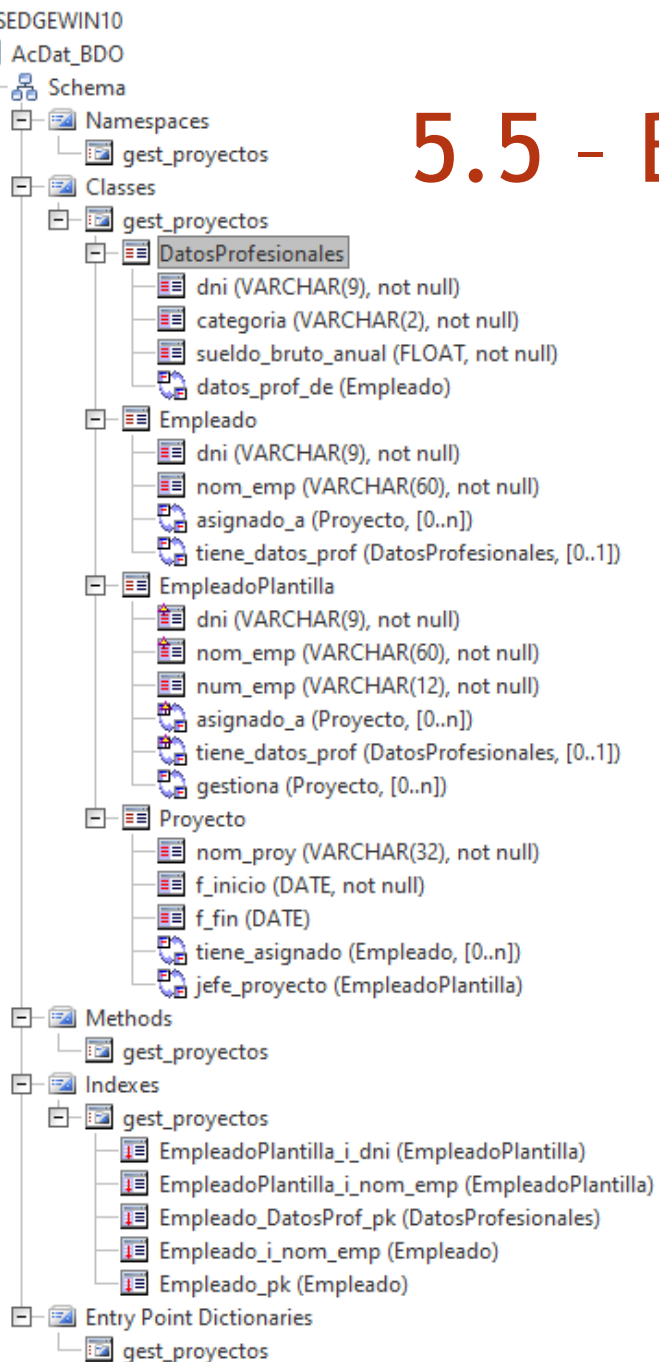
- El esquema de objetos se puede crear importando un fichero con las declaraciones en ODL.
- El nombre del módulo (gest_proyectos) es el namespace.
- El esquema se creará con el botón derecho del ratón sobre la BD recién creada, seleccionando “Schema”, “Import ODL Schema...”





5.5 - BD Matisse

CREACIÓN DE UNA BD MEDIANTE ODL EN MATISSE Esquema de objetos



Matisse
The Post-Relational SQL Database

5.5 - BD Matisse



Matisse
The Post-Relational SQL Database



ACTIVIDADES PROPUESTAS

- b) Define, utilizando el ODL de Matisse, el esquema de objetos para el modelo de objetos creado en la actividad anterior para publicaciones (libros y revistas), incluyendo todas las clases y los atributos indicados para ellas. Importa el esquema y visualízalo en el Enterprise Manager. Obtén el diagrama de clases con el Data Modeller.

5.5 - BD Matisse



Matisse
The Post-Relational SQL Database

Uso BD mediante Java binding

- Matisse tiene un Java binding muy bien documentado a través de la sección desarrolladores de su web.
- Matisse genera clases de Java para objetos persistentes (*stub clases*) a partir de sus definiciones en ODL.

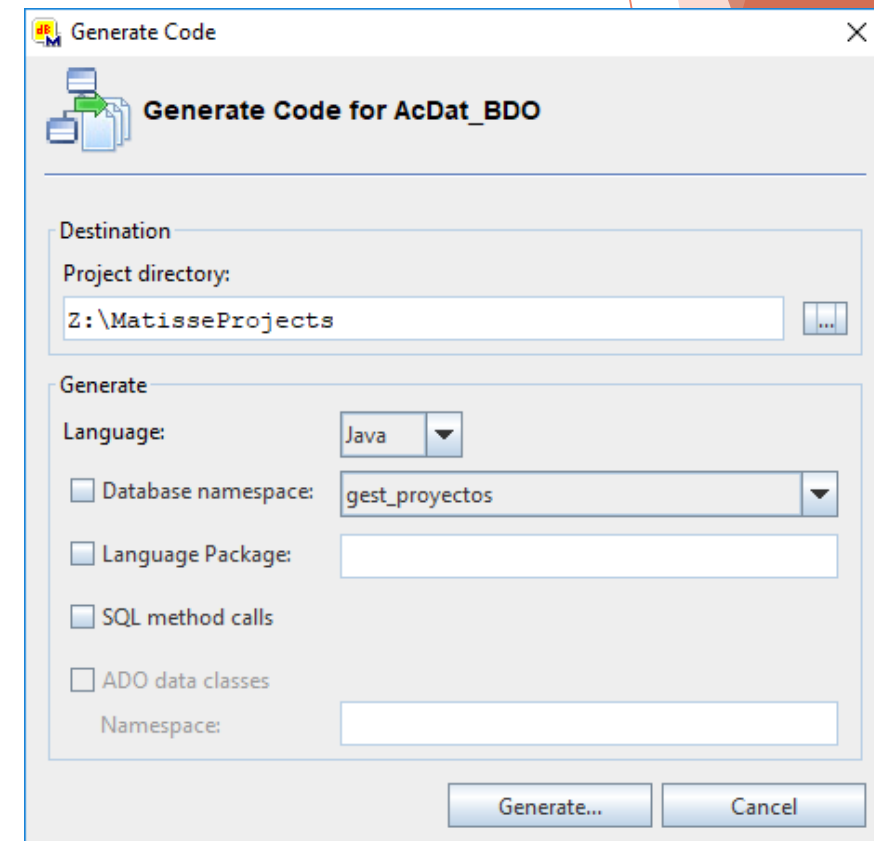
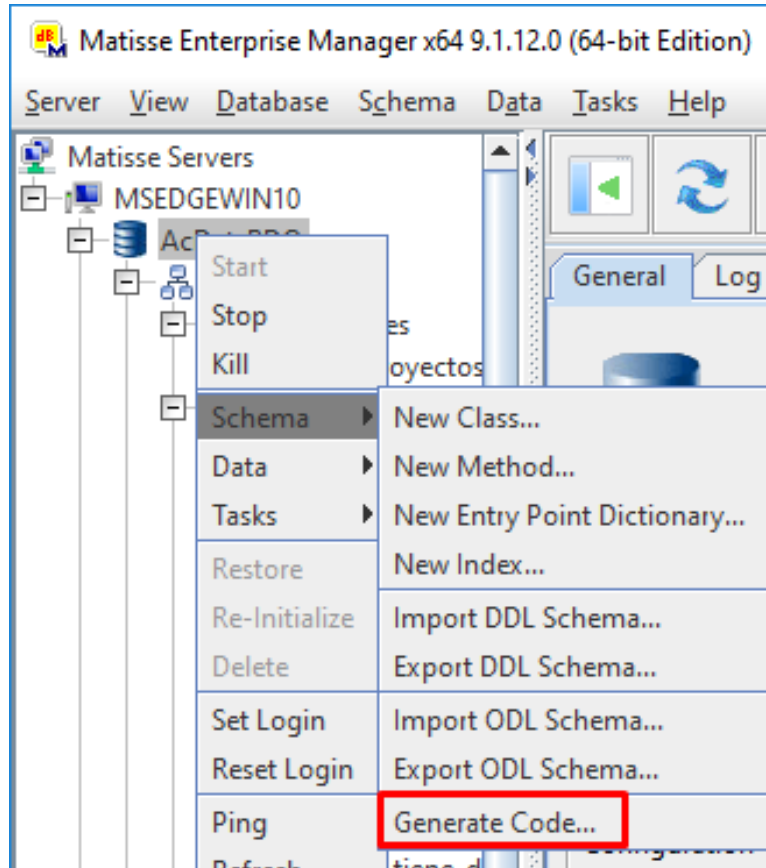
5.5 - BD Matisse



Matisse
The Post-Relational SQL Database

Uso BD mediante Java binding

- Botón derecho sobre la BD, “Schema”, “Generate Code”.



5.5 - BD Matisse



Matisse
The Post-Relational SQL Database

Uso BD mediante Java binding

- Las clases generadas contienen código que no hay que modificar entre comentarios `// BEGIN Matisse SDL Generated Code` y `// END of Matisse SDL Generated Code`.
- Este código se encarga de la persistencia de objetos.
- Fuera de ahí se pueden añadir métodos a las clases, recompilarlas, etc.

5.5 - BD Matisse



Matisse
The Post-Relational SQL Database

Uso BD mediante Java binding

- A continuación se indican los métodos más importantes disponibles en las stub clases generadas por Matisse para Java.
- Matisse no proporciona una implementación de la interfaz [org.odmg.Implementation](#) conforme al estándar ODMG 3.0.
- Matisse proporciona una clase [com.matisse.Mt-Database](#) en [matisse.jar](#).



5.5 - BD Matisse



Matisse
The Post-Relational SQL Database

Uso BD mediante Java binding

	Método	Funcionalidad
<i>Para cada clase. Son de tipo public static. Clase es el nombre de la clase. Permiten crear objetos persistentes e iterar sobre los objetos persistentes de la clase.</i>	Clase (MtDatabase db)	Crea un objeto de la clase. Clase indica el nombre de la clase.
	getInstanceNumber (MtDatabase db)	Obtienen el número de instancias de la clase existente en la BD. Aquellos métodos en cuyo nombre aparece Own solo tiene en cuenta las instancias de la propia clase, pero no de las subclases.
	getOwnInstanceNumber (MtDatabase db)	
	instanceliterator (MtDatabase db) ownInstanceliterator (MtDatabase db)	Obtiene iteradores para las instancias de la clase. El primero incluye instancias de subclases, el segundo no.



5.5 - BD Matisse



Matisse
The Post-Relational SQL Database

Uso BD mediante Java binding

	Método	Funcionalidad
<i>Para índices. Son de tipo public static. Permiten acceder directamente a objetos conociendo el valor de determinados atributos. Índice es el nombre de un índice.</i>	Clase lookupÍndice (MtDatabase db, TipoAtrib1 valorAtrib1, TipoAtrib2 valorAtrib2, ...)	Recupera objetos dados los valores para atributos incluidos en el índice. Los métodos del primer tipo permiten recuperar un objeto, y son especialmente útiles para índices únicos. Los del segundo tipo permiten recuperar varios objetos, y son de utilidad para índices no únicos.
	Clase [] lookupObjectsÍndice (MtDatabase db, TipoAtrib1 valorAtrib1, TipoAtrib2 valorAtrib2, ...)	
	ÍndiceIterator (MtDatabase db, TipoAtrib1 valorAtrib1, TipoAtrib2 valorAtrib2, ...)	Devuelve un iterador sobre objetos. Existe otra función que permite especificar opciones adicionales, y que no se describe aquí.
<i>Para cada atributo. Atr es el nombre del atributo.</i>	getAtr() setAtr()	Métodos getter y setter.
	removeAtr()	Elimina el valor del atributo y le asigna su valor por defecto.
	isAtrNull() isAtrDefault()	Comprueban si el valor del atributo es nulo (null) y si es el valor definido por defecto.



5.5 - BD Matisse



Matisse
The Post-Relational SQL Database

Uso BD mediante Java binding

	Método	Funcionalidad
<i>Para todas las relaciones. Rel es el nombre de la relación.</i>	clearRel ()	Elimina la relación con todos los sucesores según la relación. Un sucesor de un objeto según una relación es un objeto con el que está relacionado de acuerdo a esa relación. No borra los sucesores, solo elimina la relación del objeto en cuestión con ellos.
<i>Para relaciones con cardinalidad máxima 1. TipoSuc es el tipo de los sucesores según la relación.</i>	getRel () setRel (Tiposuc succ)	El primer método obtiene el sucesor según la relación. El segundo método establece la relación con un objeto dado.



5.5 - BD Matisse



Matisse
The Post-Relational SQL Database

Uso BD mediante Java binding

	Método	Funcionalidad
Para relaciones con cardinalidad máxima mayor que 1	<i>TipoSuc[]</i> getRel ()	Obtiene en un array sucesores según la relación Rel
	relliterator ()	Obtiene Iterador para sucesores según la relación Rel
	getRelSize ()	Obtiene número de sucesores según la relación Rel
	setRel (<i>TipoSuc[]</i> sucesores)	Asigna los sucesores según la relación Rel.
	prependRel (<i>TipoSuc</i> sucesores)	Añade sucesor según relación Rel en primera posición.
	appendRel (<i>TipoSuc</i> sucesores)	Añade sucesor según relación Rel en última posición.
	appendRel (<i>TipoSuc[]</i> sucesores)	Añade multiples sucesores según relación Rel.
	removeRel (<i>TipoSuc</i> sucesores)	Elimina sucesor según la relación Rel.
	removeRel (<i>TipoSuc[]</i> sucesores)	Elimina un sucesor según la relación Rel.

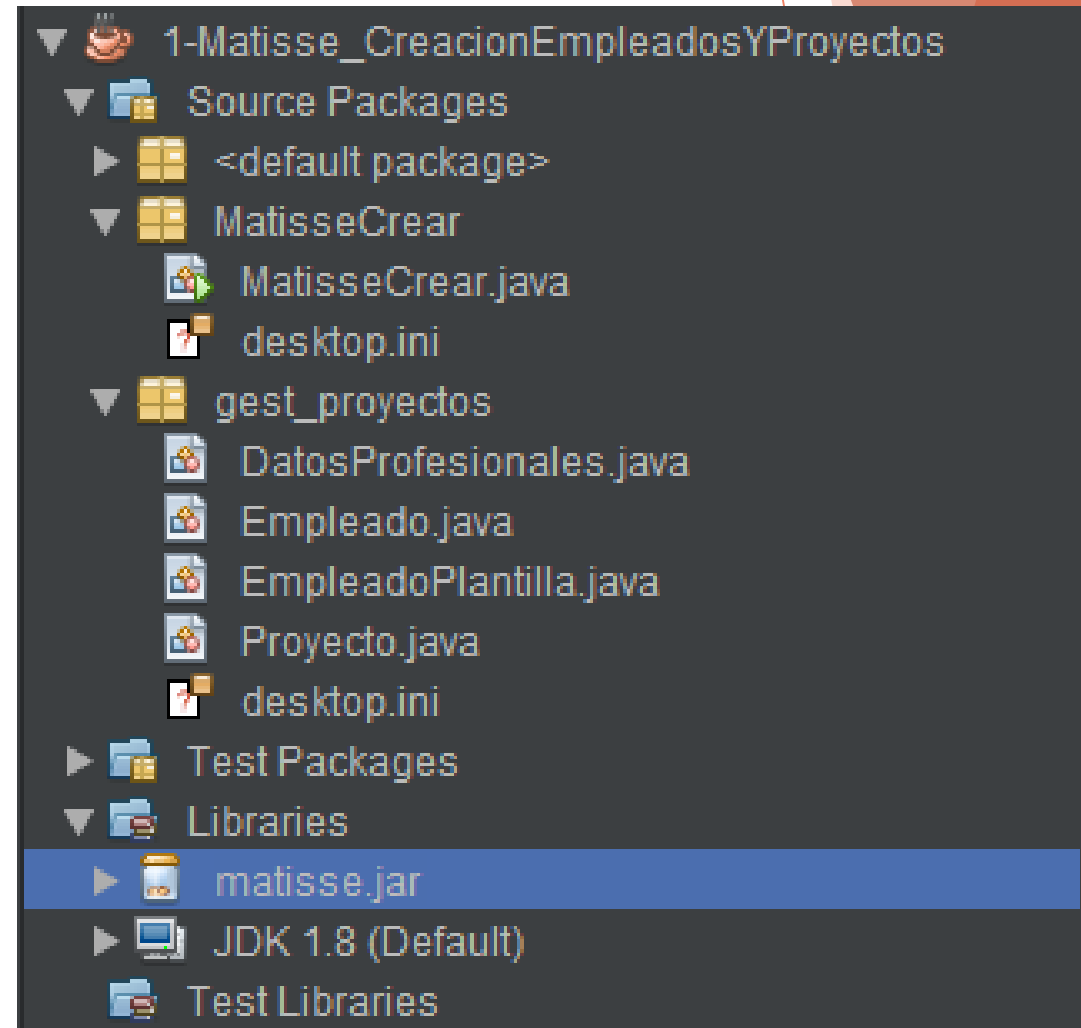
5.5 - BD Matisse



Matisse
The Post-Relational SQL Database

Uso BD mediante Java binding

- Para construir un programa que utilice Java binding hay que incluir en el proyecto **matisse.jar** (en la carpeta lib la instalación de Matisse) y las *stub classes*.
- **NOTA:** puede ser necesario incluir el path de la librería **matisse.jar** en las variables del sistema.



5.5 - BD Matisse



Matisse
The Post-Relational SQL Database

Uso BD mediante Java binding: EJEMPLO 1

- El siguiente programa abre una conexión y crea un proyecto y varios empleados, tanto de plantilla como no de plantilla, que se asignan a un proyecto.

```
1 package MatisseCrear;
2
3 import com.matisse.MtDatabase;
4 import com.matisse.MtException;
5 import gest_proyectos.*;
6
7 public class MatisseCrear {
8     public static void main(String[] args) {
9         try(MtDatabase db = new MtDatabase("localhost", "AcDat_BDO")) {
10             db.open();
11             db.startTransaction();
12             Proyecto pl = new Proyecto(db);
13             pl.setNom_proy("PAPEL ELECTRÓNICO");
14             pl.setF_inicio(new java.util.GregorianCalendar(2018,12,01));
15             EmpleadoPlantilla jpl = new EmpleadoPlantilla(db);
16             jpl.setDni("78901234X");
17             jpl.setNom_emp("NADALES");
18             jpl.setNum_emp("604202");
19             pl.setJefe_proyecto(jpl);
20             Empleado el=new Empleado(db);
21             el.setDni("56789012B");
22             el.setNom_emp("SAMPER");
23             pl.appendTiene_asignado(el);
24             EmpleadoPlantilla e2=new EmpleadoPlantilla(db);
```

Apertura de bd y
creación de
transacción

Gestión de relación entre
objetos

5.5 - BD Matisse



Matisse
The Post-Relational SQL Database

Uso BD mediante Java binding: EJEMPLO 1

- El siguiente programa abre una conexión y crea un proyecto y varios empleados, tanto de plantilla como no de plantilla, que se asignan a un proyecto.

```
24 EmpleadoPlantilla e2=new EmpleadoPlantilla(db);
25 e2.setDni("76543210S");
26 e2.setNom_emp("SILVA");
27 e2.setNum_emp("753014");
28 DatosProfesionales dp2 = new DatosProfesionales(db);
29 dp2.setDni("76543210S");
30 dp2.setCategoria("B1");
31 dp2.setSueldo_bruto_anual((float) 45200.00);
32 e2.setTiene_datos_prof (dp2);
33 pl.appendTiene_asignado(e2);
34 Empleado e3=new Empleado(db);
35 e3.setDni("89012345E");
36 e3.setNom_emp("ROJAS");
37 db.commit();
38 }
39 catch (MtException mte)
40 {
41     System.out.println("MtException : " + mte.getMessage());
42 }
43 }
44 }
```

Gestión de relación entre objetos

Confirmar la transacción

5.5 - BD Matisse



Matisse
The Post-Relational SQL Database



ACTIVIDADES PROPUESTAS

- c) Añade un método **float subeSueldoBruto (float incr)** a la clase **Empleado**, al que se le pase el porcentaje de incremento. Si el sueldo bruto está definido, debe devolver el nuevo sueldo, y null en otro caso. Escribe y verifica un programa que actualice el sueldo bruto para un empleado utilizando el nuevo método.

5.5 - BD Matisse



Matisse
The Post-Relational SQL Database



ACTIVIDADES PROPUESTAS

- d) Piensa/investiga/experimenta/verifica: todos los objetos creados por el programa del ejemplo anterior se crean como objetos persistentes. ¿Cómo se crearían objetos transitorios (es decir, no persistentes) de las mismas clases? ¿Es posible que un objeto creado como transitorio se acabe almacenando en la base de datos? ¿En qué casos? Justifica tus respuestas y verifícalas, en su caso, mediante un programa.

5.5 - BD Matisse



Matisse
The Post-Relational SQL Database



ACTIVIDADES PROPUESTAS

- e) Genera las *stub classes* para el esquema de objetos creado en la una actividad anterior para publicaciones. Crea un programa en Java que cree al menos tres libros, dos de ellos del mismo autor, y dos revistas en la base de datos y que, para terminar, muestre toda la información acerca de todos los objetos creados.

5.5 - BD Matisse



Matisse
The Post-Relational SQL Database

Uso BD mediante Java binding

- El *Object Browser* del Enterprise Manager de Matisse permite visualizar los objetos creados y las relaciones entre ellos.
- Desde un objeto se puede acceder a los relacionados siguiendo los traversal path.

The screenshot shows the 'General' tab of the Matisse Object Browser. The 'Select Objects' section is active, showing the following configuration:

- In snapshot:** Latest Version
- In namespace:** qest_proyectos
- From class:** ☐ only
- With result type:** Proyecto
- Where:** ☐ (OID =)
- Order by:** ☐ (nom_proy ASC)
- Limit:** ☐
- Offset:** ☐

The SQL statement field contains the following query:

```
SELECT REF(Proyecto) FROM Proyecto c
```

5.5 - BD Matisse



Matisse
The Post-Relational SQL Database

Object	Value	Type
[-] selection	{Length=1}	Proyecto[]
[-] [0]	{0x10a8 Proyecto}	Proyecto
[-] nom_proy	PAPEL ELECTRÓNICO	MT_STRING
[-] f_inicio	2019-01-01	MT_DATE
[-] f_fin	MT_NULL	MT_DATE
[-] tiene_asignado	{Length=2}	Empleado[]
[-] [0]	{0x10aa Empleado}	Empleado
[-] dni	56789012B	MT_STRING
[-] nom_emp	SAMPER	MT_STRING
[+] asignado_a	{Length=1}	Proyecto[]
[-] tiene_datos_prof	{Length=0}	DatosProfesionales
[-] [1]	{0x10ab EmpleadoPlantilla}	EmpleadoPlantilla
[-] dni	76543210S	MT_STRING
[-] nom_emp	SILVA	MT_STRING
[-] num_emp	753014	MT_STRING
[+] asignado_a	{Length=1}	Proyecto[]
[+] tiene_datos_prof	{Length=1}	DatosProfesionales
[-] gestiona	{Length=0}	Proyecto[]
[-] jefe_proyecto	{Length=1}	EmpleadoPlantilla
[-] [0]	{0x10a9 EmpleadoPlantilla}	EmpleadoPlantilla
[-] dni	78901234X	MT_STRING
[-] nom_emp	NADALES	MT_STRING
[-] num_emp	604202	MT_STRING
[-] asignado_a	{Length=0}	Proyecto[]
[-] tiene_datos_prof	{Length=0}	DatosProfesionales
[+] gestiona	{Length=1}	Proyecto[]

5.5 - BD Matisse



Matisse
The Post-Relational SQL Database

Uso BD mediante Java binding: EJEMPLO 2

- El siguiente programa borra y modifica algunos objetos.
- También elimina algunas relaciones entre objetos y utiliza iteradores de varios tipos.

```
1 package MatisseBorradoObjRel;
2
3 import com.matisse.MtDatabase;
4 import com.matisse.MtException;
5 import com.matisse.MtObjectIterator;
6 import gest_proyectos.*;
7
8 public class MatisseModifBorrar {
9
10     public static void muestraProyecto(Proyecto p) {
11         System.out.println("Proyecto "+p.getNom_proy()+
12             "[OID: "+p.getMtOidToHexString()+"]");
13         System.out.println("-----");
14         System.out.println("Jefe proyecto: DNI: "+p.getJefe_proyecto().getDni()+
15             ", Nombre: "+p.getJefe_proyecto().getNom_emp());
16         System.out.println("Empleados:");
17         MtObjectIterator<Empleado> itEmp = p.tiene_asignadoIterator();
18         while(itEmp.hasNext()) {
19             Empleado e = itEmp.next();
20             System.out.println("DNI: "+e.getDni()+", Nombre: "+e.getNom_emp());
21         }
22     }
23
24     public static void main(String[] args) {
```

Iterador

5.5 - BD Matisse



Matisse
The Post-Relational SQL Database

Uso BD mediante Java binding: EJEMPLO 2

- Recupera empleados de plantilla a partir de su DNI con *lookupEmpleadoPlantilla_i_dni*, que se ha creado en *EmpleadoPlantilla* porque se ha definido el índice *EmpleadoPlantilla_i_dni*.

DIAGRAMA DE OBJETOS

```
interface EmpleadoPlantilla: Empleado: persistent {
    attribute String<12> num_emp;
    relationship set<Proyecto> gestiona
    inverse Proyecto::jefe_proyecto;
    mt_index EmpleadoPlantilla_i_dni unique_key TRUE criteria { dni MT_ASCEND };
    mt_index EmpleadoPlantilla_i_nom_emp criteria { Empleado::nom_emp MT_ASCEND };
};
```

```
24 public static void main(String[] args) {
25     try(MtDatabase db = new MtDatabase("localhost", "AcDat_BDO")) {
26         db.open();
27         db.startTransaction();
28         Proyecto p = new Proyecto(db);
29         p.setNom_proy("TINTA_HOLOGRÁFICA");
30         p.setF_inicio(new java.util.GregorianCalendar(2018,12,28));
31         EmpleadoPlantilla ep = // NADALES
32             EmpleadoPlantilla.lookupEmpleadoPlantilla_i_dni(db, "78901234X");
33         p.setJefe_proyecto(ep);
34         Empleado e1 = Empleado.lookupEmpleado_pk(db, "89012345E"); // ROJAS
35         e1.setNom_emp("ROSAS");
36         e1.appendAsignado_a(p);
37         Empleado e2 = Empleado.lookupEmpleado_pk(db, "76543210S"); // SILVA
38         e2.getTiene_datos_prof().remove();
39         e2.remove();
40         Empleado e3 = Empleado.lookupEmpleado_pk(db, "56789012B"); // SAMPER
41         e3.clearAsignado_a();
42         MtObjectIterator<Proyecto> itProy = Proyecto.instanceIterator(db);
43         while(itProy.hasNext()) {
44             Proyecto unProy = itProy.next();
45             muestraProyecto(unProy);
46         }
47         db.commit();
48     }
49     catch (MtException mte)
50     {
51         System.out.println("MtException : " + mte.getMessage());
52     }
53 }
```

5.5 - BD Matisse



Matisse
The Post-Relational SQL Database

Uso BD mediante Java binding: EJEMPLO 2

- Recupera empleados a partir de su DNI con *lookup_empleado_pk()* ya que DNI se ha creado como clave primaria (*unique_key*)

DIAGRAMA DE OBJETOS

```
interface Empleado: persistent {
    attribute String<9> dni;
    attribute String<60> nom_emp;
    relationship set<Proyecto> asignado_a[0,-1]
        inverse Proyecto::tiene_asignado;
    relationship DatosProfesionales tiene_datos_prof[0,1]
        inverse DatosProfesionales::datos_prof_de;
    mt_index Empleado_pk unique_key TRUE criteria { dni MT_ASCEND };
    mt_index Empleado_i_nom_emp criteria { nom_emp MT_ASCEND };
};
```

```
24 public static void main(String[] args) {
25     try(MtDatabase db = new MtDatabase("localhost", "AcDat_BDO")) {
26         db.open();
27         db.startTransaction();
28         Proyecto p = new Proyecto(db);
29         p.setNom_proy("TINTA_HOLOGRÁFICA");
30         p.setF_inicio(new java.util.GregorianCalendar(2018,12,28));
31         EmpleadoPlantilla ep = // NADALES
32             EmpleadoPlantilla.lookupEmpleadoPlantilla_i_dni(db, "78901234X");
33         p.setJefe_proyecto(ep);
34         Empleado el = Empleado.lookupEmpleado_pk(db, "89012345E"); // ROJAS
35         el.setNom_emp("ROSAS");
36         el.appendAsignado_a(p);
37         Empleado e2 = Empleado.lookupEmpleado_pk(db, "76543210S"); // SILVA
38         e2.getTiene_datos_prof().remove();
39         e2.remove();
40         Empleado e3 = Empleado.lookupEmpleado_pk(db, "56789012B"); // SAMPER
41         e3.clearAsignado_a();
42         MtObjectIterator<Proyecto> itProy = Proyecto.instanceIterator(db);
43         while(itProy.hasNext()) {
44             Proyecto unProy = itProy.next();
45             muestraProyecto(unProy);
46         }
47         db.commit();
48     }
49     catch (MtException mte)
50     {
51         System.out.println("MtException : " + mte.getMessage());
52     }
```

Iterador para obtener
todos los proyectos

Iterador para obtener
todos los proyectos

5.5 - BD Matisse



Matisse
The Post-Relational SQL Database

Uso BD mediante Java binding: EJEMPLO 2

- Antes de borrar un empleado es necesario borrar, si existen, sus datos profesionales. Si no, se produce una excepción, pues la cardinalidad mínima del *traversal path* `datos_prof_de` es 1.

DIAGRAMA DE OBJETOS

```
interface Empleado: persistent {  
    attribute String<9> dni;  
    attribute String<60> nom_emp;  
    relationship set<Proyecto> asignado_a[0,-1]  
    inverse Proyecto::tiene_asignado;  
    relationship DatosProfesionales tiene_datos_prof[0,1]  
    inverse DatosProfesionales::datos_prof_de;  
    mt_index Empleado_pk unique_key TRUE criteria { dni MT_ASCEND };  
    mt_index Empleado_i_nom_emp criteria { nom_emp MT_ASCEND };  
};
```

```
24 public static void main(String[] args) {  
25     try(MtDatabase db = new MtDatabase("localhost", "AcDat_BDO")) {  
26         db.open();  
27         db.startTransaction();  
28         Proyecto p = new Proyecto(db);  
29         p.setNom_proy("TINTA_HOLOGRÁFICA");  
30         p.setF_inicio(new java.util.GregorianCalendar(2018,12,28));  
31         EmpleadoPlantilla ep = // NADALES  
32             EmpleadoPlantilla.lookupEmpleadoPlantilla_i_dni(db, "78901234X");  
33         p.setJefe_proyecto(ep);  
34         Empleado el = Empleado.lookupEmpleado_pk(db, "89012345E"); // ROJAS  
35         el.setNom_emp("ROSAS");  
36         el.appendAsignado a(p);  
37         Empleado e2 = Empleado.lookupEmpleado_pk(db, "76543210S"); // SILVA  
38         e2.getTiene_datos_prof().remove();  
39         e2.remove();  
40         Empleado e3 = Empleado.lookupEmpleado_pk(db, "56789012B"); // SAMPER  
41         e3.clearAsignado_a();  
42         MtObjectIterator<Proyecto> itProy = Proyecto.instanceIterator(db);  
43         while(itProy.hasNext()) {  
44             Proyecto unProy = itProy.next();  
45             muestraProyecto(unProy);  
46         }  
47         db.commit();  
48     }  
49     catch (MtException mte)  
50     {  
51         System.out.println("MtException : " + mte.getMessage());  
52     }  
53 }
```

Iterador para obtener
todos los proyectos

5.5 - BD Matisse



Matisse
The Post-Relational SQL Database

Uso BD mediante Java binding: EJEMPLO 2

- Obtiene todos los proyectos con `instanceliterador()` de la clase `Proyecto`.
- Para cada proyecto obtiene los empleados asignados con el iterador `tiene_asignadoliterador()` de la clase `proyecto`, creado para el *traversal path* `tiene_asignado`.

MÉTODO `muestraProyecto()`

```
MtObjectIterator<Empleado> itEmp = p.tiene_asignadoliterador();  
while(itEmp.hasNext()) {  
    Empleado e = itEmp.next();  
    System.out.println("DNI: "+e.getDni()+"", Nombre: "+e.getNom_emp());  
}
```

```
24 public static void main(String[] args) {  
25     try(MtDatabase db = new MtDatabase("localhost", "AcDat_BDO")) {  
26         db.open();  
27         db.startTransaction();  
28         Proyecto p = new Proyecto(db);  
29         p.setNom_proy("TINTA_HOLOGRÁFICA");  
30         p.setF_inicio(new java.util.GregorianCalendar(2018,12,28));  
31         EmpleadoPlantilla ep = // NADALES  
32             EmpleadoPlantilla.lookupEmpleadoPlantilla_i_dni(db, "78901234X");  
33         p.setJefe_proyecto(ep);  
34         Empleado e1 = Empleado.lookupEmpleado_pk(db, "89012345E"); // ROJAS  
35         e1.setNom_emp("ROSAS");  
36         e1.appendAsignado_a(p);  
37         Empleado e2 = Empleado.lookupEmpleado_pk(db, "76543210S"); // SILVA  
38         e2.getTiene_datos_prof().remove();  
39         e2.remove();  
40         Empleado e3 = Empleado.lookupEmpleado_pk(db, "56789012B"); // SAMPER  
41         e3.clearAsignado_a();  
42         MtObjectIterator<Proyecto> itProy = Proyecto.instanceliterador(db);  
43         while(itProy.hasNext()) {  
44             Proyecto unProy = itProy.next();  
45             muestraProyecto(unProy);  
46         }  
47         db.commit();  
48     }  
49     catch (MtException mte)  
50     {  
51         System.out.println("MtException : " + mte.getMessage());  
52     }  
53 }
```

Iterador para obtener
todos los proyectos

Iterador para obtener
todos los proyectos

5.5 - BD Matisse



Matisse
The Post-Relational SQL Database



ACTIVIDADES PROPUESTAS

- f) Crea un programa que cree un nuevo libro y le asigne como autor uno de los autores ya existentes. El programa debe también crear un nuevo autor, que no tendrá ningún libro asociado. El programa debe borrar uno de los dos libros del autor para el que había dos libros. Por último, como comprobación, debe mostrar toda la información acerca de todos los objetos existentes. Se sugiere mostrar autor a autor, y para cada autor sus libros.

5.5 - BD Matisse



Matisse
The Post-Relational SQL Database



ACTIVIDADES PROPUESTAS

- g) Existe un método `deepRemove()` en cada clase que por defecto llama a `remove()`. Este se puede cambiar, cuando tenga sentido, para borrar objetos subordinados antes de llamar a `remove()` para borrar el propio objeto. Cambia `deepRemove()` en la clase `Empleado` para borrar antes los datos profesionales. Modifica el programa de ejemplo anterior para que utilice este método para borrar el empleado.



5.5 - BD Matisse



Matisse
The Post-Relational SQL Database

Consultas mediante el SQL de Matisse

- Matisse no proporciona soporte para OQL conforme a ODMG 3.0.
- En su lugar tiene un lenguaje para consulta, definición y manejo de datos al que llama SQL, similar al SQL, pero adaptado para BDO y un driver JDBC.
- El lenguaje tiene sentencias SELECT, INSERT, UPDATE y DELETE.



5.5 - BD Matisse



Matisse
The Post-Relational SQL Database

Consultas mediante el SQL de Matisse

- No hay que dejarse confundir por el nombre ni su similitud con el SQL estándar, así como por el hecho de que se denomine base de datos posrelacional.
- Matisse es una BDO y su lenguaje SQL no es realmente SQL.
- En la parte para desarrolladores de la web de Matisse existe gran cantidad de documentación.



5.5 - BD Matisse



Matisse
The Post-Relational SQL Database

Consultas mediante el SQL de Matisse

- La clase para el driver JDBC es `com.matisse.sql.MtDriver`.
- Se puede cargar el driver a la manera antigua con `Class.forName("com.matisse.sql.MtDriver")` o, lo más sencillo, obtener una conexión JDBC con `getJDBCConnection()` de `MtDatabase`.

```
Connection jdbcCon = db.getJDBCConnection();  
Statement stmt = jdbcCon.createStatement(); {  
:
```

5.5 - BD Matisse



Matisse
The Post-Relational SQL Database

Consultas mediante el SQL de Matisse: EJEMPLO 3

- En el siguiente ejemplo se crea un nuevo empleado y lo asigna a todos los proyectos de los que es jefe un determinado empleado, identificado por su DNI.

```
1 //Recuperación de datos mediante JDBC y modificaciones sobre datos obtenidos
2 package MatisseConsultaConJDBCModifConJavaBinding;
3
4 import com.matisse.MtDatabase;
5 import com.matisse.MtException;
6 import java.sql.Connection;
7 import java.sql.Statement;
8 import java.sql.ResultSet;
9 import java.sql.SQLException;
10 import gest_proyectos.*;
11
12 public class MatisseConsultaConJDBCModifConJavaBinding {
13
14     public static void muestraErrorSQL(SQLException e) {
15         System.err.println("SQL ERROR mensaje: " + e.getMessage());
16         System.err.println("SQL Estado: " + e.getSQLState());
17         System.err.println("SQL código específico: " + e.getErrorCode());
18     }
19
20     public static void main(String[] args) {
21         try {
22             MtDatabase db = new MtDatabase("localhost", "AcDat_BDO") {
23                 db.open();
24                 db.startTransaction();
25                 Empleado e = new Empleado(db);
26                 e.setDni("65432109F");
27                 e.setNom_emp("LUQUE");
```




5.5 - BD Matisse

Consultas mediante el SQL de Matisse: EJEMPLO 3

```
20 public static void main(String[] args) {
21     try (
22         MtDatabase db = new MtDatabase("localhost", "AcDat_BDO")) {
23         db.open();
24         db.startTransaction();
25         Empleado e = new Empleado(db);
26         e.setDni("65432109F");
27         e.setNom_emp("LUQUE");
28         try (
29             Connection jdbcCon = db.getJDBCConnection();
30             Statement stmt = jdbcCon.createStatement()) {
31             String commandText = "SELECT REF(p) FROM gest_proyectos.Proyecto p WHERE p.jefe_proyecto.dni='78901234X'";
32             try (ResultSet rset = stmt.executeQuery(commandText)) {
33                 Proyecto p;
34                 while (rset.next()) {
35                     p = (Proyecto) rset.getObject(1);
36                     System.out.println("Proyecto: " + p.getNom_proy() +
37                                     ", jefe: [" + p.getJefe_proyecto().getDni() + "] " +
38                                     p.getJefe_proyecto().getNom_emp());
39                     if (p.getJefe_proyecto().getDni().equals("78901234X")) {
40                         p.appendTiene_asignado(e);
41                         System.out.println("Asignado nuevo empleado.");
42                     }
43                 }
44             }
45             db.commit();
46         } catch (SQLException sqle) {
47             muestraErrorSQL(sqle);
48         }
49     } catch (MtException mte) {
50         System.out.println("MtException : " + mte.getMessage());
51     }
```

Conexión con JDBC

Consulta

- Para obtener los proyectos como objetos hay que utilizar en la consulta **REF(p)**, que devuelve una referencia a cada objeto de la clase **Proyecto** que se obtiene con **getObject()**.



Matisse
The Post-Relational SQL Database

5.5 - BD Matisse

Consultas mediante el SQL de Matisse: EJEMPLO 3

- El planteamiento con una BD estrictamente conforme a ODMG 3.0 sería análogo:
 - Recuperar los proyectos mediante una consulta OQL
 - Hacer los cambios pertinentes sobre ellos.
 - Confirmar los cambios.

RESUMEN



- ✓ Como solución al desfase objeto-relacional, han surgido diversas soluciones. Entre ellas, las BDO, BDOR y la correspondencia objeto-relacional.
- ✓ Las BDO permiten almacenar directamente objetos.
- ✓ La organización ODMG desarrolló estándares para BDO, entre ellos están ODL y OQL. No se incluye entre ellos ningún lenguaje para manipulación de objetos, sino *language bindings* para lenguajes de POO (entre ellos Java), que hacen posible la persistencia transparente.

RESUMEN



- ✓ La persistencia transparente permite gestionar de igual manera los objetos persistentes y transitorios, y en el momento de confirmar una transacción se reflejan los cambios realizados sobre los objetos en la base de datos.
- ✓ En SQL:99 se introdujeron tipos estructurados definidos por el usuario, entre los que están los tipos objetos (clases), además de tipos de colecciones, tablas anidadas y referencias.
- ✓ Las BDOR, como Oracle, se basan en SQL:99 y amplían la funcionalidad de una base de datos relacional con tipos estructurados definidos por el usuario, entre los que están los objetos.

Acceso a Datos

FIN DE LA UNIDAD
GRACIAS