Andres Yassier Rivero Ledo

Haz un resumen del tema 4 (Resumen bases de datos objeto-relacionales y orientadas al objeto) en una página explicando en breves párrafos los siguientes conceptos:

Base de Datos Objeto-Relacional:
La Base de Datos Objeto-Relacional es una extensión de la base de datos relacional tradicional, a la cual se le añade características de la programación orientada a objetos.

Base de Datos Orientada Objetos:
Las Base de Datos Orientada a Objetos (BDOO) fueron diseñadas para que se puedan integrar directamente con aplicaciones desarrolladas con lenguajes orientado a objetos. Tiene las mismas características de la programación orientada a objetos como son encapsulación, identidad, herencia y polimorfismo, además del control de tipos y persistencia. También, el sistema de gestor de base de datos permitirá las características principales como son la persistencia, concurrencia, recuperación ante fallos, gestión de almacenamiento secundario y facilidad de consultas.

¿Qué es y de dónde sale ORD?

Son los objetos relacionales en base de datos. Aquellos que se pueden tratar como objetos sin tener necesariamente que almacenar información pero simulan las propiedades de un elemento de BD.

Pon ejemplos de código SGBD-OO de ODL y OQL, por ejemplo, HQL (Hibernate Query Language) o JPQL (Java Persistence Query Language)

Lenguaje ODL:

El lenguaje de definición de datos (ODL) en un SGBDOO es empleado facilitar la portabilidad de los esquemas de las bases de datos. Este ODL no es un lenguaje de programación completo, define las propiedades y los prototipos de las operaciones de los tipos, pero no los métodos que implementan esas operaciones.

```
class Persona                          //nombre de la clase (intención)
    (extent gente key idP) {           //nombre del archivo que contiene los datos (extensión)
                                       //si no se indica extent es interface
                                       //y opcional llave candidata del objeto(OID es automático)
    attribute idP int;                 //atributo atómico (int, real, char, string, boolean, enum)
    attribute nombre string;
    attribute StructDirec(string calle, string ciudad, dtring estado, string cp) direccion;
                                       //atributo tipo estructurado (struct, set, list, array, bag,
                                       // dictionary), se puede usar después como
                                       //       Persona::Direc
    attribute telefono string;
    string getNombre();                //método, sólo la firma, implementación en LP anfitrión,
                                       //los parámetros pueden ser IN, OUT ó IN/OUT
                                       //puede usar self
                                       //pueden sobrecargarse y sobrescribirse
    void setNombre(string nuevoNombre);
};
```

```
class Estudiante extends Persona //nombre de clase que hereda de Persona
    (extent estudiantes) {
    attribute creditos int;
    int getCreditos();
    void addCreditos(int numCreditos);
    relationship Set(<Grupo> tomaClase inverse Grupo::tieneEstudiantes;
                        //relación Estudiante toma un conjunto de clases en un Grupo
                        //y su inversa Grupo tieneEstudiantes Estudiante (M:M)
}
class Profesor extends Persona (extent prof) {
    attribute enum NivelProf {instructor, asistente, asociado, académico} nivel;
                                //atributo enumerado
attribute salario real(8,2);
string getNivel();
real getSalario();
relationship Departamento perteneceA Inverse Departamento::tieneProfesor;
                        //relación bidireccional Profesor perteneceA un Depto
                        //y su inversa Departamento tieneProfesor Profesor (1:M)
relationship Set<Grupo> daClaseEn Inverse Grupo::tieneProfesor;
                        //relación Profesor daClase en un conjunto de Grupo
                        //y su inversa Grupo tieneProfesor profesor
```

```
class Grupo (extent grupos) {
    attribute grupo string;
    relationship Set<Estudiante> tieneEstudiantes inverse Estudiante::tomaClase;
    relationship Profesor tieneProfesor inverse Profesor::daClaseEn;
}
```

Lenguaje OQL

El lenguaje de consulta propuesto por ODMG-93, presenta las siguientes características

* No es computacionalmente completo. Sin embargo, las consultas pueden invocar métodos, e inversamente los métodos escritos en cualquier lenguaje de programación pueden incluir consultas.
* Tiene una sintaxis abstracta.
* Su semántica formal puede definirse fácilmente.
* Proporciona un acceso declarativo a los objetos.
* Se basa en el modelo de objetos de ODMG-93.
* Tiene una sintaxis concreta al estilo SQL, pero puede cambiarse con facilidad.
* Puede optimizarse fácilmente.

* No proporciona operadores explícitos para la modificación, se basa en las operaciones definidas sobre los objetos para ese fin.
* Proporciona primitivas de alto nivel para tratar con conjuntos de objetos, pero no restringe su utilización con otros constructores de colecciones.

```
Select e.idP, e.creditos      //muestra todos los id y créditos en el archivo(extent) estudiantes
From estudiantes;

Select g.getNombre()          //muestra los nombres de todas las personas en extent gente
From g in gente;              //otra forma de alias

Select e.idP, e.tomaClase     //obtiene el conjunto de objetos relacionados al objeto
From estudiantes as e         //mediante la relación e.tomaclase, muestra todas las
Where e.idP = 'E999';         //clases que toma el estudiante E999

Select p.nombre               //encuentra registros por su referencia (d–var iterador)
From departamentos as d, d.tieneProfesores as p     //muestra los profesores del
Where d.nomDepto = 'Biología'                       //departamento de Biología
Order By p.nombre;                                  //ordenador por nombre
```

```
Select p.nombre               //misma consulta, usando subquery, obtiene todos los
From (Select d                //departamentos referenciados por el iterador b y
      From d in departamentos //por cada b obtiene un iterador p
      Where d.nomDepto = 'Biología') as b,
      b.tieneProfesor as p
Order By p.nombre;

Avg(Select p.salario          //obtiene el promedio de salarios de profesores que
    From prof as p            //pertenecen al departamento de historia
    Where p.perteneceA = 'Historia');

Select g                      //encuentra los grupos que tengan más de 30 estudiantes
From grupos as g
Where count(g.tieneEstudiantes > 30);
```

HQL (Hibernate Query Language)

El Hibernate Query Languaje (HQL) es el lenguaje de consultas que usa Hibernate para obtener los objetos desde la base de datos. Su principal particularidad es que las consultas se realizan sobre los objetos java que forman nuestro modelo de negocio, es decir, las entidades que se persisten en Hibernate. Ésto hace que HQL tenga las siguientes características:

- Los tipos de datos son los de Java.
- Las consultas son independientes del lenguaje de SQL específico de la base de datos
- Las consultas son independientes del modelo de tablas de la base de datos.
- Es posible tratar con las colecciones de Java.
- Es posible navegar entre los distintos objetos en la propia consulta.

Veamos ahora un sencillo ejemplo de consulta en HQL

```
SELECT c FROM Ciclo c ORDER BY nombre
```

¿Qué diferencias podemos ver entre HQL y SQL?

- Es necesario definir el alias `c` de la clase Java `Ciclo`.
- Tras la palabra `SELECT` se usa el alias en vez del "`*`".
- Al ordenar los objetos se usa la propiedad `nombre` de la clase `Ciclo` en vez de la columna `nombreCiclo` de la tabla `CicloFormativo`.

Respecto a la sensibilidad de las mayúsculas y minúsculas , el lenguaje HQL sí que lo es, pero con matices.

- Las palabras clave del lenguaje **NO** son sensibles a las mayúsculas o minúsculas.
  - Las siguientes 2 consultas son equivalentes.

```
select count(*) from Ciclo
SELECT COUNT(*) FROM Ciclo
```