Gestor de persistència genèric

TFC - Generació automàtica de programari

Antoni Sánchez Escamilla
ETIG
Consultor: Anna Queralt Calafat
Data: 18/06/2007

Dedicatòria

A les meves nenes, la meva filla i la meva dona, a la meva dona per fer tot lo possible per que aprofites el poc temps que tenia lliure per estudiar i a la meva petita per recordar-me dia a dia perquè tenia que posar-me les piles i acabar els estudis per poder gaudir mes d' ella. Espero començar a compensar-vos aviat.

Als meus pares que sempre m' han donat l' opció d' escollir encara que normalment ho feia equivocadament, espero que encara que una mica tard us pugui donar una alegria.

Resum

Actualment la programació orientada a objectes es la nota predominant, ningú dubta de les seves avantatges i es difícil trobar un desenvolupament nou que no la utilitzi. En la fase de disseny ens trobem amb el problema de aconseguir que aquestes classes siguin persistents i s' emmagatzemin habitualment en una base de dades relacional. Per aconseguir això ens veiem obligats a programar unes classes especifiques per fer de pont entre la classe i la taula que enregistrarà les seves instàncies, aquestes classes s' anomenen sovint *"Gestors de Disc*". Encara que existeixen eines per generar aquestes classes pont qualsevol modificació sobre la taula obligarà a regenerar la classe associada lo que acaba sent una manera de treballar no massa flexible. Si tenim un eina que ens permeti crear les classes necessàries per accedir a la base de dades sembla clar que aquesta serà una eina de **generació automàtica de programari**.

En aquest TFC es proposa utilitzar una única classe "*Gestor de Disc*" parametritzable per un fitxer, concretament un fitxer XML, que reculli les dades necessàries extretes de la base de dades, aquesta classe s' anomenarà "***TableMap"***. Es proposa obtenir la informació per omplir el fitxer XML a partir de les taules de diccionari de cada Sistema Gestor de Bases de Dades (a partir d' ara SGBD). Es marca com a objectiu que el fitxer XML ha de ser igual per qualsevol SGBD amb lo que aconseguirem independència de la plataforma.

Una vegada obtingut el fitxer XML l' utilitzarem com a paràmetre per instanciar un objecte a la classe Gestor de disc genèric que el treball s' anomena "*TableMap*". Posteriorment podrem treballar amb l' objecte instanciat amb mètodes que ens permetin afegir, esborrar i modificar instàncies de la classe i que realment llençaran sentencies SQL sobre la base de dades.

Índex

Dedicatoria ... 2
Resum ... 3
Índex ... 4
Introducció ... 5
- Justificació ... 5
- Objectius ... 6
  - Desenvolupament d' una jerarquia de classes per obtenir informació a partir del diccionari de la BDR. 6
  - Desenvolupament d' una classe que generi el fitxer xml de definició a partir de la informació recollida. ... 6
  - Desenvolupament classe per llegir el Xml i carregar les dades a memòria. ... 6
  - Desenvolupar gestor de disc genèric: Classe “TableMap”. ... 7
- Enfocament i mètode seguit ... 7
- Planificació ... 7
- Productes obtinguts ... 8
- Breu resum del contingut memòria ... 8

Requeriments ... 9
- Requeriments funcionals: ... 9
- Requeriments no funcionals ... 9

Disseny ... 10
- Casos d' us ... 10
- Diagrama de classes ... 11

Plataforma desenvolupament ... 13
Desenvolupament ... 14
- Obtenció d' informació a partir del diccionari: Classes DbInf, DbInfMysql i DbInformation ... 14
- Creació del fitxer Xml: Classe XmlTableMap ... 18
- Classe gestora comú: TableMap ... 19

Exemple d' us ... 20
- Creació XML ... 20
- Utilització TableMap ... 21

Capacitat d' extensió ... 23
Conclusions. ... 24
Glossari ... 25
Bibliografia ... 26
Annexos ... 27
- Disseny base de dades de proves. ... 27
- Exemple fitxer Xml generat: Obra.xml ... 27

Introducció

Justificació

Actualment existeixen multitud d' eines que generen classes per treballar amb taules de Bases de Dades Relacionals (a partir d' ara BDR) com si fossin objectes encapsulant internament les comandes sql per accedir a les dades. Aquestes eines acostumen a utilitzar el diccionari de la BDR per extreure la informació necessària mapejant els tipus de dades de la taules amb els tipus de dades corresponents en el llenguatge de programació escollit.

Una vegada generades les classes a partir del generador de codi podem crear les nostres classes heretant d' aquestes primeres, estenen la funcionalitat d' aquestes i afegint totes les capes necessàries per estructurar correctament la nostra aplicació.

Evidentment aquestes eines no son perfectes i es possible que tinguem que modificar les classes generades pel generador de codi per adaptar-nos a la realitat. Els problemes però apareixen quan es modifica la base de dades d' origen i ens veiem obligats a tornar a generar tota la classe, a sobreescriure els fitxers originals o bé a modificar manualment el codi de la classe per adaptar-lo a la nova situació. Una altre inconvenient d' aquest sistema es que per solucionar un bug o per implantar millores del generador de codi hem de tornar a generar totes les classes.

Tot i que en un projecte ben planificat i dissenyat els canvis de la base de dades haurien de ser mínims en la vida real son mes comuns de lo que voldríem.

Bona part dels problemes esmentats anteriorment es podrien solucionar separant la informació especifica obtinguda a partir del diccionari de la BDR del codi necessari per crear la capa d' accés a dades. Una opció possible podria ser el recopilar tota la informació necessària a partir del diccionari i emmagatzemar-la en fitxers XML. Podríem tenir una única classe que rebrà el fitxer XML de la taula corresponent com a paràmetre i s' encarregarà de crear la capa d' accés a dades. Tot i que tècnicament es perfectament possible obtenir al moment la informació del diccionari, i així ens estalviarem un pas, el treballar amb un fitxer xml de definició té una sèrie d' avantatges, per exemple:

- ✔ Elimina la necessitat de fer consultes al diccionari per cada operació amb lo que aconseguim un rendiment millor.
- ✔ Al ser un format fàcilment actualitzable facilita el manteniment posterior, per exemple, per afegir un camp que acabem d' afegir a la base de dades no cal tornar a generar el xml, tenim prou amb afegir la definició corresponent al xml.
- ✔ Es fàcilment extensible i podríem afegir atributs que ens siguin d' utilitat encara que no es pugin obtenir del diccionari de la base de dades. Un exemple d' això podria ser el afegir un atribut descripció o comentari del camp, treballant amb una base de dades Mysql podríem disposar d' aquest camp encara que el seu diccionari de dades no ho contempli.

- ✔ Ens dona la opció de crear el Xml a mà lo que podria ser útil al treballar amb bases de dades que o no tenen un diccionari consultable o bé no tenim el procediment per extreure les dades.

Objectius

Els objectius generals a aconseguir són els següents:

Desenvolupament d' una jerarquia de classes per obtenir informació a partir del diccionari de la BDR.

El projecte té aquest primer objectiu no fàcil d' assolir que consisteix en extreure del diccionari d' una BDR tota la informació que ens pugui resultar d' utilitat per generar el fitxer xml del qual es nutrirà la nostra classe "*TableMap*." Per cada taula sql haurem de extreure com a mínim la informació seguent:

– Informació dels camps de la taula. Tipus, tamany,requerit...

– Informació de clau primària.

– Informació de claus foranes.

– Altres dades d' interès. Comentaris de la base de dades, taules que la referencien ...

Desenvolupament d' una classe que generi el fitxer xml de definició a partir de la informació recollida.

Posteriorment a la captació d' informació es generarà un fitxer Xml per cada taula de base de dades localitzada. Aquest fitxer tindrà una estructura predefinida que haurà de permetre emmagatzemar tota la informació recollida, a més ha de ser fàcilment extensible per suportar l' addicció de nous items. D' aquesta manera la informació obtinguda i carregada en memòria passarà a ser persistent.

Desenvolupament classe per llegir el Xml i carregar les dades a memòria.

La classe genèrica abans de poder treballar amb una taula haurà de carregar la informació existent al fitxer Xml en memória. Per complir aquest objectiu desenvoluparem una classe que faci la lectura de tot el Xml i carregui la informació obtinguda en memòria utilitzant una classe creada a tal efecte.

Desenvolupar gestor de disc genèric: Classe "TableMap".

Aquesta classe haurà de tenir capacitat d' accedir a qualsevol taula de la BDR a partir de la informació enregistrada amb el xml corresponent, fent servir una terminologia similar a la utilitzada a les classes d' accés a dades. Per arribar a operar amb les dades d' una instancia d' aquesta classe que mapejarà els camps de la taula relacional necessitarem definir i implementar mètodes que crearan dinàmicament les sentencies SQL a utilitzar segons la informació obtinguda del fitxer Xml.

Enfocament i mètode seguit

S' ha enfocat el treball seguint el paradigma de la programació orientada a objectes, encara que al ser un desenvolupament per generació automàtica de programari el cicle clàssic de enginyeria del programari no es segueix amb massa fidelitat, no tenim recollida de requisits i l' anàlisis i el disseny s' han sol·lapat mes de lo habitual. En quan al desenvolupament s' ha intentat aprofitar al màxim les possibilitats de la herència en PHP encara que aquestes encara estan molt per sota de llenguatges com el C++ o el Java.

Planificació

La planificació es va realitzar en base a les tasques i objectius principals del projecte, i les dates d' entregues parcials (Pacs) es va fer coincidir la entrega de la segona entrega parcial (PAC2) amb la entrega del disseny del projecte juntament amb les classes necessàries per generar els arxius xml a partir de taules de Mysql .

Entre la segona i tercera entrega (PAC3) estava previst desenvolupar la classe o classes TableMap amb les funcionalitats necessàries per treballar amb Mysql. Per lo que en la tercera entrega (PAC3) es tenia la previsió de lliurar les classes TableMap amb la seva documentació. Entre la tercera i la memòria es tenia previst fer testing i generar aquesta documentació.

Mes concretament la planificació proposada era la següent:

TASQUES	Dates
Inici projecte	**28/02/2007**
Disseny	
Definició de requeriments	
Casos d' us	
Diagrama de classes	
Altres diagrames	
Desenvolupament	
Definició de l' entorn i llenguatge de programació escollits	
Codificació generació xml per Mysql	

Camps	
Altres	
Claus foranes	
Clau primària	
Entrega PAC2	**16/04/2007**
Codificació classe TableMap per Mysql	
Métode getById	
Métode Modify	
Métode Save	
Métode setField	
Métode getField	
Métode AddNew	
Métode Delete	
Métode DeleteById	
Entrega PAC3	**21/05/2007**
Proves de funcionament amb Mysql	
Escriure memòria i presentació	
Memòria + presentació	**18/06/2007**

Productes obtinguts

Aquestes serien les classes principals que hem obtingut a la finalització del projecte.

- ✔ Classe que obté la informació rellevant d' una taula a partir del diccionari de la base de dades, classe *"DbInformation"* . Tot i que es disposa d' una jerarquia de classes orientada a treballar amb el màxim número de BDRs en aquesta primera implementació només hi ha suport per Mysql.

- ✔ Classe "*XmlTableMap"* que enregistra la informació obtinguda amb "*DbInformation"* a un fitxer Xml.

- ✔ Classe *"XmlParse"* que fa la lectura del fitxer Xml.

- ✔ Classe *"TableMap"* que te la capacitat de treballar amb la taula representada al fitxer Xml.

Breu resum del contingut memòria

El primer pas serà comentar els requeriments que haurà de complir el programari a desenvolupar. Posteriorment, encara que aquest nos sigui un projecte convencional, es presentarà el disseny inicial que donarà les bases per realitzar el desenvolupament. Es presentarà la plataforma de desenvolupament utilitzada per portar a terme el projecte. En aquest punt ja estarem en condicions de entrar a definir en bastant detall les diferents parts del desenvolupament portat a terme. Per fixar les idees es mostrarà un exemple pràctic de funcionament. Finalment es comentarà la capacitat d' extensió de la classe *"TableMap"* i s' arribarà a unes conclusions al respecte del producte final.

Requeriments

Per la realització del projecte les classes a desenvolupar han de complir un conjunt de requisits tant funcionals com a no funcionals.

Requeriments funcionals:

- ✔ L' aplicació ha de disposar de classes i mètodes per extreure tota la informació rellevant de la base de dades.
- ✔ La informació extreta de la base de dades s' ha de poder emmagatzemar en un fitxer Xml de definició comú per qualsevol de les bases de dades suportares.
- ✔ Les crides per obtenir la informació de la base de dades haurien de ser iguales per qualsevol de les bases de dades suportares.
- ✔ El Xml resultant serà el principal parametre de la clase "*TableMap*" que permetrà operar sobre la taula especificada al fitxer Xml independentment del tipus de base de dades.
- ✔ La classe "*TableMap*" ha d' encapsular les típiques operacions sql amb mètodes de classe que puguin generar les sentencies sql de manera automàtica a partir de la informació recollida al Xml.

Requeriments no funcionals

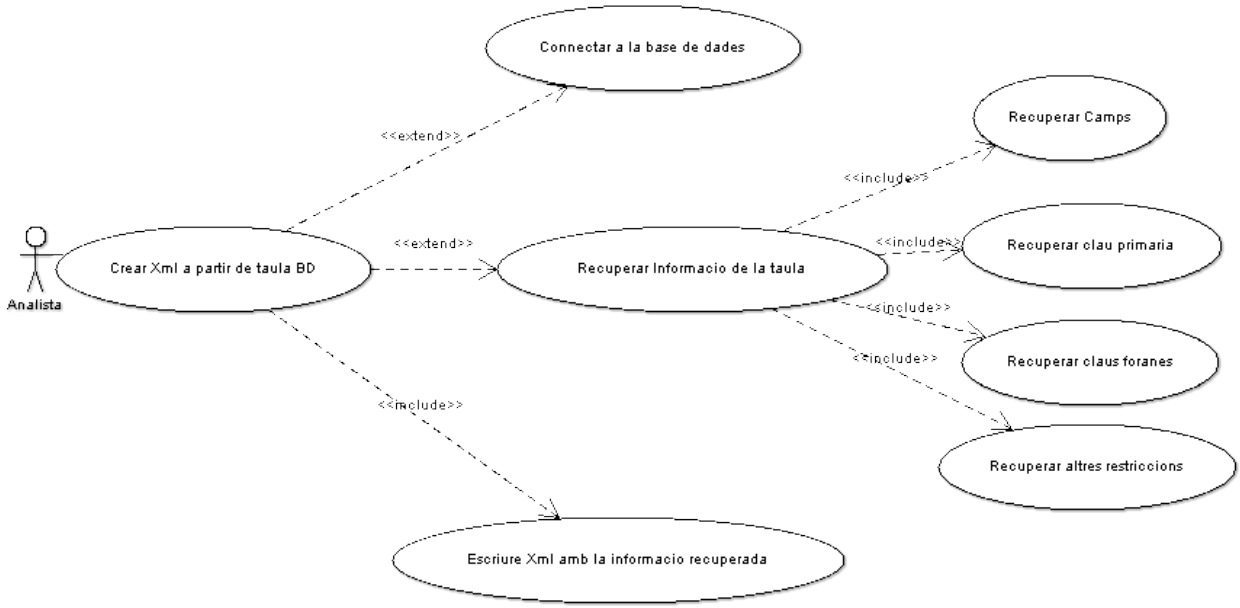
- ✔ Jerarquies de classes estructurades per suportar varies bases de dades i amb possibilitat d' escalar.
- ✔ Funcionament basat en software lliure.
- ✔ Requeriments mínims de hardware.
- ✔ Classes dissenyades per treballar amb aplicacions web.
- ✔ Multiplataforma.

Disseny

Casos d' us

Aquest es un projecte atípic del qual es difícil definir els casos d' us ja que l' actor en la majoria de casos es el analista o el programador. En tot cas podem identificar que el projecte consta de dues parts ben diferenciades com son:

- ✔ El desenvolupament de les classes necessàries per extreure les dades del SGBD i per crear el fitxer de definició en format XML. Aquesta part estaria representada en el diagrama següent:



- ✔ El desenvolupament de la classe o classes TableMap que implementaran el accés a les dades a partir del fitxer Xml de definició. Partim de la premissa que a nivell d' implementació es viable treballar amb una única classe *"TableMap*", independentment de la BDR utilitzada, encara que aquesta haurà de tenir el mitjans per generar el codi SQL adequat per cada BDR. D' aquesta part un diagrama de cas d' us no es entenedor per lo que només comentarem les funcionalitats principals que haurà de contemplar la classe.

  - **Inserir un nou registre.** A partir dels valors d' atribut (o camps) ja donats generar i executar la sentència SQL corresponent per afegir les dades.
  - **Esborrar el registre actiu.** A partir d' un registre ja carregat generar i executar la sentència SQL corresponent per esborrar-lo.

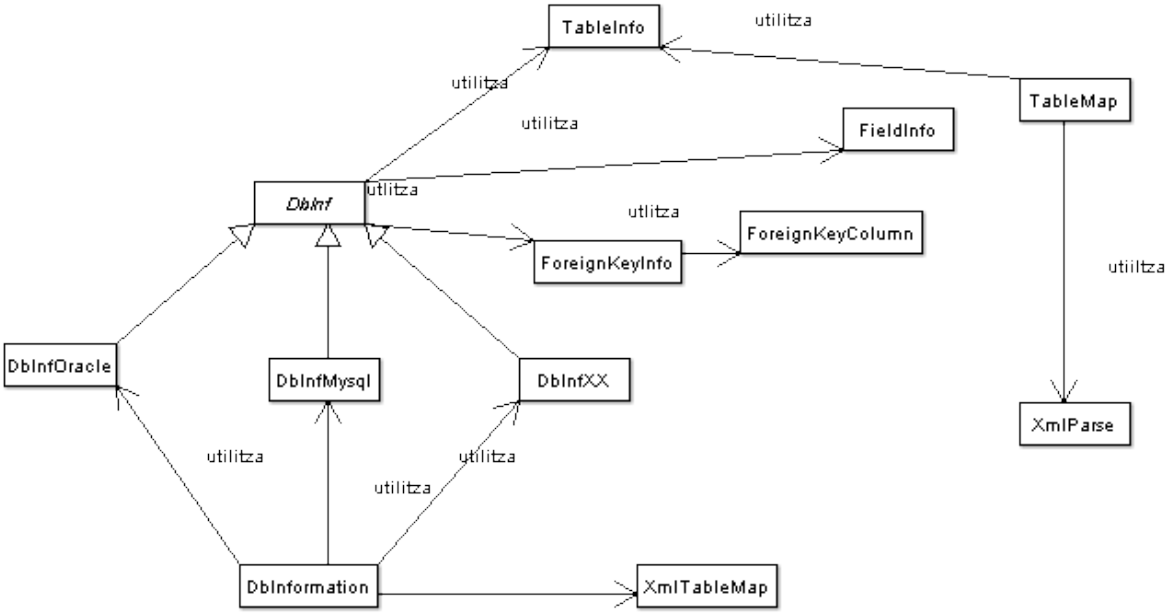
– **Esborrar el registre corresponent a la clau indicada.** Especificant com a paràmetre la clau primària generar i executar la sentència SQL corresponent per esborrar el registre que conté la clau donada.

– **Recuperar un registre a partir de la clau primària.** Especificant com a paràmetre la clau primària recuperar un registre mitjançant la sentència SQL corresponent.

– **Modificar un registre.** Poder realitzar modificacions en un o més camps del registre i fer-les persistents generant i executant la sentència SQL corresponent.

Diagrama de classes

El diagrama següent mostra les classes principals i les relacions entre elles. En la documentació generada amb el phpdocumentor (http://www.phpdoc.org) ,una utilitat similar al javadoc, es poden veure al detall les classes amb els seus atributs i mètodes.



La classe "*DbInf*" es una classe abstracte que conté en la seva majoria mètodes abstractes que hauran de ser implementats per les classes filles , tindrem una classe filla de "*DbInf*" per cada SGBD suportat.

Les classes "*TableInfo*", "*FieldInfo*", "*ForeignKeyInfo*" i "*ForeignKeyColumn*" donen suport a "*DbInf*" per permetre el emmagatzemament en memòria de la informació d' estructura estreta del SGBD. La classe "*DbInformation*" serà la utilitzada pel programador independentment del SGBD a utilitzar, aquesta farà servir la cadena de connexió per distingir entre un SGBD i un altre, i posteriorment farà la crida a la classe corresponent d' una manera transparent per l' usuari / programador.

La classe "*XmlTableMap*" s' encarregarà de escriure el Xml de definició segons la informació estreta en la utilització de la classe "*DbInformation*".

Com es pot veure la classe "*TableMap*" utilitzarà "*XmlParse*" i "*TableInfo*", la primera per recuperar la informació referent a la taula del fitxer Xml i la segona per emmagatzemar aquesta informació en memòria per millorar el rendiment.

Definim en mes detall la classe "TableMap" donada la seva importància ja que podem dir que la finalitat del projecte es arribar a implementar correctament les seves funcionalitats.

Atributs:

- ☑ **state**. Atribut per enregistrar l' estat actual de la instància de la classe, pot ser: A (Add), M (Modify) i S (Selected).
- ☑ **values**. Array associatiu amb els valors de cada camp.
- ☑ **connection**. Varible de connexio a la base de dades.
- ☑ **table**. Varible de tipus "*TableInfo*" que conté la estructura de la taula carregada del Xml.

Mètodes:

- ☑ **TableMap**. Constructor que rep la cadena de connexio a la bd i el fitxer XML de definició com a parametres.
- ☑ **setField**. Array associatiu amb els valors de cada camp.
- ☑ **getField**. Varible de connexio a la base de dades.
- ☑ **getById**. Varible de tipus "*TableInfo*" que conté la estructura de la taula carregada del Xml.
- ☑ **Modify**. Metode que prepara el objecte per modificar dades, estableix el estat a M (Modify).
- ☑ **Save**. Metode general per guardar les dades actuals del objecte actual cridant a SaveAdd si hem d' afegir un registre o SaveModify si l' hem de modificar, despres d' executar-se el objecte queda en estat S (Selected).
- ☑ **AddNew**. Metode que prepara el objecte per afegir dades, estableix el estat a A (Add) i esborra els valors del Array values.
- ☑ **Delete**. Metode que esborra el registre actual carregat per l' objecte i deixa el objecte en estat N (New).
- ☑ DeleteById. Mètode que esborra el registre corresponent a la clau passada com a paràmetre i deixa el estat com a N (New).

Plataforma desenvolupament

S' ha escollit treballar amb el llenguatge de script PHP, funcionant sobre Apache, que ja dona a partir de la versió 5, una bona cobertura a la programació orientada a objectes. Tot i que el PHP permet treballar amb molts tipus de bases de dades per aconseguir encapsular l' execució de les funcions especifiques per cada SGBD (en endavant Sistema Gestor de Bases de Dades) utilitzarem les llibreries PEAR (http://pear.php.net) concretament el paquet MDB2. Per treballar d' una manera eficaç amb els fitxer Xml es treballa amb una altre paquet del PEAR anomenat XML_Util.

Respecte als SGBD a utilitzar es donarà suport inicial a Mysql encara que s' enfocarà el projecte per aconseguir el màxim de compatibilitat possible entre diferents SGBDs.

Aquesta infraestructura (Apache+PHP+Mysql) te l' avantatge afegida de ser multiplataforma, podriem desenvolupar l' aplicació amb Windows i posar-la a producció amb la majoria de sistemes basats en Unix (inclòs Linux) sense modificar ni una sola línia de codi.

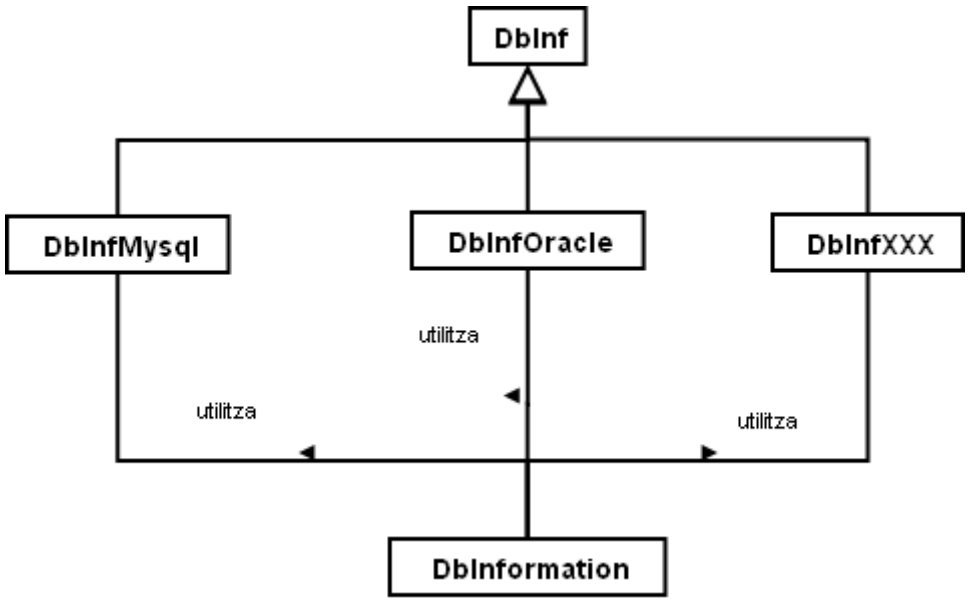
Respecte al hardware es una plataforma amb uns requeriments realment baixos, podriem tenir un obsolet Pentium III amb uns 256 Mb de memòria donant servei a mes de 30 usuaris amb un rendiment acceptable. Es una plataforma hardware que permet solucions amb servidors de tipus pc i a la vegada amb grans servidors ja siguin aquest amb arquitectura Intel o qualsevol altra.

Desenvolupament

En aquest capítol es comentaran els punts més importants del desenvolupament realitzat agafant com a base la versió final de les classes ja desenvolupades. No es pretén repassar un per un tots els atributs i mètodes sinó veure els punts mes importants per mostrar clarament la filosofia del treball. Per una referència complerta es pot consultar la documentació generada amb PHPdocumentor.

Obtenció d' informació a partir del diccionari: Classes DbInf, DbInfMysql i DbInformation

Com s' havia comentat anteriorment i es podia veure al diagrama de classes la classe *"DbInformation"* es la classe amb la qual treballarà el analista i/o programador per extreure la informació de forma de la base de dades que correspongui. Encara que aquesta no té el codi per extreure la informació de les base de dades s' encarrega de cridar a la classe corresponent segons la BDR a utilitzar. Les classes que codifiquen realment las consultes sobre el diccionari de la base de dades corresponent hereten de la classe abstracta "*DbInf*".



La classe *"DbInf"* es una classe abstracta però te codificat el constructor que s' encarrega de obrir la connexió amb la base de dades independentment del tipus d' aquesta. També codifica el mètode *"mapDataType*" que donat un tipus de dades de la BDR en qüestió retorna el tipus associat de les llibreries PEAR, aquest tipus serà el enregistrat al fitxer Xml (aquest mètode utilitza una taula de conversió especifica per cada BDR).

De les classe utilitzades per "*DbInformation" a*ctualment només tenim codificada la classe *"DbInfMysql"* que es la que s' han realitzat les proves sobre una base de dades Mysql 5.0. Comentem breument els mètodes mes importants.

El mètode getTables retorna una llista de les taules trobades a la base de dades.

```
public function getTables()
    {
        $str_sql="SELECT TABLE_NAME
                    FROM information_schema.`TABLES`
                     WHERE TABLE_SCHEMA='$this->database'";
        $conn=$this->connection;
        $res =&$conn->query($str_sql);

        $c=0;
        while (($row = $res->fetchRow())) {
            $tables[$c]=$row[0];
            $c++;
        }

        return $tables;
    }
```

El mètode getFields retorna els camps de la taula que li passem com a paràmetre.

```
public function getFields($tableName)
    {
        $str_sql="SELECT COLUMN_NAME
                      FROM information_schema.COLUMNS
                        WHERE TABLE_SCHEMA='$this->database' AND TABLE_NAME='$tableName'";

        $conn=$this->connection;
        $res =&$conn->query($str_sql);

        $c=0;
        while (($row = $res->fetchRow())) {
            $fields[$c]=$row[0];
            $c++;
        }
        $res->free();
        return $fields;
    }
```

El mètode getField carrega un objecte *"FieldInfo"* amb informació relativa al camp passat com a paràmetre.

```
public function getField($tableName,$fieldName) {
        $str_sql= "SELECT `IS_NULLABLE`, `DATA_TYPE`, `CHARACTER_MAXIMUM_LENGTH`,
`NUMERIC_PRECISION`, `NUMERIC_SCALE`
                    FROM information_schema.`COLUMNS`
                       WHERE TABLE_SCHEMA='$this->database' and TABLE_NAME='$tableName' and
COLUMN_NAME='$fieldName';";
        $conn=$this->connection;
        $res =&$conn->query($str_sql);
        $row=$res->fetchRow();
        $fieldInfo=new FieldInfo();
        $fieldInfo->setName($fieldName);
        $fieldInfo->setNull($row[0]);
        $fieldInfo->setDataType($this->mapDataType($row[1]));
```

```
        $fieldInfo->setCharMaxLen($row[2]);
        $fieldInfo->setNumPrecision($row[3]);
        $fieldInfo->setNumScale($row[4]);
        $res->free();
        return $fieldInfo;



    }
```

El mètode getPrimaryKey retorna un array amb els camps que formen part de la clau primària de la taula que li passem com a paràmetre.

```
public function getPrimaryKey($tableName)
    {

        $str_sql=" SELECT KC.`COLUMN_NAME`,KC.`ORDINAL_POSITION`
                        FROM information_schema.`KEY_COLUMN_USAGE` KC,
information_schema.`TABLE_CONSTRAINTS` TC
                        WHERE (
                         TC.`CONSTRAINT_NAME` = KC.`CONSTRAINT_NAME`
                          AND TC.`TABLE_NAME` = KC.`TABLE_NAME`
                            AND TC.`TABLE_SCHEMA` = KC.`TABLE_SCHEMA`
                            )
                            AND TC.`CONSTRAINT_TYPE` = 'PRIMARY KEY'
                            AND TC.`TABLE_SCHEMA` = '$this->database'
                            AND TC.`TABLE_NAME` = '$tableName'
                            ORDER BY KC.`ORDINAL_POSITION`";



        $conn=$this->connection;
        $res =&$conn->query($str_sql);

        $c=0;
        while (($row = $res->fetchRow())) {
            $fields[$c]=$row[0];
            $c++;
        }

        return $fields;
    }
```

El mètode getForeignKeys retorna una llista de les claus foranes de la taula que especifiquem.

```
public function getForeignKeys($tableName) {

        //Seleccionem les constraint de tipus FOREIGN KEY
        $str_sql="SELECT CONSTRAINT_NAME
                    FROM information_schema.TABLE_CONSTRAINTS TC
                    WHERE TC.`CONSTRAINT_TYPE` = 'FOREIGN KEY'
                    AND TC.`TABLE_SCHEMA` = '$this->database'
                    AND TC.`TABLE_NAME` = '$tableName'";

        $conn=$this->connection;
        //Obtenim las llista de claus foranes
        $res =&$conn->query($str_sql);

        $c=0;

        while (($row = $res->fetchRow())) {
            $foreignKeys[$c]=$row[0];
            $c++;
        }
```

```
        $res->free();

        return $foreignKeys;


    }
```

El mètode getForeignKey retorna un objecte *"ForeignKeyInfo"* amb la informació corresponent a la clau forana que passem com a paràmetre.

```
public function getForeignKey($tableName,$foreignName) {
            $foreignObject=new ForeignKeyInfo($foreignName);
            $conn=$this->connection;
            $str_sql="SELECT KC.`COLUMN_NAME`,KC.`ORDINAL_POSITION`,
KC.`REFERENCED_TABLE_NAME`, KC.`REFERENCED_COLUMN_NAME`
                        FROM information_schema.`KEY_COLUMN_USAGE` KC
                        WHERE KC.`TABLE_SCHEMA`='$this->database'
                             AND KC.`TABLE_NAME` = '$tableName'
                               AND KC.`CONSTRAINT_NAME`='$foreignName'
                          ORDER BY KC.`ORDINAL_POSITION`";

            //Recuperem la informacio de cada columna que composa la clau forana
            $res =&$conn->query($str_sql);
            $c=0;

            while (($row = $res->fetchRow())) {
                $foreignObjectColumn= new ForeignKeyColumn();
                $foreignObjectColumn->setColumnName($row[0]);
                $foreignObjectColumn->setPosition($row[1]);
                $foreignObjectColumn->setReferencedTable($row[2]);
                $foreignObjectColumn->setReferencedColumn($row[3]);

                $foreignObject->addForeignKeyColumn($foreignObjectColumn);
            }
            $res->free();
            //Guardem les dades de la forana en l' array

            return $foreignObject;


    }
```

En el constructor de la classe *"DbInformation"* s' analitza el tipus de base de dades i s' instancia una classe o un altre.

```
public function DbInformation($connectionString){
        $arDsn=MDB2::parseDSN($connectionString);
        $this->type=$arDsn['phptype'];

        switch ($this->type) {
            case 'mysql'||'mysqli':
                $this->dbinf=new DbInfMySql($connectionString);
                break;
            case 'oci8':
                die('Not implemented yet');
                break;

        }
    }
```

Com comentàvem anteriorment els mètodes de *"DbInformation"* criden al mètode de la classe corresponent.

```
public function getFields($tableName) {
        return $this->dbinf->getFields($tableName);
    }
```

Aquesta manera de treballar te l' avantatge que el programador sempre utilitzarà la mateixa classe independentment de la base de dades que utilitzi.

Creació del fitxer Xml: Classe XmlTableMap

La classe *"XmlTableMap"* s' encarrega de fer la creació del Xml de definició de taula amb la informació proporcionada per "*DbInformation".* Aquest seria un exemple d' us.

```
//Inicialitzem l' objecte per crear el fitxer xml
$xmltable=new XmlTableMap();

//Establim la cadena de connexio
$cadenaConnexio="mysql://root:@localhost/uoc2";

//Creem el Xml de la taula obra
$taula=”obra”;
$xmltable->createXml($cadenaConnexio,$taula,$SUBDIR."/xml/$taula.xml");


}
```

Com es pot observar al codi la classe "*XmlTableMap"* només necessita per construir el Xml la cadena de connexió (format package MDB2 de les llibreries PEAR), el nom de la taula i la ubicació a disc del fitxer destí.

El mètode createXml crida a la seva vegada a uns mètodes de visibilitat privada que tenen funcions mes especifiques.

```
public function createXml($connectionString,$tableName,$targetFile) {
        $this->dbinf=new DbInformation($connectionString);
        $this->tableName=$tableName;
        $sortida=$this->xmlHead();
        $sortida.=$this->xmlTableData();
        $sortida.=$this->xmlFieldData();
        $sortida.=$this->xmlPrimaryKeyData();
        $sortida.=$this->xmlForeignKeyData();
        $sortida.="</table>\n";


        $file=fopen($targetFile,"w");
        fwrite($file,$sortida);
        fclose($file);


    }
```

El mètode xmlTableData genera la part d' informació de la taula, el xmlFieldData la informació de tots els camps, el xmlPrimaryKeyData la de la clau primària i el xmlForeignKeyData la informació de les claus foranes.

Classe gestora comú: TableMap

En aquesta classe utilitzem d' alguna manera tot lo desenvolupat anteriorment per arribar a lo que considerem la finalitat del projecte: simplificar la persistència en programació orientada a objectes i bases de dades relacionals.

Al constructor li passem com a paràmetre un fitxer Xml amb la informació de la taula obtinguda a partir del "*DbInformation"* i creat per "*XmlTableMap",* i la cadena de connexió amb format paquet MDB2 de les llibreries PEAR.

```
public function TableMap($xmlFile,$connectionString){
        //Carregar dades del xml
        $parse = new XmlParse();
        $result = $parse->setInputFile($xmlFile);
        $result = $parse->parse();
        $this->table=$parse->getTableInfo();

        //Obrir la base de dades
        $conn =& MDB2::connect($connectionString);
        if (PEAR::isError($conn)) {
            die($conn->getMessage()."##".$conn->getUserInfo());
        }
        $this->connection=$conn;


    }
```

Es pot observar que la classe "*TableMap"* utilitza la classe "*XmlParse"* per llegir el fitxer Xml i que carrega la informació llegida a la classe "*TableInfo".* A mes també obrim la connexió amb la base de dades per poder enviar posteriorment sentències SQL.

La classe TableMap disposa d' un atribut anomenat "*state*" que te en tot moment l' estat del objecte instanciat A (Add), M (Modify) i S (Selected). Una altre atribut important es l' array "*values"* on es guarden temporalment els valors dels camps abans de cridar al mètode Save que executarà internament un Insert o un Update depenent de l' estat actual (atribut state).

Al capítol següent ,"Exemple d' us", trobem un exemple complert de la majoria de mètodes disponibles a la classe "*TableMap"* amb un cas real d' utilització.

Exemple d' us

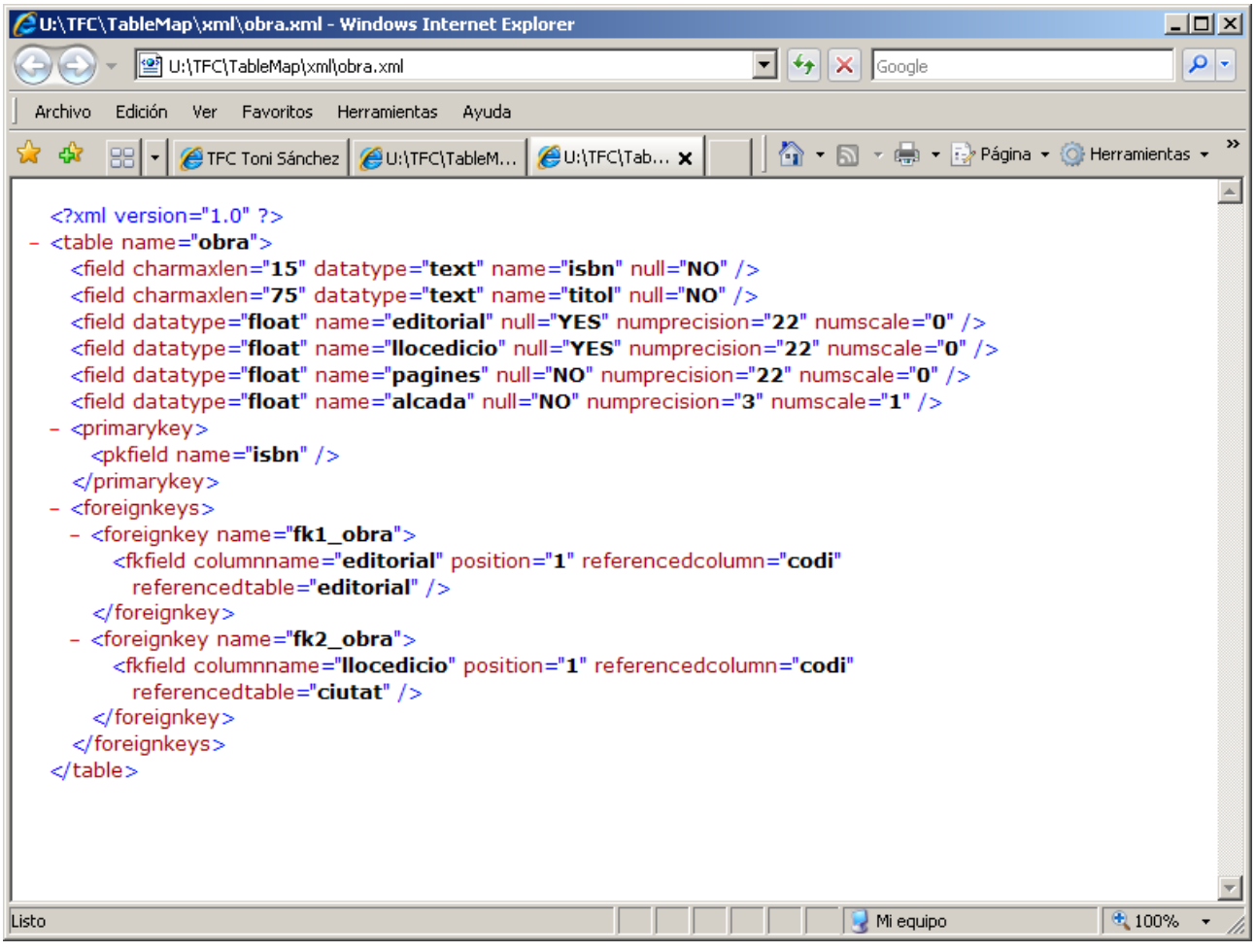
Creació XML

El primer pas per poder treballar amb la classe TableMap seria generar el XML corresponent a la taula/es que volem utilitzar. Amb el codi següent generaríem el XML de totes les taules de la base de dades que especifiquem.

```
//Inicialitzem l' objecte per crear el fitxer xml
$xmltable=new XmlTableMap();

//Establim la cadena de connexio
$cadenaConnexio="mysql://root:@localhost/uoc2";

//Obtenim la llista de taules
$inf=new DbInformation($cadenaConnexio);
$artaules=$inf->getTables();

//Per cada taula creem el Xml
for($i=0;$i<count($artaules);$i++){
  $taula=$artaules[$i];
  $xmltable->createXml($cadenaConnexio,$taula,$SUBDIR."/xml/$taula.xml");
  printf("Creat fitxer xml: %s mapeig de la taula: %s <br>","/xml/$taula.xml",$taula);

}
```

Si tot ha funcionat correctament tindrem a la carpeta /xml els fitxers generats.

Un dels fitxers generats seria el de la taula obra.

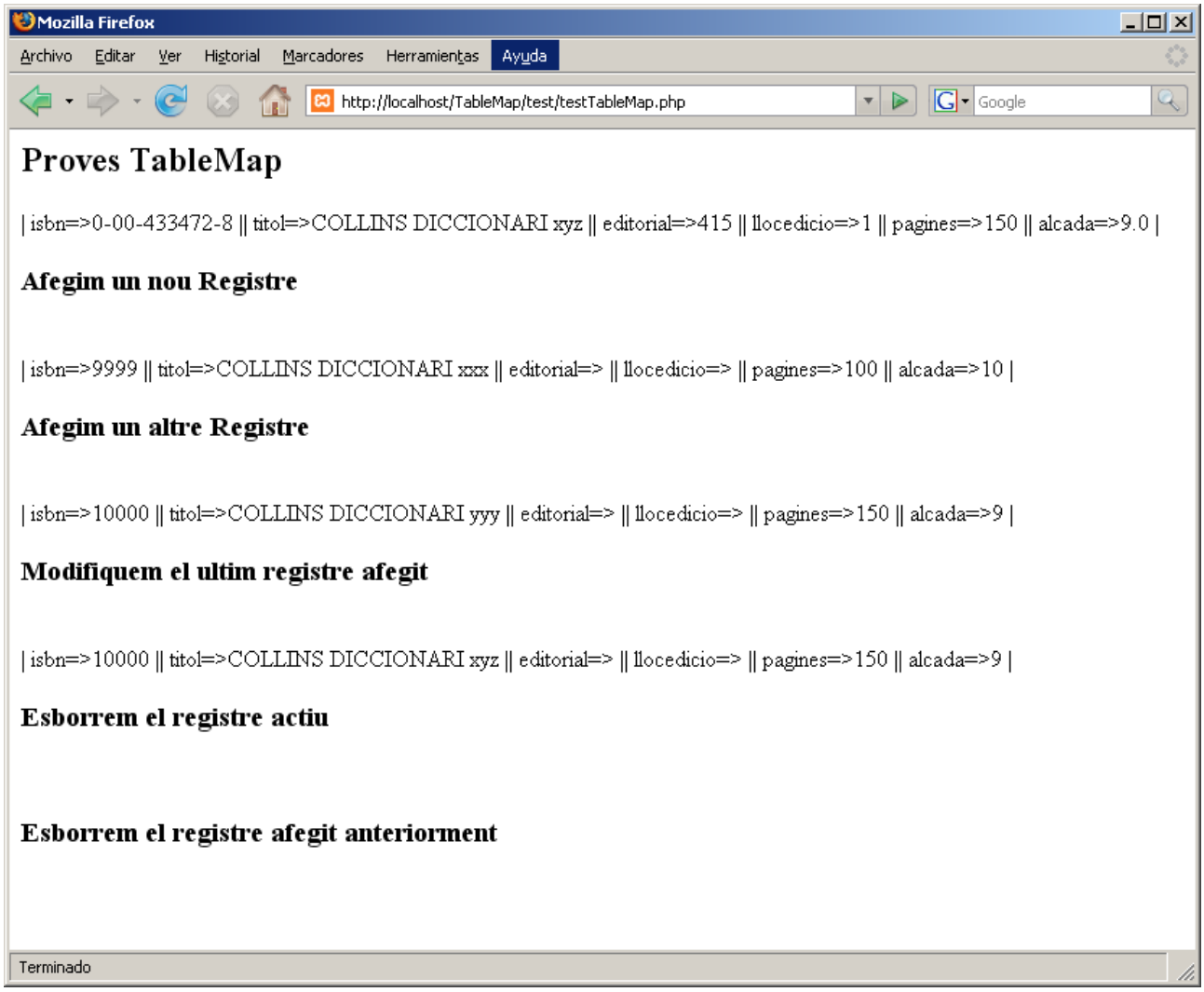


Utilització TableMap

En el codi següent utilitzem aquest fitxer en la classe TableMap.

```
echo "<h2>Proves TableMap</h2>";
//Cadena de connexio de la bd
$cadenaConnexio="mysql://root:@localhost/uoc2";

//Cridem al constructor del objecte pasant com a parametres el xml de definicio i la cadena
de connexio
$tablemap=new TableMap($SUBDIR."/xml/obra.xml",$cadenaConnexio);

//Recuperem i printem el registre per pantalla
$pkValues='0-00-433472-8';
$tablemap->getById($pkValues);
echo $tablemap->toString()."<br>";

echo "<h3>Afegim un nou Registre</h3><br>";
$tablemap->AddNew();
$tablemap->setField('isbn','9999');
$tablemap->setField('titol','COLLINS DICCIONARI xxx');
$tablemap->setField('pagines',100);
$tablemap->setField('alcada',10);

$tablemap->Save();
echo $tablemap->toString()."<br>";

echo "<h3>Afegim un altre Registre</h3><br>";
$tablemap->AddNew();
$tablemap->setField('isbn','10000');
$tablemap->setField('titol','COLLINS DICCIONARI yyy');
$tablemap->setField('pagines',150);
$tablemap->setField('alcada',9);
$tablemap->Save();
echo $tablemap->toString()."<br>";

echo "<h3>Modifiquem el ultim registre afegit</h3><br>";
$tablemap->Modify();
$tablemap->setField('titol','COLLINS DICCIONARI xyz');
$tablemap->Save();
echo $tablemap->toString()."<br>";

echo "<h3>Esborrem el registre actiu</h3><br>";
$tablemap->Delete();

echo "<h3>Esborrem el registre afegit anteriorment</h3><br>";
$tablemap->DeleteById('9999');
```

La sortida seria semblant a la següent:


Mozilla Firefox
Archivo Editar Ver Historial Marcadores Herramientas Ayuda
http://localhost/TableMap/test/testTableMap.php
Google
Proves TableMap
| isbn=>0-00-433472-8 || titol=>COLLINS DICCIONARI xyz || editorial=>415 || llocedicio=>1 || pagines=>150 || alcada=>9.0 |
Afegim un nou Registre
| isbn=>9999 || titol=>COLLINS DICCIONARI xxx || editorial=> || llocedicio=> || pagines=>100 || alcada=>10 |
Afegim un altre Registre
| isbn=>10000 || titol=>COLLINS DICCIONARI yyy || editorial=> || llocedicio=> || pagines=>150 || alcada=>9 |
Modifiquem el ultim registre afegit
| isbn=>10000 || titol=>COLLINS DICCIONARI xyz || editorial=> || llocedicio=> || pagines=>150 || alcada=>9 |
Esborrem el registre actiu
Esborrem el registre afegit anteriorment
Terminado

Capacitat d' extensió

La classe "*TableMap"* simplifica el treball amb la base de dades però en molts casos pot ser interessant estendre la seva funcionalitat ja sigui per treballar amb classes entitat independents, per tenir camps calculats des de codi, procediments específics de càlcul o simplement per separar millor la capa d' accés a dades de la de negoci.

A continuació es mostra un exemple d' extensió de la classe TableMap sobre la taula autor per incloure un camp calculat, que anomenarem "nomYtractament", que mostri a la vegada el tractament (Sr, Sra, ...) i el nom del autor.

```
class Autor extends TableMap {

  public function Autor($xmlFile,$connectionString){
    parent::TableMap($xmlFile,$connectionString);
  }

  public function get_nomYtractament(){
     return $this->getField('tractament')." ".$this->getField('nom');

  }

  public function getField($field){
    if (method_exists($this,'get_'.$field)) {
      return call_user_func(array($this,'get_'.$field));
    }else{ return parent::getField($field);}

}
```

El métode getField comprova primer si existeix un métode get_"nom_de_camp", si hi es el crida sinó es crida el mètode de la classe TableMap. Amb el codi següent recuperaríem el nou camp calculat.

```
echo "<h2>Proves Autor</h2>";
//Cadena de connexio de la bd
$cadenaConnexio="mysql://root:@localhost/uoc2";

//Cridem al constructor del objecte pasant com a parametres el xml de definicio i la cadena
de connexio
$autor=new Autor($SUBDIR."/xml/autor.xml",$cadenaConnexio);
//Recuperem i printem el registre per pantalla

$autor->getById(1);
echo $autor->getField("nomYtractament");
```

Conclusions.

Al finalitzar el projecte es pot afirmar que els objectius bàsics estan complerts, s' ha obtingut un mecanisme per consultar al diccionari de la base de dades recuperant la informació rellevant, escriure aquesta en un fitxer Xml i desenvolupar una classe que pugui accedir a qualsevol taula de la base de dades només amb la ajuda del fitxer de definició de taula generat automàticament.

El objectiu de fons era simplificar la gestió de la persistència en bases de dades relacionals i penso que realment es simplifica, amb aquestes classes no ens veiem obligats a generar manualment les consultes per accedir i actualitzar les dades, ni a utilitzar eines externes que acostumen a forçar una estructura molt estricta de les classes i del accés a les dades.

Encara que es podria pensar que el fitxer de definició Xml no es necessari per treballar, ja que seria possible fer les consultes al diccionari directament abans de generar automàticament les sentencies Sql, millora el rendiment i la seva flexibilitat fa que puguem si ens interessa afegir atributs propis no contemplats per la BDR que utilitzem, com podria ser la descripció detallada de cada camp o un atribut de descripció a visualitzar(mes entenedor que el nom real del camp). També es podria arribar a controlar la integritat amb bases de dades que no ho suportin (com Mysql amb motor MyIsam) afegint al Xml les entrades referents a les claus foranes i a les taules depenents.

Encara que aquesta primera versió de classe "*TableMap"* queden encara moltes funcionalitats per desenvolupar, per exemple:

- ✔ Controlar en les operacions els formats de camp i si són o no requerits.
- ✔ Controlar les claus foranes i les dependències entre taules.
- ✔ Afegir funcionalitats per facilitar les joins entre taules.
- ✔ Afegir funcionalitats per permetre recuperar un registre o registres per varis criteris.

Glossari

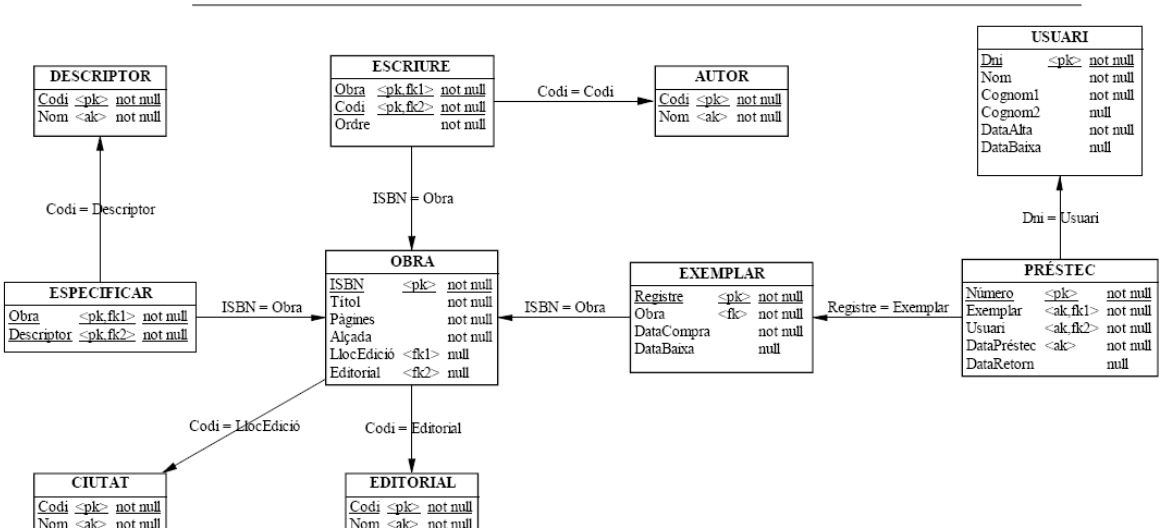
SGBDR	Sistema Gestor de Base de dades Relacions
BDR	Base de dades Relacional (per abreviar SGBDR)
PHP	Hypertext Pre-Processor. Llenguatge de script molt utilitzat per fer planes web.
PEAR	PHP Extension and Application Repository. Es un conjunt de llibreries per utilitzar en PHP. Està dividit en paquets de múltiples propòsits.
Mysql	Base de dades relacional amb versió amb codi obert.
Apache HTTP Server	Es el servidor web de codi obert mes utilitzat al mon.
PhpDocumentor	Es una eina de generació automàtica de documentació pel llenguatge PHP.
Xml	e**X**tensible **M**arkup **L**anguage. Es un llenguatge utilitzat per fer referència a altres llenguatges , es extensible i utilitza etiquetes . Es una simplificació del llenguatge SGML.
SQL	**S**tructured **Q**uery **L**anguage. Es un llenguatge d' accés a bases de dades relacionals, permet definir taules, manipular dades i llençar consultes a la base de dades per obtenir resultats d' interès.
Xampp	Distribució fàcil d' instalar amb Apache, PHP i Mysql.

Bibliografia

- ✔ Mòduls Enginyeria del programari. UOC.
- ✔ Mòduls Tècniques de desenvolupament de programari. UOC.
- ✔ PHP Manual. Varis autors.
- ✔ Manual PhpDocumentor. Varis autors.
- ✔ PEAR Manual. Varis autors.
- ✔ MDB2, API Comú per accedir a bases de dades. Varis autors.
- ✔ XML_Util, API per treballar amb fitxer Xml. Varis autors.
- ✔ XML_Parser, API per interpretar fitxers Xml. Varis autors.
- ✔ Manual Apache HTTP Server. Varis autors.
- ✔ Manual de referència Mysql..Varis autors.
- ✔ INSTALLING, CONFIGURING, AND DEVELOPING WITH XAMPP. Dalibor D. Dvorski

Annexos

Disseny base de dades de proves.



Exemple fitxer Xml generat: Obra.xml

```
<?xml version="1.0" ?>
- <table name="obra">
  <field charmaxlen="15" datatype="text" name="isbn" null="NO" />
  <field charmaxlen="75" datatype="text" name="titol" null="NO" />
  <field datatype="float" name="editorial" null="YES" numprecision="22" numscale="0" />
  <field datatype="float" name="llocedicio" null="YES" numprecision="22" numscale="0" />
  <field datatype="float" name="pagines" null="NO" numprecision="22" numscale="0" />
  <field datatype="float" name="alcada" null="NO" numprecision="3" numscale="1" />
- <primarykey>
  <pkfield name="isbn" />
  </primarykey>
- <foreignkeys>
- <foreignkey name="fk1_obra">
  <fkfield columnname="editorial" position="1" referencedcolumn="codi"
referencedtable="editorial" />
  </foreignkey>
- <foreignkey name="fk2_obra">
  <fkfield columnname="llocedicio" position="1" referencedcolumn="codi"
referencedtable="ciutat" />
  </foreignkey>
  </foreignkeys>
  </table>
```