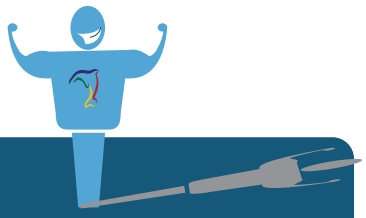
autentia

Avenida de Castilla,1 - Edificio Best Point - Oficina 21B
28830 San Fernando de Henares (Madrid)
tel./fax: +34 91 675 33 06
info@autentia.com - **www.autentia.com**

¿**Qué ofrece** Autentia Real Business Solutions S.L?

Somos su empresa de **Soporte a Desarrollo Informático**.
Ese apoyo que siempre quiso tener...

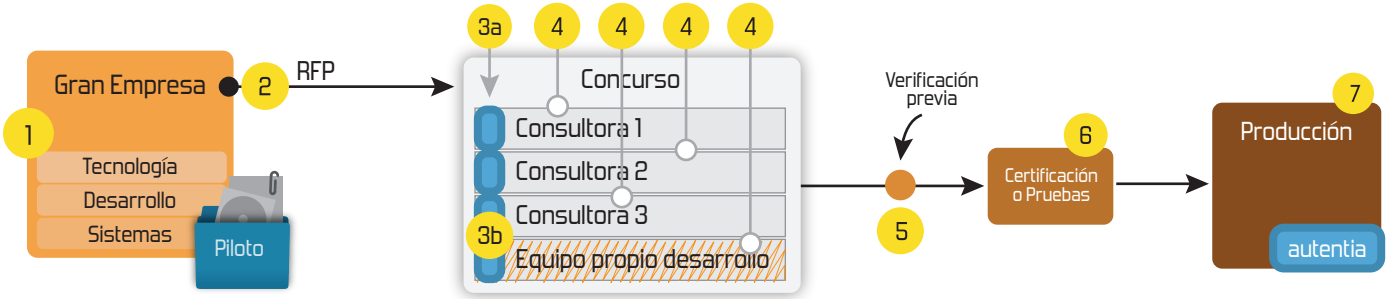
1. Desarrollo de componentes y proyectos a medida



2. Auditoría de código y recomendaciones de mejora

3. Arranque de proyectos basados en nuevas tecnologías

1. Definición de frameworks corporativos.
2. Transferencia de conocimiento de nuevas arquitecturas.
3. Soporte al arranque de proyectos.
4. Auditoría preventiva periódica de calidad.
5. Revisión previa a la certificación de proyectos.
6. Extensión de capacidad de equipos de calidad.
7. Identificación de problemas en producción.



4. Cursos de formación (impartidos por desarrolladores en activo)

- Spring MVC, JSF-PrimeFaces /RichFaces, HTML5, CSS3, JavaScript-jQuery
- Gestor portales (Liferay)
- Gestor de contenidos (Alfresco)
- Aplicaciones híbridas
- Control de autenticación y acceso (Spring Security)
- UDDI
- Web Services
- Rest Services
- Social SSO
- SSO (Cas)
- JPA-Hibernate, MyBatis
- Motor de búsqueda empresarial (Solr)
- ETL (Talend)
- Dirección de Proyectos Informáticos.
- Metodologías ágiles
- Patrones de diseño
- TDD
- Tareas programadas (Quartz)
- Gestor documental (Alfresco)
- Inversión de control (Spring)
- BPM (jBPM o Bonita)
- Generación de informes (JasperReport)
- ESB (Open ESB)

Compartimos nuestro conociemiento en:
www.adictosaltrabajo.com

Para más información visítenos en:
www.autentia.com

Carlos García de Marcos

Consultor tecnológico de desarrollo de proyectos informáticos.

Ingeniero en Informática, especialidad en Ingeniería del Software.

Puedes encontrarme en Autentia: Ofrecemos servicios de soporte a desarrollo, factoría y formación

Somos expertos en Java/J2EE

Ver todos los tutoriales del autor

Fecha de publicación del tutorial: 2015-02-13

MyBatis - SQLs Dinámicas

0. Índice de contenidos.

1. Introducción

En muchas ocasiones nos encontramos con la necesidad de generar una query dinámica para cubrir los requisitos del desarrollo, ya sea porque las condiciones de la consulta dependen del estado de alguna variable o para evitar la generación de multitud de consultas estáticas que cubran todas las posibilidades.

Cuando la capa de datos está implementada directamente con JDBC, la solución pasaría por generar dinámicamente la query que le pasamos al PreparedStatement, estableciendo las condiciones y los bucles necesarios para generarla. Al final, tenemos consultas embebidas en el código, entre multitud de condiciones y bucles, dificultando su legibilidad y mantenimiento.

En este tutorial me gustaría repasar la solución propuesta por MyBatis para la generación de estas queries dinámicas y comprobar la mejora que supone tanto en legibilidad como en mantenimiento respecto a la generación de queries dinámicas generadas por cÃ³digo, manualmente, tal como se hacía en JDBC.

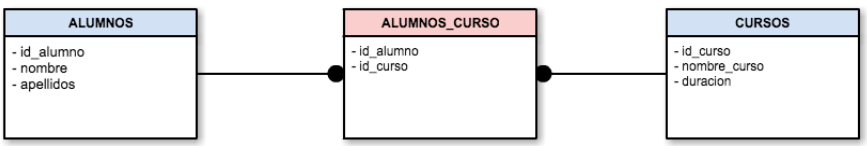
2. Entorno

Para realizar este tutorial se ha empleado el siguiente entorno de desarrollo:

- **Hardware:** Mac Book Pro 15" Intel Core i7 2,8 GHz, 16 GB RAM
- **Sistema Operativo:** Mac OS X Yosemite
- MySQL Community Server 5.6.22

3. Preparación de entorno.

Para realizar este tutorial, vamos a partir del siguiente modelo de datos:




```
CREATE TABLE ALUMNOS (
  ID_ALUMNO int(11) NOT NULL AUTO_INCREMENT,
  NOMBRE varchar(255) NOT NULL,
```

```
  APELLIDOS varchar(255),
  PRIMARY KEY (ID_ALUMNO)
);

CREATE TABLE CURSOS (
  ID_CURSO int(11) NOT NULL AUTO_INCREMENT,
  NOMBRE_CURSO varchar(50) DEFAULT NULL,
  DURACION decimal(38,0) DEFAULT NULL,
  PRIMARY KEY (ID_CURSO)
);


CREATE TABLE ALUMNOS_CURSO (
  ID_ALUMNO int(11) DEFAULT NULL,
  ID_CURSO int(11) DEFAULT NULL,
  KEY FK_ALUMNO (ID_ALUMNO),
  KEY FK_CURSO (ID_CURSO),
  CONSTRAINT FK_ALUMNO FOREIGN KEY (ID_ALUMNO) REFERENCES ALUMNOS (ID_ALUMNO),
  CONSTRAINT FK_CURSO FOREIGN KEY (ID_CURSO) REFERENCES CURSOS (ID_CURSO)
);


INSERT INTO ALUMNOS VALUES (1, 'JUAN', 'TICUARIO');
INSERT INTO ALUMNOS VALUES (2, 'PEDRO', 'MEDARIO');
INSERT INTO ALUMNOS VALUES (3, 'ALFREDO', 'REMIFA');
INSERT INTO ALUMNOS VALUES (4, 'JOHN', 'FUEGOS');
INSERT INTO ALUMNOS VALUES (5, 'AITOR', 'RETA');
INSERT INTO CURSOS VALUES (1, 'La lavadora, esa gran desconocida', 32);
INSERT INTO CURSOS VALUES (2, 'Seminario avanzado de separación de colores.', 16);
INSERT INTO ALUMNOS_CURSO VALUES (1,1);
INSERT INTO ALUMNOS_CURSO VALUES (1,2);
INSERT INTO ALUMNOS_CURSO VALUES (2,1);
INSERT INTO ALUMNOS_CURSO VALUES (2,2);
INSERT INTO ALUMNOS_CURSO VALUES (3,1);
INSERT INTO ALUMNOS_CURSO VALUES (4,2);
```

4. Limitaciones de las consultas estáticas

Como comentaba, en multitud de casos las queries estáticas no son suficientes para resolver las necesidades que se nos presentan. Es en estos casos donde necesitamos montar queries dinámicas que se formen de acuerdo a los parámetros recibidos en los distintos flujos de la aplicación.

Un ejemplo que se repite constantemente es la aplicación de filtros en informes web.

Veamos un ejemplo práctico. Imaginemos que, basándonos en el modelo de datos anterior, tenemos un informe en el que queremos mostrar para cada curso los alumnos del mismo.

Este informe lo resolveríamos con una query del siguiente estilo:

```
<select id="findAlumnosCursos" parameterType="ReportAlumnosCursosFilter" resultMap="Rep
        SELECT *
        FROM alumnos A, cursos C, alumnos_curso AC
        WHERE
            a.id_alumno = ac.id_alumno
            AND c.id_curso = ac.id_curso
</select>
```

Con la query anterior recuperaríamos toda la información, sobre la que podremos aplicar los filtros necesarios.

Ahora definamos un filtro opcional por curso, en el que podamos seleccionar o bien todos los cursos (sin indicar ninguno) o bien el id de un curso concreto.Este caso podría solucionarse facilmente basándonos en la misma query estática:

```
<resultMap type="ReportAlumnosCursos" id="ReportAlumnosCursosResult">
    <id property="idCurso" column="id_curso"/>
    <id property="idAlumno" column="id_alumno"/>
    <result property="curso" column="nombre_curso" />
    <result property="duracion" column="duracion" />
    <result property="nombre" column="nombre" />
    <result property="apellidos" column="apellidos" />
</resultMap>

<select id="findAlumnosCursos" parameterType="ReportAlumnosCursosFilter" resultMap="Repo
    SELECT *
    FROM alumnos A, cursos C, alumnos_curso AC
    WHERE
        a.id_alumno = ac.id_alumno
        AND c.id_curso = ac.id_curso
        and (#{idCurso} == null} OR c.id_curso = #{idCurso})
</select>
```

Pero supongamos que queremos incluir también un filtro por nombre o parte del nombre del curso. Este caso también lo podríamos solucionar con una query estática, pero se va complicando más el tema:

```
<select id="findAlumnosCursos" parameterType="ReportAlumnosCursosFilter" resultMap="Rep
    SELECT *
    FROM alumnos A, cursos C, alumnos_curso AC
    WHERE
        a.id_alumno = ac.id_alumno
        AND c.id_curso = ac.id_curso
        and (#{idCurso} == null} OR c.id_curso = #{idCurso})
        and (#{curso == null} OR UPPER(c.curso) like '%#{curso.toUpperCase()}%')
</select>
```

Y podríamos seguir incluyendo filtros por cada uno de los campos del informe.

Los problemas más claros de este planteamientoson las siguientes:

- Desde el punto de vista de la lógica funcional, se está depositando en la base de datos la responsabilidad de aplicar o no cada uno de los filtros, cuando debería recaer en la capa de acceso a datos.
- Desde el punto de vista del mantenimiento, la complejidad de la query generada aumenta con cada uno de los posibles filtros aplicables, lo que aumenta las posibilidades de error tanto durante el desarrollo como dura el mantenimiento.

Últimos Tutoriales del Autor

» Oracle - Importación de Datos con Data Pump Import [impdp]

» Oracle - Exportación de Datos con Data Pump Export [expdp]

- Desde el punto de vista del rendimiento, se incrementan los filtros aplicados a la query incluso en los casos en que no se quiera aplicar ningún filtro, cuando esto debería ser controlado por la capa de acceso a datos de la aplicación.

Veamos cómo MyBatis plantea la resolución de estos problemas.

5. Consultas dinámicas en MyBatis.

Para resolver este y otros casos, MyBatis proporciona una serie de tags que se pueden emplear para definir consultas dinámicas en sus mappers:

- **El tag <if>:**

  Mediante este tag podremos definir segmentos de consulta como opcionales, incluyéndose únicamente cuando se cumpla una determinada condición. Su especificación es:

```
<if test="condicion">
    -- Segmento SQL
</if>

```

  Así podríamos codificar la query anterior empleando el tag **<if>** de la siguiente manera:

```
<select id="findAlumnosCursos" parameterType="ReportAlumnosCursosFilter" resultMa
    SELECT *
    FROM alumnos A, cursos C, alumnos_curso AC
    WHERE
        a.id_alumno = ac.id_alumno
        AND c.id_curso = ac.id_curso
        <if test="idCurso != null">
            and c.id_curso = #{idCurso}
        </if>
        <if test="curso != null">
            and UPPER(c.curso) like '%#{curso.toUpperCase()}%'
</select>

```

- **El tag <choose>:**

  De igual forma, mediante este tag podremos establecer distintos segmentos de query dependiendo del valor de una variable. Su especificación es:

```
<choose>
    <when test="condicion">
        -- Segmento SQL
    </when>
    [<when test="condicion">
        -- Segmento SQL
     </when> ... ]
    [<otherwise>
        -- Segmento SQL
     </otherwise> ]
</choose>

```

  Si por ejemplo añadieramos un selector "searchBy" para indicar por qué campo queremos filtrar, la query quedaría de la siguiente manera:

```
<select id="findAlumnosCursos" parameterType="ReportAlumnosCursosFilter" resultMa
    SELECT *
    FROM alumnos A, cursos C, alumnos_curso AC
    WHERE
        a.id_alumno = ac.id_alumno
        AND c.id_curso = ac.id_curso
        <choose>
            <when test="searchBy == 'BY_CURSO_ID'">
                and c.id_curso = #{idCurso}
            </when>
            <when test="searchBy == 'BY_CURSO'">
                and UPPER(c.nombre_curso) like '%#{curso.toUpperCase()}%'
            </when>
            <when test="searchBy == 'BY_ALUMNO_ID'">
                and a.id_alumno = #{idAlumno}
            </when>

            <when test="searchBy == 'BY_ALUMNO'">
                <if test="nombre != null">
                and UPPER(a.nombre) like '%#{nombre.toUpperCase()}%'
                </if>
                <if test="apellidos != null">
                and UPPER(a.apellidos) like '%#{apellidos.toUpperCase()}%'
                </if>
            </when>
        </choose>
</select>

```

- **El tag <where>:**

  Este tag se emplea cuando todas las condiciones de la query son opcionales. Este tag lo que hace es añadir la palabra reservada **WHERE** únicamente cuando sea necesario, cuando alguna de las condiciones opcionales es añadida. De igual forma, eliminará cualquier **OR** o **AND** que le preceda. Su especificación es:

```
<where>
    -- SQL condicionado
    ...
</where>

```

  Por ejemplo, si quisieramos generar una query que nos filtrara el listado de cursos únicamente si el usuario ha seleccionado un filtro, podríamos hacerlo de la siguiente manera:

```
<select id="CursosList" parameterType="CursosFilter" resultMap="CursosResult">
    <! [CDATA[
    SELECT *
    FROM Cursos
    <where>
        <if test="searchBy=='SEARCH_BY_CURSOID'">
            id_curso = #{idCurso}
        </if>
        <if test="searchBy=='SEARCH_BY_NOMBRECURSO'">
            AND nombre_curso like '%#{curso}%'
        </if>
    </where>
</select>
```

De esta forma, si el usuario ha seleccionado un filtro por id de curso se incluirá la claúsula **WHERE** y la condición **id_curso = #{idCurso}** y si por el contrario seleccionó la busqueda por nombre de curso, incluirá la cláusula **WHERE**, eliminará la condición **AND** que le precede e incluirá la condición **nombre_curso like '%#{curso}%'**

- **El tag <trim>:**

Es bastante similar al tag **<WHERE>** pero ofrece más opciones. Veamos su especificación:

```
<trim [prefix="prefijo"] [prefixOverrides="cadena [ | cadena]"] [sufix="sufijo"] [
```

Donde **prefix** y **sufix** se emplean para añadir una cadena antes y después del segmento de sql incluido. Ambos son opcionales y no son excluyentes y se incluirán únicamente si existe un segmento de sql, al igual que el tag **<where>**. De igual forma, mediante los parámetros **prefixOverrides** y **sufixOverrides** se puede eliminar una o varias cadenas del inicio o del fin del segmento de SQL. Normalmente se emplean para eliminar los **AND** y **OR** que sobran, ya sea al inicio o al fin del segmento. Para indicar que elimine los **AND** y **OR** sobrantes, se establecería así: **prefixOverrides="AND | OR"**

.

La siguiente configuración de **TRIM** sería equivalente al tag **<where>**:

```
<trim prefix="WHERE " prefixOverrides="AND|OR" sufixOverrides="AND|OR">...</trim>
```

- **El tag <foreach>:**

Con este tag podremos iterar sobre una colección, generando un segmento de SQL para cada uno de los elementos. Si por ejemplo quisieramos buscar los cursos comunes a dos alumnos, podríamos incluir un campo List al filtro e indicar en ese campo los ids de los alumnos que buscamos. Incluiremos un **<trim>** para formar el conjunto de condiciones.La query quedaría así:

```
<select id="findAlumnosCursos" parameterType="ReportAlumnosCursosFilter" resultMa
    SELECT *
    FROM alumnos A, cursos C, alumnos_curso AC
    WHERE
        a.id_alumno = ac.id_alumno
        AND c.id_curso = ac.id_curso
        <choose>
            <when test="searchBy == 'BY_CURSO_ID'">
                and c.id_curso = #{idCurso}
            </when>
            <when test="searchBy == 'BY_CURSO'">
                and UPPER(c.nombre_curso) like '%#{curso.toUpperCase()}%'
            </when>
            <when test="searchBy == 'BY_ALUMNO_ID'">
                and a.id_alumno = #{idAlumno}
            </when>

            <when test="searchBy == 'BY_ALUMNO'">
                <if test="nombre != null">
                and UPPER(a.nombre) like '%#{nombre.toUpperCase()}%'
                </if>
                <if test="apellidos != null">
                and UPPER(a.apellidos) like '%#{apellidos.toUpperCase()}%'
                </if>
            </when>
            <when test="searchBy == 'BY_GRUPO_ALUMNOS">
                <trim prefix="AND (" sufix=")" prefixOverrides="OR">
                    <foreach item="idAlumnoItem" collection="alumnosIds">
                        OR a.id_alumno = #{idAlumnoItem}
                    </foreach>
                </trim>
            </when>
            <otherwise></otherwise>
        </choose>
</select>
```

- **El tag <set>:**

Por último, este tag es muy similar al tag pero para las sentencias **UPDATE**. En este caso, este tag incluirá la palabra reservada **SET** si cualquiera de los **<if>** internos se cumple y eliminará las comas sobrantes. Veamos como ejemplo la sentencia de actualización de los cursos:

```
<update id="updateCursos" parameterType="Curso">
    UPDATE cursos
    <set>
        <if test="curso != null">
            nombre_curso = #{curso},
        </if>
        <if test="duracion > 0">
            duracion = #{duracion}
        </if>
    </set>
</update>
```

La misma sentencia generada con un tag **<trim>** sería así:

```
<update id="updateCursos" parameterType="Curso">
    UPDATE cursos
    <trim prefix="SET " prefixOverrides="," sufixOverrides=",">
        <if test="curso != null">
            nombre_curso = #{curso},
        </if>
        <if test="duracion > 0">
            duracion = #{duracion}
        </if>
    </trim>
</update>
```

6. Conclusiones.

Como habéis podido ver, mediante los tags de MyBatis conseguimos definir SQLs dinámicas de una forma más estructurada evitando tener que incluir condiciones para ver si tenemos que incluir o no un **WHERE**, un **AND** o un **OR** en la query. Como ejemplo, a continuación podemos ver la query que hemos empleado en los ejemplos generada con MyBatis y el código equivalente si lo generáramos con JDBC:

Versión JDBC:

```
Connection conn;
PreparedStatement pstmt;
ResultSet rs;

StringBuilder sql = new StringBuilder();

...

// Generación de la query:

sql .append("SELECT * ")
    .append("FROM alumnos A, cursos C, alumnos_curso AC ")
    .append("WHERE ")
    .append(" a.id_alumno = ac.id_alumno ")
    .append(" AND c.id_curso = ac.id_curso ");

if(filter.getSearchBy().equals("BY_CURSO_ID")) {
    sql.append(" AND c.id_curso = ?");
}

if(filter.getSearchBy().equals("BY_CURSO")) {
    sql.append(" AND UPPER(c.nombre_curso) like '%?%' ");
}


if(filter.getSearchBy().equals("BY_ALUMNO_ID")) {
    sql.append(" AND a.id_alumno = ? ");
}

if(filter.getSearchBy().equals("BY_ALUMNO")) {
    if(filtro.getNombre() != null) {
        sql.append(" AND UPPER(a.nombre) like '%?%'");
    }
    if(filtro.getApellidos()!=null){
        sql.append(" AND UPPER(a.apellidos) like '%?%'");

    }
}

if(filter.getSearchBy().equals("BY_GRUPO_ALUMNOS")) {
    if(filtro.getAlumnosIds().size()>0) {
        sql.append("AND ( ");
        for(int i=0; i < filtro.getAlumnosIds().size(); i++) {
            Integer id = (Integer)filtro.getAlumnosIds().get(i);
            if(i>0) {
                sql.append(" OR ");
            }
            sql.append(" a.id_alumno = ? ");
        }
        sql.append(") ");
    }
}

pstmt = conn.prepareStatement(sql.toString());

// Establecemos los parámetros
int pos = 0;
if(filter.getSearchBy().equals("BY_CURSO_ID")) {
    pstmt.setInt(++pos, filtro.getIdCurso());
}

if(filter.getSearchBy().equals("BY_CURSO")) {
    pstmt.setString(++pos, filtro.getCurso().toUpperCase());
}

if(filter.getSearchBy().equals("BY_ALUMNO_ID")) {
    pstmt.setInt(++pos, filtro.getIdAlumno());
}

if(filter.getSearchBy().equals("BY_ALUMNO")) {
    if(filtro.getNombre() != null) {
        pstmt.setString(++pos, filtro.getNombre().toUpperCase());
    }
    if(filtro.getApellidos()!=null){
        pstmt.setString(++pos, filtro.getApellidos().toUpperCase());
    }
}

if(filter.getSearchBy().equals("BY_GRUPO_ALUMNOS")) {
    if(filtro.getAlumnosIds().size()>0) {
```

```
81          for(int i=0; i < filtro.getAlumnosIds().size(); i++) {
82              pstmt.setInt(
83      }
84  }
85
86  ...
```

Versión MyBatis:

```
1  <select id="findAlumnosCursos" parameterType="ReportAlumnosCursosFilter" resultMap="Re
2      SELECT *
3      FROM alumnos A, cursos C, alumnos_curso AC
4      WHERE
5          a.id_alumno = ac.id_alumno
6          AND c.id_curso = ac.id_curso
7          <choose>
8              <when test="searchBy == 'BY_CURSO_ID'">
9                  and c.id_curso = #{idCurso}
10             </when>
11             <when test="searchBy == 'BY_CURSO'">
12                 and UPPER(c.nombre_curso) like '%#{curso.toUpperCase()}%'
13             </when>
14             <when test="searchBy == 'BY_ALUMNO_ID'">
15                 and a.id_alumno = #{idAlumno}
16             </when>
17
18             <when test="searchBy == 'BY_ALUMNO'">
19                 <if test="nombre != null">
20                 and UPPER(a.nombre) like '%#{nombre.toUpperCase()}%'
21                 </if>
22                 <if test="apellidos != null">
23                 and UPPER(a.apellidos) like '%#{apellidos.toUpperCase()}%'
24                 </if>
25             </when>
26             <when test="searchBy == 'BY_GRUPO_ALUMNOS">
27                 <trim prefix="AND (" sufix=")" prefixOverrides="OR">
28                     <foreach item="idAlumnoItem" collection="alumnosIds">
29                         OR a.id_alumno = #{idAlumnoItem}
30                     </foreach>
31                 </trim>
32             </when>
33         </choose>
34 </select>
```

Como conclusión, comparando ambas versiones, se puede ver claramente que con el empleo de las queries dinámicas de MyBatis no solo reducimos el código, sino que además lo hacemos más legible y estructurado, y por tanto, más mantenible.

A continuación puedes evaluarlo:

Regístrate para evaluarlo

Por favor, vota +1 o compártelo si te pareció interesante

Share | 8+1 0

Anímate y coméntanos lo que pienses sobre este **TUTORIAL:**

» **Registrate** y accede a esta y otras ventajas «