Estoy diseñando los nuevos tags <select> bajo el ticket HOT-93. El ticket no está actualizado porque he avanzado sólo localmente con este análisis.

La idea principal es que actualmente los tags <select> entregan columnas sueltas. Sin embargo, estas columnas probablemente ya están modeladas como propiedades de VOs y sería ideal reutilizarlos.

Por otro lado, como parte de la configuración existente los VOs podrían tener características extra como **java-type**, **java-name**, **converter**, etc., las que podrían ser respetadas automáticamente si enlazamos los VOs con los <select>s.

Eso sería en resumidas cuentas.

Para ejemplificar los casos uso tres tablas: PERSONA, CUENTA y TRANSACCION. Ellas tienen relación 1-N entre ellas. Están descritas a continuación (código Oracle):

**create** **table** persona (

id number(10) **primary** **key** **not** **null**,

nombre varchar2(30) **not** **null**,

tipo number(3) **not** **null** -- 1:individual, 2:commercial, 3:gubernamental

);

**create** **table** cuenta (

id number(10) **primary** **key** **not** **null**,

nro\_cuenta varchar2(30) **not** **null**,

id\_persona number(10) **not** **null**,

tipo number(3) **not** **null**, -- 1:cc, 2:ahorro, 3:inversiones; 4:jubilación

activa number(1) **not** **null**, -- 0:inactiva, 1:activa

saldo number(14,2) **not** **null**,

**constraint** fk1 **foreign** **key** (id\_persona) **references** persona (id)

);

**create** **table** transaccion (

id number(18) **primary** **key** **not** **null**,

ejecutada **date** **not** **null**,

monto number(10,2) **not** **null**,

id\_cuenta number(10) **not** **null**,

tipo number(2) **not** **null**, -- 0:caja, 1:online, 2:cajero automático, 3:terceros

**constraint** fk2 **foreign** **key** (id\_cuenta) **references** cuenta (id)

);

**Caso 1 - Join actual**

Este es el caso actual de los tags <select> que automatizan una buena parte del trabajo. En el ejemplo utilizo la última sintaxis desarrollada, que usa tags <column>, <parameter> y <complement>:

<select ...>

<column name=*"tipo"* converter=*"tipo\_cuenta\_converter"* />

<column name=*"pers\_tipo"* converter=*"tipo\_persona\_converter"* />

<column name=*"activa"* converter=*"boolean\_converter"* />

<parameter name=*"idCliente"* java-type=*"java.lang.Integer"* jdbc-type=*"NUMERIC"* />

select

c.\*,

p.id "pers\_id", p.nombre, p.tipo "pers\_tipo"

from cuenta c

join persona p on (p.id) = (c.id\_persona)

<complement>

where c.id = #{idCliente}

</complement>

</select>

El join actual entrega un nuevo "VO combinado" con todas las columnas de las tablas persona y cuenta.

**Beneficios (actuales):**

* El VO combinado es generado automáticamente, sin intervención del programador.
* Las columnas tienen los nombres correctos y tipos de dato correctos, lo que es mejor que la competencia (no es un Map<String, Object>).

**Limitaciones:**

* **Colisión de nombres**: Dado que hay ambas tablas tienen columnas con el mismo nombre (id, tipo), no se puede usar "c.\*" sino que hay que nombrar cada columna directamente para evitar colisiones de nombres.
* **Redundancia**: hay que repetir la configuración especial de cada columna, que ya había sido definida en los tag <table>. Por ejemplo, para agregar los converter (columnas tipo, pers\_tipo, activa).
* **Lógica custom no está disponible**: El VO combinado no tiene relación con los VOs existentes para cada tabla (CuentaVO, ClienteVO) y, por lo tanto, toda lógica que el programador haya agregado en ellos no está disponible en el VO combinado.

**Caso 2 - SELECT entrega múltiples VOs (JOIN)**

En este caso el tag <select> hace referencia a VOs existentes.

<select ...>

<parameter name=*"idCliente"* java-type=*"java.lang.Integer"* jdbc-type=*"NUMERIC"* />

select

<columns>

<vo table=*"cuenta"* property=*"cuenta"* expression=*"c.\*"* />

<vo table=*"persona"* property=*"persona"* expression=*"p.\*"* />

</columns>

from cuenta c

join persona p on (p.id) = (c.id\_persona)

<complement>

where c.id = #{idCliente}

</complement>

</select>

El join entrega un VO correspondiente al select que tiene sólo dos propiedades, especificadas por los atributos "property":

private CuentaVO cuenta;

private PersonaVO persona;

El atributo "expression" de los tags <vo> incluye una lista de columnas del select separadas por comas, de forma similar al estándar SQL. Esta lista podría incluir sólo algunas columnas, o todas como muestra el ejemplo.

Los tags <vo> también podrían hacer referencia a vistas, en vez de tablas, usando el atributo "view".

**Beneficios (adicionales):**

* Aunque ambas tablas tienen columnas con el mismo nombre (id, tipo), el tag <select> resuelve la colisión de nombres automáticamente y el código SQL producido incluye aliases para las columnas. De esta forma todas las columnas son obtenidas por el select aunque originalmete había colisión de nombres. Mejor aún, todo ocurre tras bambalinas y no hay intervención del programador alguna.
* Toda la funcionalidad de los VOs existentes es utilizada automáticamente (java-name, converter, java-type, etc.). No hay que repetir la configuración, evitando así redundancia.
* La lógica que el programador haya agregado en los VOs está ahora disponible automáticamente ya que son las mismas clases VO (CuentaVO y PersonaVO).

**Caso 3 - SELECT entrega único VO**

La lógica del join anterior se puede usar simplificadamente (sin el join) para hacer select sobre una tabla única.

<select ...>

<parameter name=*"idCliente"* java-type=*"java.lang.Integer"* jdbc-type=*"NUMERIC"* />

select

<columns>

<vo table=*"cuenta"* expression=*"c.\*"* />

</columns>

from cuenta c

<complement>

where c.id = #{idCliente}

and (tipo = 1 and nro\_cuenta like 'SOC-%' or tipo = 3 and nro\_cuenta like 'AFP-%')

</complement>

</select>

Nótese que el tag <vo> ya no tiene el atributo "property". En este caso ya no entrega un "VO combinado" sino que entrega directamente objetos de la clase CuentaVO. Esto es posible sólo cuando se define un único tag <vo> dentro de <columns>.

No hay límite en la complejidad del SELECT SQL . Lo importante es que las columnas resultantes correspondan (parcial o totalmente) a una única tabla.

El tag <vo> también podría hacer referencia a una vista, en vez de una tabla, usando el atributo "view".

**Beneficios:**

* No se crea un VO combinado nuevo sino que se reutiliza un VO existente.
* Toda la funcionalidad del VO existente es utilizada automáticamente (java-name, converter, java-type, etc.)
* Se puede usar para hacer select usando condiciones complejas, múltiples parámetros, sub-queries, etc.
* También se puede usar Dynamic SQL (ejemplo más adelante) para selects variables.
* La lógica que el programador haya agregado en el VO está disponible automáticamente ya que es la misma clase.

**Caso 4 - SELECT entrega columnas adicionales**

En este caso el tag <select> hace referencia a VOs existentes pero incluye expresiones adicionales.

<select ...>

<parameter name=*"idCliente"* java-type=*"java.lang.Integer"* jdbc-type=*"NUMERIC"* />

select

<columns>

<vo table=*"cuenta"* property=*"cuenta"* expression=*"c.\*"* />

<vo table=*"persona"* property=*"persona"* expression=*"p.\*"* />

<expressions>

(c.saldo / decode(c.tipo, 1,15, 2,8, 3,100) as puntaje\_clase,

(c.saldo / decode(p.tipo, 1,3, 2,17, 3,5, 4,12) as puntaje\_categoria

</expressions>

</columns>

from cuenta c

join persona p on (p.id) = (c.id\_persona)

<complement>

where c.id = #{idCliente}

</complement>

</select>

El join entrega un VO combinado que tiene cuatro propiedades. Los dos VOs y dos expresiones adicionales (que tienen que tener nombres distintos a los atributos "property"):

private CuentaVO cuenta;

private PersonaVO persona;

private Double puntajeClase;

private Double puntajeCategoria;

Los tag <expressions> incluyen una lista de expresiones SQL separadas por comas. Se puede incluir múltiple tags <expression> para agregar más expresiones; esto sólo afecta el orden de las propiedades en el VO. Todas las expresiones deben proveer nombres distintos.

Los tag <expressions> no se pueden incluir cuando haya un solo tag <vo> que no tenga atributo "property"; en este caso el resultado tendría que ser del tipo VO, que no tiene lugar para las expresiones adicionales.

**Beneficios:**

* No se necesita un VO adicional, por encima del VO combinado. Éste incluye las propiedades extras.
* Todos beneficios descritos anteriormente.

**Caso 5 - Colecciones**

En este caso el tag <select> procesa el resultado entendiendo la relación 1-N entre el VO primario y el VO secundario.

<select ...>

<parameter name=*"idCliente"* java-type=*"java.lang.Integer"* jdbc-type=*"NUMERIC"* />

select

<columns>

<vo table=*"cuenta" class="CuentaExtendidaVO"* expression=*"c.\*"*>

<collection table=*"transaccion"* property=*"transacciones"* expression=*"t.\*"* />

</vo>

</columns>

from cuenta c

join transaccion t on (t.id\_cuenta) = (c.id)

<complement>

where c.id = #{idCliente}

</complement>

</select>

Es necesario definir una nueva clase VO para ubicar la collection. El join entrega una lista de CuentaExtendidaVOs (nuevo VO) que es subclase de CuentaVO. Este nuevo VO agrega la propiedad "transacciones" que es una lista de TransaccionVO. En resumen tiene la forma:

public class CuentaExtendidaVO extends CuentaVO {

private List<TransaccionVO> transacciones;

...

}

Dado que la relación es 1-N entre cuenta y transacción, la ejecución del select determina si los valores de la cuenta son repetidos y evita crear un nuevo objecto CuentaVO. En este caso sólo agrega un nuevo objeto TransaccionVO a la propiedad "transacciones" del objeto CuentaVO ya existente.

Para determinar si la cuenta es la misma, se utiliza la PK de la tabla. En este caso CUENTA.ID. Si se usa vistas en vez de tablas (que no tienen PKs) o si se utiliza una tabla que no tiene PK, entonces se debe agregar el atributo "ids" que incluye una lista de columnas (separadas por comas) que actúan como PK.

**Beneficios:**

* Procesa automáticamente relaciones 1-N. El programador no tiene que codificar un proceso extra para tratar esta relación, lo que reduce el esfuerzo de programación y, más importantemente, los errores de programación relacionados a esta tarea rutinaria.
* Dado que el nuevoVO es subclase de CuentaVO, automáticamente dispone de toda la customización incluida en CuentaVO.
* Menor uso de memoria. Los objetos CuentaVO repetidos dan lugar a un solo objeto Java, a diferencia del caso anterior en donde todas las filas repetidas siempre generan un nuevo objeto Java. Esto puede ser significativo cuando el "N" de la relación 1-N es grande, cuando la tabla primaria del join tiene muchas columnas, cuando las columnas de la tabla primaria son "pesadas" en términos de uso de memoria, o en alguna combinación de las condiciones anteriores.

**Caso 6 - Asociaciones**

En este caso el tag <select> procesa el resultado entendiendo la relación N-1 entre el VO primario y el VO secundario.

<select ...>

<parameter name=*"idCliente"* java-type=*"java.lang.Integer"* jdbc-type=*"NUMERIC"* />

select

<columns>

<vo table=*"cuenta"* *class="CuentaExtendida2VO"* expression=*"c.\*"*>

<association table=*"persona"* property=*"persona"* expression=*"p.\*"* />

</vo>

</columns>

from cuenta c

join persona p on (p.id) = (c.id\_persona)

<complement>

where c.id = #{idCliente}

</complement>

</select>

El join entrega una lista de CuentaExtendida2VOs (nuevo VO) que es subclase de CuentaVO. Este nuevo VO agrega la propiedad "persona" que apunta a un PersonaVO. En resumen tiene la forma:

public class CuentaExtendida2VO extends CuentaVO {

private PersonaVO persona;

...

}

Dado que la relación es N-1 entre cuenta y persona, la ejecución del select determina si los valores de la persona son repetidos y evita crear un nuevo objeto PersonaVO. De ser repetido, sólo apunta la propiedad "persona" de CuentaExtendida2VO al objeto PersonaVO ya instanciado anteriormente. En otras palabras, dos o más objetos CuentaExtendida2VO podrían apuntar al MISMO objeto PersonaVO.

Para determinar si la persona es la misma, se utiliza la PK de la tabla. En este caso PERSONA.ID. Si se usa vistas en vez de tablas (que no tienen PKs) o si se utiliza una tabla que no tiene PK, entonces se debe agregar el atributo "ids" que incluye una lista de columnas (separadas por comas) que actúan como PK.

**Beneficios (adicionales):**

* Procesa relaciones N-1 automáticamente. El programador no tiene que codificar un proceso extra para tratar esta relación, lo que reduce el esfuerzo de programación y, más importantemente, los errores de programación relacionados a esta tarea rutinaria.
* Menor uso de memoria. Los objetos PersonaVO repetidos dan lugar a un solo objeto Java, a diferencia del caso anterior en donde todas las filas repetidas generan un nuevo objeto Java. Nuevamente, esto puede ser significativo cuando el "N" de la relación N-1 es grande, cuando la tabla primaria del join tiene muchas columnas, cuando las columnas de la tabla primaria son pesadas en términos de uso de memoria, o en alguna combinación de las condiciones anteriores.

**Caso 7 - Colecciones y asociaciones en mútiples niveles**

<select ...>

<parameter name=*"idCliente"* java-type=*"java.lang.Integer"* jdbc-type=*"NUMERIC"* />

select

<columns>

<vo table=*"cuenta"* *class="CuentaExtendida3VO"* expression=*"c.\*"*>

<collection table=*"transaccion"* property=*"transacciones"* expression=*"t.\*"* />

<collection table=*"settlement"* property=*"settlements"* expression=*"s.\*"* />

</collection>

<association table=*"persona"* property=*"persona"* expression=*"p.\*"*>

<association table=*"empleador"* property=*"empleador"* expression=*"e.\*"* />

</association>

</vo>

</columns>

from cuenta c

join persona p on (p.id) = (c.id\_persona)

join transaccion t on (t.id\_cuenta) = (c.id)

join settlement s on (s.id\_transacción) = (s.id)

left join empleador e on (e.id) = (p.id\_empleador)

<complement>

where c.id = #{idCliente}

</complement>

</select>

**Caso 8 - Todo combinado - Colecciones, asociaciones, expresiones**

<select ...>

<parameter name=*"idCliente"* java-type=*"java.lang.Integer"* jdbc-type=*"NUMERIC"* />

select

<columns>

<vo table=*"cuenta"* *property="cuenta"* *class="CuentaExtendida2VO"* expression=*"c.\*"*>

<collection table=*"transaccion"* property=*"transacciones"* expression=*"t.\*"* />

<collection table=*"settlement"* property=*"settlements"* expression=*"s.\*"* />

</collection>

<association table=*"persona"* property=*"persona"* expression=*"p.\*"*>

<association table=*"empleador"* property=*"empleador"* expression=*"e.\*"* />

</association>

</vo>

<expressions>

(c.saldo / decode(c.tipo, 1,15, 2,8, 3,100) as puntaje\_clase,

(c.saldo / decode(p.tipo, 1,3, 2,17, 3,5, 4,12) as puntaje\_categoria

</expressions>

</columns>

from cuenta c

join persona p on (p.id) = (c.id\_persona)

join transaccion t on (t.id\_cuenta) = (c.id)

join settlement s on (s.id\_transacción) = (s.id)

left join empleador e on (e.id) = (p.id\_empleador)

<complement>

where c.id = #{idCliente}

</complement>

</select>