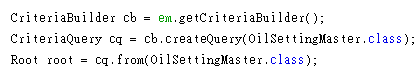
* 前言

正常寫criteria 簡單版就是針對某個entity (table)

然後要撈甚麼屬性 就順著屬性一路下去

主要目標entity



過濾屬性對象

(可能是主要目標本身 或是跑到主要目標的其他關係 或是再跑去別的地方)



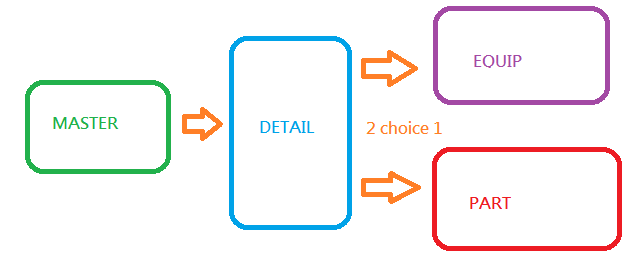
Or



* 那為何要用outer join

當table結構避不掉 得設計複雜時 也就是join 條件 有可能是空的

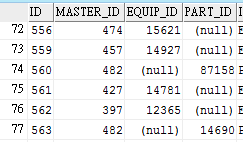
如下圖



我有一個主檔 裡面有很多明細 每個明細 基本上會有一個對象

但這個對象 有可能是 設備 也有可能是零件

(兩個只會其中一個有值 in detail)



因此當我拿某個代碼 去問時 設備or零件代碼 不知 要看使用者關心甚麼

這邊我很難設計要使用者

要看設備 請去設備欄位輸入設備

要看零件 請去零件輸入欄位輸入

更何況有人這代碼到底是設備零件搞不清楚就會找不到

不使用outer join處理 會有非預期狀態 除非用別的手法處理

如果detail / equip / part join是一般正常的join時

1. 有可能零件符合了 但該detail 沒有設備 因此 pass X(漏掉 不正常)
2. 有可能設備符合了 但該detail 沒有零件 因此pass X(漏掉 不正常)
3. Detail 設備 零件 兩種都有 而且關鍵字有符合的 O(這種情況下才正常)

* 有甚麼解決方案

1. 寫法改變 由下往上撈 (之前還沒想到說criteria原來可以寫join的解法)

先把detail 有相關的撈出來 整理出來List<Master>

最後master只要看有哪些id 是符合的

(

有利有弊 有人可能會說 這樣下面可能會很多

但某些情況下 可以透過entity去過濾資料 收斂範圍

起碼 沒有不正常情況

)

1. Criteria 增加join 機制 (後來想到有沒有可能 才發現原來可以這樣寫)

* Criteria 增加join機制 解法

1. 先針對關係 先把join給建立好(下面有圖)

這邊root是指master

第一行 我知道master 下面是有oilSettingDetailList屬性

對應的是List<Detail>

先把join給建起來

Master⬄ detail

第二行 根據第一行 得到master detail 關係

再來綁detail後面的設備

Detail⬄equip

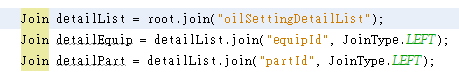
但知道有可能是空的 因此JoinType.LEFT

第三行 跟第二行很像

只是對象改成是零件

PS.detail的屬性equipId /partId 分別對應到entity EQUIP/PART

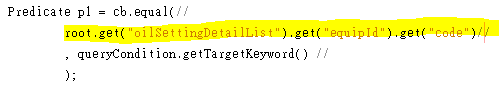
PS.我這邊get都是by string對應entity屬性



1. 重點來了

根據查詢條件撈資料時呢

正常起手式是 如下圖黃色處

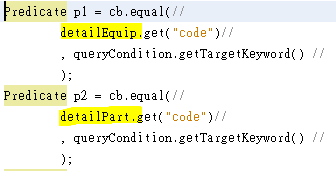


Root xxxx

但這邊要改成如下圖

注意下面的黃色處(是上面宣告的join) 改用join去抓下面的屬性

其實跟正常版很像 但差異在於join時 有刻意宣告JoinType



以上手法就可以達成想要的結果了

* SQL 比較如下 可以看一下差異性

//criterai有使用outer join 產出的sql

SELECT t1.ID, t1.CREATETIMESTAMP, t1.MODIFYTIMESTAMP, t1.COMPANY\_ID, t1.CREATOR, t1.DEP\_ID, t1.EQUIP\_ID

, t1.FACTORY\_ID, t1.MODIFIER

FROM OIL\_SETTING\_DETAIL t6 LEFT OUTER JOIN EQUIP t5 ON (t5.ID = t6.EQUIP\_ID)

LEFT OUTER JOIN PART t7 ON (t7.ID = t6.PART\_ID)

, EQUIP t4

, TC\_DEPARTMENT t3

, TC\_FACTORY t2

, OIL\_SETTING\_MASTER t1

, TC\_COMPANY t0

WHERE ((((t1.COMPANY\_ID = ?) AND (t1.FACTORY\_ID = ?))

AND (t5.CODE LIKE ? OR t7.CODE LIKE ?))

AND (((((t6.MASTER\_ID = t1.ID)

AND (t0.ID = t1.COMPANY\_ID))

AND (t2.ID = t1.FACTORY\_ID))

AND (t3.ID = t1.DEP\_ID))

AND (t4.ID = t1.EQUIP\_ID)))

ORDER BY t0.CODE ASC, t2.CODE ASC, t3.CODE ASC, t4.CODE ASC

//criteria 沒有使用outer join 直接使用path

紅色處的equip t5,t4 出現兩次

是因為master要指定設備 ,但detail對象有可能是設備

這樣的sql有點不好看

因為其中一個equip應該是要跟detail outer join綁在一起的

SELECT t1.ID, t1.CREATETIMESTAMP, t1.MODIFYTIMESTAMP, t1.COMPANY\_ID, t1.CREATOR

, t1.DEP\_ID, t1.EQUIP\_ID, t1.FACTORY\_ID, t1.MODIFIER

FROM OIL\_SETTING\_DETAIL t8

, PART t7

, OIL\_SETTING\_DETAIL t6

, EQUIP t5

, EQUIP t4

, TC\_DEPARTMENT t3

, TC\_FACTORY t2

, OIL\_SETTING\_MASTER t1

, TC\_COMPANY t0

WHERE ((((t1.COMPANY\_ID = ?)

AND (t1.FACTORY\_ID = ?))

AND ((t5.CODE = ?) OR (t7.CODE = ?)))

AND ((((((((t6.MASTER\_ID = t1.ID)

AND (t5.ID = t6.EQUIP\_ID))

AND (t8.MASTER\_ID = t1.ID))

AND (t7.ID = t8.PART\_ID))

AND (t0.ID = t1.COMPANY\_ID))

AND (t2.ID = t1.FACTORY\_ID))

AND (t3.ID = t1.DEP\_ID))

AND (t4.ID = t1.EQUIP\_ID)))

ORDER BY t0.CODE ASC, t2.CODE ASC, t3.CODE ASC, t4.CODE ASC

bind => [1, 3, 1023M01R2190-0010, 1023M01R2190-0010]

* 參考

### 可以參考java ee 6 tutor 關鍵字 Querying Relationships Using Joins

### 另外研究的過程中

### 有看到 Using the Criteria API and Metamodel API to Create Basic Typesafe Queries

### 發現有個東西叫 Metamodel

### 

### 上面黃色Pet\_ ,Owner\_ 就是 Metamodel

### 我這邊認為Metamodel就是描述entity的物件屬性

### 基本上也可以當成控制關係用的一種手段

### 只是比較麻煩一點的是 要先建立相關class 才有辦法拿來用

### ENTITY建好之後 還要再建metamodel , effort x2

