RDBMS 內部分享 1 

今天大綱

- 淺談 trigger
- 淺談 view

前言

- 所謂的「傳家之寶」……
  - 傳說中是曾曾曾祖母傳下來的
  - 用途不明
  - 絕對不能丟掉
- 沒被用的 variable ，在 go 是 compile error 的
  - 其他語言也會在 IDE 顯示 warning
- 但是沒被用上的 view / trigger 呢？

簡介 trigger

- 今天我們集中談 post action, for each row 類型的 trigger
  - 在 MySQL ，你應該看到：
    - Create trigger XXX on tableX
      after {Insert / Update / Delete}
      for each row……
- 主要用途
  - Audit log
  - Database refactoring

Audit Log with trigger

- 對於正常的 tableX ，我們想要建立 tableX_log ，去紀錄 tableX 的一切改動
- tableX_log 只能由用戶 audit_log_user 來修改，一般用戶沒有的權
  - 所以 cracker 沒法把 tableX_log 內的犯罪紀錄刪掉
- 這個 trigger 是由 tableX_log 的權限的用戶 audit_log_user 來建立的
  - 所以這個 trigger 被觸發時，是根據 audit_log_user 的權限來寫入 tableX_log
  - MySQL / postgresql / oracle 三者具體實作會有細微分別，但是原理相同

Database refactoring

- 你的 business logic 需求 db schema 的相應改動，但是你需要維持 legacy code 不會當掉。
- 除非你的 deployment 會有 down time ，否則你的系統總有一定時間，是新的和舊的版本程式並存
  - 你不可能一秒內 deploy 100 個 docker container 的

Database refactoring 例子

- 本來，貓和兔子都存於 Pets 這個資料表內
- 新的改動，需要為貓和兔子都有自己專門的屬性，所以需要分成 Cats ， Rabbits
- 為了支援 legacy code ， Pets 這個表是暫時需要保留的
- 所有 Cats 和 Rabbits 的改動，都要用 trigger 去重新 Pets 的數據，反之亦然。
  - 小心 cyclic dependency 問題！

Tigger 缺點

- trigger 本身是 functional programming ，一旦系統變複雜了便難以除錯
- 把 business logic 寫在 application layer 內，有什麼錯了還能直接連進資料庫直接跑 SQL 改正
- 一旦 business logic 寫在 RDBMS 內，有問題時很可能沒法即時做 data patch 的
- 除非你要做 audit log / database refactoring ，否則絕對別用 trigger

簡介 view

- 除非你要做 database refactoring ，否則絕對別用 updatable view
- 你可以想像： view 是單純的 placeholder ，用一個名字來替代本來的 subquery
- 理論上：
  - 用了 view 和原本的 Query 對比，其 execution plan 應該是相同的，效能也應該 100% 相同

view 的用途

- 上古時代用途：
  - 你的資料庫需要讓第三方直接連線來拿資料，但是你有些敏感的 column 不能讓他們知道
  - 你要寫報表時，其 SQL 又臭又長，你用 view 來把 SQL 拆開來變得容易閱讀

完