RDBMS 課程 v2（第四課）

by

Triton Ho 

今天大綱

- SQL:2003
- 常用的 SQL 語法
- 大型系統架構常犯錯誤
- 課程總結

SQL:2003

SQL:2003 前言

- 某程度上， opensource 世界對 RDBMS 的發展不算成功
- 明明 SQL:2003 已經超過十年歷史了， MySQL 5.7 居然還停在更遠古更不方便的 SQL:1992
  - 連 mariadb 10.2 都支援了 window function
  - 別跟我說 MySQL 8.0 支援，現在還未正式 public release 耶！
- 而各家廠商在標準 SQL 上再加進自己的東西，然後卻對標準 SQL 這法不全面支持，更加讓人容易迷失
  - 例子１： Oracle 的 (+) 語法
  - 例子２： MySQL 的 Upsert

SQL:2003

- 就是說： SQL:1992 ， SQL:1999 的之後的標準
  - ~~雖然 SQL 從來沒有標準可言.....~~
  - 理論上：這能幫忙 developer 轉換不同的 RDBMS 廠商
- 新功能
  - window function
  - MERGE
- 各家 RDBMS 的情況.....
  - MySQL 5.7 不支持 , mariadb 10.2 終於支持
  - PostgreSQL 不支持 MERGE
  - Oracle 全面支持
  - ~~MSSQL 聲稱支持.....~~

常用的 SQL 語法

前言

- 雖然 optimizer 變得越來越先進，但請別測試 optimizer 的極限
  - optimizer 有其極限，蠢材沒有.....
- 再說一次： SQL 是宣告式的語言（理想上）
  大部份情況，你的 Query 應該是短而優美的
- 如果你的 Query 很長，很有可能是你不懂得現代 SQL 的好用功能

淺談 SQL 語法

- Case when 語法
- Window function
- With clause
- Recursive query
- Merge
- Insert into..... select

Case when 語法

- 讓你能在 Query 中使用 if-then-else 和 switching
  - 補充：一般來說，在報表 / 資料修正時會比較常用這語法
- 例子：我想在學生報表中，首先輸出女生，然後再輸出男生，其中女生以身高順序排序，男生以身高倒序排序
  - Select * from students
    order by gender,
    (
        case
        when gender = 'female' then height
        when gender = 'male' then height * -1
        end
)

Case-when 語法（續）

- 我想找到吃鮪魚比鮭魚多的貓

```
select cat_id from (
    select cat_id,
    sum(case when food_type = 'tuna' then amount else 0
end) as tuna_amount,
    sum(case when food_type = 'salmon' then amount else 0
end) as salmon_amount
    from food_consumption
    where food_type in ('tuna', 'salmon')
    group by cat_id
) t
where tuna_amount > salmon_amount
```

Window function 背景

- 如果有一萬筆資料，正常人會希望做分頁。把資料排序後，每次只顯示一小部份
  - 例子：第一頁顯示第１－１００筆資料，第二頁顯示第１０１－２００筆資料
  - 英文名字： pagination
- 有笨蛋會把整個 resultset 放到 application tier ，然後把用戶想要的第１－１００筆資料傳給用戶
  - 絕對別做這種笨事

Window function

- 還沒有 SQL:2003 前，如果你需要 pagination ，你需要使用各家 RDBMS 的專用語法
  - Oracle 的 rownum ， postgreSQL 的 Limit ， MySQL 的 top
- 現在 SQL:2003 的 window function ，把 pagination 統一化，而且功能遠遠更加強大

Window function（例子１）

- 我想要 1A 班以成績排序的學生名單，我需要第１１－２０人
  - select * from (
    select class_id, student_id,
    row_number() over (order by score desc) as row_num
    from students
    where class_id = '1A'
) t
    where t.row_num between 11 and 20

Window function（例子 2 ）

- 我想要每班成績最高的首三個學生
  - （註：以下 query 如有多人同分，則每班會返回多於三個學生）

```
Select * from (
    SELECT class_id, student_id,
    rank() over (partition by class_id order by score desc) as ranking
    FROM students
) t
where t.ranking <= 3
```

With clause 還未發明前

- 對於複雜的報表， SQL:1999 很容易越來越亂，而且相同的邏輯不容易被重複利用
- 例子：我想要知道每班成績首三名的學生中，他們之間有兄弟姐妹關係的人

With clause 還未發明前 (續 1)

```
Select * from (
    Select * from (
        SELECT class_id, student_id, parent_id
        rank() over (partition by class_id order by score desc) as ranking
        FROM students
    ) t1
    where t1.ranking <= 3
) wtf1
where exists (
    Select 1 from (
        SELECT class_id, student_id, parent_id
        rank() over (partition by class_id order by score desc) as ranking
        FROM students
    ) t2
    where t2.ranking <= 3
    and wtf1.student_id != t2.student_id and wtf1.parent_id = t2.parent_id
)
```

With clause 還未發明前 (續 2)

- 在 1992 年，使用 view 把複雜而且將會重用的邏輯包起來是好主意
- 在 2018 年，除了 database refactoring 之外不應使用，原因：
  - 讓 Query 變得不再直覺化
  - view 再包裝 view 再包裝 view ，在 debug 時會讓人想死
  - 這樣的 view 一般只給單一報表應用，大量這類型的 view 對程式碼管理帶來困難
    - DBA 常常遇上的問題：這個 view 是誰在用的？能刪嗎？

With clause 還未發明前 (續 3)

```
create view top3students as
select * from (
	SELECT class_id, student_id, parent_id
	rank() over (partition by class_id order by score desc) as ranking
	FROM students
) t1
where t1.ranking <= 3;

Select * from top3students t1
where exists (
	Select 1 from top3students t2
	where t1.student_id != t2.student_id and t1.parent_id = t2.parent_id
)
```

With clause

- 能在 Query statement 中，建立只給這 statement 使用的 view
- 因為大型的報表的邏輯被 modularize 成多個 view ，讓 Query 的可讀性大幅提升
- 這些只由單一 Query 使用的 view 跟 query statement 放到一起，所以 debug 時不再用打開多個檔案了
  - 也不用再擔心報表不再使用刪除後，忘記了刪除相關的 view

With clause 例子

```
With top3students as (
    select * from (
        SELECT class_id, student_id, parent_id
        rank() over (partition by class_id order by score desc) as ranking
        FROM students
    ) t1
    where t1.ranking <= 3
)
Select * from top3students t1
where exists (
    Select 1 from top3students t2
    where t1.student_id != t2.student_id and t1.parent_id = t2.parent_id
)
```

Recursive Query 背景

- 在購物商城系統中，老闆除了種類之外，還想在種類之下建立子種類，子子種類.....
  - ~~這傢伙的中文真的沒問題嗎？~~
  - 在電腦這個種類下細分（ CPU ， RAM ，顯示卡）
    在顯示卡這個種類下細分（ AMD ， Nvidia ）
    在 AMD 這個種類下細分（ Asus ， MSI ，..... ）
- 然後某一天，老闆想要知道商城中的所有顯示卡資料.....

Recursive Query

- 這不是標準的 SQL 語法，也從未被統一
  - 雖然 Oracle, PostgreSQL, MySQL 都有支持
- 在報表中， Recursive Query 一般用來處理樹狀 (tree-like)，層次性 (hierarchical) 的資料
  - 例子：家族族譜，物種分類資料
- 好處：能讓本來需要拆碎成多個 query 報表，用單一 Recursive Query 完成
  - 資料不用在 application tier 和 RDBMS 之間傳來傳去
  - 開發上更快速（只寫一句 Query 總比多個 query 好吧）

如果沒有 Recursive Query

- step1: 找出顯示卡的 primary key
  select id from category where name = ' 顯示卡 '
- step2: 找出顯示卡下屬的子分類
  select id from category parent_id = @id
- step3: 找出顯示卡下屬的子子分類
  select id from category parent_id in (@id1, @id2...)
- step4: 找出顯示卡的記錄
  select * from items where category_id in (@id1, @id2...)

Recursive Query 例子（續）

```
with recursive all_category(id) as
(
    select id from category c
    where id = $displayCardId
    union all
    select c.id from category c
    inner join all_category parent on c.parent_id =
parent.id
)
select * from items
where items.category_id in (select id from all_category)
```

MERGE

- 雖然是 SQL:2003 的一部份，但是只有 Oracle 有限度支援
- 在 PostgreSQL, MySQL 相似的是 INSERT ... ON DUPLICATE KEY UPDATE
- Upsert 與 MVCC 不太相容
  - 所以 Oracle 的實作是很奇怪的（笑~），MERGE 有機會發生 dup_val_on_index 和 no_data_found
  - 在 Repeatable Read 時，要如何為我看不見的資料做 update ？
- 在 OLTP ，請盡量迴避 Upsert 行為，好的系統設計不應該發生 Upsert 的

Insert into..... select

- 如果你需要把資料從資料庫的一些 table 抄送到另一 table ，請不要利用 select 把資料放到 application tier ，然後再用 insert
  - Network IO 和 SQL parsing 很花時間的
- 這個語法，能把 select 得到的 resultSet 直接 insert 目標 table 上
- 對於數據處理還有報表，這語法很好用的

系統架構常犯錯誤

大型系統架構常犯錯誤

- API 設計錯誤
- 沒有預防 positive feedback
- 沒有過濾不正常的 request
- 不使用 caching
- 只使用 local cache
- 錯誤設定 caching
- 沒有思考 hotdata hotspot
- 錯誤地處理 cache miss
- 對於大型 Request，沒有使用 asynchronous
- 對於「即時性」用戶互動，沒有使用 server push
- 沒有良好備份

API 設計錯誤

- 絕對不要為了省時，把單一行動用 2 個 Request 來處理
- 例子：現在有了 /v1/{userId}/user_balances (PUT) 這個 API 來讓用戶改動其戶口金額
- 如果要做轉帳的功能，請使用 /v1/bank_transfer/(POST)
- 把單一行動拆成 2 個 Request ，會讓系統停留在不正確的中間階段 (inconsistent state) 。
  - race condition 可能性大增
  - 萬一客戶端當掉，系統數據將會長期停在不正確狀態

什麼是 positive feedback

- ~~喵的！別每次當機都說有 cracker 攻擊好嗎？你的爛系統有什麼東西值得 cracker 來攻擊~~
- 對一般用戶來說，１０秒後網頁還未顯示，便會按下 <F5>
  - 當系統繁忙時，網頁顯示緩慢
  - 網頁顯示緩慢，用戶按下 <F5> 重試
  - 用戶按下 <F5> 是不會取消還在處理中的 Request
    反而製造了新的一個 Request
  - 因為多了新的 request ，系統更繁忙
- 會有突然性的高流量的系統，特別容易進入 positive feedback
  - 例子： COSCUP 搶票

面對 positive feedback

- 在設計介面時，盡量讓用戶別按 <F5>
  - 先把網頁的一部份顯示出來，會讓用戶感覺上好一點
- 動態增加系統容量對 positive feedback 不一定有幫助
  - 因為起動新的主機需要時間 (小心新的主機讓問題雪上加霜)
- 先把用戶的 Request 接收，然後告知用戶「收到了」，而不是「處理 OK 」
  - HTTP 200(OK) 和 HTTP 202(Accepted) 的分別
  - 很多網上訂位系統，都是先收下用戶指令，然後再慢慢處理的
- 如果單純接受用戶 Request 也無法處理：
  - 對部份用戶返回 HTTP 503
  - 拒絕部份用戶，總比全面性系統當機更好

面對不正常的 request

- 如果你是 WOW 代理商，現在你的系統能讓用戶去便利商店買卡存點。
  一定有大量的 cracker 想胡亂卡號和密碼來碰運氣
- 如果每一個 request 都檢查資料庫才知道其真偽。
  這些碰運氣的 request 便會浪費大量資料庫 IO
- 建議做法：在點數卡的密碼中加入 check digit
  在 application tier 單純檢查這些 check digit 便足以證明這個 request 是錯誤的

不使用 caching 的迷思

- 5 年前某隻白目的想法：
  - RDBMS 內部也有大量記憶體作 cache ，沒必要再使用 Redis / memcache 嘛
  - Read-only 的 query 可以交給 RDBMS 的 replication ，而 RDBMS 的 replication 可以無限增加的
- 結果：某隻白目當然被打臉了

使用 caching 原因

- 用來作 caching 的主機只要記憶體夠多便好（當然 network IO 也不能太差）。而 RDBMS replication 主機的 CPU 和 disk IO 也要很好，一台 replication 主機隨時等於數台 caching 主機價格。
- 現實中 master 的 network IO 有物理上限，不能連接上無限個 replication 的。

local caching 的意思

- 這不是指把 cache 和 web server 都放到同一台主機耶！

  webserver 主要吃 CPU 和 network IO
  cache(redis / memcache) 主要吃記憶體
  所以把他們放同一主機是正常的

- 我是指每台 web server 都只使用在本台主機上的 cache
  - 很多 ORM 也會內建 cache 的功能的，像 EHcache

local caching 的問題

- 因為單一 web server 的記憶體是固定體積的，所以隨系統變大， cache hit rate 會越來越低
- 在系統繁忙時，新增加的主機的 cache 是空的。所以這主機的 Request 所需要的資料都要由 RDBMS 提供。
- 最簡單來說：這樣的設計，會讓系統在繁忙時更加繁忙

錯誤設定 caching

- 一般來說： cache 會散落在 n 台 caching 主機
- 決定一份資料應該儲存在那一台主機，最簡單的方法便是：

  mod(md5(key), n)
- 在增加主機到 m 台時，如果單純地把 n 改成 m ，將會瞬間讓全部的 cache 失效
  - 又是典型的在系統繁忙時，讓系統更加繁忙 ^^
- 在增加主機後的一段時間：

  先根據 mod(md5(key), m) ，在新位置看看有沒有資料
  如果新位置沒有資料，便從舊位置 mod(md5(key), n) 拿資料

沒有思考 hotdata hotspot

- 如果 hotdata 只根據 mod(md5(key), n) 來存到單一 redis 主機.....
  - 如果這份資料被數萬個 request 同時使用時，這數萬個 request 便會集中在單一 redis 主機上
- 再次強調：單一主機的 network IO 有其物理極限的，這 redis 主機便會成為系統效能瓶頸
- 單純把資料存到 x 台主機，只是讓系統效能瓶頸提升 x 倍，而且對於９９％不太高用量的資料，這只是單純浪費記憶體
  - 真正解決方案我還在學習中
  - facebook 花了 2 年時間去改善其 caching 演算法

沒有思考 hotdata hotspot（續）

- 例子：用 redis sorted set 來存全體用戶的分數排名榜
  - 所有用戶的分數改動都需要改動這個 sorted set
  - 單一的 sorted set 只存到單一的 redis 上
  - 所有用戶的分數改動都去了單一的 redis 上
- 方案
  - 對排名榜作 sharding by userID ？
  - 只存首 X 個高分數用戶進 redis ？

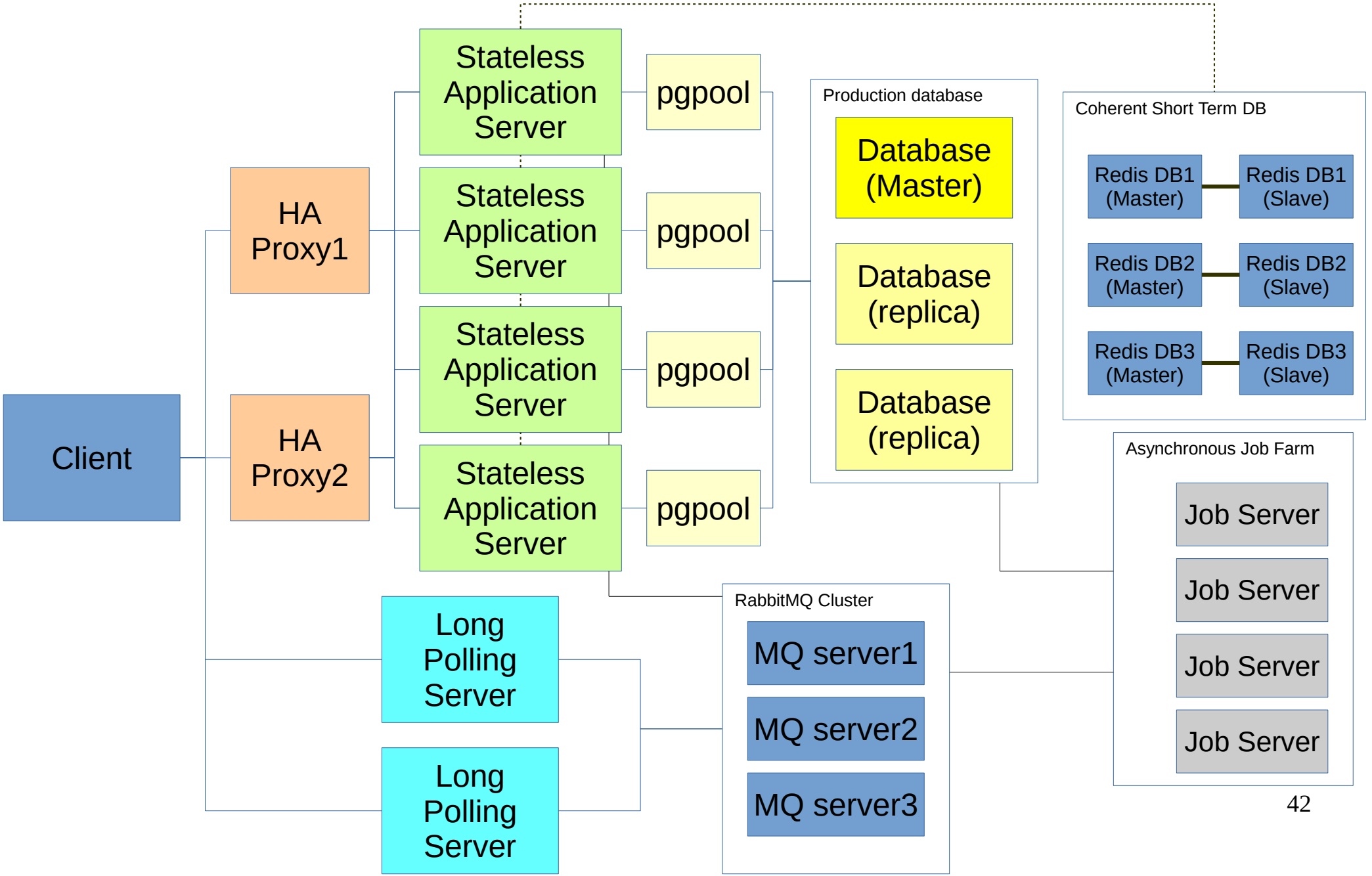
錯誤地處理 cache miss

- 正確的流程：
  - Try to get data X from cache, return if cache hit
  - Get global lock X
  - Try to get data X from cache, return if cache hit
  - Get data X from Main DB
  - Put data into cache
  - Release lock X
  - Return data

使用 Asynchronous Request

- 如果你在 web server 生產一份大型報表，需要使用二小時才能完成.....
  - 如果用戶這二小時內關掉 firefox ，便要從頭再來
  - php 的 default session 只有 24 分鐘，你需要用戶在這二小時內每 24 分鐘便活動一下.....
- 需要長時間完成的行動，先天性容易發生 positive feedback
- 理想做法：
  - 收下用戶生產報表的指令後返回 HTTP 202
  - 在背後慢慢工作
  - 完成後把報表用 email 傳給用戶

大型系統架構圖（簡化版～）


Client
HA Proxy1
HA Proxy2
Stateless Application Server
Stateless Application Server
Stateless Application Server
Stateless Application Server
pgpool
pgpool
pgpool
pgpool
Production database
Database (Master)
Database (replica)
Database (replica)
Coherent Short Term DB
Redis DB1 (Master)
Redis DB1 (Slave)
Redis DB2 (Master)
Redis DB2 (Slave)
Redis DB3 (Master)
Redis DB3 (Slave)
Asynchronous Job Farm
Job Server
Job Server
Job Server
Job Server
RabbitMQ Cluster
MQ server1
MQ server2
MQ server3
Long Polling Server
Long Polling Server

別在 web server 進行大型工作

- Web server 設計是用來處理大量 + 密集 + 小型的工作，也為這而作出專門優化的
- 如果大型工作能不受限制地在 web server 上跑，系統資源會被這些大型工作全吃光掉。嚴重影響整體系統效能。
- 所以，大型工作應該先放進 Message Queue ，然後由專屬的 worker 一個接一個地執行

Server Push

- 正常的 facebook 訊息大約是在 1 秒內便送到對方的 messenger 上的
- 如果 messenger 是每 1 秒都進行一次 GET request ，問一下伺服器
  - ９０％時間，這是 GET Request 都是沒有資料傳送，白白浪費了 network IO 還有資料庫的資源
- 所以，應該是對方發訊息時，才直接進行 server push ，把訊息主動地推到對方手機上
- android 的 GCM ， apple 的 APNS 正是 server push 的一種例子

沒有良好備份

- 使用 Raid 1 時，~~一個打翻了的泡麵引起的~~電源短路將會讓磁碟全滅
  - Raid 1 中的磁碟是好兄弟，總會同時趴掉
- 即使有了異地備份，一個~~連續工作 48 小時的工程師 /~~ 軟體臭蟲可以完全毀滅系統數據
- 只有鎖在保險櫃的磁帶 / DVD 才是真正安全的備份
  - 但是要小心一點：你寫到磁帶的數據，是否真的能拿出來

  ~~(聽過太多要做系統復原時才發現一直使用的磁帶是壞的故事了)~~
  ~~(傳說中，２００６年地震後，才發現一些磁帶是壞的)~~

課程總結

課程總結

- 這一個星期，你應該：
  - ~~更加喜歡貓咪，理解領養代替購買~~
  - 理解 RDBMS 的 ACID ，明白 noSQL 不是 RDBMS 的替代品
  - 理解 race condition ，知道如何用 RDBMS 去防範 race condition
  - 理解 natural key 和 surrogate key
  - 對 contention 概念有基本理解
  - 知道 database schema 的重要性，也知道如何做出好的 schema
  - 知道怎樣去寫優美而且容易維護的 SQL ，也懂得什麼是 SQL:2003
  - 對大型系統有入門性認知

結語

- If I have seen further, it is by standing on the shoulders of giants. Isaac Newton, 1676
- 大部份情況，你正在面對的問題早已有人面對過，解決過了。
- 請繼承前人成果，然後再向前多走一步。別輕易去 reinvent the wheel 。

時間有限，本課程不可能把全部東西都教給你。
但是，希望本課程能為你打開進階知識的大門，
憑自己的力量繼續努力。

他日閣下長大後，遇上菜鳥
請發揮騎士精神，輕輕地拉他一下唷

完

威力加強版
如果我的 RDBMS 很慢時.....

如果我的 RDBMS 很慢時.....

- 身為 DBA ，便是以自己的專業知識，去重組案情，然後解釋給普通人的老闆去理解
- 重組案情時很可能犯錯，兇手也很可能不止有一個
  - 犯錯了便坦白承認再去找，有做事便有可能犯錯嘛
  - 別用 goodest logic 去為自己辯解，那才是最可恥

CPU 用量 100%

- 先別立即告訴老闆升級系統
- 如果你的 physical storage 是用了 compression ，而你正在讀取大量 compressed data ，這才肯定是 CPU-bound
  - 經驗來說，今天的 CPU 不值錢的， CPU-bound 沒那麼容易的
- 請小心虛假的 CPU 100%

別立即叫老闆升級電腦

- 先留下問題點的狀態
  - show engine innodb status
- 去看 Network IO, Disk IO, CPU usage, CPU IO Wait 數據
- 別忘記當時的 API 流量

如果是報表類 Query 很慢

- 如果是報表類 Query 很慢
  - 看看是否有笨蛋不會寫 Query
  - 除非這個 Query 是影響到 end-users（就是付錢給你公司的人），別輕易加上 index

如果是 OLTP 行動很慢

- 先看看是卡在什麼地方
- Logical blocked 還是 Page blocked ？
  - 前者請看 application source code
  - 後者請看 application source code + database tuning
- 還是出現了 hotspot ？
- 還是 deadlock ？
  - Mysql / mariadb 很常發生