SQL Reporting 淺談 v2.0

by

Triton Ho 

今天內容

- 簡單例子
- 基礎觀念
- ISO SQL:2008 標準

前言

- 沒人叫你把一切數據分析都用 SQL 來跑
  - TB 級數據面前， SQL 有高度局限性
  - Python 和 R 的統計功能非常強大，而且能用 GPU
- 但是……
  - 一般中小型公司的數據量＜ 100GB
  - SQL 是宣告式語言，語法簡單短小
  - 用 Python 和 R 來做分析時，還是要用 SQL 先拿資料出來

簡單例子 1

- 找出每個家庭的最年長的孩子

```
Select * from residents r1
where not exists (
    select 1 from residents r2
    where r1.parent_id = r2.parent_id
    and r2.age > r1.age
)
```

簡單例子 2

- 找出在多於一個倉庫中有存貨的產品

```
Select product_id
from inventories t1
where amount > 0
group by product_id
having count(distinct warehouse_id) > 1
```

簡單例子３

- 即時計算玩家的行動點數

```
select id, least(100, candy +
trunc(extract(epoch from current_timestamp -
last_candy_time) / 60 / 8)
) as candy
from players
```

簡單例子 4

- 把用戶的身高，以每 5cm 為單位，統計用戶人數

```
select coalesce(t1.counter, 0) as counter
from generate_series(1,20) t2
left join (
    select WIDTH_BUCKET(height, 140, 200, 12) as
busket_id,
    count(*) as counter
    from users
    group by busket_id
) t1 on t1.busket_id = t2
order by t2
```

簡單例子 5

- 以每 5% 作解像度，計算用戶的身高百分位數 (percentile)

  select percentile_cont(array(SELECT j/20.0 FROM generate_series(1,20) j)) within group (order by height) as height_percentile
  from users

基礎觀念

基礎觀念

- 避免 Select *
- 避免散碎的 Query
- SQL Join 並不慢
- Seq. Scan 不一定是壞事
- Caching 和 Seq. Scan
- Query Optimizer 淺談
- 善用 CTAS 和 Temp table

避免 Select *

- 請盡量把你需要的 column 明確地寫出來
- 別浪費 Network IO
  - 特別是你的資料量是 GB 級時
- 有可能會有更好的 execution plan
  - 例子： Index-only scan

笨例子 1

```
idArray := []int{1, 43, 76, 757}
for _, id := range idArray {
    q := `select * from products where id = ?`
    db.RunQuery(q, id)
    ......
}
```

避免散碎的 Query

- 網絡通訊會吃掉時間
- SQL Parsing 和優化也吃時間
- 重要觀念：
  - 數據運算應該接近數據所在的位置
    (Computation should be close to the data)
  - 另一個說法：如果必需，先把資料移動，然後才做運算
- 簡單來說：
  - 你的 Query 次數應該跟數據量無關

好例子 1

```
idArray := []int{1, 43, 76, 757}

temp := []string{}
for _, _ := range idArray {
    temp = append(temp, `? `)
}
q := `select * from products ` +
` where id in (` + strings.Join(temp, `,`) + `)`
db.RunQuery(q, id...)
```

思考： Dijkstra's 演算法

- 如果數據量不大，乾脆把整個 graph 丟到 application 記憶體吧 ~
- 數據量很大時：
  - 寫 stored procedure(pl/pgsql) ，在資料庫上面跑算法 ~
  - 每次拿路徑 (edge) 資料時，除了目標節點 (node) ，還把相鄰的 (node) 也拿出來，然後做好 Caching

SQL Join 並不慢 1

- 例子：在顯示機票資料時，我們要顯示相關的航次資料

```
select * from flight_ticket ft
inner join flight f on ft.flight_id = f.id
where ft.id = @ticket_id
```

SQL Join 並不慢 2

- ft.id = @ticket_id 是以 Primary key 來查詢
  - 即是說：這個最多只有一個結果
- f.id 是 flight 這個表的 PK ，所以會有對應的 index
  - 所以，這個 f.id 會放在 index nested loop join 的內圈

Index nested loop join

- Step1: ft.id = @ticket_id ，對 flight_ticket 做搜查
- Step2:
  - foreach candidate ft in flight_ticket {
    f := flight.getRecordByIndex(ft.flight_id)
    if f exists, add <f, ft> into result
    }

Seq. Scan 不一定是壞事

- 每個人都知道：
  - 抄一個 10GB 檔案，比一千個 1MB 檔案更快
- 在報表類應用， RDBMS 一般會無視 secondary index(i.e. non-PK)

Seq. Scan with PK

- 只適用於 clustered index
  - Oracle Index-organized-table
  - Mariadb(innodb) table
- 如果你想拿某時段的資料
  - create_time between '2016-01-01' and '2017-12-31'
  - 做年度報表經常用上
- Extra predicate: “id between XXX and YYY” 可能有幫忙

•Caching 和 Seq. Scan

- 對 5GB table 作 Seq. Scan 時，如果你資料庫只有 4GB RAM
  - Mariadb 是用 LRU cache 的
    這情況下 cache hit rate = 0
  - Oracle / Postgresql 不會對 Seq. Scan 作 caching
- 在報表類 Query 時， RAM 是用作：
  - 暫時儲存中間的運算結果
  - 對 Random IO 作 Caching
    （例子： index join）

Query Optimizer 淺談

- 在 oracle / postgreSQL ，以下 Query 效能相同
  - Select child.id from child
    inner join parent on child.parent_id = parent.id
    where parent.name = 'TritonHo'

  - Select child.id from child
    where child.parent_id =
    (select id from parent where parent.name = 'TritonHo')

  - Select child.id from child
    where exists
    (
        select 1 from parent
        where child.parent_id = parent.id and parent.name = 'TritonHo'
)

SQL Optimizer

- 再次強調： SQL 是宣告式語言
- 好的 SQL optimizer 讓使用者不用思考
  - joining sequence
  - predicate location
- 好的 SQL optimizer 會把 sub-query 改成 join
- Optimizer Ranking
  - Oracle > Postgresql >>> Mariadb >= MySQL

SQL Optimizer （續）

- 天材有其極限，笨蛋沒有
- SQL optimizer 的工作時間 ~= O(2^n)
  n = # of table
- 如果 n 太大， optimizer 會改用猜猜樂 ~
  猜猜樂的好聽說法： heuristic ／先進預測
  - 個人看法：報表類 SQL ，不建議超過５０行

Temp. Table

- 在 TX 結束時，其內資料全部被刪除
- Temp. Table 內資料無法被其他 connection 看到
- 效能比一般的 Table 快很多
  - 有些資料庫支援以 RAM 作 Temp. Table 的儲存空間
- 能做 insert / update / delete

Temp. Table 和 CTAS

- CTAS = create table as select ……
- 使用 CTAS ，你能把中間的結果用 Temp. Table 暫存起來
- 所以，你可以把一個複雜的 SQL 變成數個簡單的
- 以下例子：要找到每個家庭的第二個孩子

Temp. Table 例子 1

- Step 1: 找到最年長的孩子，然後把結果存到 temp1

```
create temp table temp1 as
Select id from residents r1
where not exists (
    select 1 from residents r2
    where r1.parent_id = r2.parent_id
    and r2.age > r1.age
)
```

Temp. Table 例子 2

- Step 2: 把本來的 students ，扣掉 temp1 的記錄，再存到 temp2

```
create temp table temp2 as
Select id from residents r1
where r1.id not in (
    select temp1.id from temp1
)
```

Temp. Table 例子 3

- Step 3: 從 temp2 中找到其內每個家庭最年長的孩子

```
Select id from temp2 a
    where not exists (
    select 1 from temp2 b
    where a.parent_id = b.parent_id
    and b.age > a.age
)
```

ISO SQL:2008 標準

ISO SQL 里程碑

- SQL:1999
  - With-Clause
- SQL:2003
  - Window function
  - CTAS
- SQL:2008
  - Case-when expression
- MySQL 5.7 / MariaDB 10.1 目前支援度：
  - SQL:1992

沒有 with-clause 時

```
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
    where students.group_id = 2
    ) 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
    where students.group_id = 2
    ) t2
    where t2.ranking <= 3
    and wtf1.student_id != t2.student_id and wtf1.parent_id = t2.parent_id
)
```

With-clause

- 再說一次：
  - MariaDB 使用者，你可以今天後回家哭哭了 ~
- 讓你建立 Just-in-time ，單次性使用的 view
- 自動（強制性的） materialization
  - 會影響到 Execution plan ，是好事也是壞事
  - 需要手動 predicate pushing

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
            where group_id = 2
      ) 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
)
```

predicate pushing

- 把 predicate 從外層推送到內層 (i.e. subquery)
- 正常來說， query optimizer 會替你自動完成
  - 還是一句： Oracle > Postgresql >>> Mariadb >= MySQL
- query optimizer 無法 predicate pushing 例子：
  - Materialization(e.g. with-clause)
  - Aggregation(COUNT, SUM, AVG)
  - Window function

壞例子

```
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
    and t1.group_id = t2.group_id
)
and t1.group_id = 2
```

瘋狂例子

```
Select group_id, avg(score)
from students
group by group_id
having group_id = 2
```

Window-function

- 在 SQL:2003 之前，如果要做分頁 (pagination)：

  select * from students
  where …… order by id limit 50

- 沒有統一標準
  - Oracle: rownum
  - Postgresql / Mariadb: limit and offset
  - MSSQL: Top
- Window function 是統一標準，所有 ISO SQL:2003 compliant 都支援

Window-function 例子 1

- 分頁功能

```
select * from (
    select *,
    row_number() over (order by score desc ) as row_num
    from students
    where ……
) t
where t.row_num between 11 and 20
```

Window-function 例子 2

- 想拿到每班別的首三名學生：

```
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
```

- 遠遠優勝的效能
  - table 只需要被掃瞄一次
  - 不需要 joining ，也不需要 union

Window-function 例子 3

- Facebook-like thread

```
WITH RECURSIVE message_with_replies as (
  select * from (
    SELECT to_char(row_number() over(order by create_time asc), 'fm0000000') as seq, ''::text as parent_seq,
null::bigint as row_num, *
    FROM filtered_messages
    where ……
    order by create_time asc
    offset ? Limit ?
  ) t1
  UNION ALL
  select parent_seq || '-' || to_char(row_num, 'fm0000000') as seq, * from (
    SELECT p.seq as parent_seq, row_number() over(partition by r.parent_id order by r.create_time asc) as row_num,
r.*
    FROM message_with_replies p
    inner join filtered_messages r on p.id = r.parent_id
  ) t2
  where row_num <= ?
), filtered_messages as (
  select * from messages where ……
)
SELECT * from message_with_replies
order by seq
```

Window-function 例子 4

- 作為 syntax sugar

```
SELECT class_id, student_id, score
avg(score) over (partition by class_id) as avg_score
FROM students
```

- 傳統的 group-by:
  -
```
Select class_id, student_id, score, t1.avg_score
from students s1 inner join (
    select class_id, avg(score) as avg_score
    from students s2
    group by class_id
) t2 on s1.class_id = t2.class_id
```

Case-when

- 容許你在 SQL 內做到 switch
- 相似例子： coalesce

Case-when 例子

- 在拍班級照片時，你想左邊全是女生，右邊全是男生
  學生高度是兩邊向中央升高
- Select * from students
  order by gender,
  (
  case
  when gender = 'female' then height
  when gender = 'male' then height * -1
  end
)

結語

- SQL 是宣告式語言
  - 優美而且功能強大
  - 在 GB 級數據量，能快速寫好
  - 易學難精＞＜"
- MySQL / Mariadb 最大問題：
  - 只去做搶眼球的門面功夫
  - 但是卻不肯練好內功

End