Effective Data Analysis and ISO 2008 SQL Features

by

Triton Ho 

Today Contents

- Introductory Examples
- Basic Concepts on Reporting
- ISO SQL:2008 standard

Introductory Examples

Introductory Examples 1

- Find out the eldest child for each family

```
Select * from students s1
where not exists (
    select 1 from students s2
    where s1.parent_id = s2.parent_id
    and s2.age > s1.age
)
```

Introductory Examples 2

- Find out the products stored in multiple warehouses

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

Introductory Examples 3

- Just-in-time calculation of player's action point

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

Basic Concepts on Reporting

Basic Concepts on Reporting

- Avoid select *
- Cluster your query
- SQL Join is not slow
- Seq. Scan is not bad
- Caching and Seq. Scan
- Introduction to Query Optimizer
- CTAS with Temp. Table

Avoid select *

- Always explicitly list out your columns
- Avoid wasteful network IO
- Maybe better execution plan
  - Index-only-scan?

Stupid Example 1

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

Cluster your Query

- Network overhead costs time
- SQL parsing and optimization takes time
- Computation should be close to the data
  - If necessary, move your data
- In Summary: your number of query should be independent of your data size.

Good Example 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...)
```

Brainstorm: Dijkstra's algorithm

- If the dataset is small, query all edges data into memory.
- Write stored procedure and run the algorithm in RDBMS directly
- Cache the edges data in application tier

SQL Join is not slow

- Example: we want to display the flight details in additional to the flight ticket.

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

Explain the SQL

- ft.id = @ticket_id is filtering on PK
  - flight_ticket has one record only
- f.id is the PK of flight, and thus has index
  - It will be the inner loop of index nested loop join

Index nested loop join

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

Seq. Scan is not bad

- Every man knows:
  - Copying one 10GB movie file is faster than copying one thousand 1MB jpeg files
- Usually, RDBMS ignore secondary index(i.e. non-PK)

Seq. Scan with PK

- Applicable to clustered index only
  - Oracle IOT, Mariadb(innodb) table
- Sometimes, you want to get records of a period.
  - e.g. create_time between '2016-01-01' and '2017-12-31'
- Extra predicate: “id between XXX and YYY” may help to reduce the seq. scan range

Caching and Seq. Scan

- Seq. Scan on 5GB table ALWAYS get cache miss on 4GB RAM LRU cache.
  - Oracle / Postgresql on purposely skip caching on Seq. Scan
- For reporting, RAM is useful for
  - storing intermediate dataset
  - Caching random access(e.g. index join)

Introduction to Query Optimizer

- In oracle / postgreSQL, the following SQL are SAME
  - 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

- Good SQL optimizer ignores
  - joining sequence
  - Location of predicate (i.e. in where clause or as a joining condition)
- Good SQL optimizer transforms sub-query into equivalent join statement
- Optimizer Ranking
  - Oracle > Postgresql >>> Mariadb >= MySQL

SQL Optimizer(cont'd)

- The difference between genius and stupidity is that genius has its limits.
- Runtime of SQL optimizer is O(2^n)
  n = # of table involved
- If n is too large, optimizer will give up and use heuristic
  - My experience: >50 lines is dangerous for reporting

Temp. Table

- on transaction rollback / commit, automatic deleted
- Data in Temp. Table will not be shared between transactions
- Much faster than normal tables.
- You can perform insert / update / delete

Temp. Table and CTAS

- CTAS = create table as select ……
- Using CTAS, you can store intermediate dataset easily
- Thus, you can break one complex query into multiple simple query
- Example: find the 2nd child in all families

Temp. Table example

- Step 1: find the 1st child and save the result

```
create temp table temp1 as
Select id from students s1
where not exists (
    select 1 from students s2
    where s1.parent_id = s2.parent_id
    and s2.age > s1.age
)
```

Temp. Table example

- Step 2: minus the 1st child and save the result

```
create temp table t2 as
Select id from students s1
where s1.id not in (
    select temp1.id from temp1
)
```

Temp. Table example

- Step 3: get the oldest child in the dataset

```
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 standard

Great Milestone of ISO SQL

- SQL:1999
  - With-Clause
- SQL:2003
  - Window function
  - CTAS
- SQL:2008
  - Case-when expression
- MySQL / MariaDB
  - SQL:1992

Without with-clause

```
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

- Not support by MySQL / MariaDB
- Allow just-in-time view declaration
- Implicit materialization
  - Affect execution plan, both good and bad
  - Need manual predicate pushing

With-clause example

```
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

- “push“ your predicate to inner layer(i.e. subquery)
- Automatically done by query optimizer
  - Again: Oracle > Postgresql >>> Mariadb >= MySQL
- Blocked by
  - Materialization(e.g. with-clause)
  - Aggregation(COUNT, SUM, AVG)
  - Window function

Bad example

```
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
```

Crazy example

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

Window-function

- Before SQL:2003, pagination example:

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

- Pagination is not standardized
  - Oracle: rownum
  - Postgresql / Mariadb: limit and offset
  - MSSQL: Top
- Window function is standardized and supported by all SQL:2003 compliant DB
  - Once again: Sorry for Mariadb users

Window-function Example 1

- Pagination by window function

```
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 Example 2

- Top x item in each group

```
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
```

- Much higher performance
  - Single pass on the table
  - No joining, no union

Window-function Example 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 Example 4

- As a syntax sugar for aggregation

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

- Traditional 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 expression

- Allow you perform “switch“
- Similar function: coalesce

Case-when Example

- You want to get girls in ascending order, and then boys in descending order

```
Select * from students
order by gender,
(
    case
        when gender = 'female' then height
        when gender = 'male' then height * -1
    end
)
```

Conclusion

- SQL is a declarative language
  - Elegant and Powerful
  - Easy to write, hard to master
- Unless debugging, optimizer hints is bad idea

Some bad example

```
Select …….
from tableA
left join tableB on tableA.XXX = tableB.YYY
where tableB.ZZZ = ?
```

End