
Beaver
Country Day School
dbDao

dbDao

MySQL查询性能优化

汪伟华

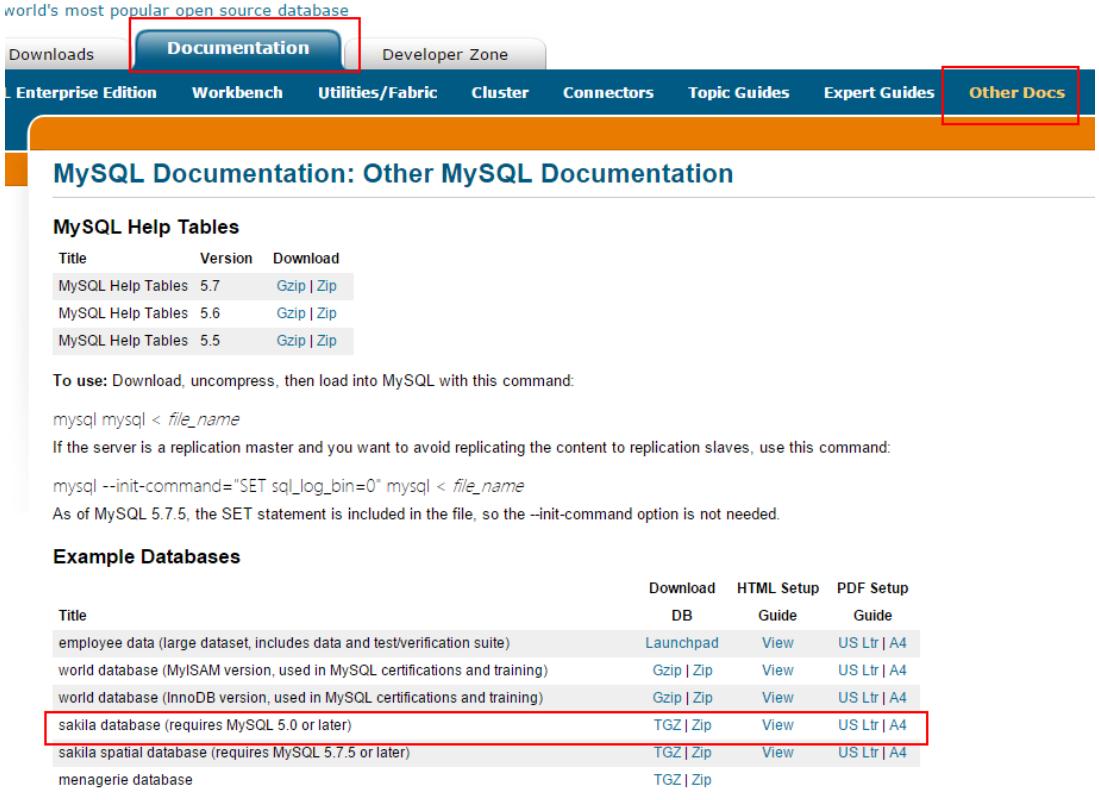
课程介绍

- 数据访问优化
- 了解MySQL查询优化过程
  - 查询的执行路径
  - 了解查询状态
  - 了解EXPLAIN命令的使用
- 通过实例来理解查询性能优化
- 最佳实践建议

课程准备

- 课程所例子都会

使用sakila数据库进行演示



sakila数据库下载地址:

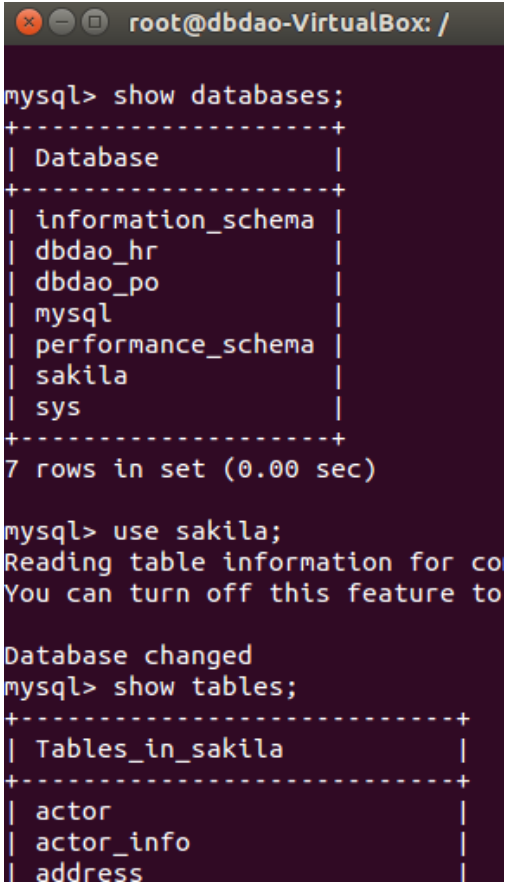
http://dev.mysql.com/doc/index-other.html

课程准备

- sakila数据库安装
  - 先下载到系统临时目录并解压
  - 之后使用root用户登陆MySQL进行数据库导入

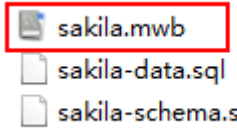
```
-- 解压安装包
shell> unzip /tmp/sakila-db.zip

-- 数据库导入
shell> mysql -uroot -p
mysql> source /tmp/sakila-db/sakila-schema.sql;
mysql> source /tmp/sakila-db/sakila-data.sql;
```

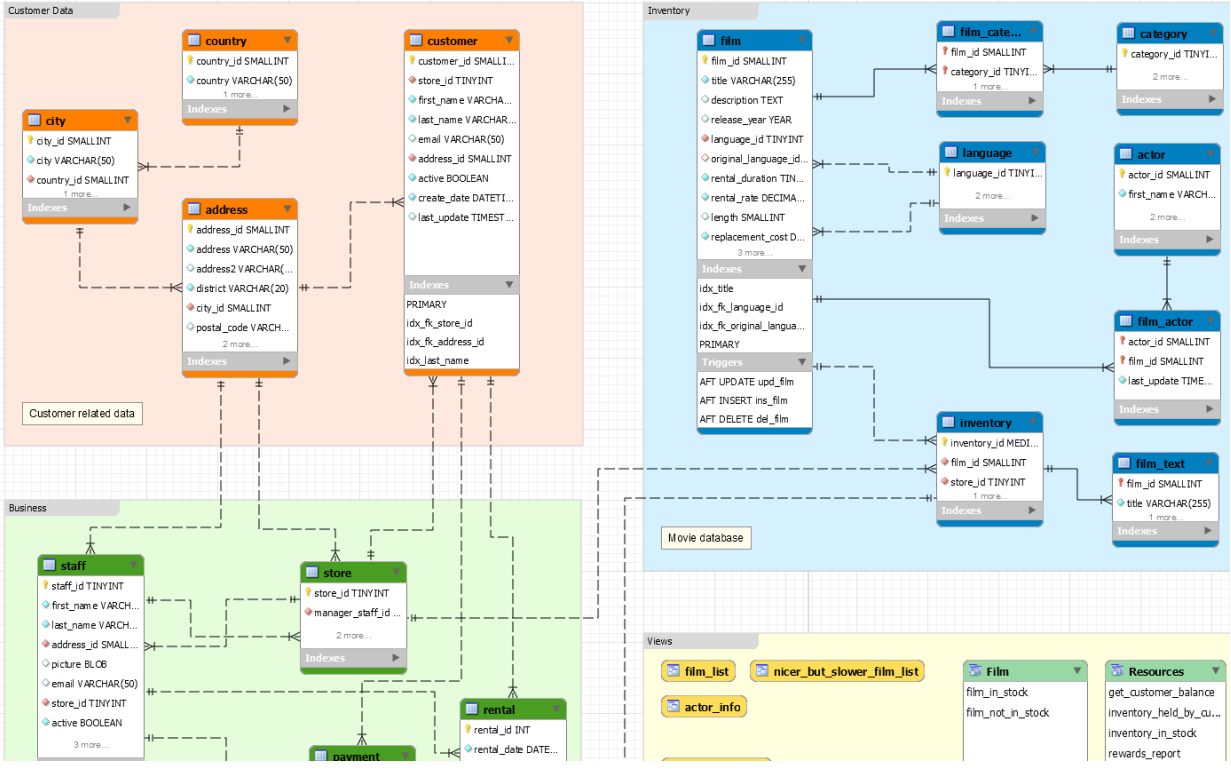


课程准备

- 举例所用表关系



akila数据库安装包中包含有sakila.mwb文件，可以通过MySQL Workbench打开查看其数据库ER关系图
(右图仅展示了其中部分关系)



数据访问优化

大量被问及的性能问题源于低质量查询，
原因是返回大量应用实际上不用的数据

开发人员常常会这样回答：

- 返回数据多了没事，如果少了，下次说不定需求要了还得改代码，留着呗
- 只是多出几列数据，没事
- 我们应用中会实现分页，所以把整个结果集都查出来
- 我们仅多取出了一些行，业务用不上也无所谓

理解业务需求，缩小查询数据结果集

- 仅查询应用所需的行
  - 相应使用WHERE语法
- 仅查询应用所需的列
  - 避免使用SELECT *, 请尽量在SELECT语句中列出列名
- 避免多次获取相同的数据
  - 使用应用缓冲来临时存取数据
- 更多在数据库层上进行ORDER BY操作
  - 在SELECT语法中使用ORDER BY，而不是到了应用中再进行排序

Demo – 1-1

```
USE sakila;

-- Regular select statement
SELECT *
  FROM actor;

-- Use WHERE clause
SELECT *
  FROM actor
 WHERE first_name LIKE 'A%';

-- List clause
SELECT actor_id, first_name, last_name
  FROM actor
 WHERE first_name LIKE 'A%';

-- Order By clause
SELECT actor_id, first_name, last_name
  FROM actor
 WHERE first_name LIKE 'A%'
 ORDER BY last_name;
```

Demo – 1-2

```
------------------------------
-- Why Star is NOT good idea
SELECT *
  FROM customer;

SELECT *
  FROM customer
 INNER JOIN rental ON rental.customer_id = customer.customer_id;

SELECT *
  FROM customer
 INNER JOIN rental ON rental.customer_id = customer.customer_id
 INNER JOIN store ON store.store_id = customer.store_id;

SELECT *
  FROM customer
 INNER JOIN rental ON rental.customer_id = customer.customer_id
 INNER JOIN store ON store.store_id = customer.store_id
 INNER JOIN staff ON staff.staff_id = rental.staff_id;
```

```
SELECT customer.*,
       staff.first_name AS Staff_First_Name,
       staff.last_name AS Staff_Last_Name
  FROM customer
 INNER JOIN rental ON rental.customer_id = customer.customer_id
 INNER JOIN store ON store.store_id = customer.store_id
 INNER JOIN staff ON staff.staff_id = rental.staff_id;

SELECT customer.first_name, customer.last_name,
       staff.first_name AS Staff_First_Name,
       staff.last_name AS Staff_Last_Name
  FROM customer
 INNER JOIN rental ON rental.customer_id = customer.customer_id
 INNER JOIN store ON store.store_id = customer.store_id
 INNER JOIN staff ON staff.staff_id = rental.staff_id;

-- too many duplicates, use distinct
SELECT DISTINCT customer.first_name,
                customer.last_name,
                staff.first_name AS Staff_First_Name,
                staff.last_name AS Staff_Last_Name
  FROM customer
 INNER JOIN rental ON rental.customer_id = customer.customer_id
 INNER JOIN store ON store.store_id = customer.store_id
 INNER JOIN staff ON staff.staff_id = rental.staff_id;
```

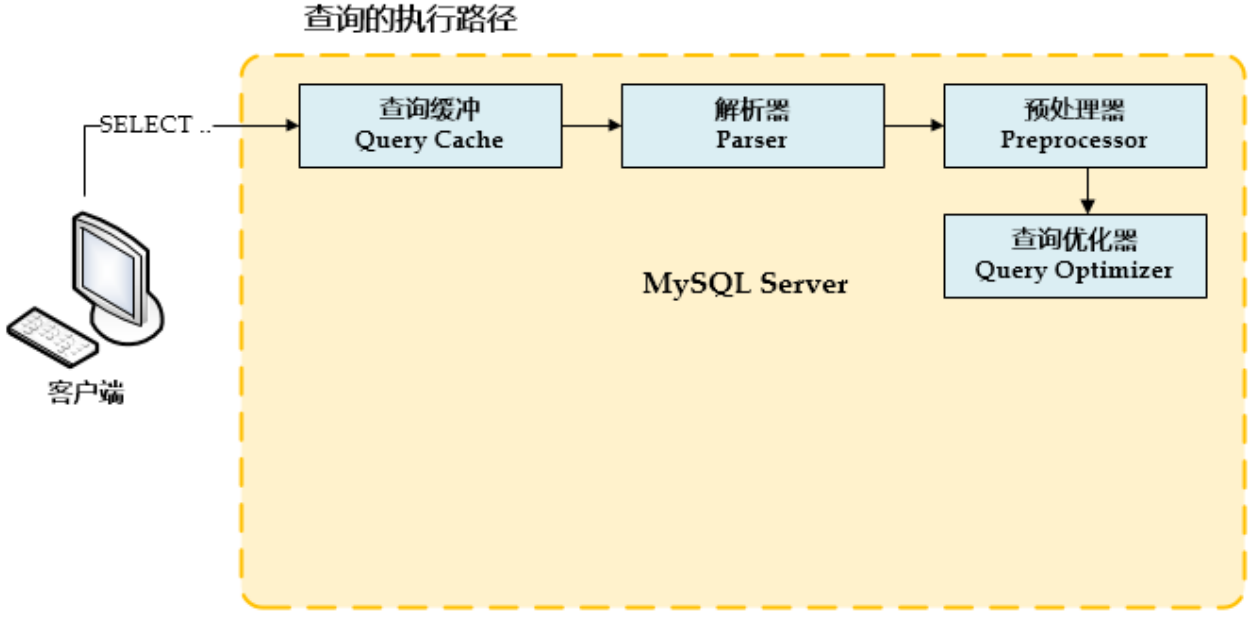
数据访问优化小结

- 避免查询返回额外的数据(行或列)
- 避免在SELECT语句中使用*(星号)进行查询
- 反复多次修改一个查询，获取了更多不需要的数据并不是个好主意，更好的建议是写新查询来代替已有查询来满足业务需要。

了解MySQL查询优化过程

查询的执行路径

- Client -> Query Cache -> Parser -> Reprocessor -> Query Optimizer -> …



查询的执行路径

- 查询缓冲 (Query Cache)

如果启用了Query Cache, MySQL会将第一次查询的结果集和查询文本放入查询缓冲中。

如果MySQL未能在Query Cache中找到严格相同的查询文本，那么就会转到解析器和预处理器，如果能找到，就会直接将对应结果集返回回客户端，而无需继续执行。

查询的执行路径

- 解析器 (Parser)

将单个查询分解为多个单词(token)，操作符和内建解析树，之后进行MySQL语法验证检查。

MySQL语法检查，即检查各种关键词及SQL语句操作符的顺序:
如ORDER BY是否在查询语句最后，FROM是否在SELECT之后等等。

查询的执行路径

- 预处理器 (Preprocessor)

在到达预处理器之前，查询已经通过了其语法检查。
之后预处理器将对权限和语句语义进行检查。

检查表是否存在在数据库中，对应表列和别名是否存在等…

尽管解析器，预处理器在最终执行前承担了重要角色，但是它们在时间上的花费则微不足道。

查询的执行路径

- 查询优化器 (Query Optimizer)

一旦到达查询优化器，那说明之前的语法验证都已通过。
查询优化器将会将从解析器、预处理器所得的解析树转换为执行计划，
这过程中将涉及查询重写，静态优化，动态优化等等优化操作，
同时查询优化器会对各种执行计划进行评估并从中找到最佳方案。

查询优化器是一个基于成本的优化器，它会选择成本最低的执行计划。

注意：基于成本计算的查询优化器不会考虑Query Cache的效果，它总是认为每次的数据读是指对磁盘IO的操作。

查询的执行路径

- 了解查询优化器的责任和限制

责任：

- ✓ 将不是最优的连接类型转换为更优的连接逻辑
- ✓ 重新定义表连接顺序
- ✓ 减少静态表达式
- ✓ 优化数学运算规则
- ✓ 逻辑短路
- ✓ 最佳索引使用
- ✓ 分组优化
- ✓ 优化聚合函数

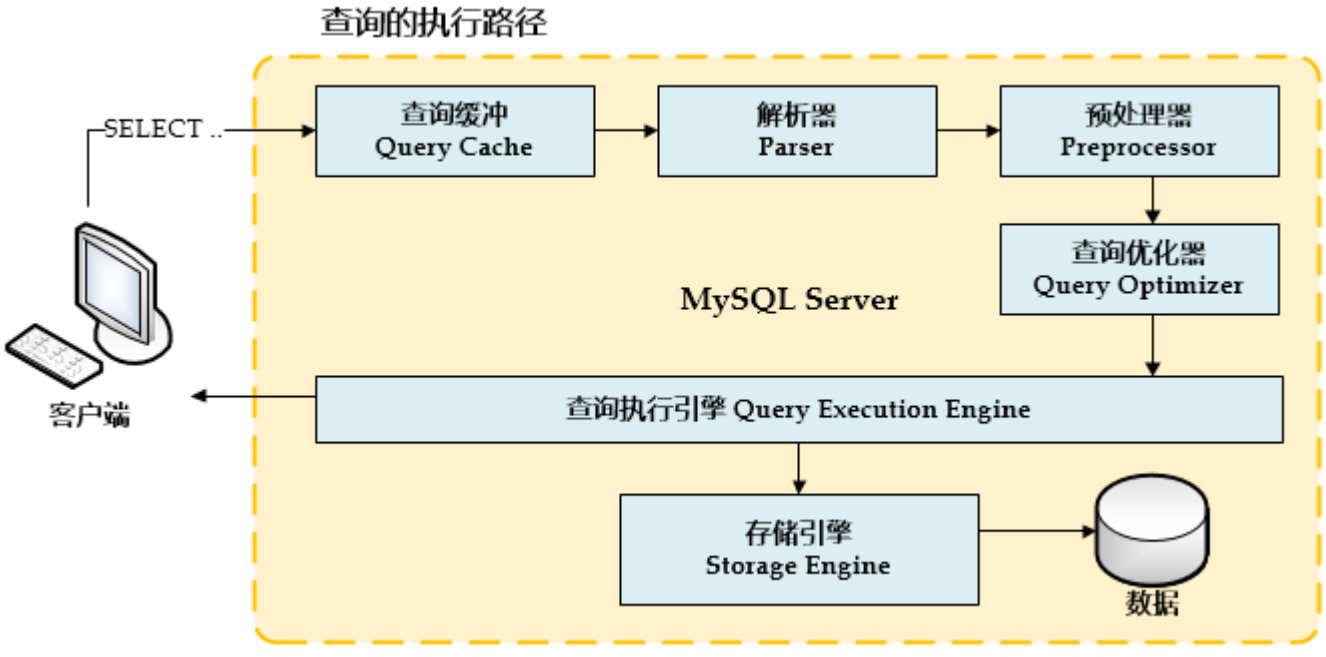
限制：

- ✓ 非并行查询执行
- ✓ 不考虑查询的并行运行情况
- ✓ 取决于存储引擎参数
- ✓ 最快的查询和最优资源的查询之间的选择
- ✓ 在进行操作成本的计算时，并不常将存储过程和功能（Stored Routing/Stored Procedure, Function）所耗成本计算在内

尽管仍有这么多限制和问题，但查询优化器在大多数情况下都能给出正确的判断和选择，最终生成最优化查询执行计划。

查询的执行路径

- 了解查询的执行路径



查询的执行路径

- 查询执行引擎 (Query Execution Engine)

引擎在接收了查询被优化后所生成的执行计划，并按其中指示来获取所需结果集。

MySQL查询执行引擎会调用存储引擎接口(Handler API)来进行操作。每个执行计划都有这样一个接口实例存在。

这些接口API会由MyISAM, InnoDB或其它存储引擎来实现最终交互并将数据返回回查询执行引擎。

注意：每个引擎有不同的结构和特点，如果某些参数在存储引擎级别未能调整好，那么你的执行计划就可能不是最优的。

查询的执行路径

最终所得到的结果集会被返回给客户端。

如果查询已经进行过了缓存，那么结果集会直接从Query Cache中读取并返回。之后的执行过程就无需进行了。

查询的执行路径

- 用好查询优化器以达到性能最大化
  - 优化数据访问
  - 理解查询优化过程
  - 查询重写

在了解查询重写之前，让我们对其他一些查询相关的状态信息及命令进行一定了解，以帮助更好地进行优化工作。

了解查询状态

- 查询状态

mysql> SHOW FULL PROCESSLIST;

- ✓ Sleep
- ✓ Query
- ✓ Locked
- ✓ Analyzing, Statistics
- ✓ Copying to tmp table
- ✓ Sorting result
- ✓ Sending data

http://dev.mysql.com/doc/refman/5.7/en/general-thread-states.html

在进行调优时，尝试执行查询以检查其不同状态，了解执行所处的大致情况。

Demo – 2-1

```
------------------
SHOW FULL PROCESSLIST;

------------------
-- Status is Refreshing
USE sakila;

SELECT * FROM actor CROSS JOIN address
 UNION
SELECT * FROM actor CROSS JOIN address
 UNION
SELECT * FROM actor CROSS JOIN address
 UNION
SELECT * FROM actor CROSS JOIN address;
```

了解EXPLAIN命令的使用

- EXPLAIN 或 EXPLAIN EXTENDED命令

EXPLAIN SELECT 查询:

Column	Meaning
id	The SELECT identifier
select_type	The SELECT type
table	The table for the output row
partitions	The matching partitions
type	The join type
possible_keys	The possible indexes to choose
key	The index actually chosen
key_len	The length of the chosen key
ref	The columns compared to the index
rows	Estimate of rows to be examined
filtered	Percentage of rows filtered by table condition
Extra	Additional information

MySQL执行计划是以树形结构展示的。可以通过EXPLAIN命令进行查看。

如果我们的表被重新组织或JOIN连接被进行了优化，MySQL执行计划将帮助使用这些优化后的结果来重建结构化查询。

通过EXPLAIN和SHOW WARNINGS我们可以了解到MySQL真实的执行语句。

在平时的调优工作中，花一些时间来理解这些重构的查询。相信这会极大提升了个人的调优能力。

Demo – 2-2

```
-- EXPLAIN
USE sakila;

SELECT *
  FROM actor;

EXPLAIN SELECT *
  FROM actor;

SELECT *
  FROM actor
 CROSS JOIN address;

EXPLAIN SELECT *
  FROM actor
 CROSS JOIN address;

SELECT *
  FROM actor
 INNER JOIN film_actor ON actor.actor_id = film_actor.actor_id;

EXPLAIN SELECT *
  FROM actor
 INNER JOIN film_actor ON actor.actor_id = film_actor.actor_id;
```

```
-- EXPLAIN EXTENDED
EXPLAIN EXTENDED SELECT *
  FROM actor
 INNER JOIN film_actor ON actor.actor_id = film_actor.actor_id;

SHOW WARNINGS;
```

小结

- MySQL优化器使用基于成本计算来找到查询的最优执行计划
- 你可以使用EXPLAIN命令来查看查询执行计划细节
- 由于查询优化取决于MySQL存储引擎相关的各种重要信息，因此所用存储引擎在查询性能中将有重要影响。

通过实例来理解性能优化

查询性能调优实例

- 我们将从对以下几点进行综合举例说明：
  - 表连接 (Join)
  - 子查询 (Subquery)
  - Union
  - 聚合函数 (Aggregate Function)
  - 分组 (Grouping)
  - 索引使用 (Index)

Demo – 3-1 在Select语句中索引的使用

```
USE sakila;

-- 在Select语句中索引的使用
SELECT *
FROM sakila.film;

SELECT *
FROM sakila.film
WHERE length < 50;

EXPLAIN SELECT *
FROM sakila.film
WHERE length < 50;

ALTER TABLE sakila.film ADD INDEX ix_length(length);
```

```
EXPLAIN SELECT *
FROM sakila.film
WHERE length < 50;

EXPLAIN SELECT *
FROM sakila.film
WHERE length < 100;

EXPLAIN SELECT film_id, length
FROM sakila.film
WHERE length < 100;

DROP INDEX ix_length ON sakila.film;
```

- 强烈推荐去掉在业务需求中不必要的查询列以使得查询使用索引。

Demo – 3-2.1 单个复杂查询 vs 多个简单查询

```
use sakila;

select *
  from film f
 left join film_actor fa on f.film_id = fa.film_id
 left join film_category fc on f.film_id = fc.film_id
 where f.film_id = 10;
show status like 'Last_Query_Cost';  -- 19.199000

select *
   from film
  where film_id = 10;     -- 1.000
show status like 'Last_Query_Cost';

select *
from film_actor
where film_id = 10;    -- 9.599000
show status like 'Last_Query_Cost';

select *
  from film_category
 where film_id = 10;     -- 1.199
show status like 'Last_Query_Cost';

-- 复杂查询 = 19.199000
-- 多个查询 = 1.000 + 9.599000 + 1.199 = 11.798
```

Demo – 3-2.2 单个复杂查询 vs 多个简单查询

```
select *
  from film f
 inner join inventory ia on f.film_id = ia.film_id
 inner join store st on st.store_id = ia.store_id
 inner join address ad on ad.address_id = st.address_id
 where f.film_id = 10;
show status like 'Last_Query_Cost';  -- 11.027403

-- 解构查询
select *
  from film
 where film_id = 10;
show status like 'Last_Query_Cost';  -- 1.000000

select *
  from inventory
 where film_id = 10;
show status like 'Last_Query_Cost';   -- 8.399000

select *
  from store
 where store_id in (1,2);
show status like 'Last_Query_Cost';   -- 2.809000

select *
  from address
 where address_id in (1,2);
show status like 'Last_Query_Cost';   -- 2.809000

-- 复杂查询 = 11.027403
-- 多个查询 = 8.399000 + 1.000000 + 2.809000 + 2.809000 = 15.017
```

- 这里没有真正的正确与错误的回答，开发者和DBA应该对不同情况进行分析。
- 这里强烈要求大家对你手中开发的查询进行下实验，看看到底是多个简单查询成本高还是单个复杂查询的成本高。并在其中找到性能最佳的解决方案。

Demo – 3-3
在建有索引的情况下，单个复杂查询 vs 多个简单查询

```
select *
  from sakila.film
 where length < 80;

explain select *
  from sakila.film
 where length < 80;

alter table sakila.film add index ix_length(length);

explain select *
  from sakila.film
 where length < 80;
```

```
explain select *
  from sakila.film
 where length < 61;

explain select *
  from sakila.film
 where length >= 61 and length <= 79;

explain select *
  from sakila.film
 where length between 61 and 79;

alter table sakila.film drop index ix_length;
```

- 通过union/union all这些小查询，最终的性能比单个复杂查询来得好。

Demo – 3-4 JOIN语法中表顺序的影响 - INNER JOIN

```sql
select *
  from film f
 inner join film_actor fa on f.film_id=fa.film_id
 inner join film_category fc on f.film_id = fc.film_id
 where f.film_id = 10;

select *
  from film f
 inner join film_category fc on f.film_id = fc.film_id
 inner join film_actor fa on f.film_id = fa.film_id
 where f.film_id = 10;

select *
  from film_category fc
 inner join film_actor fa on fc.film_id = fa.film_id
 inner join film f on f.film_id = fa.film_id
 where f.film_id = 10;
```

```sql
-- EXPLAIN
EXPLAIN EXTENDED select *
  from film f
 inner join film_actor fa on f.film_id=fa.film_id
 inner join film_category fc on f.film_id = fc.film_id
 where f.film_id = 10;

EXPLAIN EXTENDED select *
  from film f
 inner join film_category fc on f.film_id = fc.film_id
 inner join film_actor fa on f.film_id = fa.film_id
 where f.film_id = 10;

EXPLAIN EXTENDED select *
  from film_category fc
 inner join film_actor fa on fc.film_id = fa.film_id
 inner join film f on f.film_id = fa.film_id
 where f.film_id = 10;
```

- 应用不应该依赖于查询的列顺序，良好的开发中，调整列顺序不应该成为应用问题。做好绑定变量设计。

Demo – 3-5 连接语法中表顺序的影响 - OUTER JOIN

```
use sakila;

select *
  from film f
  left join film_actor fa on f.film_id = fa.film_id
  left join film_category fc on f.film_id = fc.film_id
 where f.film_id = 10;
show status like 'Last_Query_Cost';

select *
  from film f
  left join film_category fc on f.film_id = fc.film_id
  left join film_actor fa on f.film_id = fa.film_id
 where f.film_id = 10;
show status like 'Last_Query_Cost';

select *
  from film_category fc
  left join film_actor fa on fc.film_id = fa.film_id
  left join film f on f.film_id = fa.film_id
 where f.film_id = 10;
show status like 'Last_Query_Cost';
```

```
-- Explain
explain extended select *
  from film f
  left join film_actor fa on f.film_id = fa.film_id
  left join film_category fc on f.film_id = fc.film_id
 where f.film_id = 10;

explain extended select *
  from film f
  left join film_category fc on f.film_id = fc.film_id
  left join film_actor fa on f.film_id = fa.film_id
 where f.film_id = 10;

explain extended select *
  from film_category fc
  left join film_actor fa on fc.film_id = fa.film_id
  left join film f on f.film_id = fa.film_id
 where f.film_id = 10;
```

- 当我们在得到相同结果集的情况下，考虑对OUTER JOIN调整连接顺序以查看优化结果。

Demo – 3-6.1
子查询，Exists，表连接？谁是最优之选

```
use sakila;

-- subquery

select *
  from film f
 where f.film_id in (
 select i.film_id
   from inventory i
  where inventory_id < 555);

explain select *
  from film f
 where f.film_id in (
 select i.film_id
   from inventory i
  where inventory_id < 555);
```

```
-- exists

select *
  from film f
 where exists(
select i.film_id
  from inventory i
 where inventory_id < 555
   and i.film_id=f.film_id);

explain select *
  from film f
 where exists(
select i.film_id
  from inventory i
 where inventory_id < 555
   and i.film_id=f.film_id);
```

```
-- join

select distinct f.*
  from film f
 inner join inventory i on i.film_id = f.film_id
 where inventory_id < 555;

explain select distinct f.*
  from film f
 inner join inventory i on i.film_id = f.film_id
 where inventory_id < 555;
```

- 通过这样简单的例子，似乎让你感觉IN, EXISTS和JOIN的比对效果一目了然. 但是现实中结果可能因结果集和你当时的系统负载而会发生不同。

Demo – 3-6.1
子查询，Exists，表连接？谁是最优之选

```
create table filmcopy like film;
alter table filmcopy modify film_id smallint(5);
alter table filmcopy drop primary key;
```

```
delimiter //
create PROCEDURE SubQ (p1 int)
begin
   label1: loop
     set p1=p1-1;
     if p1>0 then
       insert into sakila.filmcopy
         select *
           from film f
          where f.film_id in (
         select i.film_id
           from inventory i
          where inventory_id < 555);
       ITERATE label1;
     end if;
     leave label1;
   end loop;
end//

delimiter ;
```

```
delimiter //
create PROCEDURE ExistQ (p1 int)
begin
   label1: loop
     set p1=p1-1;
     if p1>0 then
       insert into sakila.filmcopy
         select *
           from film f
          where exists(
         select i.film_id
           from inventory i
          where inventory_id < 555
            and i.film_id=f.film_id);
       ITERATE label1;
     end if;
     leave label1;
   end loop;
end//
```

```
delimiter //
create PROCEDURE JoinQ (p1 int)
begin
   label1: loop
     set p1=p1-1;
     if p1>0 then
       insert into sakila.filmcopy
         select distinct f.*
           from film f
          inner join inventory i on i.film_id = f.film_id
          where inventory_id < 555;
       ITERATE label1;
     end if;
     leave label1;
   end loop;
end//

delimiter ;
```

Demo – 3-6.1
子查询，Exists，表连接？谁是最优之选

```
-- 执行
call SubQ(1000);
truncate table sakila.filmcopy;

call ExistQ(1000);
truncate table sakila.filmcopy;

call JoinQ(1000);
truncate table sakila.filmcopy;



-- 清理
drop procedure SubQ;
drop procedure ExistQ;
drop procedure JoinQ;
drop table sakila.filmcopy;
```

Demo – 3-6.2
子查询，Exists，表连接？谁是最优之选

```
use sakila;

-- subquery

select *
  from film f
 where f.film_id in (
 select i.film_id
   from inventory i
  where inventory_id < 55);

explain select *
  from film f
 where f.film_id in (
 select i.film_id
   from inventory i
  where inventory_id < 55);
```

```
-- exists

select *
  from film f
 where exists(
select i.film_id
  from inventory i
 where inventory_id < 55
   and i.film_id=f.film_id);

explain select *
  from film f
 where exists(
select i.film_id
  from inventory i
 where inventory_id < 55
   and i.film_id=f.film_id);
```

```
-- join

select distinct f.*
  from film f
 inner join inventory i on i.film_id = f.film_id
 where inventory_id < 55;

explain select distinct f.*
  from film f
 inner join inventory i on i.film_id = f.film_id
 where inventory_id < 55;
```

Demo – 3-6.2
子查询，Exists，表连接？谁是最优之选

```
create table filmcopy like film;
alter table filmcopy modify film_id smallint(5);
alter table filmcopy drop primary key;
```

```
delimiter //
create PROCEDURE SubQ (p1 int)
begin
   label1: loop
     set p1=p1-1;
     if p1>0 then
       insert into sakila.filmcopy
         select *
           from film f
          where f.film_id in (
         select i.film_id
           from inventory i
          where inventory_id < 55);
       ITERATE label1;
     end if;
     leave label1;
   end loop;
end//

delimiter ;
```

```
delimiter //
create PROCEDURE ExistQ (p1 int)
begin
   label1: loop
     set p1=p1-1;
     if p1>0 then
       insert into sakila.filmcopy
         select *
           from film f
          where exists(
          select i.film_id
            from inventory i
           where inventory_id < 55
             and i.film_id=f.film_id);
       ITERATE label1;
     end if;
     leave label1;
   end loop;
end//

delimiter ;
```

```
delimiter //
create PROCEDURE JoinQ (p1 int)
begin
   label1: loop
     set p1=p1-1;
     if p1>0 then
       insert into sakila.filmcopy
         select distinct f.*
           from film f
          inner join inventory i on i.film_id = f.film_id
          where inventory_id < 55;
       ITERATE label1;
     end if;
     leave label1;
   end loop;
end//

delimiter ;
```

Demo – 3-6.2
子查询，Exists，表连接？谁是最优之选

```
-- 执行
call SubQ(1000);
truncate table sakila.filmcopy;

call ExistQ(1000);
truncate table sakila.filmcopy;

call JoinQ(1000);
truncate table sakila.filmcopy;



-- 清理
drop procedure SubQ;
drop procedure ExistQ;
drop procedure JoinQ;
drop table sakila.filmcopy;
```

- 你需要基于现实环境，进行试验以选出最优方案。

Demo – 3-7 聚合函数 MIN，MAX的调优

```
use sakila;

-- Optimize aggregate function - MIN and MAX

select *
  from sakila.film
 where length = 100;

---------------
-- Min
select min(film_id)
  from sakila.film
 where length = 100;

select film_id
  from sakila.film
 where length = 100
 order by film_id
 limit 1;

---------------
-- Explain Min
explain select min(film_id)
  from sakila.film
 where length = 100;

explain select film_id
  from sakila.film
 where length = 100
 order by film_id
 limit 1;
```

```
---------------
-- Max
select max(film_id)
  from sakila.film
 where length = 100;

select film_id
  from sakila.film
 where length = 100
 order by film_id desc
 limit 1;

---------------
-- Explain Max
explain select max(film_id)
  from sakila.film
 where length = 100;

explain select film_id
  from sakila.film
 where length = 100
 order by film_id desc
 limit 1;
```

Demo – 3-8 Group By语法优化

```
use sakila;

-- Optimize Group By

select title, length, count(*)
  from film
 where length < 100
 group by title, length;

select title, length, count(*)
  from film
 where length < 100
 group by film_id;

explain select title, length, count(*)
  from film
 where length < 100
 group by title, length;

explain select title, length, count(*)
  from film
 where length < 100
 group by film_id;
```

- 如果你对此数据库很了解，虽然前2个查询不同，但是实际是得出的结果是一样的。group by 中，如果列组合和其他单一列有一一对应关系。

Demo – 3-8 Group By语法优化

```
create table sakila.testtab(title varchar(100), length int, count int);

delimiter //
create procedure FirstOpt(p1 int)
begin
    label1: loop
        set p1=p1-1;
        if p1>0 then
            insert into sakila.testtab
             select title, length, count(*)
               from film
              where length < 100
              group by title, length;
              iterate label1;
        end if;
        leave label1;
    end loop label1;
end//
```

```
delimiter //
create procedure SecondOpt(p1 int)
begin
    label1: loop
        set p1=p1-1;
        if p1>0 then
            insert into sakila.testtab
             select title, length, count(*)
               from film
              where length < 100
              group by film_id;
              iterate label1;
        end if;
        leave label1;
    end loop label1;
end//

delimiter ;
```

```
-- Execute
call FirstOpt(1000);
truncate table sakila.testtab;
call SecondOpt(1000);
truncate table sakila.testtab;

-- Clean up
drop procedure FirstOpt;
drop procedure SecondOpt;
drop table sakila.testtab;
```

Demo – 3-9 使用Limit语法进行分页优化

```
use sakila;

select *
  from customer;

select *
  from customer
  limit 400;

select *
  from customer
  order by customer_id
  limit 400;

select *
  from customer
  order by customer_id
  limit 395,5;
```

- 推荐使用数据库自有功能，而不是在得出整个结果集返回应用后进行分页。

Demo – 3-10 Union和Union ALL之间的性能差别

```
use sakila;

-- Union vs Union all

select staff_id, first_name, last_name
  from staff s
  union all
select customer_id, first_name, last_name
  from customer c
  union all
select actor_id, first_name, last_name
  from actor a;

select staff_id, first_name, last_name
  from staff s
  union
select customer_id, first_name, last_name
  from customer c
  union
select actor_id, first_name, last_name
  from actor a;
```

Demo – 3-10 Union和Union ALL之间的性能差别

```
create table sakila.customertest(id int, first_name varchar(100), last_name varchar(100));

delimiter //
create procedure unionallopt(p1 int)
begin
    label1: loop
        set p1=p1-1;
        if p1>0 then
            insert into sakila.customertest
            select staff_id, first_name, last_name
              from staff s
              union all
            select customer_id, first_name, last_name
              from customer c
              union all
            select actor_id, first_name, last_name
              from actor a;
            iterate label1;
        end if;
        leave label1;
    end loop;
end//

delimiter ;
```

```
delimiter //
create procedure unionopt(p1 int)
begin
    label1: loop
        set p1=p1-1;
        if p1>0 then
            insert into sakila.customertest
            select staff_id, first_name, last_name
              from staff s
              union
            select customer_id, first_name, last_name
              from customer c
              union
            select actor_id, first_name, last_name
              from actor a;
            iterate label1;
        end if;
        leave label1;
    end loop;
end//

delimiter ;
```

Demo – 3-10 Union和Union ALL之间的性能差别

```
-- execute
call unionallopt(1000);
truncate table sakila.customertest;
call unionopt(1000);
truncate table sakila.customertest;

-- clean up
drop procedure unionallopt;
drop procedure unionopt;
drop table sakila.customertest;
```

- 当union, union all结果相同，请使用 union all , union对性能有影响。

Demo – 3-11 索引和不等号 <>

```
use sakila;

select *
  from customer
  where store_id <> 2;

explain select *
  from customer
  where store_id <> 2;

select distinct store_id
  from customer;

select *
  from customer
 where store_id = 1;

explain select *
  from customer
 where store_id = 1;
```

- 这里查询希望有索引可以使用，但是事实上索引由于 store_id<>2而没有被使用。

最佳实践建议

最佳实践建议

- 使用EXPLAIN语法检查查询执行计划
  - 查看索引的使用情况
  - 查看行扫描情况
- 当需要获取唯一一行时，请使用LIMIT 1语法
  - 对如MIN或MAX等聚合计算操作有帮助
- 避免使用SELECT *
  - 这会导致表的全扫描
  - 网络带宽会被浪费

最佳实践建议

- 将DELECT, UPDATE或INSERT中的查询语句解构成多个更小查询
- 多表列选用适当的数据类型
  - 更小的列有助于更佳的性能
- MySQL Query Cache是大小写和空格敏感的
  - 请使用严格相同的语句来进行重复查询
- WHERE语法中的列加索引（之前的例子）
- 对JOIN的表列加索引（之前的例子）

最佳实践建议

- 避免使用<>，因为这会导致索引不被使用
- 如果不存在重复行数据，请使用UNION ALL替代UNION

最佳实践建议 – 最后

- 现实中有许多方法可以查询出相同的结果集。往往一种调优方法对某个查询有效时，而对于某个其他查询，则需要其他方法来处理。

- 在部署到生产环境前，请尽量多地在开发环境中模拟现实数据情景，测试你的查询，以得出最佳方案。

使用　学习　分享

DBDAO

引导式IT在线教育　关注大数据科学


Beaver
Country Day School
dbDao