TRAE 2.0 SOLO 出道，一键贯通从灵感火花到上线部署的全程协作 立即体验

关注 6 4

MySQL中的SQL调优设计

user8288612657632 2024-04-16 771 阅读10分钟 关注

‹‹‹ TRAE 2.0 SOLO 出道，一键贯通从灵感火花到上线部署的全程协作 ›››

SQL调优有哪些基本原则?

导致SQL查询效率比较低的原因,主要包括数据量,数据访问量,数据计算,SQL语句的设计几个层面

- 减少数据量(表中数据太多可以分表，例如双11是一个小时一张订单表)
- 减少数据访问量(将全表扫描可以调整为基于索引去查询)
- 减少数据计算操作(将数据库中的计算拿到程序内存中计算)

SQL优化的基本逻辑是怎样的?

- 良好SQL编码的习惯(熟悉SQL编码规范、例如关键字大写,避免使用select *)
- 优秀SQL的编写逻辑(例如表关联时小表驱动大表)
- 定位需要优化的慢SQL语句(耗时多长时间的SQL是慢SQL)
- 调整优化策略并进行测试。(SQL结构上的调整、索引应用)
- 按业务进行分库分表。(分表可以在应用逻辑中减少单表数据量)

有哪些优秀SQL编写案例?

- 查询时尽量避免使用select *;
- 尽量避免在where子句中使用or作为查询条件。
- where 条件中尽量不要出现与null值的比较。
- 避免在查询中存在隐式转换。(... where id='1')
- 避免在where子句中使用!=或者<>操作符。
- 使用like查询条件时应尽量避免前缀使用"%"。
- 执行查询时尽量采用最左匹配原则。(where first_name='A' and email='A@t.com')
- 避免在查询条件中使用一些内置的SQL函数。(MySQL8中现在可以基于SQL函数 添加索引了)
- 假如in表达式后面的数据太多，尽量避免使用in作为查询条件。(where id in (1,2,3,4,5,....))
- 当有多个查询条件、分组条件、排序条件时，尽量使用联合索引(组合索引)
- 表连接时优先使用内连接(inner join),使用小表驱动大表。
- 进行表关联的字段尽量使用相同的编码(不能一个字段utf8,一个字段utf8mb4)

- 表设计时字段类型能用简单数据类型不用复杂类型。(例如能用 int 不用 varchar)
- 清空表中数据可优先使用truncate.(truncate删除数据时不记录日志)
- 插入多条数据时可考虑使用批量插入。(insert into xxx values (...),(...),(...))

如何基于慢SQL日志查询慢SQL?

线上环境,我们对SQL进行调优,首先要发现执行慢的SQL,然后再对SQL进行分析和优化。如何找到执行慢的SQL呢,可以通过慢查询日志进行分析,具体可以参考如下步骤:

1. 开启慢查询日志（一般默认是关闭状态）
2. 设置慢查询阀值(响应速度是多长时间被定为是慢查询)
3. 确定慢查询日志路径(日志文件在哪里)
4. 确定慢查询日志的文件名(具体日志文件是哪个)，然后对文件内容进行分析。
5. 打开慢查询日志文件，检查慢SQL。

实操演示:

查看慢查询日志的打开状态?

```sql
show variables like '%slow_query_log%';
```

默认环境下，MySQL5.7默认慢查询日志状态是关闭的(OFF)。

如何开启慢查询日志?

```sql
set global slow_query_log=ON --MySQL5.7，MySQL8.0
set slow_query_log=ON  --10.5.17-MariaDB
```

查看默认慢查询阈值 (默认为10秒,假如一个SQL查询耗时超过了10秒钟,就会认为是慢SQL)

```sql
show variables like '%long_query_time%';
```

如何设置慢查询的阈值?

MySQL5.7设置慢查询时间阀值(响应时间是多长时间是慢查询)

```sql
mysql> set  long_query_time = 1;
Query OK, 0 rows affected (0.00 sec)
```

Captured by FireShot Pro: 13 8月 2025, 10:50:30
https://getfireshot.com

- 如何知道慢查询日志路径?

慢查询日志的路径默认是 MySQL 的数据目录

```sql
mysql> show global variables like 'datadir';

+---------------+-----------------------+
| Variable_name | Value                 |
+---------------+-----------------------+
| datadir       | /mysql/data/          |
+---------------+-----------------------+
1 row in set (0.00 sec)
```

- 如何知道慢查询日志的文件名?

```sql
mysql> show global variables like 'slow_query_log_file';
+---------------------+----------------+
| Variable_name       | Value          |
+---------------------+----------------+
| slow_query_log_file | mysql-slow.log |
+---------------------+----------------+
1 row in set (0.00 sec)
```

执行一个耗时SQL，然后查看慢SQL日志,例如

```sql
select * from employees where salary between 100 and 30000
union
select * from employees where salary between 100 and 30000
```

打开日志文件，可以对日志文件中的内容进行分析，常用选项说明：

Time：慢查询发生的时间

User@Host：客户端用户和IP

Query_time：查询时间

Lock_time：等待表锁的时间

Rows_sent：语句返回的行数

Rows_examined：语句执行期间从存储引擎读取的行数

如何对慢SQL查询进行分析?

Captured by FireShot Pro: 13 8月 2025, 10:50:30
https://getfireshot.com

工欲善其事，必先利其器，分析慢查询可以通过 explain、show profile 等工具来实现。

执行计划(Explain)是什么?

执行计划是mysql优化器对SQL进行默认调优后，给出的一种执行方案，这个方案我们可以通过explain 这个指令进行查询。例如，对select语句进行分析，并输出select执行时的详细信息，开发人员可以 基于这些信息进行有针对性的优化，例如：

```sql
mysql> explain select * from employees where employee_id<100 \G;
 ****   1.   row   ***  *
id: 1
select_type: SIMPLE
table: employees
partitions: NULL
type: range
possible_keys: PRIMARY
key: PRIMARY
key_len: 4
ref: NULL
rows: 1
filtered: 100.00
Extra: Using where
1 row in set, 1 warning (0.00 sec)
```

分析执行计划的目的是什么?

分析执行计划的目的就为了了解SQL的执行逻辑,例如: - 检查关联查询、嵌套查询的执行顺序. - 查询操作的具体类型 - 哪些索引可能会命中以及实际命中的索引有哪些 - 每张表可能有多少条记录参与到了查询中

说说执行计划中几个常见的字段?

- **id**
  select 的序列号，有几个select 就有几个 id，id 的顺序是按 select 出现的顺序增长的。 即：id 越大语句执行的优先级越高，id相同则从上往下依次执行，id 为NULL最后执行。

```sql
explain
select last_name,salary
from employees
where employee_id=(
  select manager_id
  from employees
  where employee_id=206);
```

当id值相同时,优先级从上到下,例如:

```sql
explain
select m.first_name,m.salary
from employees e join employees m
on e.manager_id=m.employee_id
where e.employee_id=206
```

- **select_type表示的查询类型有哪些?**

1. SIMPLE： 表示查询语句不包含子查询或 union
2. PRIMARY：表示此查询是最外层的查询
3. UNION：表示此查询是 union 的第二个或后续的查询
4. UNION RESULT：union 的结果
5. DEPENDENT UNION：子查询中的UNION操作，UNION后的所有select都是DEPENDENT UNION。
6. SUBQUERY：SELECT 子查询语句
7. DEPENDENT SUBQUERY：子查询中的第一个SELECT,SELECT 子查询语句依赖外层查询。
8. DERIVED: from 子句后的相对比较复杂的子查询(相当于一个临时表)，当看到derivedN时，这里N表时查询id

案例分析：

select_type为SIMPLE (表示查询语句不包含子查询或 union)

```sql
explain
select *
from employees
where employee_id<100;
```

select_type为PRIMARY、SUBQUERY (PRIMARY表示最外层查询，SUBQUERY表示嵌套查询)

```sql
explain
select last_name,salary
from employees
where employee_id=(
    select manager_id
    from employees
    where employee_id=206);
```

select_type为 UNION(union操作)、UNION RESULT(union的结果)

Captured by FireShot Pro: 13 8月 2025, 10:50:30
https://getfireshot.com

```sql
explain
select first_name,hire_date,salary
from employees
where job_id='AD_VP'
union
select first_name,hire_date,salary
from employees
where  salary>15000;
```

select_type为DEPENDENT UNION (子查询中的UNION操作，UNION后的所有select都是DEPENDENT UNION。)

```sql
explain
select *
from employees e3
where first_name in (
         select first_name
         from employees e1
         where job_id = 'AD_VP'
         union
         select first_name
         from employees e2
         where salary > 15000
    )
```

select_type 为 DEPENDENT SUBQUERY

```sql
explain
select  employee_id,first_name,salary
from employees e1
where salary=(
    select max(salary)
    from employees e2
    where e1.department_id=e2.department_id);
```

说明，一般出现DEPENDENT SUBQUERY时，SQL的执行效率都会比较低，可以调整为多表查询，例如：

```sql
explain
select e2.department_id,e2.employee_id,e2.first_name,e2.salary
from (
select department_id,max(salary) max_salary
from employees
group by department_id) e1 join employees e2
on e1.department_id=e2.department_id
where  e1.max_salary=e2.salary
```

Captured by FireShot Pro: 13 8月 2025, 10:50:30
https://getfireshot.com

select_type为DERIVED(这里一般表示from后面的一个衍生表-临时表)

```sql
explain
select min(avg_salary)
from (
select avg(salary) avg_salary
from employees
group by department_id) emp;
```

- **type表示查询数据的方式。(重点)**

type是一个比较重要的一个属性，通过它可以判断出查询是全表扫描还是基于索引的部分扫描。 常用属性值如下，从上至下效率依次增强。调优时，建议type类型至少要为range，才能提高查询效率。

1. ALL：表示全表扫描，性能最差。(数据量小时无所谓)
2. index：表示基于索引的全表扫描，先扫描索引再扫描全表数据。
3. range：表示使用索引范围查询。使用 >、>=、<、<=、in 等等。
4. index_merge: 表示查询中使用到了多个索引，然后进行了索引合并
5. ref：表示使用非唯一索引进行单值查询。
6. eq_ref：一般情况下出现在多表 join 查询，表示前面表的每一个记录，都只能匹配后面表的一行结果。
7. const：表示使用主键或唯一索引做等值查询，常量查询。(效率非常高)
8. NULL：表示不用访问表，也没有索引，速度最快。(了解，例如select version())

案例分析：

type为null

```sql
explain
select now()
```

type 为const (基于主键或唯一键执行的查询)

```sql
explain
select *
from employees
where employee_id=206
```

Captured by FireShot Pro: 13 8月 2025, 10:50:30
https://getfireshot.com

```sql
explain
select first_name,email
from employees
where email='SKING';
```

type 为eq_ref (多表join，前面表的每一行记录，只能匹配后面表的一行记录)

```sql
explain
select d.department_id,d.department_name,e.first_name
from departments d join employees e on d.manager_id = e.employee_id;
```

type为 ref (使用非唯一索引进行的等值查询)

```sql
create index index_first_name on employees(first_name);
explain
select *
from employees
where first_name='Steven';
```

```sql
create index index_salary on employees (salary);
explain
select salary,first_name from employees where salary=17000;
```

type 为 index_merge (索引合并，同时应用两个索引)

```sql
create index index_salary on employees(salary);
explain
select first_name,hire_date,salary
from employees
where job_id='AD_VP' or salary>15000;
```

type 为 range (这里的range表示一个范围查询)

```sql
create index index_salary on employees (salary);
explain
select first_name,salary
from employees
where salary between 10000 and 30000;
```

Captured by FireShot Pro: 13 8月 2025, 10:50:30
https://getfireshot.com

type 为index (基于索引的全表扫描)

```sql
explain
select count(*)
from employees
group by department_id;
```

type 为 all (表示全表扫描)

```sql
explain
select *
from employees
```

- **Extra 中值 的含义是什么？** Extra 表示很多额外的信息，各种操作会在 Extra 提示相关信息，常见几种如下：

"Using where" 表示查询需要通过where条件查询数据(可能没有用到索引,也可能一部分用到了索引)。

```sql
explain
select *
from hr.employees
where salary>10000;
```

Using index 表示查询需要通过索引，索引就可以满足所需的数据(不需要再回表查询-基于普通索引找到主键，然后再基于主键查找对应纪录，当前查询中应用了覆盖索引-select列表中的值都是索引值)。

```sql
create index index_hire_date_salary on employees(hire_date,salary);
explain
select employee_id,hire_date,salary
from hr.employees
where hire_date>'2000-03-06' and salary>10000;
```

Using index condition 表示查询的记录，在索引中没有完全覆盖(可能要基于where或二级索引对应的主键再次查询-回表查询)。

```sql
create index index_hire_date_salary on employees (hire_date,salary)
explain
select employee_id,hire_date,salary,commission_pct
from hr.employees
where hire_date>'2000-03-06' and salary>10000;
```

Captured by FireShot Pro: 13 8月 2025, 10:50:30
https://getfireshot.com

Using filesort 表示查询出来的结果需要额外排序，数据量小在内存，大的话在磁盘，因此有 Using filesort 建议优化。

```sql
explain
select first_name,hire_date,salary
from hr.employees
order by hire_date
```

Using temprorary 表示查询使用到了临时表，一般出现于去重、分组等操作(这里一般也需要优化)。

```sql
explain
select first_name,salary
from hr.employees
where first_name like 'A%'
union
select first_name,salary
from hr.employees
where first_name like 'B%'
```

标签： 后端 数据库

本文收录于以下专栏

数据库 专栏目录

MySQL

0 订阅 · 15 篇文章

订阅

上一篇 常见SQL查询实践

下一篇 视图

Captured by FireShot Pro: 13 8月 2025, 10:50:30
https://getfireshot.com

评论 0

登录 / 注册 即可发布评论!

暂无评论数据

相关推荐

MySQL中的SQL查询性能调优

295阅读 · 2点赞

MySQL 性能调优与 SQL 调优指南

300阅读 · 4点赞

全解MySQL终章：这份爆肝30W字的数据库宝典赠与有缘的你！

48k阅读 · 494点赞

MySQL索引和SQL调优

28k阅读 · 617点赞

年薪近百万架构师纯手写的MySQL笔记，看完感觉之前读的都是渣渣！

505阅读 · 0点赞

Captured by FireShot Pro: 13 8月 2025, 10:50:30
https://getfireshot.com

精选内容

centos如何使用高版本gcc

苏三的开发日记 · 31阅读 · 0点赞

Java【问题 07】SSH不同版本使用jsch问题处理（7.4升级9.7及欧拉原生8.8）

yuanzhengme · 28阅读 · 1点赞

Redis 分布式锁深度解析：setnx 命令的核心作用与实现

Code季风 · 37阅读 · 0点赞

OpenSSH【安装 02】离线升级异常问题解决、无法升级时的失败恢复

yuanzhengme · 19阅读 · 0点赞

java 面试八股这一篇就够之java集合篇

小厂永远得不到... · 94阅读 · 3点赞

为你推荐

性能优化篇：SQL数据库查表速度优化

为了WLB努力 | 1年前 | 1.0k | 15 | 评论 | 后端 SQL MySQL

尚硅谷MySQL高级学习笔记 -- 4.查询截取分析

exodus3 | 4年前 | 310 | 点赞 | 评论 | 后端 笔记

MySql优化-上|8月更文挑战

耶马 | 4年前 | 248 | 1 | 评论 | MySQL 后端

MySQL调优问题与解决方案

闯闯的日常分享 | 8月前 | 142 | 3 | 评论 | 后端

MySQL查询语句优化的十个小技巧！

麒麟IT | 4年前 | 787 | 4 | 评论 | Java 后端

MySQL高级篇 - 性能优化

slowlybutsurely | 4年前 | 1.1k | 8 | 评论 | MySQL

MySQL的常用优化方案

小二上酒8 | 2年前 | 1.8k | 13 | 评论 | 后端 面试

MySQL的常用优化方案

KwingTaai | 2年前 | 1.6k | 11 | 评论 | 数据库 后端

关于 MySQL 慢查询优化的一些思路与知识点

码指星斗 | 2年前 | 3.4k | 14 | 评论 | MySQL 后端

Captured by FireShot Pro: 13 8月 2025, 10:50:30
https://getfireshot.com

[MySQL]慢SQL优化记录 -- DDL选择

Eric223 | 2年前 | 1.4k 9 1 MySQL 数据库 Go

SQL优化步骤

morningcat2018 | 2月前 | 48 1 评论 后端

PostgreSQL - SQL调优方案

SpencerGallacher | 2年前 | 666 2 评论 后端

为什么使用了索引，查询效率还是低?

月生_ | 9月前 | 375 1 2 MySQL

52条SQL语句性能优化策略，建议收藏

Spring源码项目进行时 | 4年前 | 3.0k 33 4 Java 后端 SQL

MySQL之SQL优化

试剑江湖 | 4年前 | 889 5 评论 MySQL 后端

Captured by FireShot Pro: 13 8月 2025, 10:50:30
https://getfireshot.com