

SQL優化最乾貨總結 (2021最新版)

陳哈哈 程序員閃充寶 昨天



微信搜一搜



程序員閃充寶

程序員閃充寶

作者：陳哈哈

chensj.blog.csdn.net/article/details/107020686

前言

BATJTMD等大廠的面試難度越來越高，但無論從大廠還是到小公司，一直未變的一個重點就是對SQL優化經驗的考察。一提到數據庫，先“說一說你對SQL優化的見解吧？”。

SQL優化已經成為衡量程序猿優秀與否的硬性指標，甚至在各大廠招聘崗位職能上都有明碼標註，如果你，在這個問題上能吊打面試官還是會被吊打呢？

百度_Java高级研发工程... 15000-30000元/月

补充医疗保险

定期体检

加班补助

全勤奖

年终奖

...

百度 [查看所有职位](#)

职责要求:

- 本科以上学历，2年以上Java项目开发经验
- Java基础扎实，熟悉J2EE编程，熟练使用Spring等主流框架，了解常用的设计模式
- 掌握Mysql数据库开发，了解SQL优化方法
- 熟悉Linux日常工作环境，掌握常用命令
- 熟练使用Shell、Ruby、Groovy、Python等一种脚本语言，能够在日常工作中使用脚本简化工作
- 有过大规模互联网系统架构设计经验者优先、有数据分析经验者优先
- 良好的沟通能力和团队协作精神，严谨的工作态度与高质量意识

注:如果看著模糊，可能是你擰多了

目錄

前言

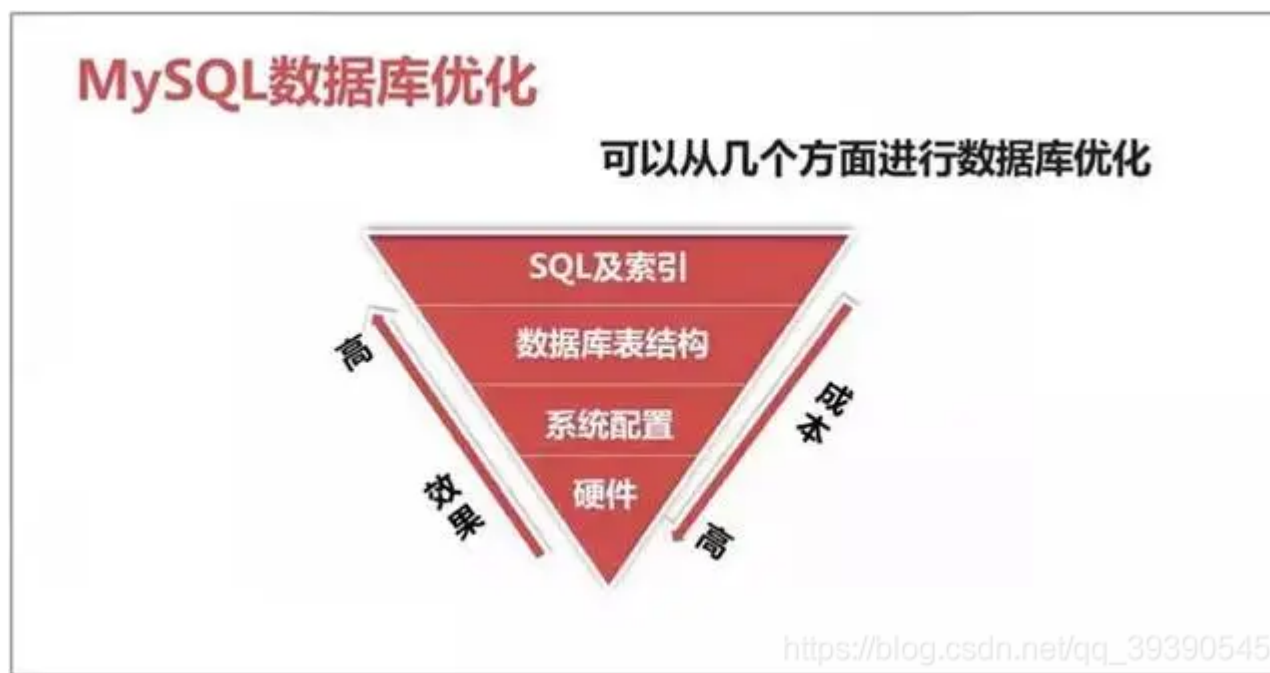
SELECT語句- 語法順序：

SELECT語句- 執行順序：

SQL優化策略

- 一、避免不走索引的場景
- 二、SELECT語句其他優化
- 三、增刪改DML 語句優化
- 四、查詢條件優化
- 五、建表優化

有朋友疑問到，SQL優化真的有這麼重要么？如下圖所示，SQL優化在提升系統性能中是：
（ 成本最低&& 優化效果最明顯 ） 的途徑。如果你的團隊在SQL優化這方面搞得很優秀，對你們整個大型系統可用性方面無疑是一個質的跨越，真的能讓你們老闆省下不止幾沓子錢。



- 優化成本：硬件>系統配置>數據庫表結構>SQL及索引。

- 優化效果：硬件<系統配置<數據庫表結構<SQL及索引。

```
String result = "嗯，不错，";

if ("SQL优化经验足") {
    if ("熟悉事务锁") {
        if ("并发场景处理666") {
            if ("会打王者荣耀") {
                result += "明天入职"
            }
        }
    }
} else {
    result += "先回去等消息吧";
}

Logger.info("面试官：" + result );
```

別看了，上面這是一道送命題。

好了我們言歸正傳，首先，對於MySQL層優化我一般遵從五個原則：

1. 減少數據訪問：設置合理的字段類型，啟用壓縮，通過索引訪問等減少磁盤IO
2. 返回更少的數據：只返回需要的字段和數據分頁處理減少磁盤io及網絡io
3. 減少交互次數：批量DML操作，函數存儲等減少數據連接次數
4. 減少服務器CPU開銷：盡量減少數據庫排序操作以及全表查詢，減少cpu 內存佔用

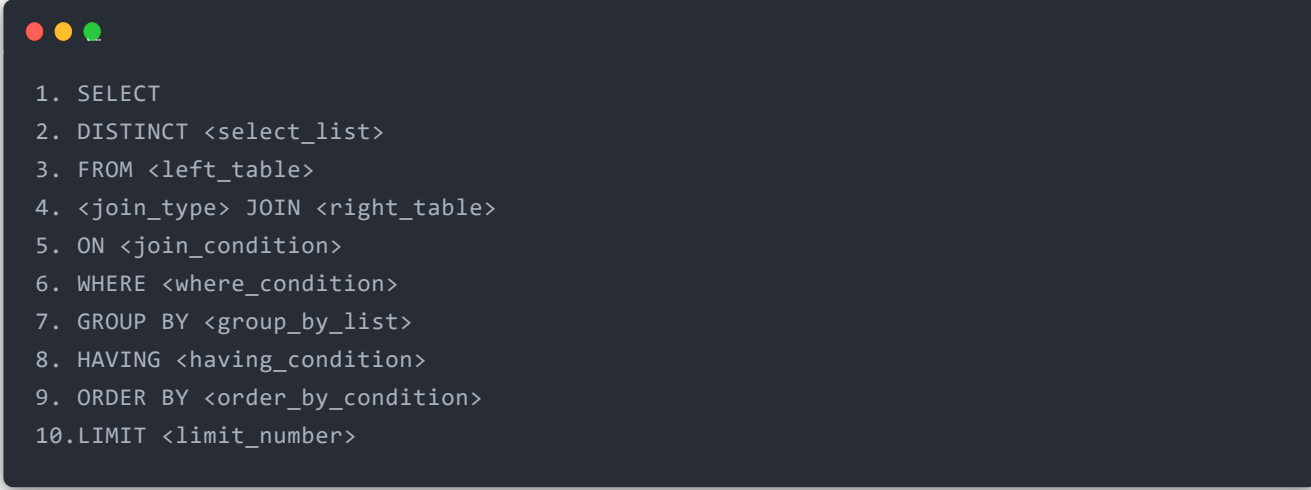
5. 利用更多資源：使用表分區，可以增加並行操作，更大程度利用cpu資源

總結到SQL優化中，就三點：

- 最大化利用索引；
- 盡可能避免全表掃描；
- 減少無效數據的查詢；

理解SQL優化原理，首先要搞清楚SQL執行順序：

SELECT語句- 語法順序：

A dark-themed terminal window with three colored window control buttons (red, yellow, green) in the top-left corner. It contains a numbered list of SQL keywords and placeholders for a SELECT statement.

```
1. SELECT
2. DISTINCT <select_list>
3. FROM <left_table>
4. <join_type> JOIN <right_table>
5. ON <join_condition>
6. WHERE <where_condition>
7. GROUP BY <group_by_list>
8. HAVING <having_condition>
9. ORDER BY <order_by_condition>
10. LIMIT <limit_number>
```

SELECT語句- 執行順序：

FROM

<表名> #選取表，將多個表數據通過笛卡爾積變成一個表。

ON

<篩選條件> #對笛卡爾積的虛表進行篩選

JOIN <join, left join, right join...>

<join表> #指定join，用於添加數據到on之後的虛表中，例如left join會將左表的剩餘數據添加到虛表中

WHERE

<where條件> #對上述虛表進行篩選

GROUP BY

<分組條件> #分組

<SUM()等聚合函數> #用於having子句進行判斷，在書寫上這類聚合函數是寫在having判斷裡面的

HAVING

<分組篩選> #對分組後的結果進行聚合篩選

SELECT

<返回數據列表> #返回的單列必須在group by子句中，聚合函數除外

DISTINCT

#數據除重

ORDER BY

<排序條件> #排序

LIMIT

<行數限制>

SQL優化策略

聲明：以下SQL優化策略適用於數據量較大的場景下，如果數據量較小，沒必要以此為準，以免畫蛇添足。

一、避免不走索引的場景

1.盡量避免在字段開頭模糊查詢，會導致數據庫引擎放棄索引進行全表掃描。如下：

```
SELECT * FROM t WHERE username LIKE '%陈%'
```

優化方式：盡量在字段後面使用模糊查詢。如下：

```
SELECT * FROM t WHERE username LIKE '陈%'
```

如果需求是要在前面使用模糊查詢，

- 使用MySQL內置函數INSTR(str,substr) 來匹配，作用類似於java中的indexOf()，查詢字符串出現的角標位置
- 使用FullText全文索引，用match against 檢索
- 數據量較大的情況，建議引用ElasticSearch、solr，億級數據量檢索速度秒級
- 當表數據量較少（幾千條兒那種），別整花里胡哨的，直接用like '%xx%'。

2.盡量避免使用in和not in，會導致引擎走全表掃描。如下：

```
SELECT * FROM t WHERE id IN (2,3)
```

優化方式：如果是連續數值，可以用between代替。如下：

```
SELECT * FROM t WHERE id BETWEEN 2 AND 3
```

如果是子查詢，可以用exists代替。如下：

```
-- 不走索引
select * from A where A.id in (select id from B);
-- 走索引
select * from A where exists (select * from B where B.id = A.id);
```

3.盡量避免使用or，會導致數據庫引擎放棄索引進行全表掃描。如下：

```
SELECT * FROM t WHERE id = 1 OR id = 3
```

優化方式：可以用union代替or。如下：

```
SELECT * FROM t WHERE id = 1
UNION
```



```
SELECT * FROM t WHERE id = 3
```

4.盡量避免進行null值的判斷，會導致數據庫引擎放棄索引進行全表掃描。如下：

```
SELECT * FROM t WHERE score IS NULL
```

優化方式：可以給字段添加默認值0，對0值進行判斷。如下：

```
SELECT * FROM t WHERE score = 0
```

5.盡量避免在where條件中等號的左側進行表達式、函數操作，會導致數據庫引擎放棄索引進行全表掃描。

可以將表達式、函數操作移動到等號右側。如下：

```
-- 全表扫描
SELECT * FROM T WHERE score/10 = 9
-- 走索引
SELECT * FROM T WHERE score = 10*9
```

6.當數據量大時，避免使用where 1=1的條件。通常為了方便拼裝查詢條件，我們會默認使用該條件，數據庫引擎會放棄索引進行全表掃描。如下：

```
SELECT username, age, sex FROM T WHERE 1=1
```

優化方式：用代碼拼裝sql時進行判斷，沒where 條件就去掉where，有where條件就加and。

搜索Java知音公眾號，回复“後端面試”，送你一份Java面試題寶典.pdf

7. 查詢條件不能用<> 或者!=

使用索引列作为条件进行查询时，需要避免使用<>或者!=等判断条件。如确实业务需要，使用到不等于符号，需要在重新评估索引建立，避免在此字段上建立索引，改由查询条件中其他索引字段代替。

8. where条件仅包含复合索引非前置列

如下：复合（联合）索引包含key_part1，key_part2，key_part3三列，但SQL语句没有包含索引前置列"key_part1"，按照MySQL联合索引的最左匹配原则，不会走联合索引。

```
select col1 from table where key_part2=1 and key_part3=2
```

9. 隱式类型转换造成不使用索引

如下SQL语句由于索引对列类型为varchar，但给定的值为数值，涉及隱式类型转换，造成不能正确走索引。

```
select col1 from table where col_varchar=123;
```

10. order by 条件要与where中条件一致，否则order by不会利用索引进行排序

```
-- 不走age索引
SELECT * FROM t order by age;

-- 走age索引
SELECT * FROM t where age > 0 order by age;
```

对于上面的语句，数据库的处理顺序是：

- 第一步：根据where条件和统计信息生成执行计划，得到数据。
- 第二步：将得到的数据排序。当执行处理数据（order by）时，数据库会先查看第一步的执行计划，看order by 的字段是否在执行计划中利用了索引。如果是，则可以利用索引顺序而直接取得已经排好序的数据。如果不是，则重新进行排序操作。
- 第三步：返回排序后的数据。

当order by 中的字段出现在where条件中时，才会利用索引而不再二次排序，更准确的说，order by 中的字段在执行计划中利用了索引时，不用排序操作。

这个结论不仅对order by有效，对其他需要排序的操作也有效。比如group by 、union 、distinct等。

11. 正确使用hint优化语句

MySQL中可以使用hint指定优化器在执行时选择或忽略特定的索引。一般而言，处于版本变更带来的表结构索引变化，更建议避免使用hint，而是通过Analyze table多收集统计信息。但在特定场合下，指定hint可以排除其他索引干扰而指定更优的执行计划。

1. USE INDEX 在你查询语句中表名的后面，添加 USE INDEX 来提供希望 MySQL 去参考的索引列表，就可以让 MySQL 不再考虑其他可用的索引。例子: SELECT col1 FROM table USE

INDEX (mod_time, name)...

2. IGNORE INDEX 如果只是单纯的想让 MySQL 忽略一个或者多个索引，可以使用 IGNORE INDEX 作为 Hint。例子: SELECT col1 FROM table IGNORE INDEX (priority) ...

3. FORCE INDEX 为强制 MySQL 使用一个特定的索引，可在查询中使用FORCE INDEX 作为 Hint。例子: SELECT col1 FROM table FORCE INDEX (mod_time) ...

在查询的时候，数据库系统会自动分析查询语句，并选择一个最合适的索引。但是很多时候，数据库系统的查询优化器并不一定总是能使用最优索引。如果我们知道如何选择索引，可以使用FORCE INDEX强制查询使用指定的索引。

例如：

```
SELECT * FROM students FORCE INDEX (idx_class_id) WHERE class_id = 1 ORDER BY id DESC;
```

二、SELECT语句其他优化

1. 避免出现select *

首先，select * 操作在任何类型数据库中都不是一个好的SQL编写习惯。

使用select * 取出全部列，会让优化器无法完成索引覆盖扫描这类优化，会影响优化器对执行计划的选择，也会增加网络带宽消耗，更会带来额外的I/O,内存和CPU消耗。

建议提出业务实际需要的列数，将指定列名以取代select *。

2. 避免出现不确定结果的函数

特定针对主从复制这类业务场景。由于原理上从库复制的是主库执行的语句，使用如now()、rand()、sysdate()、current_user()等不确定结果的函数很容易导致主库与从库相应的数据不一致。另外不确定值的函数,产生的SQL语句无法利用query cache。

3.多表关联查询时，小表在前，大表在后。

在MySQL中，执行 from 后的表关联查询是从左往右执行的（Oracle相反），第一张表会涉及到全表扫描，所以将小表放在前面，先扫小表，扫描快效率较高，在扫描后面的大表，或许只扫描大表的前100行就符合返回条件并return了。

例如：表1有50条数据，表2有30亿条数据；如果全表扫描表2，你品，那就先去吃个饭再说吧是吧。

4. 使用表的别名

当在SQL语句中连接多个表时，请使用表的别名并把别名前缀于每个列名上。这样就可以减少解析的时间并减少哪些友列名歧义引起的语法错误。

5. 用where字句替换HAVING字句

避免使用HAVING字句，因为HAVING只会在检索出所有记录之后才对结果集进行过滤，而where则是在聚合前刷选记录，如果能通过where字句限制记录的数目，那就能减少这方面的开销。HAVING中的条件一般用于聚合函数的过滤，除此之外，应该将条件写在where字句中。

where和having的区别：where后面不能使用组函数

6.调整Where字句中的连接顺序

MySQL采用从左往右，自上而下的顺序解析where子句。根据这个原理，应将过滤数据多的条件往前放，最快速度缩小结果集。

三、增刪改 DML 语句优化

1. 大批量插入数据

如果同时执行大量的插入，建议使用多个值的INSERT语句(方法二)。这比使用分开INSERT语句快（方法一），一般情况下批量插入效率有几倍的差别。

方法一：

```
insert into T values(1,2);  
  
insert into T values(1,3);  
  
insert into T values(1,4);
```

方法二：

```
Insert into T values(1,2),(1,3),(1,4);
```

选择后一种方法的原因有三。

- 减少SQL语句解析的操作，MySQL没有类似Oracle的share pool，采用方法二，只需要解析一次就能进行数据的插入操作；
- 在特定场景可以减少对DB连接次数
- SQL语句较短，可以减少网络传输的IO。

2. 适当使用commit

适当使用commit可以释放事务占用的资源而减少消耗，commit后能释放的资源如下：

- 事务占用的undo数据块；
- 事务在redo log中记录的数据块；
- 释放事务施加的，减少锁争用影响性能。特别是在需要使用delete删除大量数据的时候，必须分解删除量并定期commit。

3. 避免重复查询更新的数据

针对业务中经常出现的更新行同时又希望获得改行信息的需求，MySQL并不支持PostgreSQL那样的UPDATE RETURNING语法，在MySQL中可以通过变量实现。

例如，更新一行记录的时间戳，同时希望查询当前记录中存放的时间戳是什么，简单方法实现：

```
Update t1 set time=now() where col1=1;

Select time from t1 where id =1;
```

使用变量，可以重写为以下方式：

```
Update t1 set time=now () where col1=1 and @now: = now ();

Select @now;
```

前后二者都需要两次网络来回，但使用变量避免了再次访问数据表，特别是当t1表数据量较大时，后者比前者快很多。

4.查询优先还是更新 (insert 、 update 、 delete) 优先

MySQL 还允许改变语句调度的优先级，它可以使来自多个客户端的查询更好地协作，这样单个客户端就不会由于锁定而等待很长时间。改变优先级还可以确保特定类型的查询被处理得更快。我们首先应该确定应用的类型，判断应用是以查询为主还是以更新为主的，是确保查询效率还是确保更新的效率，决定是查询优先还是更新优先。

下面我们提到的改变调度策略的方法主要是针对只存在表锁的存储引擎，比如 MyISAM 、 MEMROY 、 MERGE，对于Innodb 存储引擎，语句的执行是由获得行锁的顺序决定的。MySQL 的默认的调度策略可用总结如下：

- 1) 写入操作优先于读取操作。
- 2) 对某张数据表的写入操作某一时刻只能发生一次，写入请求按照它们到达的次序来处理。
- 3) 对某张数据表的多个读取操作可以同时地进行。MySQL 提供了几个语句调节符，允许你修改它的调度策略：

- LOW_PRIORITY关键字应用于DELETE、INSERT、LOAD DATA、REPLACE和UPDATE；
- HIGH_PRIORITY关键字应用于SELECT和INSERT语句；
- DELAYED关键字应用于INSERT和REPLACE语句。

如果写入操作是一个 LOW_PRIORITY (低优先级) 请求，那么系统就不会认为它的优先级高于读取操作。在这种情况下，如果写入者在等待的时候，第二个读取者到达了，那么就允许第二

个读取者插到写入者之前。只有在没有其它的读取者的时候，才允许写入者开始操作。这种调度修改可能存在 LOW_PRIORITY写入操作永远被阻塞的情况。

SELECT 查询的HIGH_PRIORITY (高优先级) 关键字也类似。它允许SELECT 插入正在等待的写入操作之前，即使在正常情况下写入操作的优先级更高。另外一种影响是，高优先级的SELECT 在正常的 SELECT 语句之前执行，因为这些语句会被写入操作阻塞。如果希望所有支持LOW_PRIORITY 选项的语句都默认地按照低优先级来处理，那么 请使用--low-priority-updates 选项来启动服务器。通过使用 INSERTHIGH_PRIORITY 来把 INSERT 语句提高到正常的写入优先级，可以消除该选项对单个INSERT语句的影响。

搜索Java知音公众号，回复“后端面试”，送你一份Java面试题宝典.pdf

四、查询条件优化

1. 对于复杂的查询，可以使用中间临时表 暂存数据

2. 优化group by语句

默认情况下，MySQL 会对GROUP BY分组的所有值进行排序，如 “GROUP BY col1 , col2 , ...;” 查询的方法如同在查询中指定 “ORDER BY col1 , col2 , ...;” 如果显式包括一个包含相同的列的 ORDER BY子句，MySQL 可以毫不减速地对它进行优化，尽管仍然进行排序。

因此，如果查询包括 GROUP BY 但你并不想对分组的值进行排序，你可以指定 ORDER BY NULL禁止排序。例如：

```
SELECT col1, col2, COUNT(*) FROM table GROUP BY col1, col2 ORDER BY NULL ;
```

3. 优化join语句

MySQL中可以通过子查询来使用 **SELECT** 语句来创建一个单列的查询结果，然后把这个结果作为过滤条件用在另一个查询中。使用子查询可以一次性的完成很多逻辑上需要多个步骤才能完成的 **SQL** 操作，同时也可以避免事务或者表锁死，并且写起来也很容易。但是，有些情况下，子查询可以被更有效率的连接(**JOIN**)..替代。

例子：假设要将所有没有订单记录的用户取出来，可以用下面这个查询完成：

```
SELECT col1 FROM customerinfo WHERE CustomerID NOT in (SELECT CustomerID FROM salesinfo )
```

如果使用连接(**JOIN**).. 来完成这个查询工作，速度将会有所提升。尤其是当 **salesinfo**表中对 **CustomerID** 建有索引的话，性能将会更好，查询如下：

```
SELECT col1 FROM customerinfo
LEFT JOIN salesinfo ON customerinfo.CustomerID=salesinfo.CustomerID
WHERE salesinfo.CustomerID IS NULL
```

连接(**JOIN**).. 之所以更有效率一些，是因为 **MySQL** 不需要在内存中创建临时表来完成这个逻辑上的需要两个步骤的查询工作。

4. 优化union查询

MySQL通过创建并填充临时表的方式来执行**union**查询。除非确实要消除重复的行，否则建议使用**union all**。原因在于如果没有**all**这个关键词，**MySQL**会给临时表加上**distinct**选项，这会导致对整个临时表的数据做唯一性校验，这样做的消耗相当高。


高效：

```
SELECT COL1, COL2, COL3 FROM TABLE WHERE COL1 = 10

UNION ALL

SELECT COL1, COL2, COL3 FROM TABLE WHERE COL3= 'TEST';
```

低效：



```
SELECT COL1, COL2, COL3 FROM TABLE WHERE COL1 = 10

UNION

SELECT COL1, COL2, COL3 FROM TABLE WHERE COL3= 'TEST';
```

5. 拆分复杂SQL为多个小SQL，避免大事务

- 简单的SQL容易使用到MySQL的QUERY CACHE；
- 减少锁表时间特别是使用MyISAM存储引擎的表；
- 可以使用多核CPU。

6. 使用truncate代替delete

当删除全表中记录时，使用delete语句的操作会被记录到undo块中，删除记录也记录binlog，当确认需要删除全表时，会产生大量的binlog并占用大量的undo数据块，此时既没有很好的效率也占用了大量的资源。

使用truncate替代，不会记录可恢复的信息，数据不能被恢复。也因此使用truncate操作有其极少的资源占用与极快的时间。另外，使用truncate可以回收表的水位，使自增字段值归零。

7. 使用合理的分页方式以提高分页效率

使用合理的分页方式以提高分页效率 针对展现等分页需求，合适的分页方式能够提高分页的效率。

案例1：

```
select * from t where thread_id = 10000 and deleted = 0
order by gmt_create asc limit 0, 15;
```

上述例子通过一次性根据过滤条件取出所有字段进行排序返回。数据访问开销=索引IO+索引全部记录结果对应的表数据IO。因此，该种写法越翻到后面执行效率越差，时间越长，尤其表数据量很大的时候。

适用场景：当中间结果集很小（10000行以下）或者查询条件复杂（指涉及多个不同查询字段或者多表连接）时适用。

案例2：

```
select t.* from (select id from t where thread_id = 10000 and deleted = 0
order by gmt_create asc limit 0, 15) a, t
where a.id = t.id;
```

上述例子必须满足t表主键是id列，且有覆盖索引secondary key:(thread_id, deleted, gmt_create)。通过先根据过滤条件利用覆盖索引取出主键id进行排序，再进行join操作取出其他字段。数据访问开销=索引IO+索引分页后结果（例子中是15行）对应的表数据IO。因此，该写法每次翻页消耗的资源和时间都基本相同，就像翻第一页一样。

适用场景：当查询和排序字段（即where子句和order by子句涉及的字段）有对应覆盖索引时，且中间结果集很大的情况时适用。

五、建表优化

1. 在表中建立索引，优先考虑where、order by使用到的字段。

2. 尽量使用数字型字段（如性别，男：1 女：2），若只含数值信息的字段尽量不要设计为字符型，这会降低查询和连接的性能，并会增加存储开销。

这是因为引擎在处理查询和连接时会 逐个比较字符串中每一个字符，而对于数字型而言只需要比较一次就够了。

3. 查询数据量大的表 会造成查询缓慢。主要的原因是扫描行数过多。这个时候可以通过程序，分段分页进行查询，循环遍历，将结果合并处理进行展示。要查询100000到100050的数据，如下：

```
SELECT * FROM (SELECT ROW_NUMBER() OVER(ORDER BY ID ASC) AS rowid,*  
FROM infoTab)t WHERE t.rowid > 100000 AND t.rowid <= 100050
```

4. 用varchar/nvarchar 代替 char/nchar

尽可能的使用 `varchar/nvarchar` 代替 `char/nchar`，因为首先变长字段存储空间小，可以节省存储空间，其次对于查询来说，在一个相对较小的字段内搜索效率显然要高些。

不要以為NULL 不需要空間，比如：`char(100)` 型，在字段建立時，空間就固定了，不管是否插入值（NULL也包含在內），都是佔用100個字符的空間的，如果是`varchar`這樣的變長字段，`null` 不佔用空間。



- [今天，又被Java8的時間庫噁心到了，有同感的舉手...](#)
- [Docker從入門到干活，看這一篇足矣](#)
- [程序員需知的58 個網站](#)
- [騰訊Code Review 規範出爐！你還敢亂寫代碼？？](#)
- [誰總結的JavaWeb會話技術了？太全面了...](#)
- [2萬字20個實例解析Java8 Stream，帶你玩轉集合四大點！](#)



好文章，我在看

喜歡此內容的人還喜歡

強大：MyBatis 流式查詢



Java後端



師兄，為什麼刪除數據後，Redis內存佔用依然很高？

碼猿技術專欄



網頁端收消息，究竟是推還是拉？

架構師之路

