# 第五篇 DB

## 5.1 MySQL

### 5.1.1 发展

MySQL 1.0（1996）

1996年，MySQL 1.0发布 ，只有CRUD基本操作

1999-2000年，MySQL AB在瑞典成立。 开发出了 Berkeley DB引擎, MySQL从此开始支持事务处理。

2000 年 MySQL 公布了自己的源代码，并采用GPL许可协议，正式进入开源世界。

MySQL 3.23（2001）

实现了SQL 查询系统，MyISAM 代替了老旧且有诸多限制的 ISAM 引擎。InnoDB 引擎也已经可以使用，Mysql真正“诞生”的时刻。

MySQL 4.0（2003）

不支持存储过程、触发程序、服务器端指针或视图，这时候的MySQL还不是一个企业级数据库。

MySQL 5.0（2006）

出现了一些“企业级”特性：视图、触发器、存储过程和存储函数。同时 Oracle 收购的 InnoDB Oy发布了 InnoDB plugin。

5.1（2008）

Sun 收购 MySQL AB 以后发布的首个版本，研发时间长达五年。

2008年 MySQL被Sun公司收购。

2009年Oracle收购Sun 公司，MySQL 转入Oracle 门下。Oracle将MySQL分成了社区版和企业版。

MySQL 5.5（2010）

默认InnoDB plugin引擎。具有提交、回滚和Crash恢复功能、ACID兼容。

行级锁

MySQL 5.6（2012）

MySQL 5.7（2015）

### 4.3.2 逻辑架构



查询的生命周期

客户端🡪服务器🡪解析🡪生成执行计划🡪执行🡪客户端

SQL生命周期每个环节都有足够的优化空间，优化的本质是减少SQL对资源的消耗和依赖

执行：调用storage engine和调用后的数据处理，包括排序、分组等。

服务层

为请求做连接处理，授权认证，安全等。

核心层

* SQL查询解析，分析，优化，缓存，
* 提供内建函数;
* 存储过程，触发器，视图。

Storage engine

存储引擎。data/index的save和query，事务处理。

### 4.3.4 存储引擎

影响引擎性能的因素：数据大小，I/O请求量，主键还是二级索引。

InnoDB

采用了**行级锁**。优先选择InnoDB。

* 事务：崩溃恢复
* 聚簇索引：
* 热备份

MyISAM

* select性能高？但随着应用压力上升，各种锁争用，崩溃数据丢失问题
* 表级锁，不支持事务。
* 缓存索引，而不缓存数据。

TokuDB

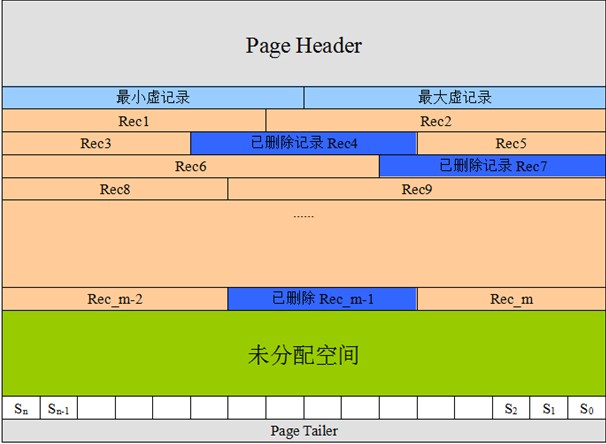
适合读少写多，提升了在高负载情况下的 InnoDB 的性能

### 5.1.4 页面

TODO 页大小为什么是4K

页面格式

[Reference](https://www.cnblogs.com/vinchen/archive/2012/09/10/2679478.html)



页内的有序性

按序存储：插入导致记录移动

链表：查找需要逐个比较

slot和每个record对应，只需保证slot按序存储，record链表存储即可。

**slot区**：slot是一些页面有效记录的指针，每个slot占两个字节，存储了记录相对页面首地址的偏移。如果页面有n条有效记录，那么slot的数量就在n/8+2~n/4+2之间。下一节详细介绍slot区，它是记录页面有序和二分查找的关键。

### 4.3.5 主从分离架构



#### 主从备份

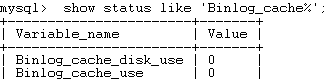
#### Binlog

[Reference](https://www.cnblogs.com/martinzhang/p/3454358.html)

记录了所有的DDL和DML(除了数据查询语句)语句

查看

 show status like 'Binlog\_cache%';



使用场景

1.主从复制：MySQL Replication在Master端开启binlog，Master把它的二进制日志传递给slaves来达到master-slave数据一致的目的。

2.数据恢复：使用mysqlbinlog工具来恢复数据。

### 4.3.6 Mgr

连接：mysql –uroot [-h slc11fsp] -p

启动服务 service mysqld start

版本**：**mysql –V；或者select version();

information\_schema

indexs

Show: show **index** **from** fund\_history\G;

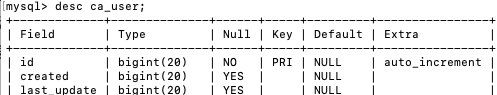
Create: create **index** idx\_fund\_name on fund(name);

tables

show databases/tables;

查看表结构

desc table\_name;



Show DDl: show create table ld\_cart;

**Status**: show [**global**/session] status like '%xxx%';

**Variables:** show variables like '%max\_connections%';

**Innodb:** show [**global**] status like 'Innodb\_%'

GRANT ALL PRIVILEGES ON \*.\* TO 'root'@'%' IDENTIFIED BY '123456' WITH GRANT OPTION;

PLSQL

**优势**

1 SQL语言只是访问、操作数据库的语言。PLSQL具备高级语言的特性，比如模块化，流程控制，异常。

2.集成在数据库中，调用更快。减少了网络的交互，从而减少网络传输时间。

### 4.3.5 监控

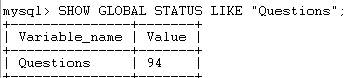
mysql提供的指标很多，主要监控4大指标

[Reference](http://blog.oneapm.com/apm-tech/754.html)

#### 查询吞吐量

QPS

Questions ： 查询的总数（统计开始时间点为service start）。



TPS

TPS = (Com\_commit + Com\_rollback) / seconds

mysql > show status like 'Com\_commit';

mysql > show status like 'Com\_rollback';

mysql workBench可查看

#### 查询执行性能

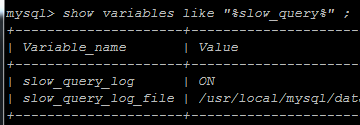
#### 慢查询日志

Slow\_queries： 超过long\_query\_time数量。（show variables like "%long\_query\_time%"）;

show variables like 'log%';

show query

show variables like "%slow\_query%" ;



开启日志

set global slow\_query\_log=ON;

分析

mysqldumpslow /usr/local/mysql/data/mysql/slc11fsp-slow.log

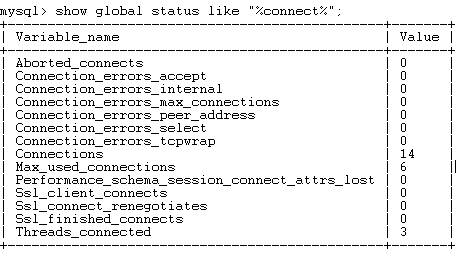
#### 连接情况

Connections

show full processlist;



每一条代表一个Thread，一个Tcp连接。



Aborted\_connects 尝试已经失败的MySQL服务器的连接的次数。

Threads\_connected：当前连接数。只有数量值;

Threads\_running：正在运行线程数。

connections：试图连接mySQL服务器的次数

Max\_used\_connections

Uptime 服务器工作了多少秒。

#### 缓冲池使用情况

key Buffer 命中率

key\_buffer\_read\_hits = (1-key\_reads / key\_read\_requests) \* 100%

key\_buffer\_write\_hits = (1-key\_writes / key\_write\_requests) \* 100%

mysql> show status like 'Key%';

InnoDB Buffer命中率

innodb\_buffer\_read\_hits = (1 - innodb\_buffer\_pool\_reads / innodb\_buffer\_pool\_read\_requests) \* 100%

mysql> show status like 'innodb\_buffer\_pool\_read%';

Query Cache命中率

Query\_cache\_hits = (Qcahce\_hits / (Qcache\_hits + Qcache\_inserts )) \* 100%;

mysql> show status like 'Qcache%';

Table Cache状态量

mysql> show status like 'open%';

Thread Cache 命中率

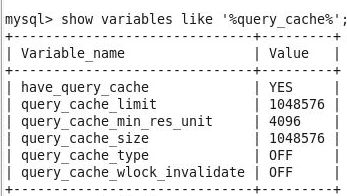
Thread\_cache\_hits = (1 - Threads\_created / connections ) \* 100%

mysql> show status like 'Thread%';

mysql> show status like 'Connections';

Query

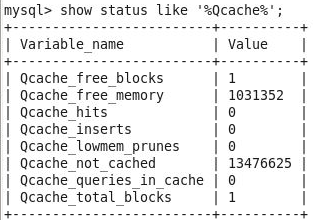
show variables like '%query\_cache%'



query\_cache\_limit 缓冲区大小，缺省为1M；

query\_cache\_min\_res\_unit缓存块大小，默认4KB，设置值大对大数据查询有好处，但如果你的查询都是小数据 查询，就容易造成内存碎片和浪费；

Cache Status



Qcache\_free\_blocks：缓存中相邻内存块的个数。数目大说明可能有碎片。FLUSH QUERY CACHE会对缓存中的碎片进行整理，从而得到一个空闲块。

Qcache\_free\_memory：缓存中的空闲内存。

Qcache\_hits：每次查询在缓存中命中时就增大

Qcache\_inserts：每次插入一个查询时就增大。命中次数除以插入次数就是不中比率。

Qcache\_lowmem\_prunes：缓存出现内存不足并且必须要进行清理以便为更多查询提供空间的次数。这个数字最好长时间来看;如果这个 数字在不断增长，就表示可能碎片非常严重，或者内存很少。(上面的 free\_blocks和free\_memory可以告诉您属于哪种情况)

Qcache\_not\_cached：不适合进行缓存的查询的数量，通常是由于这些查询不是 SELECT 语句或者用了now()之类的函数。

Qcache\_queries\_in\_cache：当前缓存的查询(和响应)的数量。

Qcache\_total\_blocks：缓存中块的数量。

### 5.6 Nested-Loop Join

假设join types

Table Join Type

t1 range

t2 ref

t3 ALL

Simple Nested-Loop Join

1. **for** each row in t1 matching range {
2. **for** each row in t2 matching reference key {
3. **for** each row in t3 {
4. **if** row satisfies join conditions, send to client
5. }
6. }
7. }

Block Nested-Loop Join

## 5.2 DB设计

### 4.1.1 数据类型

* varchar：较char占用更少的存储空间,提升性能。但更新时引起额外的工作。varchar宽度越长，会消耗更多内存(mysql为优化查询，在内存中对字符串使用固定的宽度)。
* unsigned 不允许负值，可使正数的上限提高一倍.
* timestamp占4个字节， datetime用8个。而TIMESTAMP只能表示1970 - 2038年
* date 三个字节
* text 处理方式决定了他的性能要低于char或者是varchar类型。
* blob

### 4.4.2 范式

[Reference](http://blog.csdn.net/dongnan591172113/article/details/7910988)

范式和反范式

范式需要联表，代价昂贵。还可能使一些索引策略无效，比如覆盖索引。

### 4.4.3 Integrity Constraints

完整性约束。约束保证数据的正确性，可靠性和一致性。包括域约束，基本表约束等。

[Reference](https://www.indiastudychannel.com/resources/150128-Integrity-Constraints-DBMS.aspx)

#### Domain Integrity

域完整性

* CHECK(检查)约束：用于限制列中值得范围
* DEFAULT(默认值)约束
* NOT NULL(非空)约束

#### Entity integrity constraints

实体约束，表约束。保证Table的记录都是唯一的 。

* PRIMARY KEY(主键)约束：唯一识别每一条记录的标志，可以有多列共同组成
* IDENTITY(自增)约束：列值自增，一般使用此属性设置的列作为主键
* UNIQUE(唯一)约束

#### Referential Integrity

参照完整性。保证关联table的数据是一致的。 T1(id,name),T2(id,t1\_id),T2的外键t1\_id,在T1里必须是可用的。

* FOREIGN KEY(外键)

## 5.3 事务

一组操作完成一个任务。在计算机术语中，事务通常就是指数据库事务。

语法

1. start **transaction** | **begin**；
2. DML...
3. **commit**;

begin会挂起autocommit，commit后恢复原来的值。set autocommit = 0也可开启。

提交即commit

Autocommit

自动提交事务，对当前connect生效，相当于开启事务。

1. --查看
2. show variables like 'autocommit' / **select** @@autocommit

set autocommit = 0 关闭自动提交。

查看事务innodb\_trx

1. **select** \* **FROM** information\_schema.innodb\_trx\G;



trx\_id

trx\_state: RUNNING/LOCK WAIT

trx\_mysql\_thread\_id

trx\_stared 事务开始时间

trx\_wait\_stated:block事务wait stated

trx\_query:

trx\_rows\_locked 锁定row行数

trx\_tables\_locked 锁定table数量

### 5.3.1事务特性

Atomic（原子性）

要么全部成功，要么全部失败。执行的完整性。

Consistency（一致性）

数据不会因为事务的执行而遭受破坏.只有合法的数据可以被写入数据库，否则事务应该将其回滚到最初状态。

实现: 完整性约束：

这里的一致性偏向于数据的正确性。

Isolation（隔离性）

事务允许多个用户对同一个数据进行并发访问，而不破坏数据的正确性和完整性。

实现：事务的锁来进行控制。

Durability（持久性）

事务和锁的关系

[Reference](https://tech.meituan.com/innodb_lock.html)

TODO2

应用层加锁：方法串行化执行。等于隔离级别的Serializable。

数据库要保证最大的并发执行…读锁写锁。

Hollis: [Reference](https://www.hollischuang.com/archives/tag/%E4%BA%8B%E5%8A%A1)

### 5.3.2 隔离级别

事务的隔离级别和数据库的性能呈反比。

* Read uncommitted：。but会出现脏读。
* Read committed：。but其他事务所做的修改或删除，出现不可重复读
* Repeatable read ：其他事务update or delete，事务依然可以Repeatable read。but其他事务插入，此时发生幻像读。
* Serializable：序列化。表锁。

查看InnoDB存储引擎 系统级的隔离级别和会话级的隔离级别

1. **select** @@**global**.tx\_isolation,@@tx\_isolation;

1）补充Repeateale read

实现原理：mysql实现了多版本并发控制（MVCC），保存数据的某个时间点快照。

好处：避免加锁操作，开销更新。

Problem：多版本导致read的数据可能过期，会出现**丢失更新**(P1)。

Mysql默认隔离级别Repeate read有lost update 问题。会引起其他潜在问题,比如营业额增加，银行转账。

|  |  |  |
| --- | --- | --- |
| **时间** | **trx1** | **trx2** |
| T1 | begin;select=>100 |  |
| T2 |  | select =>m :=100元 |
| T3 |  | update :m+100 |
| T4 |  | commit |
| T5 | update :m+100 |  |
| T6 | commit |  |

解决思路：

* 加悲观锁: 。trx1和trx2在select都for update
* 乐观锁：应用层实现，比如Hibernate。
* 设置为Serializable

(Oracle可以把隔离级别设置成read-only。?)

2）select是否开启事物的区别？

Select开启事物后可保证repeateable read

### 5.3.3 事务锁

事务通过锁实现，不同隔离级别锁的实现不同。

锁定的资源包括行、表、页、簇和数据库。为了最小化锁的成本，尽量锁定比较小的对象。

行级锁

加锁粒度最小，但加锁的开销也最大。行级锁分为[共享锁](http://www.hollischuang.com/archives/923)和[排他锁](http://www.hollischuang.com/archives/923)。

共享锁(Share Lock)

T1对数据A加共享锁后，Mysql会对select结果中的每行都加共享锁。事务T2只能对A再加共享锁，不能加排他锁。用法

1. **SELECT** ... LOCK IN SHARE MODE;

排他锁（eXclusive Lock）

排他锁又称写锁，T1对数据A加排他锁后，T2不能再对A加任任何类型的锁，但是可以读。

1. **SELECT** ... **FOR** **UPDATE**;

|  |  |  |
| --- | --- | --- |
| **时间** | **trx1** | **trx2** |
| T1 | begin; |  |
| T2 | select…for update | begin; |
| T3 |  | upate will throw exception. |
| T4 | commit |  |
| T5 |  |  |

innoDB对update、insert、delete会自动加排他锁（select不加），所以T3处的trx2的update会阻塞到Lock wait timeout exceeded。

查看lock具体数据

1. **select** \* **from** information\_schema.innodb\_locks;



lock\_trx\_id找到事务

lock\_table+lock\_data找到具体锁定的数据。

Lock\_mode: X/S

lock\_type： RECORD行锁

InnoDB 行锁是通过给索引项加锁实现的，并不是直接锁记录。若没有索引，InnoDB 会通过隐藏的聚簇索引来对记录加锁。索引查询不走索引，要扫描全表，就得锁定全表数据。

意向锁

意向锁是表级锁，其设计为了在一个事务中揭示下一行将要被请求锁的类型。InnoDB中的两个表锁：

意向共享锁（IS）：表示事务准备给数据行加入共享锁，也就是说一个数据行加共享锁前必须先取得该表的IS锁

意向排他锁（IX）：类似上面，表示事务准备给数据行加入排他锁，说明事务在一个数据行加排他锁前必须先取得该表的IX锁。

意向锁是InnoDB自动加的，不需要用户干预。

1. 表级锁

lock tables tablename read|write

1. 页级锁

悲观锁&乐观锁

不是数据库中真正存在的锁，是并发控制采用的技术手段。其实不仅仅是关系型数据库系统中有乐观锁和悲观锁的概念，像memcache、hibernate、tair等都有类似的概念。

## 5.4 索引

### 4.4.1 索引是什么？

在数据之外，DB维护着满足特定查找算法的数据结构，这些数据结构以某种方式引用（指向）数据，这样就可以在这些数据结构上实现高级查找算法。这种数据结构，就是索引。索引优化了查询。本质是空间换时间。

索引的优点

1）索引大大减少了服务器需要扫描的数据量

2）索引可以帮助服务器避免排序和临时表

3）索引可以将随机I/O变成顺序I/O

哪些字段要索引

Where column, order by

InnoDB的页面分为

叶子节点：B树层次为0的页面，存储记录的所有内容

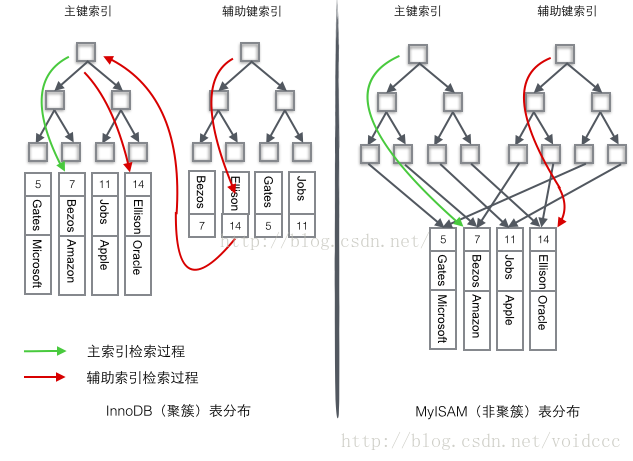
非叶子节点：B树层次大于0的页面，只存储索引键和页面指针。

### 4.4.2 索引类型

* Primary Index：数据非空值不重复（唯一），表只有一个。
* 普通索引
* Unique Index：保证数据唯一性
* 组合索引（联合索引、复合索引）：
* Hash索引
* 全文索引(Full Text)

还有多列索引、短索引（前缀索引）、覆盖索引、聚簇索引和非聚簇索引，这些都不能将单独它归为一个索引类型。

* 覆盖索引：联合索引之上的叶子节点已覆盖要查询的列，不需要访问表数据。eplain的extra显示using index。
* 聚簇索引： clustered index。行数据就储存在主键索引树的叶子节点上。
* 非聚簇索引：又称辅助索引/Secondary index 。索引和表数据存储在独立的地方。



InnoDB：数据行和索引存在一起。页节点存主键值（方便通过通过主键索引查找）。

MyISAM 主辅助索引结构上没区别，只是主索引的叶节点是Primary Key。页节点存数据的地址。

聚簇索引优点

* 聚集相关数据，比如userId做主键，相关数据只需读少量数据页，大大减少磁盘I/O。
* 使用覆盖索引扫描，可直接使用叶节点的主键值。

缺点

* 基于聚集索引插入，可能导致页裂变，
* 全表扫描慢，尤其是行比较稀疏，或也分裂导致存储不连续。
* 二级索引包含主键值，可能更大，索引时需要两次查找，而不是一次。

Multiple-Column Indexes

这里指在多个列上分别建索引。建索引的一个误区是在where后面条件每个列建一个索引，这样最好的情况也只能用到一个索引。生效原则：

Or：index merge。耗费性能做数据合并排序。特别是返回数据量较大时

And：

索引顺序:

### 4.4.4 查询算法

B-Tree

[Reference](https://blog.csdn.net/Roy_70/article/details/76581028)

即Balanced Tree, 有别于二叉查找树（Binary Search Tree）。B+Tree实现其索引结构。

二叉搜索树的查询时间复杂度是O(log(n))，从算法逻辑上来讲，无论查找速度还是比较次数都是最小的，但在数据库的索引通常十分庞大，需要以文件形式存储，而磁盘IO的存取次数就是评价一个数据库索引优劣的关键性指标。

B-Tree优化了磁盘IO

## 5.5 InnoDB

并发控制，MVCC，锁

## 5.7 性能优化

information\_schema/performance\_schema

4.5.2 优化策略

不同业务场景有不同的侧重，比如、读写比例、并发量、数据量和数据增长数据。

* 读写比例：根据具体业务读写比例，降低主库压力
* 高并发：集群
* 数据是否需要强一致🡺NOSQL
* 数据量：数据增长快。

1.针对数量大的表进行历史表分离（如交易流水表）。

2.做分表分库，单表行数超过 500 万行或者单表容量超过 2GB，才推荐进行分库分表。说明：如果预计三年后的数据量根本达不到这个级别，不要在创建表时就分库分表

### 4.6.1 库表结构优化

MySQL基于Row存储，数据文件和索引文件，以文件形式存在磁盘，而IO时以page（block）读到内存。磁盘IO相对于内存存取，I/O存取的消耗要高几个数量级。减少IO次数手段：

* 读数据：每条记录占用的空间量减小，每页中可存放的数据行数增大。进而减少IO次数。
* 索引查找：查找过程中产生磁盘I/O消耗，IO的存取次数是判断索引性能的关键性指标。

选择合适的数据类型

* 更小的通常更好：占用更少的磁盘、内存和CPU缓存，CUP处理周期。vachar宽度。
* 简单就好：简单类型CPU处理快。例如整形比字符操作代价低。
* not null：确保 default值不是 NULL 。null使索引更复杂，影响效率。

适当拆分

text或varchar大字段：若访问不到，拆分到独立地表，以减少常用数据所占用的存储空间。

好处：每个数据块中可以存储更多的数据，既减少物理 IO 次数，提高内存中的缓存命中率。

适当冗余

通过空间换时间，不用连表，

缓存

缓存重复查询和获取比较慢的数据。

### 4.6.2 索引优化

索引增加写数据的额外消耗，频繁写的表不宜建索引。

三星系统

[Reference](https://yq.aliyun.com/articles/5556)

第一颗星：在where条件里的列建索引，避免直接扫描DB，避免全表扫描

第二颗星：order by 中的排序和索引顺序是否一致

第三颗星：覆盖索引。（where条件中column）索引行包含所有select的列。

索引失效

[Reference](https://mp.weixin.qq.com/s?__biz=MzIwMTY0NDU3Nw==&mid=2651937946&idx=1&sn=e0797f6334ccd51860f3e6db70e2878f&chksm=8d0f33d4ba78bac2f26f688221235b18a636d43a736b31bf442b1ab53c1b789980d148338ad9&mpshare=1&scene=1&srcid=#rd])

* 前导模糊查询：不能命中索引。like %value
* 负向条件：可优化为in查询。负向条件有：!=、<>、not in、not exists、not like等。
* 复合索引的情况下，查询条件不包含索引列最左边部分。
* 用or分割开的条件，如果or前的条件中列有索引，而后面列中没有，那么涉及到的索引都不会被用到

### 4.6.3 SQL优化

优化数据访问

1.select必要的列和行。

2.explain确保MySQL服务层分析必要的行。

解决方法

建索引：type判断是否全表扫描，rows越小越好，理想情况扫描行数=返回行数。

汇总表：group by 或count/一些 会引起全表扫描，必要时使用单独的汇总表或者放Memcached。

缓存：缓存重复查询和获取比较慢的数据。

批量操作。

分解关联查询

复杂查询 or 简单查询？

大的语句会锁住很多数据，占满整个事务日志，耗尽系统资源，阻塞很多小而重要查询。

传统实现总是强调DB完成尽可能的工作，原因在于以前网络通信，而且每个SQL的查询解析和优化代价高。

用单表还是联表，看具体业务场景。在高并发高读环境，遇到性能瓶颈时。好处

* 减少锁竞争 （TODO1? 查询有锁竞争？）
* 方便优化，单表建覆盖索引。
* 缓存更高效，dao层的复用率更好，更好维护，后期好扩展。数据库好拆分。

分解方法

将联表分解成单表冗余字段的形式

建汇总表

需要联表查询

* 连表时，保证被关联的字段需要有索引，一般在连表顺序第二个create index。
* 数据类型必须绝对一致
* 考虑对相对应的查询条件做索引。加快查询速度。
* 一般联表消耗更多系统资源，还可能使索引失效(?)，超过三个表禁止join。

### 4.6.3 Case

##### 分页

Mysql的分页查询十分简单，但是当数据量大的时候一般的分页就吃不消了。

原理：先读取前面n条记录，然后抛弃前n条，读后面m条想要的，所以n越大，偏移量越大，性能就越差。

1. **select** id,**name** **from** table\_name limit 866613, 20

优化

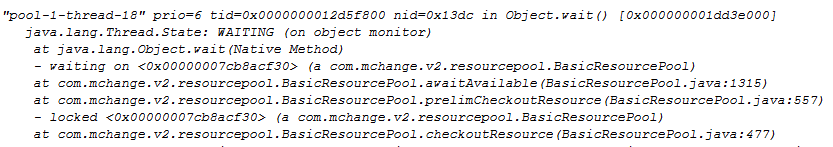
1. **select** id,**name** **from** table\_name **where** id> 866612 limit 20

TODO exists

##### insert时间长 (远程)

每个请求占用一个request/缓解mysql压力，将数据存入redies。然后批量插入mysql

2．未关闭事务，导致系统无响应。



WAITING等待连接池连接。

##### 搜索

页面搜索严禁左模糊或者全模糊，如果需要走搜索引擎来解决。

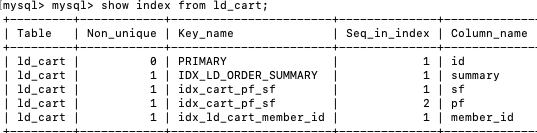
说明：索引文件具有 B-Tree 的最左前缀匹配特性，如果左边的值未确定，那么无法使用此索

引。

## 附录 CMD

### Explain执行计划

Ceate index



Explain SQL



* id：id越大则优先级越高，越先会被执行。如果id相同，则执行顺序从上至下。
* select\_type：simple/ primary/ derived
* type：all < index < range < ref < eq\_ref < const < system < NULL
* possible\_keys :基于SQL的访问列和where中使用的比较符(=><)
* Key：The index actually chosen。
* key\_len：使用索引最大byte
* ref：The ref column shows which columns or constants are compared to the index named in the key column to select rows from the table.
* rows：the number of rows MySQL believes it must examine to execute the query。
* Extra：Additional information about how MySQL resolves the query。

#### select\_type

simple: 不使用UNION或子查询等

primary: 最外层的select

subquery，

derived:里层的子查询



union，

unionresult

#### Type

all < index < range < ref < eq\_ref < const < system < NULL

* **ALL**：即full table scan，（到存储引擎）遍历全表
* **Index**

full index scan，遍历全部的索引树。



没过滤条件导致，相比full table scan好处是避免排序。

* **range**： 1.使用索引 2.只检索给定范围的行



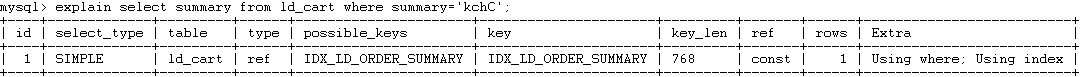
* **ref**： 使用了索引

1) Using Where



先走索引🡪回表查询数据，回表次数太多会严重影响SQL性能，不如直接全部扫描。

2) Using where;Using index;



这里的Using where**指在server层面过滤index。只用一次IO。**

* **eq\_ref:** 类似ref，最多返回一条记录。区别:索引列是primary key或者unique index
* **const，system**: mysql能对查询部分优化转成变量，比如id在where子句，mysql将查询转成常量，然后就可高效将表从联表执行中移除。



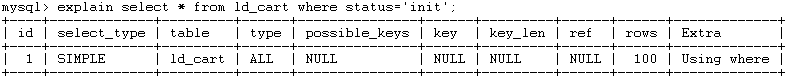
* **NULL：**表示在执行语句中，不用查表或索引

#### Extra

Additional information about how MySQL resolves the query。

* Using index：使用了索引。
* Using where: 过滤了数据。包括index和data
* Using index condition： v5.6加入。可以在store engine过滤。
* using temporay表示用临时表来存储结果集，常见于排序和分组查询
* using filesort，mysql中无法用索引完成的排序成为文件排序。

**Using where**

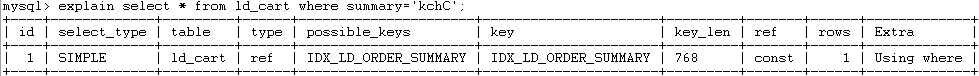


Type All全表扫描，Extra：扫描的数据在server层面过滤。

Using index condition

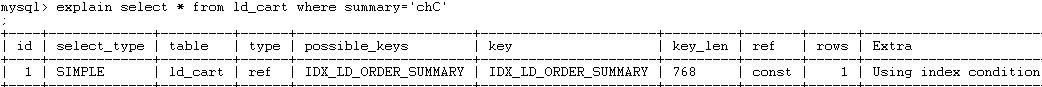
Tables are read by accessing index tuples and testing them first to determine whether to read full table rows. In this way, index information is used to defer (“push down”) reading full table rows unless it is necessary

先看5.6之前版本，

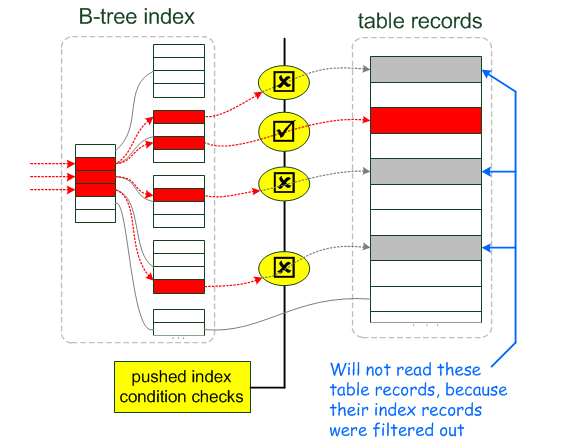


server先读到index在server层面过滤，再到store engine读到full rows。没有Using index因为index没有覆盖。

v5.6



storage engine 根据index 做判断, 再read full rows. 这样节省一次disk IO.



SQL执行过程：

以前是server 命令 storage engine 按 index 把where后column相应的数据从数据表读出, 传给server, server来按 where条件做选择; 现在 ICP则是在 可能的情况下, 让storage engine 根据index 做判断, 如果不符合条件无须读数据表. 这样节省了disk IO.

Using union

1. mysql> explain **select** \* **from** ld\_cart **where** summary='kchC' or member\_id=13\G;
2. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
3. id: 1
4. select\_type: SIMPLE
5. **table**: ld\_cart
6. type: index\_merge
7. possible\_keys: IDX\_LD\_ORDER\_SUMMARY,idx\_ld\_cart\_member\_id
8. **key**: IDX\_LD\_ORDER\_SUMMARY,idx\_ld\_cart\_member\_id
9. key\_len: 768,9
10. ref: NULL
11. **rows**: 2
12. Extra: Using **union**(idx\_summary,idx\_member\_id); Using **where**

同时扫描summary和member\_id两个列索引，并将结果合并。

Where

Mysql应用where条件记录，从好到坏依次为

1）存储索引层，在索引中使用where条件过滤(Using index condition)。过滤覆盖索引。

2) 在mysql服务器层, 扫描索引过滤并返回记录。无需在回表查询记录(Using where，Using index)。**这里过滤的是索引**。

3) 在mysql服务器层，先从数据表读出记录然后过滤(Using Where)

1 2 有待验证

### Transaction

ReadOnly

查看 show global variables like "%read\_only%";

设置

### Transaction Lock

。

Timeout

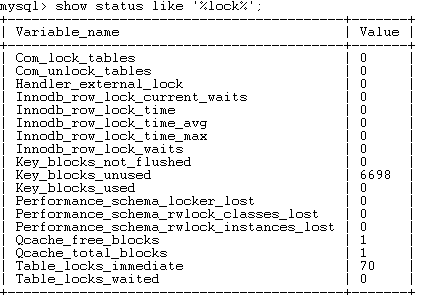
等待TimeOut： show variables like 'innodb\_lock\_wait\_timeout';

Syntax: show status like '%Innodb\_row\_lock%';



Innodb\_row\_lock\_current\_waits 当前阻塞线程数

Innodb\_row\_lock\_waits 总共wait次数



### performance\_schema

#### 数据大小

select data\_length,index\_length from information\_schema.tables where table\_schema='bigdata' and table\_name = 'ld\_cart';

默认单位是byte，优化显示

select concat(round(sum(data\_length/1024/1024),2),'MB') as data\_length\_MB,concat(round(sum(index\_length/1024/1024),2),'MB') as index\_length\_MB from information\_schema.tables where table\_schema='bigdata' and table\_name = 'fund\_history';

idex

show index from bigdata.fund;

#### 统计

DDL统计

Innodb\_rows\_read/inserted/deleted/updated

Innodb\_data\_read/writes

show status like 'innodb\_log\_waits';

### MySQL Workbench



Key Efficiency

<https://gxnotes.com/article/72479.html>