**数据库学习**

# Mysql存储引擎（storage engines）：

由于开源，有很多其他的开源团体对其贡献，mysql的存储引擎很丰富。

存储引擎主要涉及到存储算法和IO操作方式。

1. ISAM

查詢远大于更新，不占用大量的内存和储存资源。不支持事务，不能容错。使用ISAM必须经常备份事实数据

1. MyISAM

ISAM的升级版，使用表格锁定优化并发读写操作，需要经常运行optimize table来恢复更新浪费的空间，经常需要做实时备份。快速读操作，写操作效率低。重大缺陷：不能在表格损坏后恢复数据。不支持事务。数据越多，写操作越低，要同时维护数据和索引信息，索引越多，相对效率越低。MyISAM是非聚集索引，支持全文索引。

1. InnoDB（当前默认用的就是innodb，重点）

MYSQL++API实现，支持事务（提交、回滚、崩溃恢复）和外键foreign key，查询性能比前两种引擎低，5.5以上使用。Sql查询中可以自由的将Innodb类型的表与其他mysql表类型混合起来，在同一查询中也可混合。5.7版本中的Innodb存储引擎管理的数据文件为两个，frm、idb。Innodb是聚集索引，主键不应过大，不支持全文索引，避免全表扫描。

特点：

支持事务

数据多版本读取

锁定机制改进，使用行锁设计

支持外键

最有效地利用以及使用内存和CPU

提供一直性非锁定读。

1. Memory存储引擎

内存存储引擎，将数据放在内存中，仅在磁盘上存放表结构相关信息，不支持事务、支持索引

1. NDBCluster存储引擎：分布式集群环境中使用
2. Merge存储引擎

# 相关命令

查看数据库支持的引擎：

Show engines

查看当前数据库引擎

Show variables likes ‘%storage\_engine%’

可在mysql配置文件中找到 window下my.ini，linux下my.cnf

查看建表详情

Show create table table\_name

修改表的存储引擎

Alter table table\_name engine=engine\_name

修改默认的存储引擎

在配置文件中修改

default-storage-engine=INNODB

# mysql索引

MySQL索引的建立对于MySQL的高效运行是很重要的，索引可以大大提高MySQL的检索速度。索引的作用是便捷化检索表中行和列的子集，而不需要检查表中的每一行。索引分单列索引和组合索引。单列索引，即一个索引只包含单个列，一个表可以有多个单列索引，但这不是组合索引。组合索引，即一个索引包含多个列。索引的目的是提高查詢效率。

索引的优点：

1. 通过创建唯一性索引，可以保证数据库中每一行数据的唯一性（主键）。
2. 可以大大加快数据检索数据，创建索引的主要原因。
3. 可以加快表和表的连接，特别是在实现数据的参看完整性有特别意义。
4. 使用分组排序字句进行数据检索时，能减少分组排序时间。
5. 通过索引能在查询过程中使用查询优化器，提高系统性能。

缺点

1. 创建和维护索引需要耗时，这种时间随着数据量增加而增加。
2. 索引占用物理空间，如果建立聚簇簇索引，那么需要的空间更大。
3. 当对表中的数据进行增加，删除和修改时，索引需要动态维护，降低了索引的维护速度

## 索引创建：

alter table department add index dept\_name\_idx(name);

create index indexName on mytable(username(length));

创建索引需要注意的点：

1. 在经常需要搜索的列上加索引会加快搜索速度
2. 在作为主键的列上，强制该列的唯一性和组织表中数据的排列结构
3. 在经常用在连接的列上，这些列主要是一些外键，可以加快连接速度
4. 在经常需要根据范围进行搜索的列上创建索引，因为索引已经排序，指定范围是连续的能加快搜索速度。
5. 在经常需要排序的列上创建索引，因为索引已经排序，这样查询可以利用缩影的排序
6. 在经常使用where字句中的列上创建缩影，加快条件判断。

对于那些查询中很多使用或者参考的列不应该建立缩影

对于那些只有很少数据值（比如性别，只有男和女）的列不应该建立索引

对于定义为text，image和bit数据类型的列不应该建立索引，这些列数据要么数据量大，要么取值少。

当修改性能远大于检索性能时，不应该建立索引。

## Mysql索引种类

1.B-tree索引（平衡搜索树）

按照blance tree的数据结构来存储，优点是减少定位记录所经历的中间过程，加快取值。

b-tree中每个节点包含：

1. 本节点所含的关键字的个数（主键）
2. 指向父节点的指针
3. 关键字
4. 子节点的指针

对于一个m阶的b-tree每个节点最多有m个子节点，各节点的关键字和可以拥有的自己子节点的个数都有限制，规定m阶的b-tree中，根节点至少两个，除非根节点为叶子节点。根节点的关键字个数为1~m-1个。非根节点至少有[m/2](向上取整)

特点：

1. 关键字的集合分布在整棵树中
2. 任何一个关键字出现且只出现在一个节点中
3. 搜索可有在非叶子节点结束
4. 搜索新能等价于关键字在全集内的一次二分查找
5. 自动层次控制

2.full-text索引

全文索引，存储结构也是b-tree，主要为了解决”XXXX%”在需要like查询的低效问题

## Mysq索引管理

1. 普通索引

默认使用btree索引，建议规定字段缩影长度

1. 查看索引

show index from table\_name

show keys from table\_name (在mysql中可用)

1. 删除索引

drop index index\_name on table\_name

alter table table\_name drop index index\_name

alter table table\_name drop primary key

1. 唯一索引

与普通索引类似，不同的是索引列的值必须唯一，允许为空（主键不允许为空），如果是组合索引，列值组合必须唯一。

1. 全文索引（fulltext）,仅适合MyISAM表，仅可从CHAR、varchar、text中列中作为语句一部分被创建
2. 组合索引：将一张表中多个列作为索引

create index index\_name table\_name(column\_1(length),column\_2(length))

## 索引优化

1. 索引不会包含null值得列，null值列不会包含在索引中，组合索引中只要有一列含有null值，那么这一列对于组合索引就是无效的。设计数据库时不要让字段默认值为null。
2. 尽量使用短索引，提高查询速度，节省磁盘空间

create index index\_name on table\_name(column(length))

1. 索引列排序，尽量不要使用order by排序操作（避免全表扫描），避免使用多个列排序。
2. 避免使用Like操作，不要使用‘%XXX%’(不会用索引)，使用‘XXX%（会用到索引）
3. 不要在列上进行计算

Mysql只对一下操作符才会使用索引：<,<=,>=,>between,in以及某些like(不以%开头的模糊匹配)，建议一张表的索引数量不超过6个。

SQL优化方向：执行计划、索引、SQL改写。

1. 避免全表扫描
2. 避免判断null值
3. 避免不等值判断
4. 避免使用or逻辑（可以使用union all）
5. 慎用in 和not in
6. 注意模糊查询（like 中不要使用%XXX%）
7. 避免查询条件中的字段计算
8. 避免查询条件中对字段进行函数操作
9. where字句左边不要进行函数算数运算或其他表达式运算，否则系统无法正确使用索引
10. 合理使用组合索引
11. 不要定义无意义的查询（where 1=1）
12. 使用exists代替in
13. 索引中有大量重复数据时索引可能会失效
14. 表中字段类型合理选择
15. 查询语法中避免使用\*，不要返回用不到的数据
16. 程序中避免大的事务操作，提高系统并发能力。

主键与唯一索引：

1. 主键一定是唯一性索引，唯一性索引并不一定就是主键。数据库表中只允许存在一个主键，但允许存在多个索引。
2. 一个表中可以有多个唯一性索引，但只能有一个主键。
3. 主键列不允许空值，而唯一性索引列允许空值。
4. 索引可以提高查询的速度。

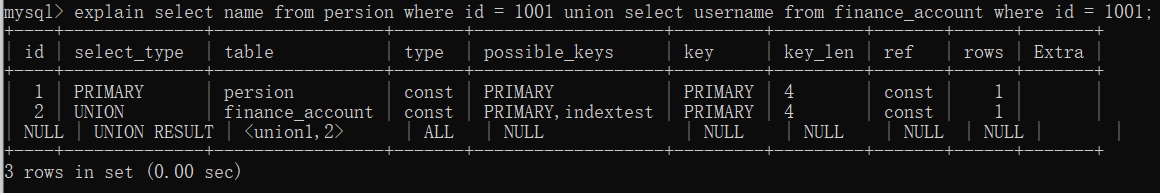
# 执行计划（explain）

当我们的系统上线后数据库的记录不断增加，之前写的一些SQL语句或者一些ORM操作效率变得非常低。我们不得不考虑SQL优化，SQL优化大概是这样一个流程：1.定位执行效率低的SQL语句（定位）,2.分析为什么这段SQL执行的效率比较低（分析），3.最后根据第二步分析的结构采取优化措施（解决）。EXPLAIN命令的作用就是帮助我们分析SQL的执行情况，属于第二步，EXPLAIN命令是查看查询优化器如何决定执行查询的主要的方法。学会解释EXPLAIN将帮助我们了解SQL优化器是如何工作的。执行计划可以告诉我们SQL如何使用索引，连接查询的执行顺序，查询的数据行数。

用法：

要使用EXPLAIN（执行计划）,只需要在查询的SELECT关键字之前增加EXPLAIN这个词。

输出结果分析：



1. id：包含一组数字，表示查询中执行select子句或操作表的顺序。id相同，执行顺序由上至下，如果是子查询，id的序号会递增，id值越大优先级越高，越先被执行。id如果相同，可以认为是一组，从上往下顺序执行；在所有组中，id值越大，优先级越高，越先执行
2. select\_type: 查询中每个select子句的类型

**a. SIMPLE：查询中不包含子查询或者UNION**  
**b. 查询中若包含任何复杂的子部分，最外层查询则被标记为：PRIMARY**  
**c. 在SELECT或WHERE列表中包含了子查询，该子查询被标记为：SUBQUERY**  
**d. 在FROM列表中包含的子查询被标记为：DERIVED（衍生）用来表示包含在from子句中的子查询的select，mysql会递归执行并将结果放到一个临时表中。服务器内部称为"派生表"，因为该临时表是从子查询中派生出来的**  
**e. 若第二个SELECT出现在UNION之后，则被标记为UNION；若UNION包含在FROM子句的子查询中，外层SELECT将被标记为：DERIVED**  
**f. 从UNION表获取结果的SELECT被标记为：UNION RESULT**

1. type: **表示MySQL在表中找到所需行的方式，又称“访问类型”,常见类型：**
2. ALL **Full Table Scan， MySQL将遍历全表以找到匹配的行**
3. **index：Full Index Scan，index与ALL区别为index类型只遍历索引树**
4. **range:索引范围扫描，对索引的扫描开始于某一点，返回匹配值域的行。显而易见的索引范围扫描是带有between或者where子句里带有<, >查询。当mysql使用索引去查找一系列值时，例如IN()和OR列表，也会显示range（范围扫描）,当然性能上面是有差异的。**
5. **ref：使用非唯一索引扫描或者唯一索引的前缀扫描，返回匹配某个单独值的记录行**
6. **eq\_ref：类似ref，区别就在使用的索引是唯一索引，对于每个索引键值，表中只有一条记录匹配，简单来说，就是多表连接中使用primary key或者 unique key作为关联条件**
7. **const、system：当MySQL对查询某部分进行优化，并转换为一个常量时，使用这些类型访问。如将主键置于where列表中，MySQL就能将该查询转换为一个常量**
8. **NULL：MySQL在优化过程中分解语句，执行时甚至不用访问表或索引，例如从一个索引列里选取最小值可以通过单独索引查找完成。**
9. possible\_key: **指出MySQL能使用哪个索引在表中找到记录，查询涉及到的字段上若存在索引，则该索引将被列出，但不一定被查询使用**
10. **key:** **MySQL在查询中实际使用的索引，若没有使用索引，显示为NULL**
11. **key\_len:** **表示索引中使用的字节数，可通过该列计算查询中使用的索引的长度（key\_len显示的值为索引字段的最大可能长度，并非实际使用长度，即key\_len是根据表定义计算而得，不是通过表内检索出的）**
12. **ref:** **表示上述表的连接匹配条件，即哪些列或常量被用于查找索引列上的值**
13. **row:** **表示MySQL根据表统计信息及索引选用情况，估算的找到所需的记录所需要读取的行数**
14. **extra:** **包含不适合在其他列中显示但十分重要的额外信息**

在sql优化中重点关注3，4，5列。

Sql改写需要注意的地方：

1.在表中建立索引，优先考虑where.group by使用到的字段。

2.查询条件中，一定不要使用select \*，因为会返回过多无用的字段会降低查询效率。应该使用具体的字段代替\*，只返回使用到的字段。

3.不要在where条件中使用左右两边都是%的like模糊查询，如：

SELECT \* FROM t\_order WHERE customer LIKE '%zhang%'

这样会导致数据库引擎放弃索引进行全表扫描。

优化：尽量在字段后面使用模糊查询。如下：

SELECT \* FROM t\_order WHERE customer LIKE 'zhang%'

4.尽量不要使用in 和not in，会造成全表扫描。如下：

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

SELECT \* FROM t\_order1 WHERE customer IN (SELECT customer FROM t\_order2)

优化：

对于连续的数值，能用 between 就不要用 in ，如下：SELECT \* FROM t\_order WHERE id BETWEEN 2 AND 3

对于子查询，可以用exists代替。如下：SELECT \* FROM t\_order1 WHERE EXISTS (SELECT \* FROM t\_order2 WHERE t1.customer = t2.customer)

5.尽量不要使用or，会造成全表扫描。如下：

SELECT \* FROM t\_order WHERE id = 1 OR id = 3

优化：可以用union代替or。如下：

SELECT \* FROM t\_order WHERE id = 1

UNION

SELECT \* FROM t\_order WHERE id = 3

6.尽量不要在 where 子句中对字段进行表达式操作，这样也会造成全表扫描。如：

select id FROM t\_order where num/2=100

应改为:

select id FROM t\_order where num=100\*2

7.where条件里尽量不要进行null值的判断，null的判断也会造成全表扫描。如下：

SELECT \* FROM t\_order WHERE score IS NULL

优化：

给字段添加默认值，对默认值进行判断。如：

SELECT \* FROM t\_order WHERE score = 0

8.尽量不要在where条件中等号的左侧进行表达式.函数操作，会导致全表扫描。如下：

SELECT \* FROM t\_order2 WHERE score/10 = 10

SELECT \* FROM t\_order2 WHERE SUBSTR(customer,1,5) = 'zhang'

优化：

将表达式.函数操作移动到等号右侧。如下：

SELECT \* FROM t\_order2 WHERE score = 10\*10

SELECT \* FROM t\_order2 WHERE customer LIKE 'zhang%'

9.尽量不要使用where 1=1的条件

有时候，在开发过程中，为了方便拼装查询条件，我们会加上该条件，这样，会造成进行全表扫描。如下：

SELECT \* FROM t\_order WHERE 1=1

优化：

如果用代码拼装sql，则由代码进行判断，没where加where，有where加and

如果用mybatis，请用mybatis的where语法。

10.程序要尽量避免大事务操作，提高系统并发能力。

11.一个表的索引数最好不要超过6个，如果索引太多的话，就需要考虑一下那些不常使用到的列上建的索引是否有必要。

架构优化方向：高可用架构、高性能架构、分库分表。

# 数据库分库分表

作用：缓解数据压力，提高查询效率，读写分离

mysql读写分离实现：mysql proxy/atlas/tddl，减小主库读的压力。

存在的问题：数据同步存在延迟，主库宕机从库无法工作。

## 分库类型：

垂直

当表中字段较多时，将表中的字段拆分到不同的表中。表（或库）的结构不一样，但都共有的一个字段（一般是主键）

水平

当表中数据行数很多时，将不同的行按一定的策略拆分到不同的表中。每个表（或库）的结构都一样，但数据都不一样。

二者相同点：每个表或库的并集是整个应用的全量数据。

分库策略：

1. hash：对表中的某个字段进行哈希
2. range：通过数据的范围进行分库分表，例如将id=0到10000的数据放在某个库
3. list：预定义，根据未来数据量的预估来分析该分多少张表多少个库。

分库分表存在的问题：

1. 未实现高可用（HA）
2. 分布式事务问题
3. 分布式全局唯一性id（不能使用自增）
4. sql改写问题
5. 数据异构问题（如果后台需要统计数据，由于不确定数据放在哪里，需要查询所有库中的表），使用搜索引擎来解决