# MySQL知识点

<https://mp.weixin.qq.com/s/HMvnpvRRq8VTTxV6im_08g>

#### 为什么要优化：

1. 系统的吞吐量瓶颈往往出现在数据库的访问速度上
2. 随着应用程序的运行，数据库中的数据会越来越多，处理时间会相应变慢
3. 数据存放在磁盘上，读写速度无法和内存相比

#### 如何优化：

1. 设计数据库时：数据库表、字段的设计，存储引擎的选择。
2. 利用好MySQL自身提供的功能，如索引等
3. 横向扩展：MySQL集群，负载均衡，读写分离
4. SQL语句优化（效果甚微）

**字段设计原则：**

1. 尽量用整型表示字符串
2. 定长与非定长数据的选择
3. 尽可能选择小的数据类型和指定短的长度
4. 尽可能使用not null
5. 单表字段不宜过多（极限20-30）
6. 可以预留字段

**引擎选择：**默认使用InnoDB

如果没有特别的需求，使用默认的Innodb即可。

MyISAM：以读写插入为主的应用程序，比如博客系统、新闻门户网站。

Innodb：更新（删除）操作频率也高，或者要保证数据的完整性；并发量高，支持事务和外键保证数据完整性。比如OA自动化办公系统。

**范式：**

第一范式：字段原子性（保证每个字段是不可以再分的）

第二范式：每个表要有唯一主键

第三范式：消除依赖传递

**索引：**

索引为什么那么快：1、关键字对应数据本身，数据量小。2、关键字是有序的，二分查找速度更快。

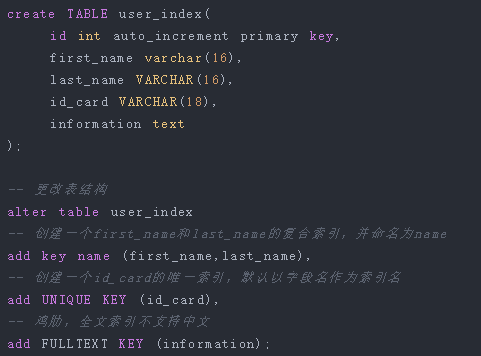
MySQL索引类型：普通索引（key），唯一索引（unique key），主键索引（primary key），全文索引（fulltext key）

普通索引：对关键字没有限制

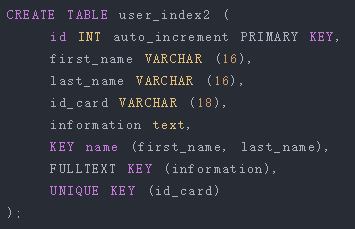
唯一索引：要求记录提供的关键字不能重复（字段内容不能有重复）

主键索引：要求关键字唯一且不为null（如主键id）

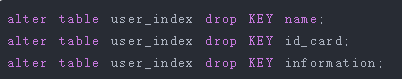
索引创建方式：



或

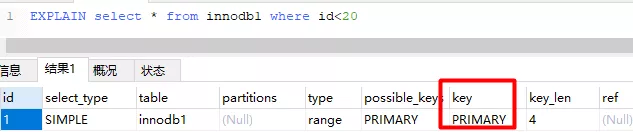


删除索引：



执行计划explain：

我们可以通过explain selelct来分析SQL语句执行前的执行计划：



由上图可看出此SQL语句是按照主键索引来检索的。

执行计划是：当执行SQL语句时，首先会分析、优化，形成执行计划，在按照执行计划执行。

**索引使用场景，如何更好创建索引：**

**1、where之后经常被用来当做条件的字段可以添加索引**

**2、order by 因为其本身就是排序，如在该字段上添加索引就不需要取出所有数据再进行排序可以直接索引对应的数据。**

**3、join 对join语句匹配关系（on）涉及的字段建立索引能够提高效率**

**4、索引覆盖：**

如果一个索引包含（或者说覆盖）所有需要查询的字段的值，如：

Index索引index（first\_name,last\_name,date）且刚好select查询刚好包括索引所 有的字段如select first\_name,last\_name,date from...时，此时select的数据列只用从 索引中就能够取得，不必从表中读取数据行，称为索引覆盖。她减少了从表中读到 内存的IO操作从而可以提升效率。

在EXPLAIN的Extra列可以看到“Using index”的信息

**索引语法细节：**

1. **字段要独立出现**

**比如下面两条SQL语句在语义上相同，但是第一条会使用主键索引而第二条不会。**

select \* from user where id = 20-1;

select \* from user where id+1 = 20;

1. **like 查询，不能以通配符开头 如下第一条不触发第二条出发：**

select \* from article where title like '%mysql%';

select \* from article where title like 'mysql%';

1. **复合索引只对第一个字段有效：**

**建立符合索引:**

alter table person add index(first\_name,last\_name);

其原理是将索引先按照first\_name中提取的关键字排序，如果无法确定再按照last\_name提取关键字排序。

其使用的场景如下：

比如对于select \* person from first\_name = ? and last\_name = ?，复合索引就比对first\_name和last\_name单独建立索引要高效些。很好理解，复合索引首先二分查找与first\_name = ?匹配的记录，再在这些记录中二分查找与last\_name匹配的记录，只涉及到一张索引表。而分别单独建立索引则是在first\_name索引表中二分找出与first\_name = ?匹配的记录，再在last\_name索引表中二分找出与last\_name = ?的记录，两者取交集

1. **使用or时只要一边没有索引就会进行全表检索**
2. **状态值时不要使用索引因重复字段太多**

#### 索引的存储结构

1. BTree

btree（多路平衡查找树）是一种广泛应用于磁盘上实现索引功能的一种数据结构，也是大多数数据库索引表的实现。

1. B+Tree聚簇结构

聚簇结构（也是在BTree上升级改造的）中，关键字和记录是存放在一起的。

在MySQL中，仅仅只有Innodb的主键索引为聚簇结构，其它的索引包括Innodb的非主键索引都是典型的BTree结构

1. 哈希索引

在索引被载入内存时，使用哈希结构来存储。

#### 数据库事务

**四个特征：**

1. **原子性（atomicity）：**整个数据库事务是不可分割的单位，即要么都做要么都不做。就如操作一笔数据要么一起成功要么一起失败。
2. **一致性（consistency）：**指事务将数据库从一种状态变为另一种一致的状态。也就是说事务执行之前和执行之后必须是处于一致状态。如两个账户转账不管怎么相互转最后的总和是不会变的。
3. **隔离性（isolation）：**也称并发控制或者串行化、锁等。指并发多个事务时只要有一个事务在进行，其它事务必须在该事务执行完成后才会执行。如T1和T2两个事务，在T1看来，T2要么在T1开始前已经结束了，要么T1结束才开始，事务之间是不可见的。
4. **持久性（durability）：**指事务一旦提交，其结果就是永久性的。

**事务隔离级别：**

**事务并发问题：**

1. **脏读：**a事务读取到b事务未提交的数据，然后b事务回滚导致a事务读到的数据为 脏数据。
2. **不可重复读：**事务a多次读取同一条数据（一个事务范围内多次读取），事务b在事务a多次读取过程中，对数据进行了更新并提交，导致事务a多次读取数据时，结果不一致。
3. **幻读（虚读）：系统管理员A将数据库中所有学生的成绩从具体分数改为ABCDE等级，但是系统管理员B就在这个时候插入了一条具体分数的记录，当系统管理员A改结束后发现还有一条记录没有改过来，就好像发生了幻觉一样，这就叫幻读。**
4. **总结：不可重复读的和幻读很容易混淆，不可重复读侧重于修改，幻读侧重于新增或删除。解决不可重复读的问题只需锁住满足条件的行，解决幻读需要锁表。**

**隔离级别表：**

|  |  |  |  |
| --- | --- | --- | --- |
| **隔离级别** | **脏读** | **不可重复读** | **幻读** |
| **（未提交读）**  **Read Uncommitted** | **是** | **是** | **是** |
| **（读已提交）**  **Read Committed** | **否** | **是** | **是** |
| **（可重复读）Repeatable Read** | **否** | **否** | **是** |
| **（串行化）Serializable** | **否** | **否** | **否** |