MySql语句练习：<https://blog.csdn.net/u012467492/article/details/46790205>

<https://blog.csdn.net/flycat296/article/details/63681089>

Mysql优化：<https://blog.csdn.net/samjustin1/article/details/52314813>

分库分表：<https://www.cnblogs.com/bluebluesky/articles/6413831.html>

索引注意事项：<https://www.cnblogs.com/heyonggang/p/6610526.html>

C3p0 , dbcp , druid数据库连接池：<https://blog.csdn.net/wawa3338/article/details/81380662>

jdbc的流程

加载数据库驱动Class.forname

建立数据库连接DuriveManager.getConnection

获取传输器对象Statement

通过传输器对象传输sql语句或者结果集对象result

关闭连接等资源

## 主键 超键 候选键 外键

主 键：数据库表中对储存数据对象予以唯一和完整标识的数据列或属性的组合。一个数据列只能有一个主键，且主键的取值不能缺失，即不能为空值（Null）。

超 键：在关系中能唯一标识元组的属性集称为关系模式的超键。一个属性可以为作为一超键，多个属性组合在一起也可以作为一个超键。超键包含候选键和主键。

候选键：是最小超键，即没有冗余元素的超键。

外 键：在一个表中存在的另一个表的主键称此表的外键。

### 数据库事务的四个特性及含义

数据库事务transanction正确执行的四个基本要素。ACID,原子性(Atomicity)、一致性(Correspondence)、隔离性(Isolation)、持久性(Durability)。

原子性:整个事务中的所有操作，要么全部完成，要么全部不完成，不可能停滞在中间某个环节。事务在执行过程中发生错误，会被回滚（Rollback）到事务开始前的状态，就像这个事务从来没有执行过一样。

一致性:在事务开始之前和事务结束以后，数据库的完整性约束没有被破坏。

隔离性:隔离状态执行事务，使它们好像是系统在给定时间内执行的唯一操作。如果有两个事务，运行在相同的时间内，执行 相同的功能，事务的隔离性将确保每一事务在系统中认为只有该事务在使用系统。这种属性有时称为串行化，为了防止事务操作间的混淆，必须串行化或序列化请 求，使得在同一时间仅有一个请求用于同一数据。

持久性:在事务完成以后，该事务所对数据库所作的更改便持久的保存在数据库之中，并不会被回滚。

### MySQL常用的引擎主要就是2个：Innodb和MyIASM。

**Innodb**引擎，Innodb引擎提供了对数据库ACID事务的支持。并且还提供了行级锁和外键的约束。它的设计的目标就是处理大数据容量的数据库系统。它本身实际上是基于Mysql后台的完整的系统。Mysql运行的时候，Innodb会在内存中建立缓冲池，用于缓冲数据和索引。但是，该引擎是不支持全文搜索的。同时，启动也比较的慢，它是不会保存表的行数的。当进行Select count(\*) from table指令的时候，需要进行扫描全表。所以当需要使用数据库的事务时，该引擎就是首选。由于锁的粒度小，写操作是不会锁定全表的。所以在并发度较高的场景下使用会提升效率的。

**MyIASM**引擎，它是MySql的默认引擎，但不提供事务的支持，也不支持行级锁和外键。因此当执行Insert插入和Update更新语句时，即执行写操作的时候需要锁定这个表。所以会导致效率会降低。不过和Innodb不同的是，MyIASM引擎是保存了表的行数，于是当进行Select count(\*) from table语句时，可以直接的读取已经保存的值而不需要进行扫描全表。所以，如果表的读操作远远多于写操作时，并且不需要事务的支持的。可以将MyIASM作为数据库引擎的首先。

大容量的数据集时趋向于选择Innodb。因为它支持事务处理和故障的恢复。Innodb可以利用数据日志来进行数据的恢复。主键的查询在Innodb也是比较快的。

大批量的插入语句时（这里是INSERT语句）在MyIASM引擎中执行的比较的快，但是UPDATE语句在Innodb下执行的会比较的快，尤其是在并发量大的时候。

#### 两种引擎所使用的索引的数据结构是什么？

都是B+树!

MyIASM引擎，B+树的数据结构中存储的内容实际上是实际数据的地址值。也就是说它的索引和实际数据是分开的，只不过使用索引指向了实际数据。这种索引的模式被称为非聚集索引。

Innodb引擎的索引的数据结构也是B+树，只不过数据结构中存储的都是实际的数据，这种索引有被称为聚集索引。

### 索引

索引的目的在于提高查询效率，与我们查阅图书所用的目录是一个道理：先定位到章，然后定位到该章下的一个小节，然后找到页数。相似的例子还有：查字典，查火车车次，飞机航班等

本质都是：通过不断地缩小想要获取数据的范围来筛选出最终想要的结果，同时把随机的事件变成顺序的事件，也就是说，有了这种索引机制，我们可以总是用同一种查找方式来锁定数据。

mysql的索引分为单列索引(主键索引,唯一索引,普通索引)和组合索引.

单列索引：一个索引只包含一个列,一个表可以有多个单列索引.

普通索引：这个是最基本的索引,

唯一索引：与普通索引类似,但是不同的是唯一索引要求所有的类的值是唯一的,这一点和主键索引一样，但是他允许有空值。

主键索引：不允许有空值,(在B+TREE中的InnoDB引擎中,主键索引起到了至关重要的地位)

主键索引建立的规则是 int优于varchar，一般在建表的时候创建,最好是与表的其他字段不相关的列或者是业务不相关的列.一般会设为 int 而且是 AUTO\_INCREMENT自增类型的

组合索引：一个组合索引包含两个或两个以上的列, 一个表中含有多个单列索引不代表是组合索引,通俗一点讲 组合索引是:包含多个字段但是只有索引名称

#### 索引的最左匹配特性（即从左往右匹配）

当b+树的数据项是复合的数据结构，比如(name,age,sex)的时候，b+数是按照从左到右的顺序来建立搜索树的，比如当(张三,20,F)这样的数据来检索的时候，b+树会优先比较name来确定下一步的所搜方向，如果name相同再依次比较age和sex，最后得到检索的数据；但当(20,F)这样的没有name的数据来的时候，b+树就不知道下一步该查哪个节点，因为建立搜索树的时候name就是第一个比较因子，必须要先根据name来搜索才能知道下一步去哪里查询。比如当(张三,F)这样的数据来检索时，b+树可以用name来指定搜索方向，但下一个字段age的缺失，所以只能把名字等于张三的数据都找到，然后再匹配性别是F的数据了， 这个是非常重要的性质，即索引的最左匹配特性。

mysql的索引是最左前缀原则，最左边的索引先匹配，后面的可以不用匹配，像你这个索引index(key1,key2)相当于包括key1单独索引。

#### 索引类型hash与btree

hash类型的索引：查询单条快，范围查询慢

btree类型的索引：b+树，层数越多，数据量指数级增长（我们就用它，因为innodb默认支持它）

#不同的存储引擎支持的索引类型也不一样

InnoDB 支持事务，支持行级别锁定，支持 B-tree、Full-text 等索引，不支持 Hash 索引；

MyISAM 不支持事务，支持表级别锁定，支持 B-tree、Full-text 等索引，不支持 Hash 索引；

Memory 不支持事务，支持表级别锁定，支持 B-tree、Hash 等索引，不支持 Full-text 索引；

NDB 支持事务，支持行级别锁定，支持 Hash 索引，不支持 B-tree、Full-text 等索引；

Archive 不支持事务，支持表级别锁定，不支持 B-tree、Hash、Full-text 等索引；

#### 使用索引的优点

1.可以通过建立唯一索引或者主键索引,保证数据库表中每一行数据的唯一性.

2.建立索引可以大大提高检索的数据,以及减少表的检索行数

3.在表连接的连接条件 可以加速表与表直接的相连

4.在分组和排序字句进行数据检索,可以减少查询时间中 分组 和 排序时所消耗的时间(数据库的记录会重新排序)

5.建立索引,在查询中使用索引 可以提高性能

#### 使用索引的缺点

1.在创建索引和维护索引 会耗费时间,随着数据量的增加而增加

2.索引文件会占用物理空间,除了数据表需要占用物理空间之外,每一个索引还会占用一定的物理空间

3.当对表的数据进行 INSERT,UPDATE,DELETE 的时候,索引也要动态的维护,这样就会降低数据的维护速度,(建立索引会占用磁盘空间的索引文件。一般情况这个问题不太严重，但如果你在一个大表上创建了多种组合索引，索引文件的会膨胀很快)。

#### 使用索引需要注意的地方

1.在经常需要搜索的列上,可以加快索引的速度

2.主键列上可以确保列的唯一性

3.在表与表的而连接条件上加上索引,可以加快连接查询的速度

4.在经常需要排序(order by),分组(group by)和的distinct 列上加索引 可以加快排序查询的时间, (单独order by 用不了索引，索引考虑加where 或加limit)

5.在一些where 之后的 < <= > >= BETWEEN IN 以及某个情况下的like 建立字段的索引(B-TREE)

6.like语句的 如果你对nickname字段建立了一个索引.当查询的时候的语句是 nickname lick '%ABC%' 那么这个索引讲不会起到作用.而nickname lick 'ABC%' 那么将可以用到索引

7.索引不会包含NULL列,如果列中包含NULL值都将不会被包含在索引中,复合索引中如果有一列含有NULL值那么这个组合索引都将失效,一般需要给默认值0或者 ' '字符串

8.使用短索引,如果你的一个字段是Char(32)或者int(32),在创建索引的时候指定前缀长度 比如前10个字符 (前提是多数值是唯一的..)那么短索引可以提高查询速度,并且可以减少磁盘的空间,也可以减少I/0操作.

9.不要在列上进行运算,这样会使得mysql索引失效,也会进行全表扫描

10.选择越小的数据类型越好,因为通常越小的数据类型通常在磁盘,内存,cpu,缓存中 占用的空间很少,处理起来更快

#### 什么情况下不创建索引

1.查询中很少使用到的列 不应该创建索引,如果建立了索引然而还会降低mysql的性能和增大了空间需求.

2.很少数据的列也不应该建立索引,比如 一个性别字段 0或者1,在查询中,结果集的数据占了表中数据行的比例比较大,mysql需要扫描的行数很多,增加索引,并不能提高效率

3.定义为text和image和bit数据类型的列不应该增加索引,

4.当表的修改(UPDATE,INSERT,DELETE)操作远远大于检索(SELECT)操作时不应该创建索引,这两个操作是互斥的关系

#### 事务总结:

事务的特性: ACID

原子性:事务里面的操作单元不可切割,要么全部成功,要么全部失败

一致性:事务执行前后,业务状态和其他业务状态保持一致.

隔离性:一个事务执行的时候最好不要受到其他事务的影响

持久性:一旦事务提交或者回滚.这个状态都要持久化到数据库中

#### 不考虑隔离性会出现的读问题

脏读:在一个事务中读取到另一个事务没有提交的数据

不可重复读:在一个事务中,两次查询的结果不一致(针对的update操作)

虚读(幻读):在一个事务中,两次查询的结果不一致(针对的insert操作)

#### 事务隔离级别分为四种（级别递减）：

1、Serializable （串行化）：最严格的级别，事务串行执行，资源消耗最大；

2、REPEATABLE READ（重复读） ：保证了一个事务不会修改已经由另一个事务读取但未提交（回滚）的数据。避免了“脏读取”和“不可重复读取”的情况，但不能避免“幻读”，但是带来了更多的性能损失。

3、READ COMMITTED （提交读）：大多数主流数据库的默认事务等级，保证了一个事务不会读到另一个并行事务已修改但未提交的数据，避免了“脏读取”，但不能避免“幻读”和“不可重复读取”。该级别适用于大多数系统。

4、Read Uncommitted（未提交读） ：事务中的修改，即使没有提交，其他事务也可以看得到，会导致“脏读”、“幻读”和“不可重复读取”。

数据库锁

出现的目的：处理并发问题

并发控制的主要采用的技术手段：乐观锁、悲观锁和时间戳。

锁分类

从数据库系统角度分为三种：排他锁、共享锁、更新锁。

从程序员角度分为两种：一种是悲观锁，一种乐观锁

#### 悲观锁（Pessimistic Lock）

每次去拿数据的时候都认为别人会修改，所以每次在拿数据的时候都会上锁，这样别人拿这个数据就会block（阻塞），直到它拿锁。

传统的关系数据库里用到了很多这种锁机制，比如行锁、表锁、读锁、写锁等，都是在操作之前先上锁。

# 悲观锁按使用性质划分

#### 共享锁（Share Lock）

S锁，也叫读锁，用于所有的只读数据操作。共享锁是非独占的，允许多个并发事务读取其锁定的资源。

性质

1. 多个事务可封锁同一个共享页；

2. 任何事务都不能修改该页；

3. 通常是该页被读取完毕，S锁立即被释放。

#### 排他锁（Exclusive Lock）

X锁，也叫写锁，表示对数据进行写操作。如果一个事务对对象加了排他锁，其他事务就不能再给它加任何锁了。（某个顾客把试衣间从里面反锁了，其他顾客想要使用这个试衣间，就只有等待锁从里面打开了。）

性质

1. 仅允许一个事务封锁此页；

2. 其他任何事务必须等到X锁被释放才能对该页进行访问；

3. X锁一直到事务结束才能被释放。

#### 更新锁

U锁，在修改操作的初始化阶段用来锁定可能要被修改的资源，这样可以避免使用共享锁造成的死锁现象。

性质

1. 用来预定要对此页施加X锁，它允许其他事务读，但不允许再施加U锁或X锁；

2. 当被读取的页要被更新时，则升级为X锁；

3. U锁一直到事务结束时才能被释放。

行锁：锁的作用范围是行级别。

表锁：锁的作用范围是整张表。

举个例子，一个用户表user，有主键id和用户生日birthday。

当你使用update … where id=?这样的语句时，数据库明确知道会影响哪一行，它就会使用行锁；

当你使用update … where birthday=?这样的的语句时，因为事先不知道会影响哪些行就可能会使用表锁。

# 乐观锁（Optimistic Lock）

顾名思义，就是很乐观，每次去拿数据的时候都认为别人不会修改，所以，不会上锁。但是在更新的时候会判断一下在此期间别人有没有更新这个数据，可以使用版本号等机制。

数据版本：为数据增加一个版本标识，在基于数据库表的版本解决方案中，一般是通过为数据库表增加一个 “version” 字段来实现。读取出数据时，将此版本号一同读出，之后更新时，对此版本号加一。此时，将提交数据的版本数据与数据库表对应记录的当前版本信息进行比对，如果提交的数据版本号大于数据库表当前版本号，则予以更新，否则认为是过期数据。

#### 并发控制会造成两种锁

活锁

定义：指的是T1封锁了数据R，T2同时也请求封锁数据R，T3也请求封锁数据R，当T1释放了锁之后，T3会锁住R，T4也请求封锁R，则T2就会一直等待下去。

解决方法：采用“先来先服务”策略可以避免。

死锁

定义：就是我等你，你又等我，双方就会一直等待下去。比如：T1封锁了数据R1，正请求对R2封锁，而T2封住了R2,正请求封锁R1，这样就会导致死锁，死锁这种没有完全解决的方法，只能尽量预防。

预防方法：

1. 一次封锁法，指的是一次性把所需要的数据全部封锁住，但是这样会扩大了封锁的范围，降低系统的并发度；

2. 顺序封锁法，指的是事先对数据对象指定一个封锁顺序，要对数据进行封锁，只能按照规定的顺序来封锁，但是这个一般不大可能的。

系统判定死锁的方法：

超时法：如果某个事物的等待时间超过指定时限，则判定为出现死锁；

等待图法：如果事务等待图中出现了回路，则判断出现了死锁。

对于解决死锁的方法，只能是撤销一个处理死锁代价最小的事务，释放此事务持有的所有锁，同时对撤销的事务所执行的数据修改操作必须加以恢复。