**MySQL面试题**

1. **MySQL三范式**
   1. 一范式：有主键，列不可再分；
   2. 二范式：列依赖全部主键；
   3. 三范式：非主键互不依赖
2. **执行顺序：from->join->where->group by->select->order by->limit**

先是from和join把要查询的表迪卡儿乘积变成虚表->用where对上述虚表进行筛选->筛选之后用group by进行分组->再用having对分组之后进行聚合筛选->select选择要返回的单列->order by对返回内容进行排序->limit对排序之后的内容限制行数后再显示

1. **group by-分组查询**

分组查询的字段一般为聚合函数和分组字段，查询其他字段无任何意义。

按多段分组记录条数就是分组内所有种类的笛卡尔乘积

1. **where与having区别：**
2. 执行时机不同：where是分组之前进行过滤，不满足where条件，不参与分组；而having是分组 之后对结果进行过滤。
3. 判断条件不同：where不能对聚合函数进行判断，而having可以。
4. **有几种约束（6种）**

主键、外键、非空、唯一、默认、检查约束

1. **删除更新行为（5种）**
   1. no action：有外键不允许删除
   2. restrict：有外键不允许删除
   3. cascade：级联，父表删子表也删
   4. set null：父表删，子表变为null
   5. set default：父表删，子表变为默认值
2. **多表查询**

内连接join（显、隐），外连接left join（左、右），自连接（内、外）

内连接是交集，左外连接是左边和交集，自连接可以是内外连接

1. **子查询**

标量（单值）、列（列），表（多行多列）

1. **事务**

事务是一组操作的集合，要么同时成功，要么同时失败。如果执行过程中报错，我们可以回滚事务，把数据恢复到事务开始之前的状态。当有多个事务并发执行时，我们还需要考虑事务隔离的概念。

（start transaction/begin commit rollback）

1. **事务四大特性ACID**
   1. 原子性（Atomicity）：事务是不可分割的最小操作单元，要么全部成功，要么全部失败。
   2. 一致性（Consistency）：事务完成时，必须使所有的数据都保持一致状态。（比如转账后钱不能少）
   3. 隔离性（Isolation）：数据库系统提供的隔离机制，保证事务在不受外部并发操作影响的独立 环境下运行。
   4. 持久性（Durability）：事务一旦提交或回滚，它对数据库中的数据的改变就是永久的。

其中原子性、一致性和持久性通过重做日志（redo）和回滚日志（undo）来保证。隔离性则通过锁和MVCC来保证。

1. **并发事务问题**
   1. 脏写：一个事务修改了另一个未提交的事务修改过的数据。
   2. 脏读：一个事务读到另外一个事务还没有提交的数据。
   3. 不可重复读：一个事务先后读取同一条记录，但两次读取的数据不同，称之为不可重复读。
   4. 幻读：一个事务按照条件查询数据时，没有对应的数据行，但是在插入数据时，又发现这行数据 已经存在，好像出现了 "幻影"。
2. **事务隔离级别**

默认情况下，InnoDB在 REPEATABLE READ事务隔离级别运行，解决了脏写、脏读和不可重复读的问题。 然后使用临键锁进行索引扫描，以防止幻读。

RU解决脏写，RC解决脏读、RR解决不可重复读、串行化解决幻读（都向前兼容）

事务的隔离级别越高，数据越安全，但性能越低

（RU:Read Uncommitteed RC:Read Committed RR:Repeatable Read Serializable:串行化）

1. **MySQL体系结构**

客户端->连接层->服务层->引擎层->存储层

1. **InnoDB、MyASAM、Memory**
   1. InnoDB是默认存储引擎，支持事务、外键、行级锁。存储分为表空间、段、区（1M）、页（16K）、行。
   2. MyISAM不支持事务，不支持外键、不支持行锁只支持表锁，访问速度快 ，主要用于高负荷的查询表
   3. Memory引擎使用hash索引，表数据存储在内存中的，只能作为临时表或缓存使用。与Redis类似，不过Redis的性能更高所以一般会使用Redis代替。
2. **索引**

索引是帮助MySQL高效获取数据的有序的数据结构。

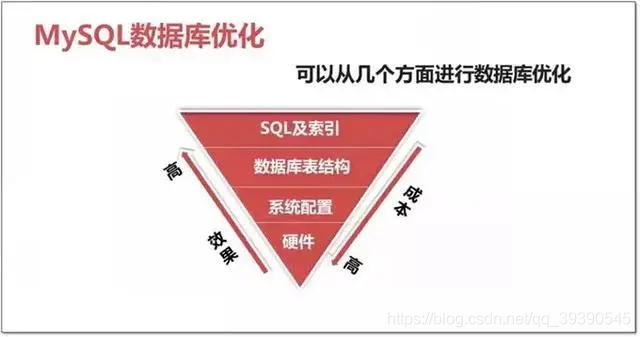
1. **B树、B-树、B+树、Hash**
   1. B（B-）树也叫多路平衡查找树，例如5阶B树表示每个节点最多有5个子节点，每个节点有4个key5个指针，key的后面会存放数据，指针则指向子节点的位置。子节点不会有和父节点相同的key
   2. B+树所有的数据都会出现在叶子节点，非叶子节点不包含数据，只起到索引的作用，所以可以用更小的磁盘空间存下索引，减少磁盘io次数。叶子节点形成一个单向链表。MySQL对B+树进一步优化，叶子节点形成的是一个循环双向链表，利于排序和范围查询。B+树更加矮胖，检索时间稳定。
   3. Hash索引就是采用一定的hash算法，将键值换算成新的hsh值保存在hash表中，这个值可以映射到对应的存储位值上。当出现hash冲突时，MySQL通过链表的方式来解决。hash索引对等比较查询效率很高但是不支持范围查询和排序操作。
   4. 自适应hash索引：如果观察到在特定的条件下hash索引可以提升速度，则自动建立hash索引，称之为自适应hash索引。
   5. R-tree索引主要用于地理类型的数据
   6. 全文索引（Full-text）主要用于快速匹配文档
2. **聚集索引和非聚集索引**

聚集索引是索引结构和数据一起存放且只能有一个，默认是主键，而非聚集索引不和数据一起存放且可以有多个。非聚集索引可以查询到聚集索引，当需要回表查询时，再根据聚集索引查找数据。

**优化**

1. **数据库优化**

可以提升数据库性能的方面有：硬件、系统配置、数据库表结构、SQL及索引。成本最低且效果最明显的就是SQL和索引的优化了。SQL优化应该最大化利用索引、减少无效数据的查询、尽可能避免全表扫描



1. **在select之前使用explain或desc 获取 MySQL 如何执行 SELECT 语句的信息（可以检查索引是否命中）**
2. **索引失效的情况**
   1. 使用不等于（<>或!=）
   2. 使用in和not in，范围查询最好使用between and才不会失效
   3. 在索引列上进行运算操作
   4. 发生类型转换（字符串类型的字段不加引号）
   5. 头部模糊查询（尾部模糊查询不会）
   6. or前后没有同时使用索引都会失效
   7. 联合索引不遵守最左前缀法则（索引左边列要在SQL里出现）
   8. 联合索引中，用>和<，右侧的列索引失效（它本身不失效）。使用>= 或 <= 时，索引不会失效
   9. 联合索引跳过中间列，后边的索引不会被用到
   10. 如果MySQL评估使用全表查询更快，则不使用索引。（*is null和is not null 根据此类情况讨论*）
3. **索引设计原则** 
   1. 针对于数据量较大，且查询比较频繁的表建立索引。
   2. 针对于常作为查询条件（where）、排序（order by）、分组（group by）操作的字段建立索引。
   3. 尽量选择区分度高的列作为索引，尽量建立唯一索引，区分度越高，使用索引的效率越高。 （唯一索引找到后不会继续检索）
   4. 如果是字符串类型的字段，字段的长度较长，可以针对于字段的特点，建立前缀索引。 （用了前缀索引就用不了覆盖索引）
   5. 根据查询情况建立联合索引，减少单列索引，查询时，联合索引很多时候可以覆盖索引，节省存储空间，避免回表，提高查询效率。
   6. 要控制索引的数量，索引并不是多多益善，索引越多，维护索引结构的代价也就越大，会影响增删改的效率。
   7. 尽量使用非空约束加默认值。当优化器知道每列是否包含NULL值时，它可以更好地确定哪个索引最有效地用于查询。
4. **SQL优化**
   1. 首先要想查询速度快我们应该建立合适的索引、或者是覆盖索引，然后用SQL提示让MySQL走我们预期的索引。
   2. 在优化之前可以使用show profiles命令查看查询时间都耗费到哪里去了（*show profiles;+show profiles for query ID;*）
   3. 批量插入可以减少数据库解析次数和连接次数
   4. 主键优化：主键要顺序插入，乱序插入会导致页分裂（*后一半数据分到其它页*），耗费性能。尽量降低主键的长度选择自增主键，不使用uuid和身份证等自然主键。避免对主键的修改
   5. order by优化：MySQL两种排序方式： 第一种通过索引顺序直接返回数据的方式效率最高，而另一种则是把数据复制到缓冲区中排序然后再将排好序的数据返回性能低。所以我们应该给经常排序的字段建立合适的索引。尽量使用覆盖索引(order后的字段顺序最好和建立联合索引时的顺序一致，不一致就会filesort） 。如果不可避免的要使用缓冲区排序时，可以适当增大排序缓冲区的大小 (默认256k)。
   6. group by优化：默认是先排序后分组，依然可以先建立合适的索引，或者在不需要排序的时候设置它。
   7. limit优化 ：一般采用子查询的形式进行优化。可以理解为在子表先对id排序和分页再和主表联合查询找数据。

*例如查询2000000条到2000010条（limit 2000000,10）*

*explain select \* from tb\_sku t , (select id from tb\_sku orde r by id*

*limit 2000000,10) a where*  ***t.id = a.id; （此步对主表进行筛选）***

* 1. update优化：InnoDB的行锁是针对索引加的锁，不是针对记录加的锁 ,并且该索引不能失效，否则会从行锁升级为表锁 。 回表时就会变为表锁，此时并发开启多个事务时，性能将大大降低。
  2. count优化 ：MySQL不会自动计数，需要用count（）函数计数，效率：count(字段) < count(主键) < count(1) <count(\*)，mysql对count（\*）做了优化，所以尽 量使用 count(\*)。

（*没有主键的时候count（1）比count（\*）快*）

1. **对大批量数据的优化**
2. 分库分表（垂直分记录不完整，水平分记录完整）
3. 读写分离
4. 对大表查询时需带限定范围（*比如查询历史记录自动限定在两个月内*）
5. **视图的作用：**
   1. 简化用户的理解，帮助用户屏蔽表结构的变化
   2. 安全性：可以屏蔽其它字段
6. **不符合视图创建条件的数据不会插入视图但是会插入基表中,怎样做到满足条件才能插入基表中？**

使用with casecaded/local check option。caesecade会级联检查多个嵌套视图；local只会检查当前视图

1. **存储过程**

可以存储SQL语句，以后可以多次调用，相同的SQL语句不用多次输入。可以接受参数

***游标（CURSOR）：****用来存储查询结果集的数据类型 , 在存储过程和函数中可以使用游标对结果集进行循环的处理*

***存储函数：****有返回值的存储过程，存储函数的参数只能是IN类型的。*

1. **触发器（trigger）**

触发器是与表有关的数据库对象，指在insert/update/delete之前(BEFORE)或之后(AFTER)，触发并执行触发器中定义的SQL语句集合。现在触发器还只支持行级触发，不支持语句级触发。

1. **事务的并发读写问题**

采用MVCC的方式，读-写操作彼此并不冲突，性能更高。

加锁方式的话，读-写需要排队执行，影响性能

1. **锁**

锁是计算机协调多个进程或线程并发访问某一资源的机制。按照对数据的操作类型可以分为共享锁和排他锁也叫读写锁，在InnoDB引擎中，读写锁既可以加在行上也可以加在表上。然后按照锁的粒度可以分为行级锁、页级锁和表级锁，需要注意的是，行锁是基于索引实现的，当索引失效时行锁就会转变为表锁，影响性能。行级锁又可分为行读写锁,间隙锁、临键锁、插入意向锁4种；表级锁也可分为表级读写锁、意向锁、自增锁、元数据锁四种，

1. **行级锁**

（1） 行级读写锁，读-读兼容，其他互斥

（2）间隙锁不允许在加锁的记录之前再插入数据，防止出现幻读的情况

（3）临建锁其实就是行锁和间隙锁的组合，既可以锁住当前行，又可以锁住前边的间隙，防止插入。

（4）插入意向锁是指间隙已经被锁住，但是自己还想插入就需要排队等待。

1. **表级锁**

（1）标记读写锁，读-读兼容，其他互斥

（2）意向锁也分为意向共享锁和意向排他锁，意向锁主要解决行锁与表锁的冲突，使得表锁不用检查每行数据是否加锁，意向锁之间都不互斥；与普通表锁的关系：读和意向读不互斥，其它都互斥。（例：某行加写锁，表自动加意向排它锁，当其它事务想要来加表锁的时候就不用逐行检查是否有排它锁了）

（3）自增锁主要是在有自增主键的情况下使用，因为每次只能加1，当有不同事务同时添加记录时就需要自增锁去约束

（4）元数据锁是系统自动控制，指的是一张表上有未提交的事务时，不能修改表的结构。

1. **死锁（和操作系统的一样）**
   1. 两个或两个以上事务
   2. 每个事务都持有锁并且申请新锁
   3. 锁资源同时只能被同一个资源持有
   4. 事务之间因为持有锁又想获得对方的锁而循环等待
2. **处理死锁**
   1. 设置超时后事务自动取消然后回滚
   2. 使用死锁检测进行死锁处理
3. **InnoDB引擎**

**存储**

每张表都会有一个表空间（table1.ibd），表空间内可分为段， 索引段即为B+树的非叶子节点，数据段就是B+树的叶子节点。段又可分为区，每个区的大小为1M。区又分为页， 默认情况下， InnoDB存储引擎页大小为16K， 即一 个区中一共有64个连续的页。页中的行就是所存放的记录，每行都有两个隐藏字段

Trx\_id：最近修改这条记录的事务的id。

Roll\_pointer：回滚指针，指向在undo日志中这条记录的上一个版本

Row\_Id：隐藏主键，没有主键时将会自动生成此字段

**架构-内存**

（1）缓冲池（Buffer Pool）：存放经常操作的数据，然后再以一定频率刷新到磁盘，从而减少磁盘IO，提升性能

（2）更改缓冲池（Change Buffer）存放被更改的数据，之后再恢复到缓冲池

（3）自适应哈希索引（Adaptive Hash Index）：用于优化对缓冲池数据的查询，系统自动支持

（4）日志缓冲区（Log Buffer）：保存要写入磁盘的日志

**架构-磁盘**

（1）双写缓冲区（Double Buffer Files）：刷新到磁盘前，先将数据页写入双写缓冲区文件中，便于系统异常时恢复数据

（2）系统表空间（System Tablespace）：包含表和索引数据

（3）撤销表空间（Undo Tablespace）：存储回滚日志（Undo log）

（4）临时表空间（Temporary Tablespace）：存放临时数据

（5）重做日志：用于存放重做日志（Redo log）

*（6）File-Per-Table Tablespaces*

*（7）通用表空间General Tablespaces*

**线程**

（1）核心线程：调度其它线程；将数据刷新到磁盘

（2）IO线程：处理IO请求

（3）Purge Thread：回收事务执行后的回滚日志

（4）Page Cleaner Threed：协助主线程将脏页刷新到磁盘

1. **日志种类**

（1）重做日志（redo log）：记录修改信息，异常时用于进行数据恢复

（2）回滚日志（undo log）：记录数据修改之前的信息，用于回滚事务和多版本并发控制（MVCC）

（3）二进制日志（bin log）：记录DDL和DML语句，用于主从复制和灾难时的数据恢复

（4）错误日志：记录错误信息

（5）查询日志：记录客户端所有的操作语句

（6）慢查询日志：可以配置查询时间超过多少秒时就记录它

1. **当前读和快照读**

当前读读取的是记录的最新版本，读取时还要保证其他并发事务不能修改当前记录，会对读取的记录进行加锁。

而简单的select（不加锁）就是快照读，快照读，读取的是记录数据的可见版本，有可能是历史数据，不加锁，是非阻塞读。

1. **MVCC**

多版本并发控制。指维护一个数据的多个版本，使得读写操作没有冲突，需要依赖于数据库记录中的三个隐式字段、回滚日志（undo log）和读视图（ReadView） 。

1. **主从复制**

主从复制是指将主数据库的 DDL 和 DML 操作通过二进制日志传到从库服务器中，从库记录到自己的终极日志后重新执行这些操作，从而使得从库和主库的数据保持同步。

优点：（1）主库宕机可以用从库提供服务；

（2）实现读写分离；

（3）可以在从库进行备份

1. **读写分离**

把对数据库的读和写操作分开,以对应不同的数据库服务器。主数据库提供写操作，从数据库提供读操作，这样能有效地减轻单台数据库的压力。

1. **多主多从**

一个主机 Master1 用于处理所有写请求，它的从机 Slave1 和另一台主机 Master2 还有它的从机 Slave2 负责所有读请求。当 Master1 主机宕机后，Master2 主机负责写请求，Master1 、Master2 互为备机。

1. **分库分表**

当数据量变得非常大时，只在一个主机上管理这些数据非常吃力，性能跟不上。

垂直分库：把一个数据库内不同的表分到不同的主机上

垂直分表：把一张表上不同的列分到不同的主机上

水平分库：所有数据库的表都一样，表内数据不一样

水平分表：将一个表的数据拆分到多个表中，表结构一样，数据不一样

1. **读视图ReadView**

进行快照读时产生的视图