# 数据库相关 面试积累

# SQL和数据库基础

SQL是最基本的数据库考核和实践的内容。基础中的基础。

|  |  |
| --- | --- |
|  |  |
| 范式 | **第一范式就是无重复的列**  **第二范式就是非主属性非部分（即完全）依赖于主关键字，不能出现只依赖关键字的一部分的属性。**  **第三范式就是属性不包含其它表中的非主属性。（我的理解是消除冗余，员工表中除了部门编号之外不要再有部门名称等信息了。）** |
| 内连接  **join** | * 有点像交集。 join 或者是 inner join。   select \* from table1 join table2 on table1.id=table2.id |
| 外连接  left join  left outer join | * 左连接：返回左表所有行，右边对应行没有匹配的话为null。   select \* from table1 **left join** table2 on table1.id=table2.id   * 右连接：包含table2的所有，   select \* from table1 **right join** table2 on table1.id=table2.id   * 完整外连接：返回左表和右表中的所有行。当某行在另一个表中没有匹配行时，则另一个表的选择列表列包含空值   select \* from table1 full join table2 on table1.id=table2.id |
|  |  |
| 删除 | drop直接删掉表 。 truncate删除表中数据，再插入时自增长id又从1开始 。 delete删除表中数据，可以加where字句。 |

# Sql优化

1. 避免在where子句中使用!= 或> < ，对字段进行null值判断，not in ， exists等否则将放弃索引使用全盘扫描。
2. 避免使用存储过程和触发器（pass 都是用代码来整合。）避免多表关联查询，因为这个依赖数据库去实现，高并发情况下可能导致数据库瘫痪。
3. Where条件里面必须有一个带索引。

**如何用explain分析sql执行效率？**

explain select \* from news;

输出： | id | select\_type | table | type | possible\_keys | key | key\_len | ref | rows | Extra |

# Mysql引擎

**InnoDB和MyIsam的区别？**

Mysql5.5之前MyIsam是默认引擎，不支持事务和行级锁。崩溃后无法恢复。

Mysql5.5之后引入了InnoDB（事务性数据库引擎），5.5之后的默认引擎为InnoDB。

1. 是否支持行级锁：Myisam只有表锁，InnoDB支持行锁和表锁。
2. 是否支持事务崩溃后的安全恢复：MyIsam强调性能，执行速度比InnoDB快，但不支持事务。InnoDB支持事务，可以回滚和崩溃修复。
3. 是否支持外键：MyIsam不支持，InnoDB支持外键。
4. 知否支持MVCC：只有InnoDB支持。

**MVCC？多版本并发控制。**

* 1. 高并发事务，MVCC比加锁更高效。
  2. 只在Read-committed和Repeatable-read两个隔离级别下工作。
  3. 可以实用乐观锁和悲观锁来实现。
  4. 各种数据库中MVCC的实现方式不统一。

# 索引

**为什么使用索引？**

加快数据的检索速度。

创建唯一索引，保证数据库表每一行数据的唯一性。

可以将随机IO变为顺序IO

帮助服务器避免排序和临时表

**为何不对每个字段都创建索引？**

1. 创建和维护索引（增删数据索引都要改）很耗费时间。
2. 索引占用物理空间。

**索引是如何提升查询速度的？**

将无序的数据变成相对有序的数据（目录一样）

**索引失效的情况？**

1. 条件中有 or 即使有条件带索引也不会使用。除非or条件中的每个列都加索引。
2. 多列索引，不是使用的第一部分，则不会使用索引（最左前缀原则）
3. Like查询以%开头，索引失效。
4. 条件类型是字符串，条件需用引号括起来。不括起来需要类型转换，类型转换索引失效。
5. 索引列使用了函数，索引失效。
6. 当Mysql觉得全表扫描比使用索引快，则索引失效。比如数据少的时候。

**什么是聚簇索引和非聚簇索引？**

聚簇索引：数据与索引在一起，找到索引即找到数据。

非~~~~：索引结构的叶子节点指向数据行。（MyInsm存地址，InnoDB存主键）

InnoDB中，在聚簇索引上面创建的是辅助索引。访问数据需要（根据主键）二次查找。InnoDB的非聚簇索引都是辅助索引，复合索引、前缀索引、唯一索引、辅助索引的叶子节点存储的不再是行的物理位置，而是主键值。

Mysql中默认设置主键为聚簇索引。没有主键选择非空索引，没有非空索引，隐式定义主键作为聚簇索引。

**索引使用的数据结构？**

哈希索引：哈希表，单条记录查询性能最快。应用场景比较少。

有序数组：等值查询和范围查询性能优秀，二分查找可快速找到O（logN）,但是插入的数据成本高。所以有序数组适用于静态存储。

BTree索引：Mysql的BTree索引使用的B+Tree结构，MyIsam和InnoDB两种引擎的实现方式不同。

**MyISAM和InnoDB实现Btree索引方式的区别？**

MyISAM是非聚簇索引，树的叶子data域存的是地址。指向数据行。

InnoDB数据文件本身就是索引文件。叶子存着完整的数据。索引的key就是主键，因此InnoDB数据文件本身就是主索引，是聚簇索引。其他索引都是辅助索引，辅助索引的data域存主键而不是地址。所以根据辅助索引查找的时候，需要回表，先取主键值，再走一遍主索引。主键不能太长，最好是单调的。

**BTree与B+Tree的区别是？ 为啥B+Tree更适合做文件索引。**

BTree根据键值生成树，每个节点都带有数据。

B+Tree中，所有数据都是根据键值大小顺序存在一层叶子节点上的。非叶子节点只存key值信息。内部节点小，同样大小的盘块能容纳更多的索引结构，一次性读入内存的关键字信息更多，IO次数更少。

**什么是覆盖索引，举例？**

当索引包含所需查询字段的值，就叫覆盖索引。因为InnoDB如果走辅助索引，需要回表。覆盖索引就是把要查询的列加索引，不需要做回表操作。

创建索引（username，age）

查询sql：select username,age from user where username=”zhanghao” and age =24;

要查询的列和索引是对应的，不需要回表。

**最左前缀原则的理解？**

Mysql的索引可以以一定顺序引用多列，这种索引**叫联合索引**。最左前缀原则就是查询的时候匹配左边连续一列或几列，索引就可以被用到。（查询的时候如果顺序不一致，那么查询引擎可以自动优化为匹配联合索引的顺序。也是可以命中索引的。）

# 事务

**什么是事务？**

事务是逻辑上的一组操作，要么执行，要么不执行。

**事务的四个特性：**

* 原子性（Auto）：最小的单位，不可再分。要么全部完成，要么不起作用。
* 一致性（Consistency）：一个事务必须从数据库的一致状态变换到另一个一致状态。（其他三个特性都是为一致性服务的。跟并发有关。所以保证并发条件下的一致性，需要事务具有隔离性，实现隔离级别，引出了乐观锁和悲观锁。）
* 隔离性（Isolation）：并发访问数据的时候，一个事务不被其他事务干扰。
* 持久性（Durability）：事务提交之后，数据库中的数据改变是持久的。

**事务的并发问题有哪些？**

* 脏读：读取了另一个还未提交的事务中的数据，读到的是个错误数据。
* 丢失修改：事务修改的数据被其他事务修改后提交，前事务的修改便丢失了。
* 不可重复读：多次读取结果不一致，中间被其他事务修改了。（某条数据前后不一致）
* 幻读：读取到了其他事务插入的数据（数据行发生了改变）

**事务的隔离级别？**

* 读未提交：查询语句不加锁，可能会读到未提交的行（产生脏读）
* 读已提交：写的时候加锁，读的时候没有阻塞update，可能前后读不一致（不可重复读）
* 可重复读：读事务阻塞其他写事务（包括update和delete 但是只给存在的行上加了锁），不包括insert，因为新行不存在，没有办法加锁。可能两次读取，行数变多了（幻读）
* 可串行化：InnoDB隐式的将全部的查询语句加了共享锁，解决了幻读问题。

**隔离级别如何实现？**

* **锁** 
  + 共享锁（读锁）
  + 互斥锁（写锁）
* **MVCC 多版本并发控制**
  + 维护很多个版本的数据，可以读取旧版数据。
  + 读不加锁，读写没有冲突。 但是每行记录都要额外的存储空间。

**Mysql的默认隔离级别？在实际企业中一般使用的隔离级别？**

Mysql默认是可重复读。

互联网项目一般用读已提交。

不采用串行化，因为每次读都要加锁，快照读失效。性能不佳。

可重复读隔离级别下，有间隙锁，出现死锁的几率比读已提交更大。

select \* from test where id <3 for update;

可重复读，会锁住间隙 防止其他事务插入。

读已提交，不存在间隙锁，其他事务可以插入。

可重复读隔离级别下，条件列未命中索引会锁表。在读已提交，只锁行。

**项目中用读已提交，那么不可重复读问题需要解决吗？**

不用解决，毕竟数据已经提交了。ORACLE的默认隔离级别就是读已提交，没见人改过。

# 锁

锁是实现事务的关键，锁可以保证事务的完整性和并发性

**请举出可能形成数据库死锁的原因？如何能避免死锁？**

死锁就是操作系统中两个或者以上的进程因抢夺资源而造成的互相等待的现象。

原因1：事务对资源访问顺序交替。

是程序BUG造成的，在数据库多表操作的时候，应该尽量按照相同的顺序进行处理，尽量避免同时锁住两个资源，必须同时锁住两个资源时，要保证任何时刻都应该按照相同的顺序来锁定资源。

原因2：并发修改同一条记录。[一个离奇的死锁](http://www.bimowu.com/kstudy-web/clienttopic/insSelfView.do?type=javaweb&topicId=784" \l "%E5%8F%A6%E4%B8%80%E8%B5%B7%E5%9F%BA%E4%BA%8E%E7%B4%A2%E5%BC%95%E7%9A%84%E5%8A%A0%E9%94%81%E5%92%8C%E6%AD%BB%E9%94%81%E9%97%AE%E9%A2%98)

使用乐观锁，写-写并发。？？？

使用悲观锁，依赖数据库的锁机制。

原因3：索引不当导致死锁

当全盘扫描次数过多时（全盘扫描将行级锁上升为了表级锁数据量非常庞大，而索引建的过少或者不合适都会发生全盘扫描，最终发生阻塞或死锁。）

解决： SQL语句中不要使用太复杂的关联度表查询，对于有全盘扫描的SQL语句，应该建立相应的索引进行优化。

**如何避免死锁？**

以固定的顺序访问表和行，这样就不会造成互相等待，按顺序申请锁。

尽量将大事物拆成小事物，大的事物会更容易死锁

在同一个事物中尽可能一次性锁定所需要的所有资源。

降低隔离级别，从可重复读变成读已提交，可以避免很多因为间隙锁造成的死锁。

为表添加合理的索引，如果不走索引，将会为表的每一行记录都添加上否字符的概率会大大增加。

**数据库中的乐观锁和悲观锁有什么区别？各适用于什么场景？**

悲观所就是很悲观的，每次去拿数据都会认为别人会修改，所以每次拿数据的时候都会上锁，这样别人想拿到这个数据就会阻塞，直到他拿到锁，比如传统的行锁表所读所写所都是在操作之前先上锁。

乐观锁很乐观，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这期间别人有没有去更新这个数据，可以使用版本号等机制，它适用于多读的类型，这样可以提高吞吐量。

乐观锁适用于写比较少的情况，这样省去了锁的开销，加大了系统的吞吐量，响应速度快。

如果经常产生冲突，上层的应用会不断的进行重试，这样反倒降低了性能，这种情况下用悲观锁就比较合适。如果重试代价大的话建议采用悲观锁。

**请结合你的开发经历，谈谈数据库中的乐观锁和悲观锁事具体如何被应用的。**

悲观锁：

首先关闭Mysql数据库的自动提交属性。 当执行一个更新操作后，my circle会立刻将结果进行提交。想要使用事物的话就要设置为非auto commit的模式。

*set autocommit=0;*

*//设置完autocommit后，我们就可以执行我们的正常业务了。具体如下：*

*//开始事务*

*begin;/begin work;/start transaction; (三者选一就可以)*

*//查询出商品信息*

*select status from items where id=10000 for update;*

*//根据商品信息生成订单*

*insert into orders (id,item\_id) values (null,10000);*

*//修改商品status为2*

*update items set status=2 where id=10000;*

*//提交事务*

*commit;*

从begin开始到commit结束是一个事务。

乐观锁：

加一个version字段，每次修改version++

*//查询出商品信息*

*select (status,version) from items where id=#{id}*

*//修改商品status为2，表示已被下单。*

*update items set status=2,version=version+1 where id=#{id} and version=#{version};*

*//以库存数作为乐观锁,只要保证库存数大于0即可。*

*update item set*

*quantity=quantity-#sub\_quantity#*

*where*

*item\_id = #id#*

*and quantity-#sub\_quantity# > 0*

# 日志

**事务日志：**  
基于InnoDB存储引擎的MySQL之所以可以从崩溃中恢复，正是依赖于事务日志，当数据库实例宕机后，重启时MySQL会自行检查事务日志，然后依次处理；  
事务日志分为redo log和undo log两种：  
(1)、对于事务日志中未正常提交的事务，则会记录到**undo log**中，因为事务未正确执行完，因此必须回滚，从而保证数据一致性  
(2)、对于事务日志中已正常提交但未同步到持久化存储上时，则会记录到**redo log**中，因此MySQL会重新执行一遍事务，然后让数据存储到磁盘上，从而保证数据一致性