# Mysql 复习笔记

## 1.数据库范式：

第一范式：每一列是独立的，不存在多个列名及属性值

第二范式：所有非主属性依赖于主键，不存在不依赖于主键的属性

第三范式：非主属性不存在传递依赖

BCNF：所有属性均依赖于主属性

## 2.InnoDB与MyISAM区别

|  |  |  |
| --- | --- | --- |
| InnoDB | MyISAM | 备注 |
| 支持事务 | 不支持事务 |  |
| 支持外键 | 不支持外键 |  |
| 聚集索引 | 非聚集索引 |  |
| 不保存表的具体行数 | 用一个变量保存了整个表的行数 | select count(\*) from table  InnoDB需要全表扫描 |
| 最小的锁粒度是行锁 | 最小的锁粒度是表锁 | MyISAM并发受限，所以Innodb |
|  |  |  |

InnoDB：MySQL默认的事务型引擎，也是最重要和使用最广泛的存储引擎。它被设计成为大量的短期事务，短期事务大部分情况下是正常提交的，很少被回滚。InnoDB的性能与自动崩溃恢复的特性，使得它在非事务存储需求中也很流行。除非有非常特别的原因需要使用其他的存储引擎，否则应该优先考虑InnoDB引擎。

MyISAM：在MySQL 5.1 及之前的版本，MyISAM是默认引擎。MyISAM提供的大量的特性，包括全文索引、压缩、空间函数（GIS）等，但MyISAM并不支持事务以及行级锁，而且一个毫无疑问的缺陷是崩溃后无法安全恢复。正是由于MyISAM引擎的缘故，即使MySQL支持事务已经很长时间了，在很多人的概念中MySQL还是非事务型数据库。尽管这样，它并不是一无是处的。对于只读的数据，或者表比较小，可以忍受修复操作，则依然可以使用MyISAM（但请不要默认使用MyISAM，而是应该默认使用InnoDB）MySQL存储引擎－－MyISAM与InnoDB区别

1、 存储结构

MyISAM：每个MyISAM在磁盘上存储成三个文件。分别为：表定义文件、数据文件、索引文件。第一个文件的名字以表的名字开始，扩展名指出文件类型。.frm文件存储表定义。数据文件的扩展名为.MYD (MYData)。索引文件的扩展名是.MYI (MYIndex)。

InnoDB：所有的表都保存在同一个数据文件中（也可能是多个文件，或者是独立的表空间文件），InnoDB表的大小只受限于操作系统文件的大小，一般为2GB。

2、 存储空间MyISAM： MyISAM支持支持三种不同的存储格式：静态表(默认，但是注意数据末尾不能有空格，会被去掉)、动态表、压缩表。当表在创建之后并导入数据之后，不会再进行修改操作，可以使用压缩表，极大的减少磁盘的空间占用。

InnoDB： 需要更多的内存和存储，它会在主内存中建立其专用的缓冲池用于高速缓冲数据和索引。

3、 可移植性、备份及恢复

MyISAM：数据是以文件的形式存储，所以在跨平台的数据转移中会很方便。在备份和恢复时可单独针对某个表进行操作。

InnoDB：免费的方案可以是拷贝数据文件、备份 binlog，或者用 mysqldump，在数据量达到几十G的时候就相对痛苦了。

4、 事务支持

MyISAM：强调的是性能，每次查询具有原子性,其执行数度比InnoDB类型更快，但是不提供事务支持。

InnoDB：提供事务支持事务，外部键等高级数据库功能。 具有事务(commit)、回滚(rollback)和崩溃修复能力(crash recovery capabilities)的事务安全(transaction-safe (ACID compliant))型表。

5、 AUTO\_INCREMENT

MyISAM：可以和其他字段一起建立联合索引。引擎的自动增长列必须是索引，如果是组合索引，自动增长可以不是第一列，他可以根据前面几列进行排序后递增。InnoDB：InnoDB中必须包含只有该字段的索引。引擎的自动增长列必须是索引，如果是组合索引也必须是组合索引的第一列。6、 表锁差异MyISAM： 只支持表级锁，用户在操作myisam表时，select，update，delete，insert语句都会给表自动加锁，如果加锁以后的表满足insert并发的情况下，可以在表的尾部插入新的数据。InnoDB： 支持事务和行级锁，是innodb的最大特色。行锁大幅度提高了多用户并发操作的新能。但是InnoDB的行锁，只是在WHERE的主键是有效的，非主键的WHERE都会锁全表的。7、 全文索引MySql全文索引MyISAM：支持 FULLTEXT类型的全文索引InnoDB：不支持FULLTEXT类型的全文索引，但是innodb可以使用sphinx插件支持全文索引，并且效果更好。8、表主键MyISAM：允许没有任何索引和主键的表存在，索引都是保存行的地址。InnoDB：如果没有设定主键或者非空唯一索引，就会自动生成一个6字节的主键(用户不可见)，数据是主索引的一部分，附加索引保存的是主索引的值。9、表的具体行数MyISAM： 保存有表的总行数，如果select count() from table;会直接取出出该值。InnoDB： 没有保存表的总行数，如果使用select count(\*) from table；就会遍历整个表，消耗相当大，但是在加了wehre条件后，myisam和innodb处理的方式都一样。10、CRUD操作MyISAM：如果执行大量的SELECT，MyISAM是更好的选择。InnoDB：如果你的数据执行大量的INSERT或UPDATE，出于性能方面的考虑，应该使用InnoDB表。11、 外键MyISAM：不支持InnoDB：支持

## 3.事务

—— 数据库事务是指作为单个逻辑工作单元执行的一系列操作，这些操作要么全做，要么全部做，是一个不可分割的工作单元。

—— 事务的开始与结束可以由用户显示的控制。如果用户没有显式地定义事务，则由DBMS按照默认的规定自动划分事务。事务分为 原子性、一致性、独立性 及 持久性 等特点。

事务的原子性是指一个事务要么全部执行，要么不执行。也就是说，一个事物不可能只执行了一半就停止了。比如你从银行取钱，这个事务可以分为两个步骤：①存折减款，②拿到现金。不可能存折钱少了，而钱没取出来。这两步必须同时完成，要么都不完成。

事务的一致性是指事务的运行并不改变数据库中数据的一致性。例如，完整性约束了a+b=10，一个事务改变了a, 那么b 也应该随之改变。

事务的独立性是指两个以上的事务不会出现交错执行的状态，因为这样可能会导致数据不一致。

事务的持久性是指事务运行成功后，系统的更新是永久的，不会无缘无故地回滚。

## 4.索引及其类型

聚集索引：cluster 在聚集索引中，表中行的物理顺序与键值的逻辑（索引）顺序相同。一个表只能包含一个聚集索引。 如果某索引不是聚集索引，则表中行的物理顺序与键值的逻辑顺序不匹配。与非聚集索引相比，聚集索引通常提供更快的数据访问速度。

唯一索引：UNIQUE 表明此索引的每一个索引值只对应唯一的数据记录，对于单列惟一性索引，这保证单列不包含重复的值。对于多列惟一性索引，保证多个值的组合不重复。

主键索引：primary key数据库表经常有一列或列组合，其值唯一标识表中的每一行。该列称为表的主键。 在数据库关系图中为表定义主键将自动创建主键索引，主键索引是唯一索引的特定类型。该索引要求主键中的每个值都唯一。当在查询中使用主键索引时，它还允许对数据的快速访问。

注意：索引是在存储引擎中实现的，也就是说不同的存储引擎，会使用不同的索引。MyISAM和InnoDB存储引擎：只支持BTREE索引，也就是说默认使用BTREE，不能够更换。MEMORY/HEAP存储引擎：支持HASH和BTREE索引。

1、索引我们分为四类来讲单列索引(普通索引，唯一索引，主键索引)、组合索引、全文索引、空间索引、

1.1、单列索引：一个索引只包含单个列，但一个表中可以有多个单列索引。 这里不要搞混淆了。

1.1.1、普通索引：MySQL中基本索引类型，没有什么限制，允许在定义索引的列中插入重复值和空值，纯粹为了查询数据更快一点。

1.1.2、唯一索引：索引列中的值必须是唯一的，但是允许为空值，

1.1.3、主键索引：是一种特殊的唯一索引，不允许有空值。（主键约束，就是一个主键索引）

1.2、组合索引：在表中的多个字段组合上创建的索引，只有在查询条件中使用了这些字段的左边字段时，索引才会被使用，使用组合索引时遵循最左前缀集合。例如，这里由id、name和age3个字段构成的索引，索引行中就按id/name/age的顺序存放，索引可以索引下面字段组合(id，name，age)、(id，name)或者(id)。如果要查询的字段不构成索引最左面的前缀，那么就不会是用索引，比如，age或者（name，age）组合就不会使用索引查询

1.3、全文索引：全文索引，只有在MyISAM引擎上才能使用，只能在CHAR,VARCHAR,TEXT类型字段上使用全文索引，介绍了要求，说说什么是全文索引，就是在一堆文字中，通过其中的某个关键字等，就能找到该字段所属的记录行，比如有"你是个大煞笔，二货 ..." 通过大煞笔，可能就可以找到该条记录。这里说的是可能，因为全文索引的使用涉及了很多细节，我们只需要知道这个大概意思。

1.4、空间索引：空间索引是对空间数据类型的字段建立的索引，MySQL中的空间数据类型有四种，GEOMETRY、POINT、LINESTRING、POLYGON。在创建空间索引时，使用SPATIAL关键字。要求，引擎为MyISAM，创建空间索引的列，必须将其声明为NOT NULL。可能跟游戏开发有关。

## 5. B树、B+树区别，索引为何使用B+树

<https://blog.csdn.net/xlgen157387/article/details/79450295>

## 6.数据库崩溃时事务的恢复机制（REDO日志和UNDO日志

Undo Log:

Undo Log是为了实现事务的原子性，在MySQL数据库InnoDB存储引擎中，还用了Undo Log来实现多版本并发控制(简称：MVCC)。

事务的原子性(Atomicity)事务中的所有操作，要么全部完成，要么不做任何操作，不能只做部分操作。如果在执行的过程中发生了错误，要回滚(Rollback)到事务开始前的状态，就像这个事务从来没有执行过。

原理Undo Log的原理很简单，为了满足事务的原子性，在操作任何数据之前，首先将数据备份到一个地方（这个存储数据备份的地方称为UndoLog）。然后进行数据的修改。如果出现了错误或者用户执行了ROLLBACK语句，系统可以利用Undo Log中的备份将数据恢复到事务开始之前的状态。

之所以能同时保证原子性和持久化，是因为以下特点：

更新数据前记录Undo log。

为了保证持久性，必须将数据在事务提交前写到磁盘。只要事务成功提交，数据必然已经持久化。

Undo log必须先于数据持久化到磁盘。如果在G,H之间系统崩溃，undo log是完整的， 可以用来回滚事务。

如果在A-F之间系统崩溃,因为数据没有持久化到磁盘。所以磁盘上的数据还是保持在事务开始前的状态。

缺陷：每个事务提交前将数据和Undo Log写入磁盘，这样会导致大量的磁盘IO，因此性能很低。

如果能够将数据缓存一段时间，就能减少IO提高性能。但是这样就会丧失事务的持久性。因此引入了另外一种机制来实现持久化，即Redo Log。

Redo Log:

原理和Undo Log相反，Redo Log记录的是新数据的备份。在事务提交前，只要将Redo Log持久化即可，不需要将数据持久化。当系统崩溃时，虽然数据没有持久化，但是Redo Log已经持久化。系统可以根据Redo Log的内容，将所有数据恢复到最新的状态。

## 7.mysql优化，读写分离、主从复制

<https://blog.51cto.com/13555423/2068071>

查询中用到的关键词主要包含六个，并且他们的顺序依次为 select--from--where--group by--having--order by

其中select和from是必须的，其他关键词是可选的，这六个关键词的执行顺序 与sql语句的书写顺序并不是一样的，而是按照下面的顺序来执行

**8.查询语句不同元素（where、jion、limit、group by、having等等）执行先后顺序**

from:需要从哪个数据表检索数据

where:过滤表中数据的条件

group by:如何将上面过滤出的数据分组

having:对上面已经分组的数据进行过滤的条件

select:查看结果集中的哪个列，或列的计算结果

order by :按照什么样的顺序来查看返回的数据

**9. drop/delete/truncate**

<https://blog.csdn.net/shadow_zed/article/details/78252494>

**10.聚集索引与非聚集索引（使用非聚集索引的查询过程）**

<https://www.cnblogs.com/aspnethot/articles/1504082.html>

<https://juejin.im/post/6844904117081473031>

<https://www.cnblogs.com/Arlen/articles/1751312.html>

**11.死锁判定原理和具体场景**

<https://blog.csdn.net/XiaHeShun/article/details/81393796>

## 12.mvcc

https://cloud.tencent.com/developer/news/310780

**13.乐观锁与悲观锁（一般都是悲观）**

<https://juejin.im/post/6844903639207641096>

# Redis详解

<https://blog.csdn.net/ThinkWon/article/details/103522351>

# 操作系统

## 1. 进程间通信方式

<https://www.jianshu.com/p/c1015f5ffa74>

## 2.信号量的使用（非互斥锁实现）

<https://www.jianshu.com/p/4fdad407068b>