MySQL

1.MySQL数据库

1.1 mysql的常用存储引擎

MyISAM

它不支持事务，也不支持外键，尤其是访问速度快，对事务完整性没有要求或者以SELECT、INSERT为主的应用基本都可以使用这个引擎来创建表。
每个MyISAM在磁盘上存储成3个文件，其中文件名和表名都相同，但是扩展名分别为：

- .frm(存储表定义)
- MYD(MYData，存储数据)
- MYI(MYIndex，存储索引)

数据文件和索引文件可以放置在不同的目录，平均分配IO，获取更快的速度。要指定数据文件和索引文件的路径，需要在创建表的时候通过DATA DIRECTORY和INDEX DIRECTORY语句指定，文件路径需要使用绝对路径。

每个MyISAM表都有一个标志，服务器或myisamchk程序在检查MyISAM数据表时会对这个标志进行设置。MyISAM表还有一个标志用来表明该数据表在上次使用后是不是被正常的关闭了。如果服务器以为当机或崩溃，这个标志可以用来判断数据表是否需要检查和修复。如果想让这种检查自动进行，可以在启动服务器时使用--myisam-recover现象。这会让服务器在每次打开一个MyISAM数据表是自动检查数据表的标志并进行必要的修复处理。MyISAM类型的表可能会损坏，可以使用CHECK TABLE语句来检查MyISAM表的健康，并用REPAIR TABLE语句修复一个损坏到MyISAM表。

MyISAM的表还支持3种不同的存储格式：

- 静态(固定长度)表
- 动态表
- 压缩表

其中静态表是默认的存储格式。静态表中的字段都是非变长字段，这样每个记录都是固定长度的，这种存储方式的优点是存储非常迅速，容易缓存，出现故障容易恢复；缺点是占用的空间通常比动态表多。静态表在数据存储时会根据列定义的宽度定义补足空格，但是在访问的时候并不会得到这些空格，这些空格在返回给应用之前已经去掉。同时需要注意：在某些情况下可能需要返回字段后的空格，而使用这种格式时后面到空格会被自动处理掉。

动态表包含变长字段，记录不是固定长度的，这样存储的优点是占用空间较少，但是频繁到更新删除记录会产生碎片，需要定期执行OPTIMIZE TABLE语句或myisamchk -r命令来改善性能，并且出现故障的时候恢复相对比较困难。

压缩表由myisamchk工具创建，占据非常小的空间，因为每条记录都是被单独压缩的，所以只有非常小的访问开支。

InnoDB

InnoDB是一个健壮的事务型存储引擎，这种存储引擎已经被很多互联网公司使用，为用户操作非常大的数据存储提供了一个强大的解决方案。我的电脑上安装的MySQL 5.6.13版，InnoDB就是作为默认的存储引擎。InnoDB还引入了行级锁定和外键约束，在以下场合下，使用InnoDB是最理想的选择：

1.更新密集的表。InnoDB存储引擎特别适合处理多重并发的更新请求。
2.事务。InnoDB存储引擎是支持事务的标准MySQL存储引擎。
3.自动灾难恢复。与其它存储引擎不同，InnoDB表能够自动从灾难中恢复。
4.外键约束。MySQL支持外键的存储引擎只有InnoDB。
5.支持自动增加列AUTO_INCREMENT属性。

一般来说，如果需要事务支持，并且有较高的并发读取频率，InnoDB是不错的选择。

MEMORY

使用MySQL Memory存储引擎的出发点是速度。为得到最快的响应时间，采用的逻辑存储介质是系统内存。虽然在内存中存储表数据确实会提供很高的性能，但当mysqld守护进程崩溃时，所有的Memory数据都会丢失。获得速度的同时也带来了一些缺陷。它要求存储在Memory数据表里的数据使用的是长度不变的格式，这意味着不能使用BLOB和TEXT这样的长度可变的数据类型，VARCHAR是一种长度可变的类型，但因为它在MySQL内部当做长度固定不变的CHAR类型，所以可以使用。

一般在以下几种情况下使用Memory存储引擎：

1.目标数据较小，而且被非常频繁地访问。在内存中存放数据，所以会造成内存的使用，可以通过参数max_heap_table_size控制Memory表的大小，设置此参数，就可以限制Memory表的最大大小。

2.如果数据是临时的，而且要求必须立即可用，那么就可以存放在内存表中。

3.存储在Memory表中的数据如果突然丢失，不会对应用服务产生实质的负面影响。

Memory同时支持散列索引和B树索引。B树索引的优于散列索引的是，可以使用部分查询和通配查询，也可以使用<、>和>=等操作符方便数据挖掘。散列索引进行“相等比较”非常快，但是对“范围比较”的速度就慢多了，因此散列索引值适合使用在=和<>的操作符中，不适合在<或>操作符中，也同样不适合用在order by子句中。

可以在表创建时利用USING子句指定要使用的版本。例如：

复制代码代码如下:

```
create table users
(
   id smallint unsigned not null auto_increment,
   username varchar(15) not null,
   pwd varchar(15) not null,
   index using hash (username),
   primary key (id)
)engine=memory;
```

上述代码创建了一个表，在username字段上使用了HASH散列索引。下面的代码就创建一个表，使用BTREE索引。

复制代码代码如下:

```
create table users
(
   id smallint unsigned not null auto_increment,
   username varchar(15) not null,
   pwd varchar(15) not null,
   index using btree (username),
   primary key (id)
)engine=memory;
```

MERGE

MERGE存储引擎是一组MyISAM表的组合，这些MyISAM表结构必须完全相同，尽管其使用不如其它引擎突出，但是在某些情况下非常有用。说白了，Merge表就是几个相同MyISAM表的聚合器；Merge表中并没有数据，对Merge类型的表可以进行查询、更新、删除操作，这些操作实际上是对内部的MyISAM表进行操作。Merge存储引擎的使用场景。

对于服务器日志这种信息，一般常用的存储策略是将数据分成很多表，每个名称与特定的时间端相关。例如：可以用12个相同的表来存储服务器日志数据，每个表用对应各个月份的名字来命名。当有必要基于所有12个日志表的数据来生成报表，这意味着需要编写并更新多表查询，以反映这些表中的信息。与其编写这些可能出现错误的查询，不如将这些表合并起来使用一条查询，之后再删除Merge表，而不影响原来的数据，删除Merge表只是删除Merge表的定义，对内部的表没有任何影响。

ARCHIVE

Archive是归档的意思，在归档之后很多的高级功能就不再支持了，仅仅支持最基本的插入和查询两种功能。在MySQL 5.5版以前，Archive是不支持索引，但是在MySQL 5.5以后的版本中就开始支持索引了。Archive拥有很好的压缩机制，它使用zlib压缩库，在记录被请求时会实时压缩，所以它经常被用来当做仓库使用。

存储引擎的一些问题

1.如何查看服务器有哪些存储引擎可以使用?
为确定你的MySQL服务器可以用哪些存储引擎，执行如下命令：

复制代码代码如下:

```
show engines;
```

这个命令就能搞定了。

2.如何选择合适的存储引擎?
（1）选择标准可以分为：
（2）是否需要支持事务；
（3）是否需要使用热备；
（4）崩溃恢复：能否接受崩溃；
（5）是否需要外键支持；
然后按照标准，选择对应的存储引擎即可。

1.2 mysql存储引擎比较

MyISAM：默认的MySQL插件式存储引擎，它是在Web、数据仓储和其他应用环境下最常使用的存储引擎之一。

· InnoDB：用于事务处理应用程序，具有众多特性，包括ACID事务支持。(提供行级锁)

· BDB：可替代InnoDB的事务引擎，支持COMMIT、ROLLBACK和其他事务特性。

· Memory：将所有数据保存在RAM中，在需要快速查找引用和其他类似数据的环境下，可提供极快的访问。

· Merge：允许MySQL DBA或开发人员将一系列等同的MyISAM表以逻辑方式组合在一起，并作为1个对象引用它们。对于诸如数据仓储等VLDB环境十分适合。

· Archive：为大量很少引用的历史、归档、或安全审计信息的存储和检索提供了完美的解决方案。

· Federated：能够将多个分离的MySQL服务器链接起来，从多个物理服务器创建一个逻辑数据库。十分适合于分布式环境或数据集市环境。

· Cluster/NDB：MySQL的簇式数据库引擎，尤其适合于具有高性能查找要求的应用程序，这类查找需求还要求具有最高的正常工作时间和可用性。 · Other：其他存储引擎包括CSV（引用由逗号隔开的用作数据库表的文件），Blackhole（用于临时禁止对数据库的应用程序输入），以及Example引擎（可为快速创建定制的插件式存储引擎提供帮助）。

一般来说不使用事务的话，请使用MyISAM引擎，使用事务的话，一般使用InnoDB ------，通过更改STORAGE_ENGINE配置变量，能够方便地更改MySQL服务器的默认存储引擎。

1.3 范式

什么是范式：简言之就是，数据库设计对数据的存储性能，还有开发人员对数据的操作都有莫大的关系。所以建立科学的，规范的的数据库是需要满足一些 规范的来优化数据数据存储方式。在关系型数据库中这些规范就可以称为范式。

什么是三大范式：

第一范式：当关系模式R的所有属性都不能在分解为更基本的数据单位时，称R是满足第一范式的，简记为1NF。满足第一范式是关系模式规范化的最低要 求，否则，将有很多基本操作在这样的关系模式中实现不了。

第二范式：如果关系模式R满足第一范式，并且R得所有非主属性都完全依赖于R的每一个候选关键属性，称R满足第二范式，简记为2NF。

第三范式：设R是一个满足第一范式条件的关系模式，X是R的任意属性集，如果X非传递依赖于R的任意一个候选关键字，称R满足第三范式，简记为3NF. 注：关系实质上是一张二维表，其中每一行是一个元组，每一列是一个属性

1.4 事务以及MYSQL如何支持事务

事务就是一段sql 语句的批处理，但是这个批处理是一个原子 ，不可分割，要么都执行，要么回滚（rollback）都不执行。

事务具体四大特性，也就是经常说的ACID ：

1.原子性（所有操作要么全部成功，要么全部失败回滚）

2.一致性（事务执行之前和执行之后都必须处于一致性状态。）

3.隔离性（数据库为每一个用户开启的事务，不能被其他事务的操作所干扰，多个并发事务之间要相互隔离）

4.持久性（一个事务一旦被提交了，那么对数据库中的数据的改变就是永久性的，即使遭遇故障依然能够通过日志恢复最后一次更新） 在 MySQL 中只有使用了 Innodb 数据库引擎的数据库或表才支持事务

MYSQL 事务处理主要有两种方法：

1、用 BEGIN, ROLLBACK, COMMIT来实现 BEGIN 开始一个事务 ROLLBACK 事务回滚 COMMIT 事务确认

2、直接用 SET 来改变 MySQL 的自动提交模式: SET AUTOCOMMIT=0 禁止自动提交 SET AUTOCOMMIT=1 开启自动提交

1.5 数据库的触发器，函数，视图，存储过程

触发器：触发器是一个特殊的存储过程，它是MySQL在insert、update、delete的时候自动执行的代码块。

```
create trigger trigger_name

    after/before insert /update/delete on 表名
```

for each row

begin

sql语句：（触发的语句一句或多句）

end

函数：MySQL中提供了许多内置函数，还可以自定义函数（实现程序员需要sql逻辑处理）

自定义函数创建语法：

创建：CREATE FUNCTION 函数名称(参数列表)

RETURNS 返回值类型　函数体

修改： ALTER FUNCTION 函数名称 [characteristic ...]

删除：DROP FUNCTION [IF EXISTS] 函数名称

调用：SELECT 函数名称(参数列表)

视图：视图是由查询结果形成的一张虚拟表，是表通过某种运算得到的一个投影

create view view_name as select 语句

存储过程：把一段代码封装起来，当要执行这一段代码的时候，可以通过调用该存储过程来实现（经过第一次编译后再次调用不需要再次编译，比一个个执行sql语句效率高）

create procedure 存储过程名(参数,参数,...)

begin

//代码

end

1.6 存储引擎的数据结构

- MyISAM引擎

  不支持事务

  支持表级锁（MySql支持两种表级锁，表共享读锁和表独占写锁），但不支持行级锁

  存储表的总行数

  一个MyISAM表有三个文件：索引文件（.MYI），表结构文件(.frm)，数据文件(.MYD)

  采用非聚集索引：即索引文件和数据文件是分开的，索引文件的数据域存储指向数据文件的指针

  跨平台应用更方便（表保存为文件形式）

  支持三种不同的存储格式：

  （1）静态表：存储迅速，容易缓存，出现故障容易恢复；但是占用内存多，因为会按列宽度补足空格

  （2）动态表：占用空间少，但是频繁更新和删除容易产生碎片，需要定期整理（OPTIMIZE TABLE），并且出现故障难以恢复。

（3）压缩表：占据的磁盘空间非常小（每个记录被单独压缩，所以访问开支很小）。

- InnoDB引擎

  支持事务

  支持行级锁（仅在条件语句中包括主键索引时）

  内存使用率低

  查询效率和写的效率更低

  采用聚集索引，索引和数据存在一起，叶子结点直接存的是数据。

  支持外键

  注：MyISAM在查询时的性能比InnoDB高，因为它采用的辅索引和主键索引类似，所以通过辅索引查找数据时只需要通过辅索引树就可以查找到，而InnoDB需要先通过辅索引查找到主索引，再通过主索引树查找到数据。

- MEMORY

  只对应一个磁盘文件.frm，用来存储表结构

  访问速度非常快，因为他的数据存在内存中，但是一旦服务关闭，数据就会丢失

  可以指定Hash索引或BTREE索引

  默认存储数据大小不超过16MB，但可以调整

  应用场景：比如作为统计操作的的中间结果表，便于高效地对中间结果分析并得到最终结果。

- MERGE

  一组MyISAM表的组合，这些MyISAM表结构必须完全相同

  对MERGE表的操作实际上是对其子表进行的

  可以通过指定INSERT_METHOD=LAST来制定插入数据的表（这里指定为最后一个表）

2. MySQL索引

2.1 什么是索引

索引**是一个排序的列表**，在这个列表中存储着索引的值和包含这个值的数据所在行的物理地址，在数据十分庞大的时候，索引可以大大加快查询的速度，这是因为使用索引后可以不用扫描全表来定位某行的数据，而是先通过索引表找到该行数据对应的物理地址然后访问相应的数据。

2.2 MySQL中索引的语法

创建索引

在创建表的时候添加索引

```
CREATE TABLE mytable(
    ID INT NOT NULL,
    username VARCHAR(16) NOT NULL,
    INDEX [indexName] (username(length))
);
```

在创建表以后添加索引

```
ALTER TABLE my_table ADD [UNIQUE] INDEX index_name(column_name);
或者
CREATE INDEX index_name ON my_table(column_name);
```

注意：

1、索引需要占用磁盘空间，因此在创建索引时要考虑到磁盘空间是否足够

2、创建索引时需要对表加锁，因此实际操作中需要在业务空闲期间进行

根据索引查询

```
具体查询：
SELECT * FROM table_name WHERE column_1=column_2;(为column_1建立了索引)

或者模糊查询
SELECT * FROM table_name WHERE column_1 LIKE '%三'
SELECT * FROM table_name WHERE column_1 LIKE '三%'
SELECT * FROM table_name WHERE column_1 LIKE '%三%'

SELECT * FROM table_name WHERE column_1 LIKE '_好_'

如果要表示在字符串中既有A又有B，那么查询语句为：
SELECT * FROM table_name WHERE column_1 LIKE '%A%' AND column_1 LIKE '%B%';

SELECT * FROM table_name WHERE column_1 LIKE '[张李王]三';  //表示column_1中有匹配张三、李三、王三的都可以
SELECT * FROM table_name WHERE column_1 LIKE '[^张李王]三';  //表示column_1中有匹配除了张三、李三、王三的其他三都可以

//在模糊查询中，%表示任意0个或多个字符；_表示任意单个字符（有且仅有），通常用来限制字符串长度；[]表示其中的某一个字符；[^]表示除了其中的字符的所有字符

或者在全文索引中模糊查询
SELECT * FROM table_name WHERE MATCH(content) AGAINST('word1','word2',...);
```

删除索引

```
DROP INDEX my_index ON tablename;
或者
ALTER TABLE table_name DROP INDEX index_name;
```

查看表中的索引

SHOW INDEX FROM tablename

查看查询语句使用索引的情况

```
//explain 加查询语句
explain SELECT * FROM table_name WHERE column_1='123';
```

2.3 索引的分类

常见的索引类型有：主键索引、唯一索引、普通索引、全文索引、组合索引

1、主键索引：即主索引，根据主键pk_clolum（length）建立索引，不允许重复，不允许空值；

```
ALTER TABLE 'table_name' ADD PRIMARY KEY pk_index('col');
```

2、唯一索引：用来建立索引的列的值必须是唯一的，允许空值

```
ALTER TABLE 'table_name' ADD UNIQUE index_name('col');
```

3、普通索引：用表中的普通列构建的索引，没有任何限制

```
ALTER TABLE 'table_name' ADD INDEX index_name('col');
```

4、全文索引：用大文本对象的列构建的索引（下一部分会讲解）

```
ALTER TABLE 'table_name' ADD FULLTEXT INDEX ft_index('col');
```

5、组合索引：用多个列组合构建的索引，这多个列中的值不允许有空值

```
ALTER TABLE 'table_name' ADD INDEX index_name('col1','col2','col3');
```

*遵循“最左前缀”原则，把最常用作为检索或排序的列放在最左，依次递减，组合索引相当于建立了col1,col1col2,col1col2col3三个索引，而col2或者col3是不能使用索引的。

*在使用组合索引的时候可能因为列名长度过长而导致索引的key太大，导致效率降低，在允许的情况下，可以只取col1和col2的前几个字符作为索引

```
ALTER TABLE 'table_name' ADD INDEX index_name(col1(4),col2（3));
```

表示使用col1的前4个字符和col2的前3个字符作为索引

3. 索引的实现原理

MySQL支持诸多存储引擎，而各种存储引擎对索引的支持也各不相同，因此MySQL数据库支持多种索引类型，如**BTree索引，B+Tree索引，哈希索引，全文索引**等等，

3.1 哈希索引

只有memory（内存）存储引擎支持哈希索引，哈希索引用索引列的值计算该值的hashCode，然后在hashCode相应的位置存执该值所在行数据的物理位置，因为使用散列算法，因此访问速度非常快，但是一个值只能对应一个hashCode，而且是散列的分布方式，因此哈希索引不支持范围查找和排序的功能

3.2 全文索引

FULLTEXT（全文）索引，仅可用于MyISAM和InnoDB，针对较大的数据，生成全文索引非常的消耗时间和空间。对于文本的大对象，或者较大的CHAR类型的数据，如果使用普通索引，那么匹配文本前几个字符还是可行的，但是想要匹配文本中间的几个单词，那么就要使用LIKE %word%来匹配，这样需要很长的时间来处理，响应时间会大大增加，这种情况，就可使用时FULLTEXT索引了，在生成FULLTEXT索引时，会为文本生成一份单词的清单，在索引时及根据这个单词的清单来索引。FULLTEXT可以在创建表的时候创建，也可以在需要的时候用ALTER或者CREATE INDEX来添加：

```
//创建表的时候添加FULLTEXT索引
CTREATE TABLE my_table(
    id INT(10) PRIMARY KEY,
    name VARCHAR(10) NOT NULL,
    my_text TEXT,
    FULLTEXT(my_text)
)ENGINE=MyISAM DEFAULT CHARSET=utf8;

//创建表以后，在需要的时候添加FULLTEXT索引
ALTER TABLE my_table ADD FULLTEXT INDEX ft_index(column_name);
```

全文索引的查询也有自己特殊的语法，而不能使用LIKE %查询字符串%的模糊查询语法

SELECT * FROM table_name MATCH(ft_index) AGAINST('查询字符串');

注意：

*对于较大的数据集，把数据添加到一个没有FULLTEXT索引的表，然后添加FULLTEXT索引的速度比把数据添加到一个已经有FULLTEXT索引的表快。

*5.6版本前的MySQL自带的全文索引只能用于MyISAM存储引擎，如果是其它数据引擎，那么全文索引不会生效。5.6版本之后InnoDB存储引擎开始支持全文索引

*在MySQL中，全文索引支队英文有用，目前对中文还不支持。5.7版本之后通过使用ngram插件开始支持中文。

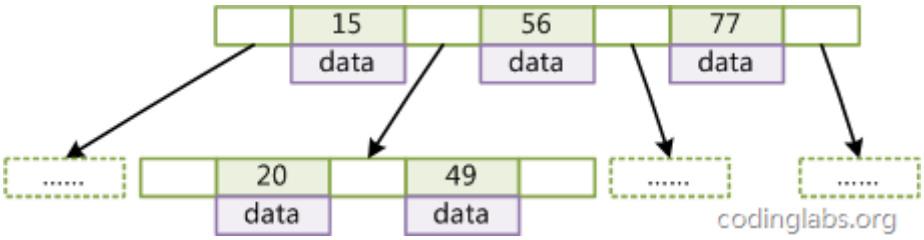
*在MySQL中，如果检索的字符串太短则无法检索得到预期的结果，检索的字符串长度至少为4字节，此外，如果检索的字符包括停止词，那么停止词会被忽略。

3.3 BTree索引和B+Tree索引

- BTree是平衡搜索多叉树，设树的度为2d（d>1），高度为h，那么BTree要满足以一下条件：

  每个叶子结点的高度一样，等于h；
  每个非叶子结点由n-1个key和n个指针point组成，其中d<=n<=2d,key和point相互间隔，结点两端一定是key；
  叶子结点指针都为null；
  非叶子结点的key都是[key,data]二元组，其中key表示作为索引的键，data为键值所在行的数据；

BTree的结构如下：



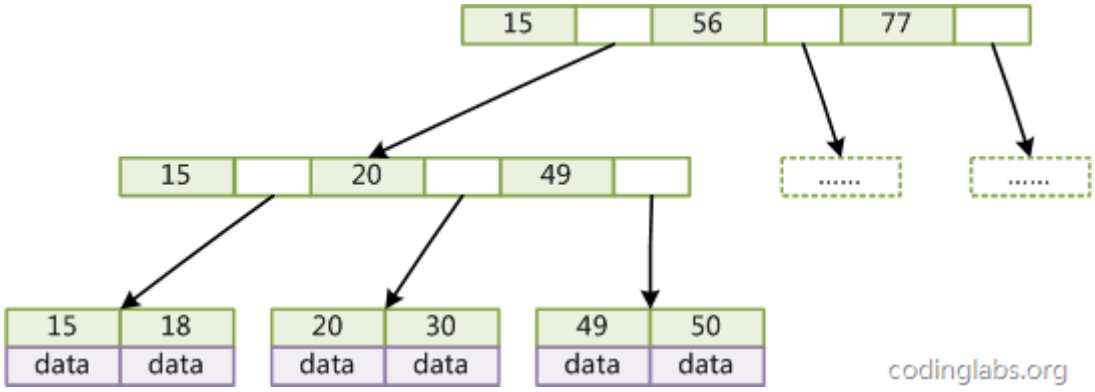
在BTree的机构下，就可以使用二分查找的查找方式，查找复杂度为h*log(n)，一般来说树的高度是很小的，一般为3左右，因此BTree是一个非常高效的查找结构。

- B+Tree 索引

  B+Tree是BTree的一个变种，设d为树的度数，h为树的高度，B+Tree和BTree的不同主要在于：

  ```
  B+Tree中的非叶子结点不存储数据，只存储键值；
  B+Tree的叶子结点没有指针，所有键值都会出现在叶子结点上，且key存储的键值对应data数据的物理地址；
  B+Tree的每个非叶子节点由n个键值key和n个指针point组成；
  ```

B+Tree的结构如下：



B+Tree对比BTree的优点：

1、磁盘读写代价更低

一般来说B+Tree比BTree更适合实现外存的索引结构，因为存储引擎的设计专家巧妙的利用了外存（磁盘）的存储结构，即磁盘的最小存储单位是扇区（sector），而操作系统的块（block）通常是整数倍的sector，操作系统以页（page）为单位管理内存，一页（page）通常默认为4K，数据库的页通常设置为操作系统页的整数倍，因此索引结构的节点被设计为一个页的大小，然后利用外存的“预读取”原则，每次读取的时候，把整个节点的数据读取到内存中，然后在内存中查找，已知内存的读取速度是外存读取I/O速度的几百倍，那么提升查找速度的关键就在于尽可能少的磁盘I/O，那么可以知道，每个节点中的key个数越多，那么树的高度越小，需要I/O的次数越少，因此一般来说B+Tree比BTree更快，因为B+Tree的非叶节点中不存储data，就可以存储更多的key。

2、查询速度更稳定

由于B+Tree非叶子节点不存储数据（data），因此所有的数据都要查询至叶子节点，而叶子节点的高度都是相同的，因此所有数据的查询速度都是一样的。

- 带顺序索引的B+TREE

  很多存储引擎在B+Tree的基础上进行了优化，**添加了指向相邻叶节点的指针**，形成了带有顺序访问指针的B+Tree，这样做是为了**提高区间查找的效率**，只要找到第一个值那么就可以顺序的查找后面的值。

  B+Tree的结构如下：

4. 聚簇索引和非聚簇索引

MySQL中最常见的两种存储引擎分别是MyISAM和InnoDB，分别实现了非聚簇索引和聚簇索引。

聚簇索引的解释是:聚簇索引的顺序就是数据的物理存储顺序

非聚簇索引的解释是:索引顺序与数据物理排列顺序无关

（这样说起来并不好理解，让人摸不着头脑，清继续看下文，并在插图下方对上述两句话有解释）

首先要介绍几个概念，在索引的分类中，我们可以按照索引的键是否为主键来分为“主索引”和“辅助索引”，使用主键键值建立的索引称为“主索引”，其它的称为“辅助索引”。因此主索引只能有一个，辅助索引可以有很多个。

MyISAM——非聚簇索引

MyISAM存储引擎采用的是非聚簇索引，非聚簇索引的主索引和辅助索引几乎是一样的，只是主索引不允许重复，不允许空值，他们的叶子结点的key都存储指向键值对应的数据的物理地址。
非聚簇索引的数据表和索引表是分开存储的。
非聚簇索引中的数据是根据数据的插入顺序保存。因此非聚簇索引更适合单个数据的查询。插入顺序不受键值影响。
只有在MyISAM中才能使用FULLTEXT索引。(mysql5.6以后innoDB也支持全文索引)

*最开始我一直不懂既然非聚簇索引的主索引和辅助索引指向相同的内容，为什么还要辅助索引这个东西呢，后来才明白索引不就是用来查询的吗，用在那些地方呢，不就是WHERE和ORDER BY 语句后面吗，那么如果查询的条件不是主键怎么办呢，这个时候就需要辅助索引了。

InnoDB——聚簇索引

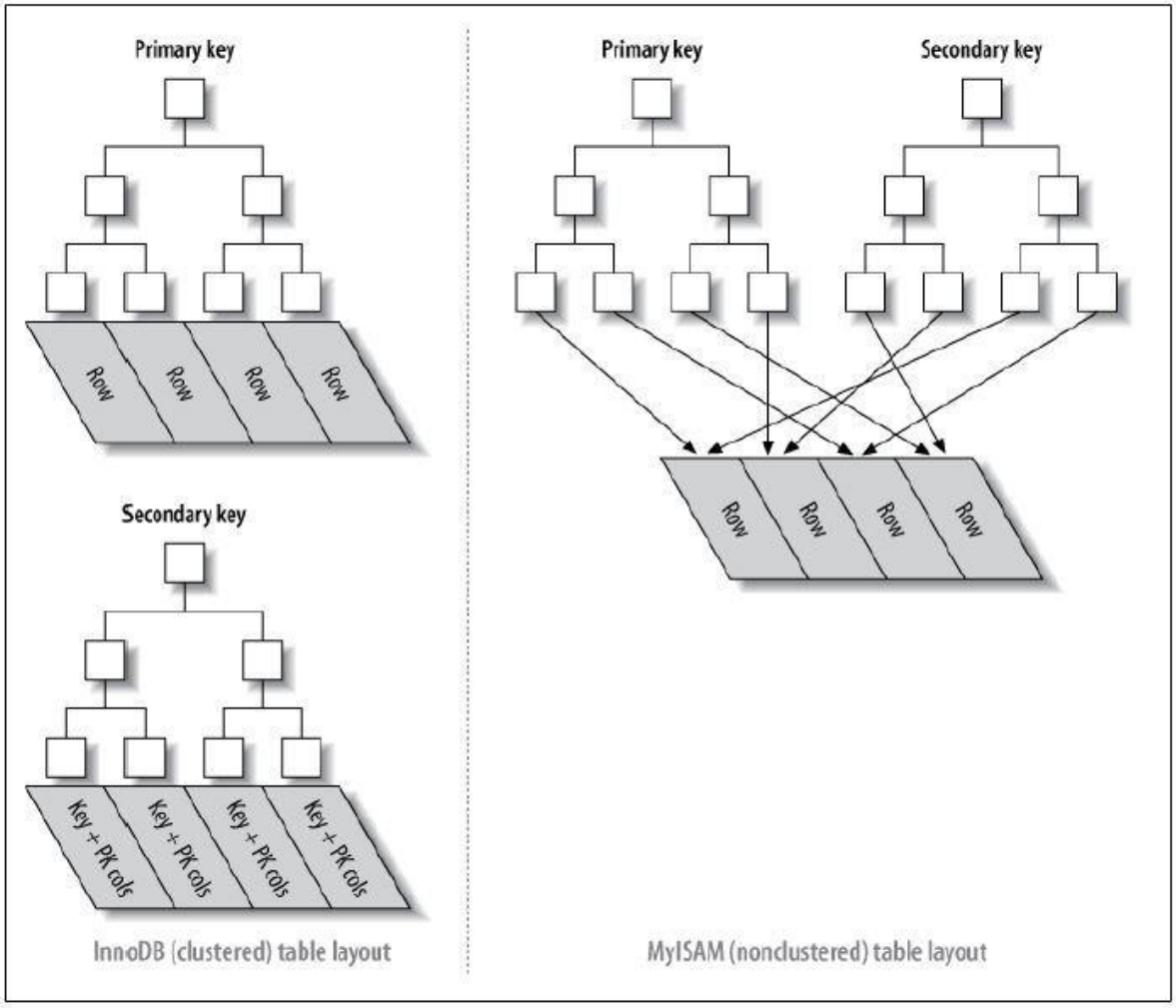
聚簇索引的主索引的叶子结点存储的是键值对应的数据本身，辅助索引的叶子结点存储的是键值对应的数据的主键键值。因此主键的值长度越小越好，类型越简单越好。
聚簇索引的数据和主键索引存储在一起。
聚簇索引的数据是根据主键的顺序保存。因此适合按主键索引的区间查找，可以有更少的磁盘I/O，加快查询速度。但是也是因为这个原因，聚簇索引的插入顺序最好按照主键单调的顺序插入，否则会频繁的引起页分裂，严重影响性能。
在InnoDB中，如果只需要查找索引的列，就尽量不要加入其它的列，这样会提高查询效率。

*使用主索引的时候，更适合使用聚簇索引，因为聚簇索引只需要查找一次，而非聚簇索引在查到数据的地址后，还要进行一次I/O查找数据。

*因为聚簇辅助索引存储的是主键的键值，因此可以在数据行移动或者页分裂的时候降低成本，因为这时不用维护辅助索引。但是由于主索引存储的是数据本身，因此聚簇索引会占用更多的空间。

*聚簇索引在插入新数据的时候比非聚簇索引慢很多，因为插入新数据时需要检测主键是否重复，这需要遍历主索引的所有叶节点，而非聚簇索引的叶节点保存的是数据地址，占用空间少，因此分布集中，查询的时候I/O更少，但聚簇索引的主索引中存储的是数据本身，数据占用空间大，分布范围更大，可能占用好多的扇区，因此需要更多次I/O才能遍历完毕。

下图可以形象的说明聚簇索引和非聚簇索引的区别



从上图中可以看到聚簇索引的辅助索引的叶子节点的data存储的是主键的值，主索引的叶子节点的data存储的是数据本身，也就是说数据和索引存储在一起，并且索引查询到的地方就是数据（data）本身，那么索引的顺序和数据本身的顺序就是相同的；

而非聚簇索引的主索引和辅助索引的叶子节点的data都是存储的数据的物理地址，也就是说索引和数据并不是存储在一起的，数据的顺序和索引的顺序并没有任何关系，也就是索引顺序与数据物理排列顺序无关。

此外MyISAM和innoDB的区别总结如下：

	MyISAM	innoDB
索引类型	非聚簇	聚簇
支持事务	是	否
支持表锁	是	是
支持行锁	否	是
支持外键	否	是
支持全文索引	是	是
适用操作类型	大量select下使用	大量insert、delete和update下使用

```
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；
```

此外，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；

5 索引的使用策略及优化

5.1 索引的使用策略

什么时候要使用索引?

```
主键自动建立唯一索引；
经常作为查询条件在WHERE或者ORDER BY 语句中出现的列要建立索引；
作为排序的列要建立索引；
查询中与其他表关联的字段，外键关系建立索引
高并发条件下倾向组合索引；
用于聚合函数的列可以建立索引，例如使用了max(column_1)或者count(column_1)时的column_1就需要建立索引
```

什么时候不要使用索引?

```
经常增删改的列不要建立索引；
有大量重复的列不建立索引；
表记录太少不要建立索引。只有当数据库里已经有了足够多的测试数据时，它的性能测试结果才有实际参考价值。如果在测试数据库里只有几百条数据记录，它们往往在执行完第一条查询命令之后就被全部加载到内存里，这将使后续的查询命令都执行得非常快--不管有没有使用索引。只有当数据库里的记录超过了1000条、数据总量也超过了MySQL服务器上的内存总量时，数据库的性能测试结果才有意义。
```

索引失效的情况:

```
在组合索引中不能有列的值为NULL，如果有，那么这一列对组合索引就是无效的。
在一个SELECT语句中，索引只能使用一次，如果在WHERE中使用了，那么在ORDER BY中就不要用了。
LIKE操作中，'%aaa%'不会使用索引，也就是索引会失效，但是‘aaa%’可以使用索引。
在索引的列上使用表达式或者函数会使索引失效，例如：select * from users where YEAR(adddate)<2007，将在每个行上进行运算，这将导致索引失效而进行全表扫描，因此我们可以改成：select * from users where adddate<’2007-01-01’。其它通配符同样，也就是说，在查询条件中使用正则表达式时，只有在搜索模板的第一个字符不是通配符的情况下才能使用索引。
在查询条件中使用不等于，包括<符号、>符号和！=会导致索引失效。特别的是如果对主键索引使用！=则不会使索引失效，如果对主键索引或者整数类型的索引使用<符号或者>符号不会使索引失效。（经erwkjrfhjwkdb同学提醒，不等于，包括&lt;符号、>符号和！，如果占总记录的比例很小的话，也不会失效）
在查询条件中使用IS NULL或者IS NOT NULL会导致索引失效。
字符串不加单引号会导致索引失效。更准确的说是类型不一致会导致失效，比如字段email是字符串类型的，使用WHERE email=99999 则会导致失败，应该改为WHERE email='99999'。
在查询条件中使用OR连接多个条件会导致索引失效，除非OR链接的每个条件都加上索引，这时应该改为两次查询，然后用UNION ALL连接起来。
如果排序的字段使用了索引，那么select的字段也要是索引字段，否则索引失效。特别的是如果排序的是主键索引则select * 也不会导致索引失效。
尽量不要包括多列排序，如果一定要，最好为这队列构建组合索引；
```

5.3 索引的优化

1、最左前缀

索引的最左前缀和和B+Tree中的“最左前缀原理”有关，举例来说就是如果设置了组合索引<col1,col2,col3>那么以下3中情况可以使用索引：col1，<col1,col2>，<col1,col2,col3>，其它的列，比如<col2,col3>，<col1,col3>，col2，col3等等都是不能使用索引的。

根据最左前缀原则，我们一般把排序分组频率最高的列放在最左边，以此类推。

2、带索引的模糊查询优化

在上面已经提到，使用LIKE进行模糊查询的时候，'%aaa%'不会使用索引，也就是索引会失效。如果是这种情况，只能使用全文索引来进行优化（上文有讲到）。

3、为检索的条件构建全文索引，然后使用

SELECT * FROM tablename MATCH(index_colum) ANGAINST(‘word’);

4、使用短索引

对串列进行索引，如果可能应该指定一个前缀长度。例如，如果有一个CHAR(255)的 列，如果在前10个或20 个字符内，多数值是惟一的，那么就不要对整个列进行索引。短索引不仅可以提高查询速度而且可以节省磁盘空间和I/O操作。

6 SQL

6.1 sql是什么

SQL (Structured Query Language)
是具有数据操纵和数据定义等多种功能的数据库语言，这种语言具有交互性特点，能为用户提供极大的便利，数据库管理系统应充分利用SQL语言提高计算机应用系统的工作质量与效率。SQL语言不仅能独立应用于终端，还可以作为子语言为其他程序设计提供有效助力，该程序应用中，SQL可与其他程序语言一起优化程序功能，进而为用户提供更多更全面的信息。

6.2 sql分类

n 数据定义语言：简称DDL(Data Definition Language)，用来定义数据库对象：数据库，表，列等。关键字：create，alter，drop等

n 数据操作语言：简称DML(Data Manipulation Language)，用来对数据库中表的记录进行更新。关键字：insert，delete，update等

n 数据控制语言：简称DCL(Data Control Language)，用来定义数据库的访问权限和安全级别，及创建用户。

n 数据查询语言：简称DQL(Data Query Language)，用来查询数据库中表的记录。关键字：select，from，where等

6.3 数据库类型

分类	类型名称	说明
整数类型	tinyInt	很小的整数
	smallint	小的整数
	mediumint	中等大小的整数
	int(integer)	普通大小的整数
小数类型	float	单精度浮点数
	double	双精度浮点数
	decimal（m,d)	压缩严格的定点数
日期类型	year	YYYY 1901~2155
	time	HH:MM:SS -838:59:59~838:59:59
	date	YYYY-MM-DD 1000-01-01~9999-12-3
	datetime	YYYY-MM-DD HH:MM:SS 1000-01-01 00:00:00~ 9999-12-31 23:59:59
	timestamp	YYYY-MM-DD HH:MM:SS 1970~01~01 00:00:01 UTC~2038-01-19 03:14:07UTC
文本、二进制类型	CHAR(M)	M为0~255之间的整数
	VARCHAR(M)	M为0~65535之间的整数
	TINYBLOB	允许长度0~255字节
	BLOB	允许长度0~65535字节
	MEDIUMBLOB	允许长度0~167772150字节
	LONGBLOB	允许长度0~4294967295字节
	TINYTEXT	允许长度0~255字节
	TEXT	允许长度0~65535字节
	MEDIUMTEXT	允许长度0~167772150字节
	LONGTEXT	允许长度0~4294967295字节
	VARBINARY(M)	允许长度0~M个字节的变长字节字符串
	BINARY(M)	允许长度0~M个字节的定长字节字符串

6.4 数据库基本操作

2.数据库的操作（CURD） 创建数据库（重点）

```
1. 创建数据库的语法
    * 基本语法 ：create database 数据库名称
    * 正宗语法 ：create database 数据库名称 character set 编码 collate 校对规则
2.校对规则：决定当前数据库的属性
```

3.查看数据库

```
1. show databases;
2. use 数据库名称
3. show create database 数据库名称
4. select database();       查询当前正在使用的数据库
```

4.删除数据库

```
1. drop database 数据库名称;
```

5.表的创建

1. 语法：
    create table 表名称(
      字段一   类型（长度） 约束
      );
2. 注意
  * 创建表的时候，后面用小括号，后面分号。
  * 编写字段，字段与字段之间使用都好，最后一个字段不能使用都好
  * 如果声明字符串数据的类型，长度都必须是指定的
  * 如果不指定数据的长度，又默认值的，int的长度为11

6.数据库的数据类型

1. 字符串型 VARCHAR ：长度是可变的。 CHAR ：长度是不可变的，不够的会用空格来补充
  * 长度不确定饿话使用VARCHAR
  * CHAR 适用于固定长度，节省空间，效率跟高
2. 大数据类型
  * BLOB ：字节
  * TEXT ：字符
3. 数值型
  * TINYINIT
  * SMALLINT
  * INT
  * BIGINT
  * FLOAT
  * DOUBLE
4. 逻辑性
  * BIT 在java中是true或者false 在数据库中bit的类型（1 或者 0）
5. 日期型（重点）
  * DATE ：只包含日期（年月日）
  * TIME ： 只包含时间（时分秒）
  * DATETIME ： 包含日期和时间，如果插入数据的时候，字复置为空，字段值九尾空了
  * TIMESTAMP ：包含日期和时间，如果插入数据的时候，这只字段的值为空，默认获取当前的系统时间

7.表中加入

1. 添加一个新字段
  * alter table 表名称 add 字段名称 数据类型 约束;
2. 修改字段的数据类型，长度或者约束
  * alter table 表名称 modify 字段名称 数据类型 约束;
3. 删除某一个字段
  * alter table 表名称 drop 字段名称;
4. 修改字段的名称
  * alter table 表名称 change 旧字段 新字段 数据类型 约束
5. 修改表的名称
  * rename table 旧表名字 to 新表名

8.表删除

```
1. 删除语句的语法
  * delete from 表 where 条件;
    * 如果不加where，默认删除所有数据
2. 删除所有的数据
  * delete from 表:
    * 一行一行的数据
    * 支持事物的操作，事务是数据库的特性
  * truncate 表:
    * 先把表删除，数据也删除，创建一个与原来一模一样的表
```

9.查询数据select（重点）

```
1. 查询语句的用法
  * select * from 表;
  * select 字段一，字段二，。。。from 表;
  * DISTINCT  ---去掉重复关键字

  * 查询语句中可以用as的关键字，起别名。
    * 别名的真正的用法，采用多表的查询，为了区分每一张表，表起别名
    * as的关键字可以不写，中间需要使用空格
```

10.where字句

```
1.常用符号
*   > < <= >= = <>(不等于)
* in ---代标范围

* like -- 模糊查询
  * '张_'     匹配张飞，张思
  * '张%'     匹配张飞，张一的
  * '%张'     以张结尾的
  * '%张%'    只要包含张的
```

11.order by 排序

```
1. 语法
  * order by 字段 asc | desc
      * asc   升序
      * desc  降序
2. 注意
  * order by自己 放在select句尾
```

12.聚集函数

```
1. count()      ---求数量
2. sum()        ---求和
  * sum函数可以忽略null
3. avg()        ---平均值
4. max()
5. min()
```

13.分组查询

```
1. group by
  * where 后面不能跟聚合函数  having后面可以跟随聚合函数
```

14.外键约束

```
1. 保证表结构的完整性
  * 通过设计外键的约束来避免表直接删除
  * alert table emp add foreign key () references dept ();
```

15.多表的设计

```
1. 一对一
2. 多对多
3. 一对多
```

16.多表查询内连接

```
1. 普通内链接
    * 语法  关键字 .... inner join ... on 条件;
    * 注意:
        * 再inner join 关键字之前写表一
        * 再inner join关键字之后写表2
        * on的后面写条件:
    * 语句:  select * from dept inner join emp on dept.did =emp.dno;
2. 隐士内链接
    * 语法:select ... from 表一,表二 where 表一.字段 = 表二.字段
    * 语句:select * from dept , emp on dept.did =emp.dno;
    * 指定字段 * 改为想要的字段名称
```

17.多表查询外连接

```
1. 左外链接
    * 语法:  ... 表1 left outer join 表2 on 表一.字段 = 表二.字段
    * 特点:按坐标,默认把坐标中的全部数据都查询出来,再查询有关联的数据。
2. 右外链接
    * 语法 ... 表一 right outer join 表1.字段 = 表2.字段
```

7 数据库优化

7.1 数据库优化

总体思路从以下几个方面:

1、选取最适用的字段属性

2、使用连接（JOIN）来代替子查询(Sub-Queries)

3、使用联合(UNION)来代替手动创建的临时表

4、事务（当多个用户同时使用相同的数据源时，它可以利用锁定数据库的方法来为用户提供一种安全的访问方式，这样可以保证用户的操作不被其它的用户所干扰）

5.锁定表（有些情况下我们可以通过锁定表的方法来获得更好的性能）

6、使用外键（锁定表的方法可以维护数据的完整性，但是它却不能保证数据的关联性。这个时候我们就可以使用外键）

7、使用索引

8、优化的查询语句（绝大多数情况下，使用索引可以提高查询的速度，但如果SQL语句使用不恰当的话，索引将无法发挥它应有的作用）

7.2 sql优化

(1)、explain出来的各种item的意义；
select_type
表示查询中每个select子句的类型
type
表示MySQL在表中找到所需行的方式，又称“访问类型”
possible_keys
指出MySQL能使用哪个索引在表中找到行，查询涉及到的字段上若存在索引，则该索引将被列出，但不一定被查询使用
key
显示MySQL在查询中实际使用的索引，若没有使用索引，显示为NULL
key_len
表示索引中使用的字节数，可通过该列计算查询中使用的索引的长度
ref
表示上述表的连接匹配条件，即哪些列或常量被用于查找索引列上的值
Extra
包含不适合在其他列中显示但十分重要的额外信息

(2)、profile的意义以及使用场景；
查询到 SQL 会执行多少时间, 并看出 CPU/Memory 使用量, 执行过程中 Systemlock, Table lock 花多少时间等等

8 数据库备份与还原

备份: 将当前已有的数据或者记录保留

还原: 将已经保留的数据恢复到对应的表中

为什么要做备份还原?

```
防止数据丢失: 被盗, 误操作
保护数据记录
```

数据备份还原的方式有很多种: 数据表备份, 单表数据备份, SQL备份, 增量备份.

数据表备份

不需要通过SQL来备份: 直接进入到数据库文件夹复制对应的表结构以及数据文件, 以后还原的时候,直接将备份的内容放进去即可.

数据表备份有前提条件: 根据不同的存储引擎有不同的区别.

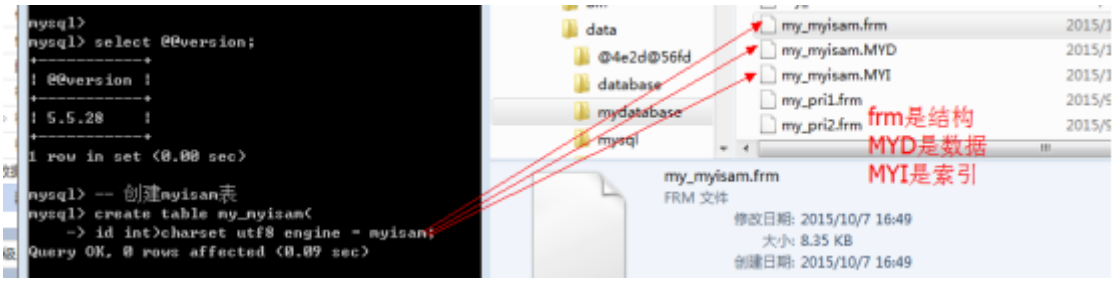
存储引擎: mysql进行数据存储的方式: 主要是两种: innodb和myisam(免费)

特点	Myisam	InnoDB	BDB	Memory	Archive
批量插入的速度	高	低	高	高	非常高
事务安全		支持	支持		
全文索引	支持	5.5版本支持			
锁机制	表锁	行锁	页锁	表锁	行锁
存储限制	没有	64TB	没有	有	没有
B树索引	支持	支持	支持	支持	
哈希索引		支持		支持	
集群索引		支持			
数据缓存		支持		支持	
索引缓存	支持	支持		支持	
数据可压缩	支持				支持
空间使用	低	高	低	N/A	非常低
内存使用	低	高	低	中等	低
支持外键		支持			

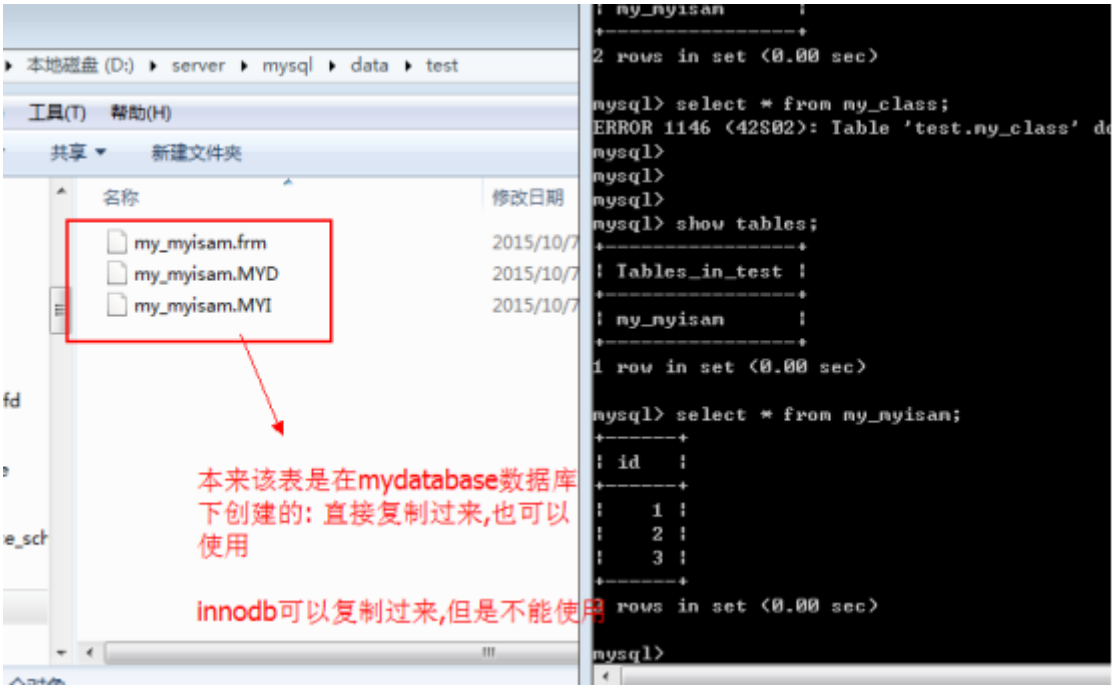
对比myisam和innodb: 数据存储方式

Innodb: 只有表结构,数据全部存储到ibdata1文件中

Myisam: 表,数据和索引全部单独分开存储



这种文件备份通常适用于myisam存储引擎: 直接复制三个文件即可, 然后直接放到对应的数据库下即可以使用.



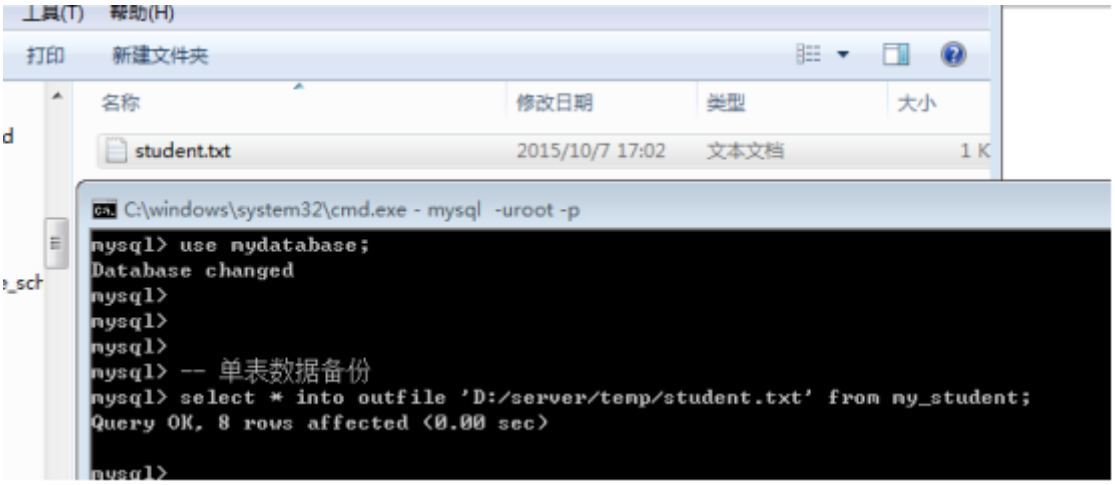
单表数据备份

每次只能备份一张表; 只能备份数据(表结构不能备份)

通常的使用: 将表中的数据进行导出到文件

备份: 从表中选出一部分数据保存到外部的文件中(outfile)

Select */字段列表 into outfile 文件所在路径 from 数据源; -- 前提: 外部文件不存在



高级备份: 自己制定字段和行的处理方式

Select */字段列表 into outfile 文件所在路径 fields 字段处理 lines 行处理 from 数据源;

Fields: 字段处理

Enclosed by: 字段使用什么内容包裹, 默认是'',空字符串

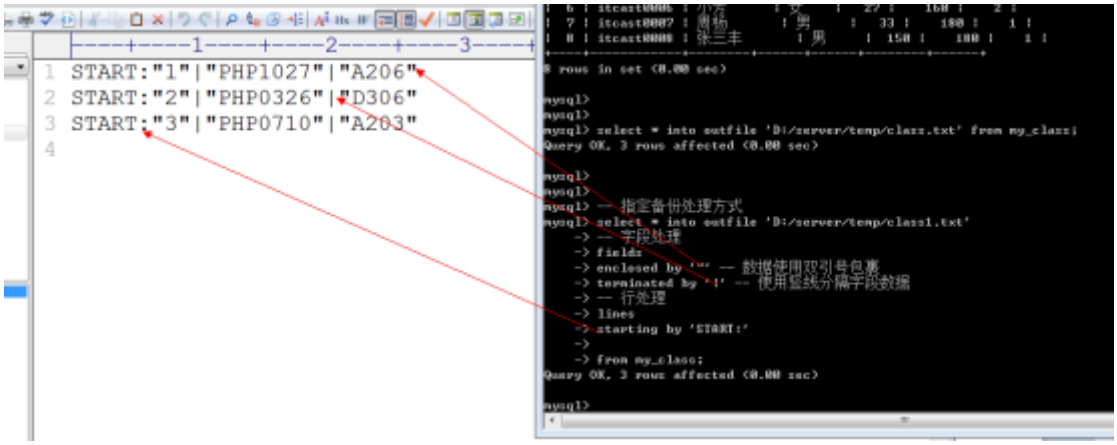
Terminated by: 字段以什么结束, 默认是"\t", tab键

Escaped by: 特殊符号用什么方式处理,默认是'\', 使用反斜杠转义

Lines: 行处理

Starting by: 每行以什么开始, 默认是'',空字符串

Terminated by: 每行以什么结束,默认是"\r\n",换行符



数据还原: 将一个在外部保存的数据重新恢复到表中(如果表结构不存在,那么sorry)

Load data infile 文件所在路径 into table 表名[(字段列表)] fields字段处理 lines 行处理; -- 怎么备份的怎么还原

SQL备份

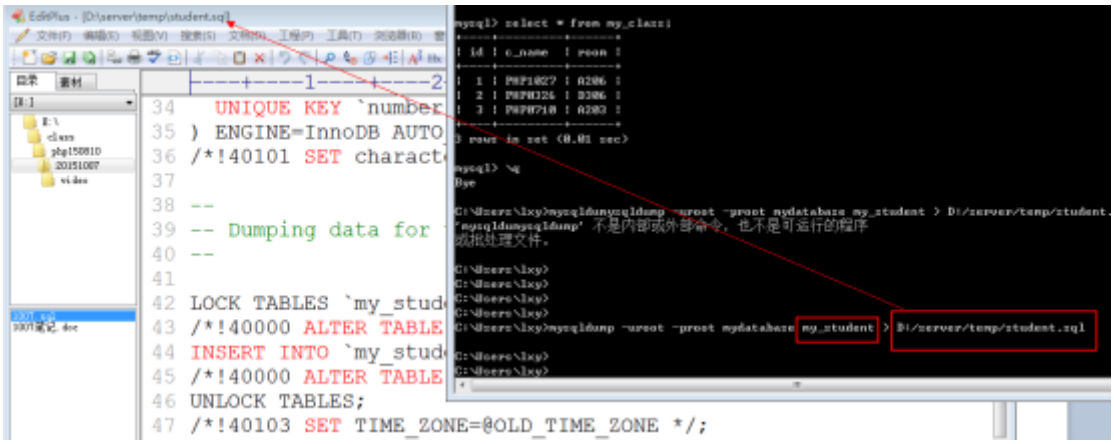
备份的是SQL语句: 系统会对表结构以及数据进行处理,变成对应的SQL语句, 然后进行备份: 还原的时候只要执行SQL指令即可.(主要就是针对表结构)

备份: mysql没有提供备份指令: 需要利用mysql提供的软件: mysqldump.exe

Mysqldump.exe也是一种客户端,需要操作服务器: 必须连接认证

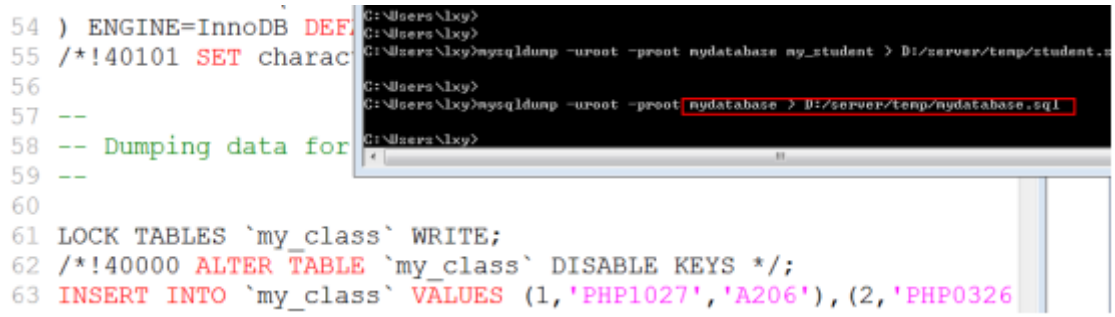
Mysqldump/mysqldump.exe -hPup 数据库名字 [数据表名字1[数据表名字2...]] > 外部文件目录(建议使用.sql)

单表备份



整库备份

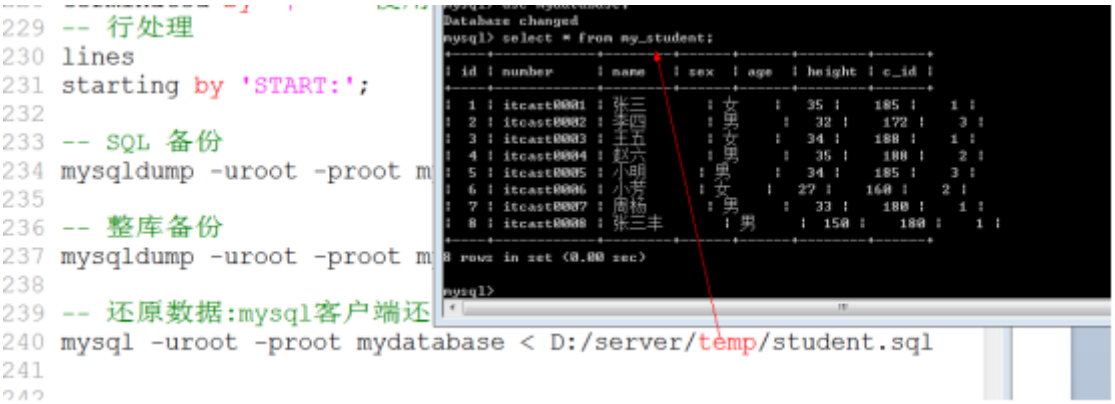
Mysqldump/mysqldump.exe -hPup 数据库名字 > 外部文件目录



SQL还原数据: 两种方式还原

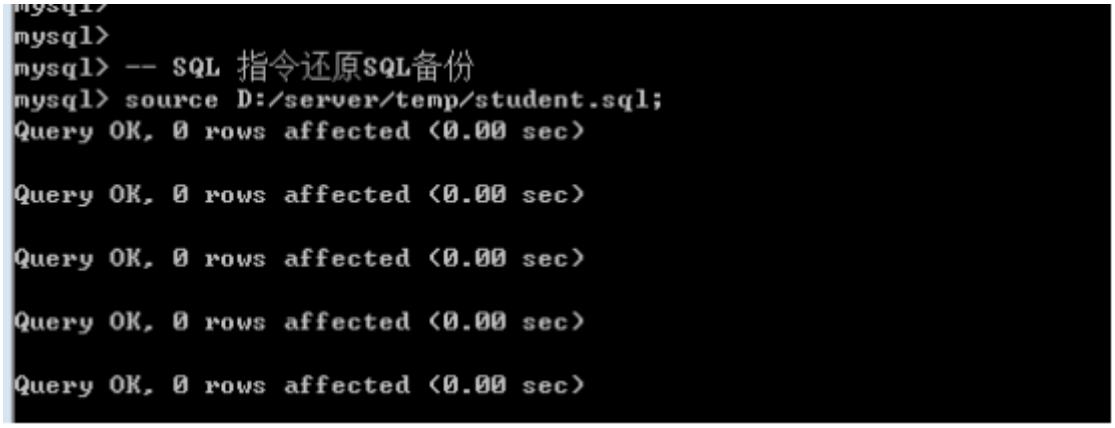
方案1: 使用mysql.exe客户端还原

Mysql.exe/mysql -hPup 数据库名字 < 备份文件目录



方案2: 使用SQL指令还原

Source 备份文件所在路径;



SQL备份优缺点

```
优点: 可以备份结构
缺点: 会浪费空间(额外的增加SQL指令)
```

增量备份

不是针对数据或者SQL指令进行备份: 是针对mysql服务器的日志文件进行备份

增量备份: 指定时间段开始进行备份., 备份数据不会重复, 而且所有的操作都会备份(大项目都用增量备份)