# mysql结构

## 系统默认字符集

show variables like 'character\_set\_%' 查看mysql是不是utf-8

character\_set\_connection | utf8   连接层字符集

character\_set\_database   | latin1           当前选中数据库的默认字符集

character\_set\_filesystem | binary

character\_set\_results    | utf8       查询结果字符集

character\_set\_server    | latin1       默认的内部操作字符集

character\_set\_system  | utf8   系统元数据(字段名等)字符集

character\_sets\_dir       | /usr/local/mysql-5.5.23-osx10.6-x86\_64/share/charsets/

查看到：

character\_set\_database和character\_set\_server依然是latin1的字符集，也就是说mysql后续创建的表都是latin1字符集的，不是utf8，会造成一些麻烦。所以有必要修改my.cnf，在修改my.cnf之前一定要关闭mysql进程，不然会遇到mysql的sock不能连接的问题。

### 设置默认字符集

临时设置：会话级别

SET NAMES 'utf8';

它相当于下面的三句指令：

SET character\_set\_client = utf8;

SET character\_set\_results = utf8;

SET character\_set\_connection = utf8;

## databases

 数据库(DataBase):简称DB，按照某一特定数据结构来组织，存储数据的仓库。

show databases; 显示所有的db；

show CREATE database wwl; 查询wwl数据库详细的创建情况

### 创建数据库

有三个属性：字符集和排序规则

IF NOT EXISTS ：这关键字用来判断数据库名字，有了这关子健：有同名也不会报错，（可有可无）

CHARACTER SET: 字符集

COLLATE： 排序规则 (后续有兴趣继续深究这里不写了)

CREATE DATABASE IF NOT EXISTS `wwl3` CHARACTER SET 'utf8' COLLATE 'utf8\_general\_ci';

## table

MySQL的数据表类型很多，其中比较重要的是MyISAM，InnoDB这两种。

### 表类型

#### **InnoDB**

InnoDB是MySQL4.0之后推出的一种比较新的数据表类型，这种类型是事务安全的。它与BDB类型具有相同的特性，它们还支持外健。InnoDB表格速度很快具有比BDB还丰富的特性，因此如果需要一个事物安全的存储引擎，建议使用它。如果你的数据执行大量的insert或者update，处于性能方面的考虑，同样应该使用InnoDb表。对于支持事物的InnoDB类型的表来说，影响速度的主要原因是autcommit设置是打开的，而且程序没有显式调用begin开始事物，导致每插入一条都自动提交，严重影响了速度。可以在执行sql前调用begin，多条sql想成一个事物，将大大提高性能。

#### MyISAM

MyISAM基于ISAM代码，可以说是ISAM的衍生品，不过增加了不少好用的扩展。它是MySQL的默认数据表类型，基于了传统的ISAM类型，ISAM是Indexed Sequential Access Method(有索引的顺序访问方法)的缩写，一般来说，它是存储记录和文件的标准方法。与其他存储引擎相比，MyISAM具有检查和修复表格的大多数工具。ISAM表格可以被压缩，而且它们支持全文索引，不过它们不是事务安全的，也不支持外健。如果事务会滚将会造成不完全会滚，从而不具备原子性。假如忽略事物以及访问并发性的话，并且需要执行大量的select检索语句的话，MyISAM将是最好的选择。

#### 区别

（1）事物处理：

MyISAM是非事物安全型的，而InnoDB是事物安全型的（支持事物处理等高级处理）；

（2）锁机制不同：MyISAM是表级锁，而InnoDB是行级锁。

表级锁：锁住整个表，行级锁：只锁表中字段一行

（3）select，update，insert，delete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

（4）查询表的行数不同：

MyISAM：select count(\*) from table,MyISAM只是简单的读出保存好的行数，注意的是，当count（\*）语句包含where条件时，两种表的操作是一样的。

InnoDB：InnoDB中不保存表的具体行数，也就是说，执行select count（\*）from table时，InnoDB要扫描一遍整个表来计算有多少行

5）外健支持：

MyISAM表不支持外健，而InnoDB支持

为什么MyISAM会比InnoDB的查询速度快？

InnoDB在做select的时候，要维护的东西比MyISAM引擎多很多；

1）数据块，InnoDB要缓慢，MyISAM只缓存索引快，这中间还有换进换出的减少；

2）InnoDB寻址要映射到块，再到行，MyISAM记录的直接是文件的offset，定位比InnoDB要快

3）InnoDB还需要维护MVCC一致；虽然你的场景没有，但他还是需要去检查和维护

MVCC（Muti-Version Concurrency Control）多版本并发控制

#### 应用场景

MyISAM适合：（1）做很多count的计算；（2）插入不频繁，查询非常频繁；（3）没有事务。

InnoDB适合：（1）可靠性要求比较高，或者要求事务；（2）表更新和查询都相当频繁，并且行锁定的机会比较大的情况。

### 创建表

use wwl; //先进入数据库

CREATE TABLE `user` (

`id` int(10) NOT NULL AUTO\_INCREMENT , //not null :不能为空，auto\_increment:自动增长

`name` varchar(10) NOT NULL DEFAULT '默认内容' COMMENT '这是注释关键字' ,

`user\_id` int(10) NOT NULL,

PRIMARY KEY (`id`) // PRIMARY KEY：设置主键

CONSTRAINT `ccc` FOREIGN KEY (`user\_id `) REFERENCES `users` (`userid`) //设置字段user\_id和users表的userid外键关联名字：ccc

)

ENGINE=InnoDB //引擎(表类型)

DEFAULT CHARACTER SET=utf8 //字符集编码

AUTO\_INCREMENT=1; //上面设置的自动增长：这里设置自动增长次数

注意：标点符号的使用：`user`这里的标点符号不是单引号而是键盘数字1旁边的

### 约束

#### 主键约束

唯一索引：此表中唯一的，不能有重复，一般用于外键和主键

#### 外键约束

MySQL有两种常用的引擎类型:MyISAM和InnoDB。目前只有InnoDB引擎类型支持外键约束。

1. CASCADE: 从父表中删除或更新对应的行，同时自动的删除或更新自表中匹配的行。ON DELETE CANSCADE和ON UPDATE CANSCADE都被InnoDB所支持。

2. RESTRICT(默认值): 拒绝删除或者更新父表。指定RESTRICT（或者NO ACTION）和忽略ON DELETE或者ON UPDATE选项的效果是一样的。

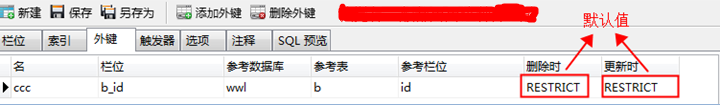
3. SET NULL: 从父表中删除或更新对应的行，同时将子表中的外键列设为空。注意，这些在外键列没有被设为NOT NULL时才有效。ON DELETE SET NULL和ON UPDATE SET SET NULL都被InnoDB所支持。

4. NO ACTION: InnoDB拒绝删除或者更新父表。

5. SET DEFAULT: InnoDB目前不支持。

外键约束使用步骤：

创建好表后设置外键默认值 ON UPDATE RESTRICT ON DELETE RESTRICT更新和删除都是RESTRICT默认值



例子 一个表是姓名表(主表)，另一个表是地址表 (从表)

创建子表a:3个字段id、地址address、外键b\_id, ,维护着b表的主键，有外键就属于子表

create table `a`(id int(25) not null AUTO\_INCREMENT,address varchar(100), b\_id int(25) not null,

PRIMARY KEY (`id`),

CONSTRAINT `ccc` FOREIGN KEY (`b\_id`) REFERENCES `b` (`id`) 外键的设置

)ENGINE=InnoDB DEFAULT CHARACTER SET=utf8 AUTO\_INCREMENT=1;

创建父亲表b：两个字段 主键id 和姓名name

create table `b`(id int(25) not null AUTO\_INCREMENT,name varchar(20),

PRIMARY KEY (`id`) // PRIMARY KEY：设置主键

) ENGINE=InnoDB DEFAULT CHARACTER SET=utf8 AUTO\_INCREMENT=1;

测试

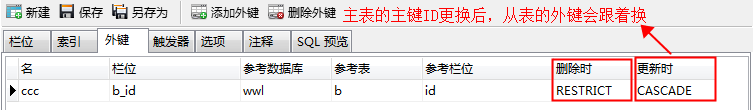
插入：必须先插入父亲表b(id有值)，不然子表无法插入：提示父亲表不能为空

删除：必须先删除子表，才能删除父亲表

修改：子表外键不能修改父表没有的主键ID

1）(常用)父表更新时子表也必须更新(外键)，父表删除时如果子表有匹配的项，删除失败；

ON UPDATE CASCADE ON DELETE RESTRICT



更换外键约束命令

--先删除外键

ALTER TABLE a DROP FOREIGN KEY ccc

--后添加外键

ALTER TABLE `a` ADD CONSTRAINT `ccc` FOREIGN KEY ( `b\_id` ) REFERENCES `b` ( `id` )

ON DELETE RESTRICT ON UPDATE CASCADE

结合上面默认的例子：最大变化就是：父表b主键id更换，子表外键id跟这主键id关联的也会跟着变

2）父表更新时子表也更新，父表删除时子表匹配的项也删除

ON UPDATE CASCADE ON DELETE CASCADE

结论：

1、如果子表试图创建一个在父表中不存在的外键值，InnoDB会拒绝任何INSERT或UPDATE操作。

2、如果父表试图UPDATE或者DELETE任何子表中存在或匹配的外键值，最终动作取决于外键约束定义中的ON UPDATE和ON DELETE选项。

# SQL语句

SQL语句最常见的分类一般就是3类：

DDL：数据定义语言（CREATE，ALTER，DROP）

DML:数据操作语言（SELECT，INSERT，DELETE，UPDATE）

DCL：数据控制语言（GRANT，REVOKE，COMMIT，ROLLBACK）

## DML

CRUD是指在做计算处理时的增加(Create)、读取查询(Retrieve)、更新(Update)和删除(Delete)几个单词的首字母简写。crud主要被用在描述软件系统中数据库或者持久层的基本操作功能。

### Retrieve【查询】

#### 单表查询

#### 联表查询

(左连接)LEFT JOIN：先查询左边表(主表)所有的值，匹配右边表(从表)的值，无则null显示

语句格式：SELECT 两表要显示的字段FROM`主表 ` LEFT JOIN `从表` ON 条件语句 后续可以where

SELECT b.billsdate, b.money, d.otherDescribe FROM `bills` b LEFT JOIN `describes` d ON b.billsid = d.bills\_Id

运用场景：如查询用户下了几张单

注意 左连接：右边表(从表)查询出来的属性不会有重复的，左边表(主表)会有重复的

(内连接)INNER JOIN：查询两表之间相等的值，匹配显示两边表，无则都不显示

语句格式：SELECT 两表要显示的字段FROM`主表 ` inner JOIN `从表` ON 条件语句 后续可以where

SELECT b.billsdate, b.money, d.otherDescribe FROM `bills` b INNER JOIN `describes` d ON b.billsid = d.bills\_Id

(右连接) RIGHT JOIN：顾名思义就和左连接相反

## DDL

## Other

导入sql语句名利

先进入数据库：use wwl; 然后使用命令：source 路径/文件名.sql;

# 主从同步数据库配置

作用：减轻数据库的压力

创建两个服务器，主和从；

主服务器：用来进行增删改操作

从服务器：用来查询操作(进行配置后’从’数据库会同步’主’的数据)

## 第一步 ‘主’的mysql配置

1、在mysql中my.ini中标签[mysqld]配置如下：

‘主’的mydql版本是5.6的

[mysqld]

server-id=1

log-bin=master-bin

log-bin-index=master-bin.index

character-set-server=utf8

2、进入mysql验证命令：show master status显示如下：



3、创建用户用来和‘从’的mysql进行交互用

1、grant replication slave on \*.\* to ‘wwl’@’ 192.168.0.104’ identified by ‘123456’

意思是创建账号wwl用于来连接’从’mysql( 192.168.0.104)的操作，权限是\*.\*：所有库所有表的

2、flush privileges ; 刷新一下让此账户生效

## 第二步 ‘从’的mysql配置

1、在mysql中my.inf中标签[mysqld]配置如下：

注意：我的’从’数据库mysql是5.7版本的，my.inf位置在：/etc/mysql/my.inf

打开后发生都是空的，没有[client]、[mydql]、[mysqld]三个标签的：自己添加进去即可

[client]

default\_character\_set=utf8

[mysql]

default\_character\_set=utf8

[mysqld]

server-id=2 这三个是”从的配置”：

relay-log-index=slave-relay-bin.index

relay-log=slave-relay-bin

character\_set\_server=utf8

2、连接‘主’的mysql操作

命令如下

1、change master to master\_host='192.168.0.101',master\_port=3306,master\_user='wwl',master\_password='123456',

master\_log\_file='master-bin.000001',master\_log\_pos=0;

上面意思是：指定主的ip地址、端口、账号、密码

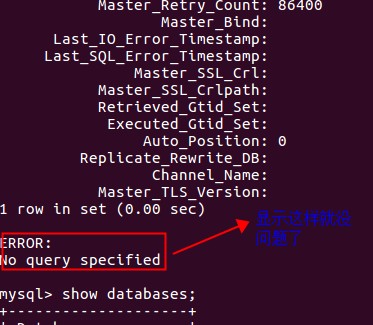
master\_log\_file：在主数据命令下：show master status 可以获取到该数据

master\_log\_pos：从log中哪个位置开始复制主的数据，以后’从’服务器挂掉可以指定哪里复制主的数据

2、start slave; 开启主从跟踪，stop slave; 关闭主从跟踪

3、show slave status \G; \G的参数意思是竖向显示

**第三步命令最后显示没有任何错误即成功连接上了**



## 第三步 测试操作

在’主’服务器进行添加数据库

cteate database ccc; 创建数据库 ccc，创建成功后，可以看到’从’也会跟着自动创建了数据库ccc

# 事务

## 四大特性

四大特性 ACID

**1、原子性(Atomicity)** 事务是一个不可分割的工作单位，多条语句操作数据库要么都成功，要么都失败

**2、一致性(Consistency)** 事务前后数据的完整性必须保持一致，如： 不要有员工却没有部门表

**3、持久性(Durability)** 一个事务一旦提交，数据库中的数据改变就是永久性，哪怕数据库故障也不受影响

**4、隔离性(Isolation)：**当多个用户并发访问数据库操作同一张表时，为每一个用户开启事务，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。

## 事务安全问题

隔离性(Isolation) 需要自己操心的，当并发操作数据库有下面3个情况：

1、两个线程并发修改或增加: 数据库都会加锁(排他锁)，等一个修改完，另一个才能进行修改，没有线程安全问题

2、两个线程并发查询:没有线程安全问题(数据都不变)

3、两个线程一个修改或插入，一个查询：就有可能会导致下面3个问题发生

**脏读：**一个事务读到另一个事务未提交的事务

B事务进行修改或提交，还未commit(提交事务)，A事务就能查询到B事务未commit的数据

**不可重复读：**在一个事务内读取表中的某一行数据，多次读取结果不同**(线程并发一个修改一个查询)**

A事务先查询数据是2 ,并发中B事务把A查询的数据2修改为3进行了commit;A还没有commit：再次查询变成3：这就是不可重复读.

在某些情况下，不可重复读并不是问题，比如我们多次查询某个数据当然以最后查询得到的结果为主。但在另一些情况下就有可能发生问题，例如对于同一个数据A和B依次查询就可能不同，A和B就可能打起来了……

不可重复读和脏读的区别是:脏读是某一事务读取了另一个事务未提交的脏数据，而不可重复读则是读取了前一事务提交的数据

**虚读(幻读)：**在一个事务内查询到了别的事务插入的数据，导致前后读取不一致**(线程并发一个插入一个查询)**

A事务查询到数据只有一个2，并发中B事务插入了一条3进行了commit;A还没有commit：再次查询变成了2,3的数据，这就是虚读(幻读)

另一个种理解

幻读是事务非独立执行时发生的一种现象。事务A对一个表中所有的行的某个数据项做了从“1”修改为“2”的操作，这时事务B又对这个表中插入了一行数据项，而这个数据项的数值还是为“1”并且提交给数据库。而操作事务A的用户如果再查看刚刚修改的数据，会发现还有一行没有修改，其实这行是从事务B中添加的，就好像产生幻觉一样，这就是发生了幻读。

幻读和不可重复读都是读取了另一条已经提交的事务（这点就脏读不同），所不同的是不可重复读查询的都是同一个数据项，而幻读针对的是一批数据整体（比如数据的个数）。

解决方案：脏读、虚读、幻读的问题数据库提供了四大隔离级别

Read ybcimmutted 不防止任何隔离性问题。

Read committed 可以防止脏读问题，但是不能防止不可重复读/虚读(幻读)的问题

Repeatable read 可以防止脏读、不可重复读问题，不能防止虚读(幻读)问题

A事务查询到数据是2 ,B事务进行修改跟A一样的数据为3进行了commit，虽然是真的修改了数据库内容，但A还没有commit：再次查询还是2，这就是**解决了不可重复读问题**

Serializable 数据库被设计为单线程数据库，可以防止上述所有问题，但是效率最低

A事务查询到数据的期间，B事务要进行插入或修改操作，都必须等A事务commit结束后才能操作

## 锁机制

**共享锁：**两个条件：①Serializable隔离级别下、② 查询操作，才会加共享锁

特点：共享锁和共享锁可以共存(两个查询可以共同操作)，但是共享锁和排他锁不能共存；

**排他锁：**两个条件：①所有的隔离级别、②增删改操作 。都会加排它锁：只要有一个线程进行了增删改操作，其它线程就没法进行增删改

特点：和任意的锁都不能共存