MYSQL 九阳神功

开发前的基础

摘要

一个人静静坐在电脑面前写代码的感觉，那是什么感觉？那是武林高手闭关修炼的感觉。

c3gen_小胖

2016-12-9　　于武汉

MySQL 九阳神功

目录

前言 2
第一章 mysql 的安装 3
1.1 mysql 版本的说明和下载 3
1.2 mysql 的硬安装和软安装 4
第二章 sql 语句基础 5
2.1 mysql 基础知识 5
2.2 mysql 的文件结构 6
2.3 表之间的关系 7
一对一 1:n 7
一对多 1:m 8
多对多 n:m 9
2.4 SQL 操作 10
2.4.1 库操作 10
2.4.2 表操作 15
2.4.3 数据操作 20
2.5 mysql 基本语法 23
2.5.1 数据类型 23
2.5.2 列属性 28
2.5.3 字段属性 29
2.5.4 常用的查询子句 41
2.5.5 联合查询 union 57
2.5.5 连接查询 join 60
第三章 sql 高级部分 66
3.1 变量 66
3.1.1 系统变量 66
3.1.2 自定义变量 68
3.2 函数 70
3.2.1 系统函数 70
3.2.2 自定义函数 73
3.3 代码执行结构 76
分支结构 76
循环结构 77
3.4 视图 78
创建视图 79
查看视图 79
使用视图 80
修改视图 80
删除视图 81

视图数据操作 insert,update,delete.....81
3.5 事务和锁.....83
锁(un)lock table 85
事务操作.....86
事务原理.....89
回滚点.....89
自动事务处理.....90
3.6 存储过程.....92
创建过程.....92
查看过程.....92
调用存储过程.....93
修改存储过程&删除存储过程 93
过程参数.....94
3.7 触发器.....96
创建触发器.....97
查看触发器.....98
使用触发器.....99
修改触发器&删除触发器 99
触发器记录.....100
3.8 备份与还原.....101
数据表备份.....101
单表数据备份.....103
SQL 备份 105
定时任务备份.....111
3.9 常见请求编码乱码(重要) 114

前言

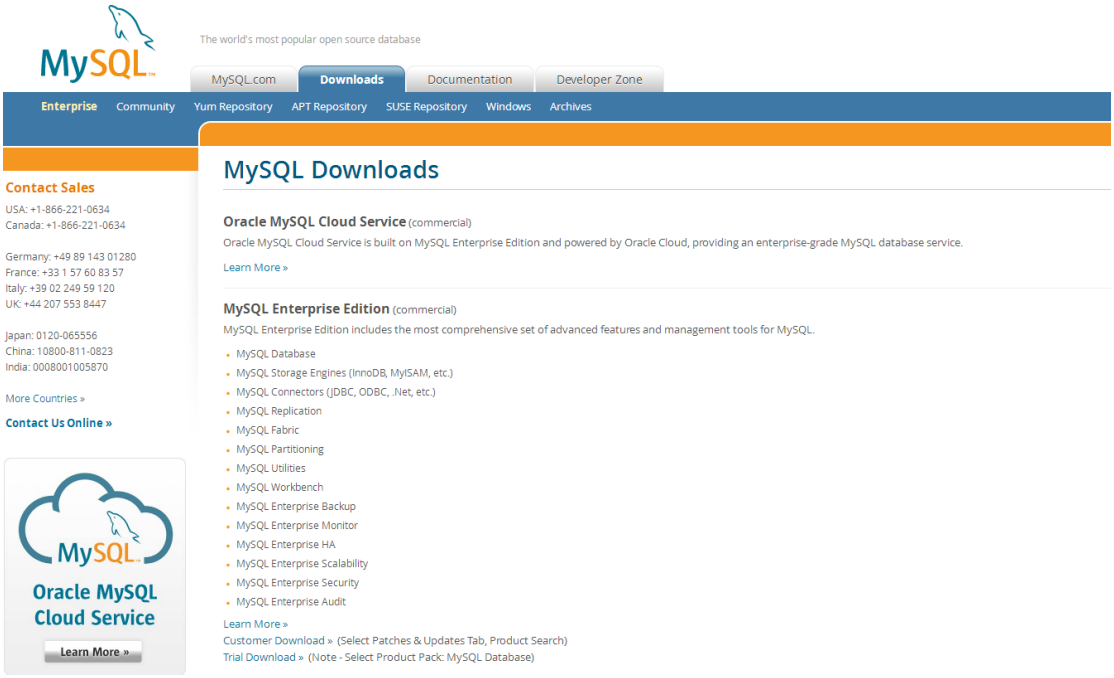
现在，这本秘籍已经出现在你面前，由小胖业余编写，根据自己的一点点开发经验和官网的api(点击可以访问)，如果你任何问题，可以私信我 Email：bugcoder@163.com,一起学习交流，如果你是大神，或者 mysql 了解的很深了，那么这基础部分你就当和我一起复习复习。这本秘籍的第一版主要是基础部分，也就是开发前的准备工作。

工欲善其事必先利其器，首先还是说下 mysql 的下载和安装。

第一章 mysql 的安装

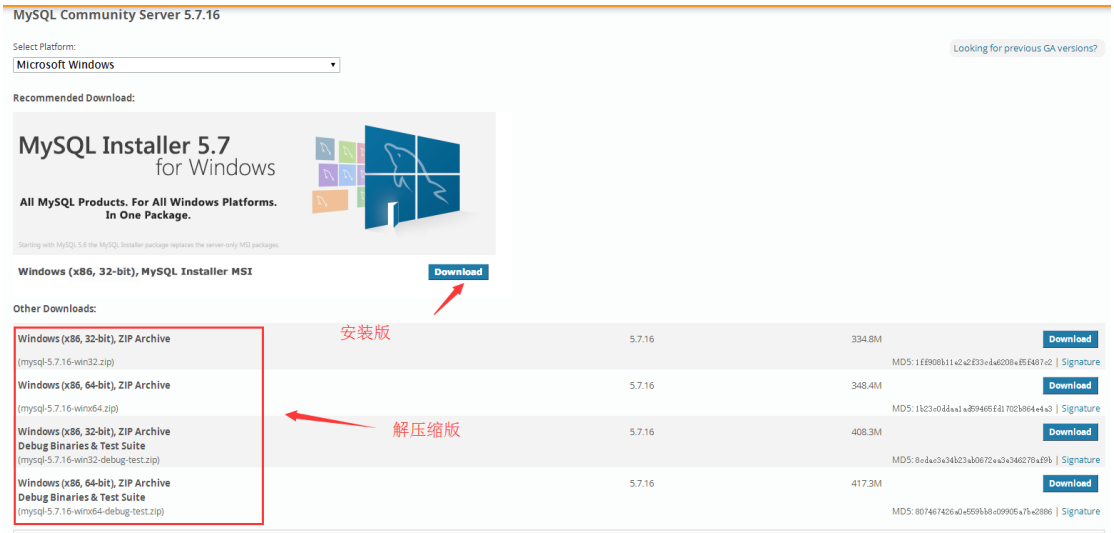
1.1 mysql 版本的说明和下载

http://www.mysql.com/ 官网地址附上。有时候可能访问不了。



- Oracle MySQL Cloud Service (commercial)
  mysql 的云服务，商业版，服务器在 oracle 的云上，企业级的云数据库
- MySQL Enterprise Edition (commercial)
  MySQL 企业版（商业版），这版本包括了 mysql 的高级功能和工具，比如 mysql 的企业备份，企业监控器、审计等高级功能。
- MySQL Cluster CGE (commercial)
  MySQL 集群 CGE（商业），MySQL 集群是一个实时的开放源码的事务数据库设计为快速，总是在获得高吞吐量的条件下的数据。这版本包括 mysql 集群和集群管理以及 mysql 企业版的所有内容。
- MySQL Community Edition (GPL)
  MySQL 社区版（GPL），这个版本是我们通常开发和项目使用的免费版。mysql 的一些基本功能都有。
  notes：商业版和免费版的区别在于，商业版 oracle 官方提供技术支持，提供了更多的功能，免费版就没有官方的技术支持了，在使用上，其实两者差别不大。
  ******我们下载这个版本

1.2 mysql 的硬安装和软安装



为了方便简单，我们选择安装的，下载后一步步的下一步就可以。
默认安装版的配置文件以及 data 文件在 C:\ProgramData\MySQL\MySQL Server 5.7 默认隐藏了。需要在查看文件的选项里打开隐藏文件夹。
/**********免安装配置如下**********/
*****解压缩版安装

免安装版配置主要包括以下几步：

1) 第一步解压文件，随便放到一个文件夹下面，如：D:\mysql-5.7.15-winx64

2) 配置环境变量，在系统变量 path 后面追加 D:\mysql-5.7.15-winx64\bin ，我的习惯是定义一个 MYSQL_HOME = D:\mysql-5.7.15-winx64 ，然后在 path 里面加上 &MYSQL_PATH&\bin,这样可操作性更高，推荐。

3) 复制 my-default.ini,命名为 my.ini,打开 my.ini,加入

```
basedir = D:\mysql-5.7.15-winx64
datadir = D:\mysql-5.7.15-winx64\data
server_id =mysql
```

port =3306 （注意修该内容将前面的#去掉）

4) 以管理员身份运行 cmd,进入到 D:\mysql-5.7.15-winx64\bin(输入 D:回车*** 输入 cd D:\mysql-5.7.15-winx64\bin 回车)

5) 这一步很重要创建 data 文件夹，以前有些版本有 data 文件夹，但现在没有，在 cmd 中 D:\mysql-5.7.15-winx64\bin 后面输入

mysqld --initialize-insecure --user=mysql

这样就创建了 data 文件夹，

6) 执行 mysqld -install 安装服务，安装成功会得到提示：Service successfully installed

7) 启动服务，执行 net start mysql

/*****************************登录*************************************/

8) 同样在 cmd 中进入 D:\mysql-5.7.15-winx64\bin 这个文件，执行 mysql -u root -p

出现 enter password：

直接回车，以 root 身份登录。

9) 修改密码，进入 D:\mysql-5.7.15-winx64\bin，执行 mysqladmin -uroot -p password 命令，提示输入原来的密码，

原来密码为空，直接回车，输入新密码，然后确认密码，就可以登录了。

*****如果你解压缩版安装遇到，启动服务时提示，本地计算机上的 mysql 服务器启动后停止的错误，那么你可以看看我的博客。

碍于篇幅，我也不讲什么数据库分类，什么 mysql 的历史，这些你如果要了解可以百度，谷歌。

以下例子都是用的安装版来演示。进入安装版的/bin 目录下 也可以执行 mysql 命令。

第二章 sql 语句基础

2.1 mysql 基础知识

连接和操作 mysql，常见的有两种方式，cmd 命令行直连和客户端工具代理。(主要在 cmd 命令窗口下操作，sql 语句是熟练出技巧的，客户端可以方便你开发，但是学习还是 cmd 把。)

我使用的是安装版，cmd 进入 bin 目录 ，然后执行 mysql –uc3gen –p 输入密码进可以

/*****************设置环境变量参照上面的免安装版的设置************************/

a) 访问数据库

密码设置成功后，执行 mysql –uroot –p 然后输入密码 进入 mysql 的控制台

```
C:\Program Files\MySQL\MySQL Server 5.7\bin>mysql -uroot -p
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 6
Server version: 5.7.14 MySQL Community Server (GPL)

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

查看一下 目前有多少数据库

执行命令 show databases;

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
4 rows in set (0.00 sec)
```

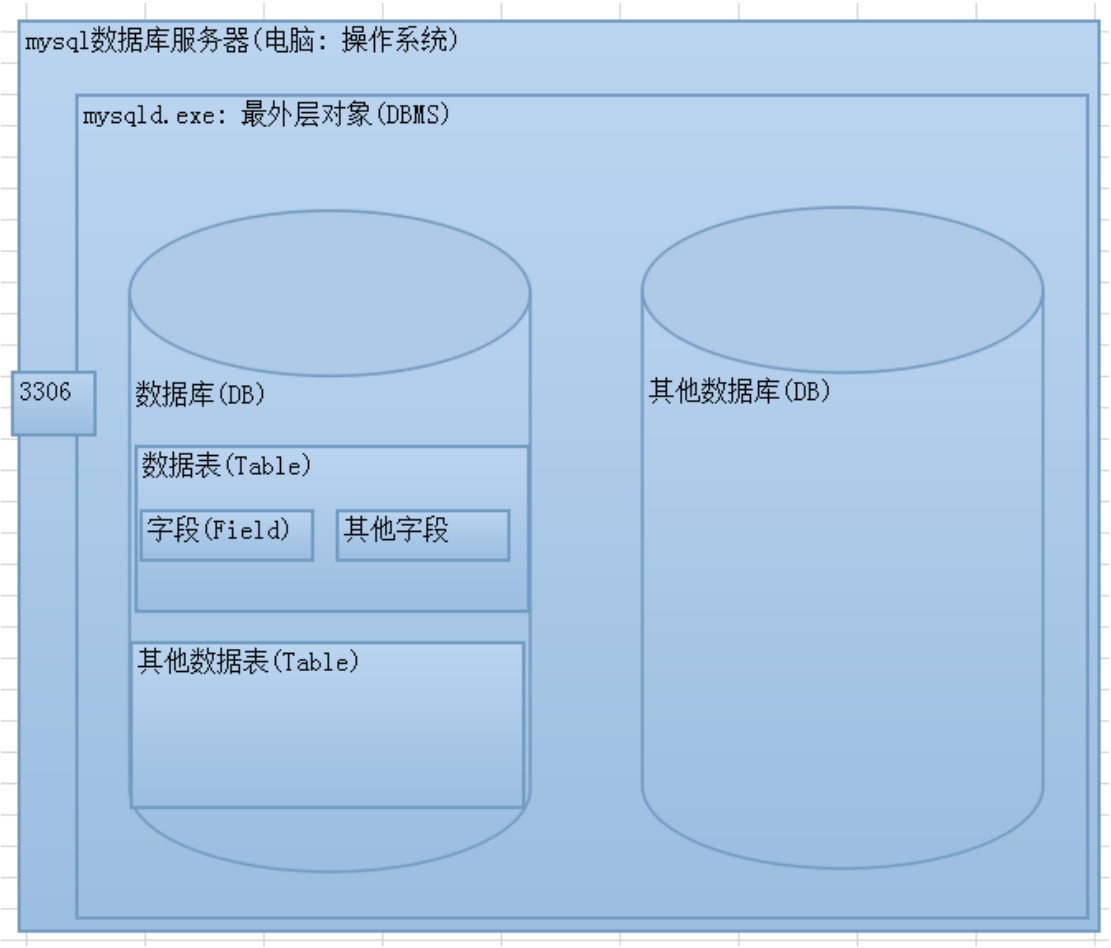
退出当前会话窗口，当然你是可以直接关闭 cmd 的窗口的，但是如果紧紧是退出当前会话，那么你可以执行 命令 exit/ quit / \q

```
mysql> \q
Bye
```

2.2 mysql 的文件结构

为了后面的学习，先对 mysql 的文件结构或者说内部对象有个整体认识。

将 mysql 服务器内部对象分成了四层: 系统(DBMS) -> 数据库(DB) -> 数据表(Table) -> 字段(field)



2.3 表之间的关系

学习 sql 语句之前，应该对表之间的关系有一定的了解：

一对一 1:n

一对一：一张表的一条记录一定只能与另外一张表的一条记录进行对应；反之亦然.

学生表：姓名,性别,年龄,身高,体重,婚姻状况，籍贯，家庭住址,紧急联系人

Id(P)	姓名	性别	年龄	体重	身高	婚姻	籍贯	住址	联系人

表设计成以上这种形式：符合要求. 其中姓名,性别,年龄,身高,体重属于常用数据；但是婚姻,籍贯,住址和联系人属于不常用数据. 如果每次查询都是查询所有数据,不常用的数据就会影响效率，实际又不用.

解决方案：将常用的和不常用的信息分离存储,分成两张表
常用信息表

Id(P)	姓名	性别	年龄	体重	身高
1					

不常用信息表: 保证不常用信息与常用信息一定能够对应上: 找一个具有唯一性(确定记录)的字段来共同连接两张表

Id(P)	婚姻	籍贯	住址	联系人
2				
1				

一个常用表中的一条记录: 永远只能在一张不常用表中匹配一条记录;反过来,一个不常用表中的一条记录在常用表中也只能匹配一条记录: 一对一的关系

一对多 1:m

一对多: 一张表中有一条记录可以对应另外一张表中的多条记录; 但是返回过, 另外一张表的一条记录只能对应第一张表的一条记录. 这种关系就是一对多或者多对一.

母亲与孩子的关系: 母亲,孩子两个实体

妈妈表

ID(P)	名字	年龄	性别

孩子表

ID(P)	名字	年龄	性别

以上关系: 一个妈妈可以在孩子表中找到多条记录(也有可能是一条); 但是一个孩子只能找到一个妈妈: 是一种典型的一对多的关系.

但是以上设计: 解决了实体的设计表问题, 但是没有解决关系问题: 孩子找不出妈,妈也找不到孩子.

解决方案: 在某一张表中增加一个字段,能够找到另外一张表的中记录: 应该在孩子表中增加一个字段指向妈妈表: 因为孩子表的记录只能匹配到一条妈妈表的记录.

妈妈表

ID(P)	名字	年龄	性别

孩子表

ID(P)	名字	年龄	性别	妈妈 ID
				妈妈表主键

多对多 n:m

多对多: 一张表中(A)的一条记录能够对应另外一张表(B)中的多条记录; 同时 B 表中的一条记录也能对应 A 表中的多条记录: 多对多的关系

老师教学: 老师和学生
老师表

T_ID(P)	姓名	性别
1	A	男
2	B	女

学生表

S_ID(P)	姓名	性别
1	张三	男
2	小芳	女

以上设计方案: 实现了实体的设计, 但是没有维护实体的关系.
一个老师教过多个学生; 一个学生也被多个老师教过.

解决方案: 在学生表中增加老师字段: 不管在哪张表中增加字段, 都会出现一个问题: 该字段要保存多个数据, 而且是与其他表有关系的字段, 不符合表设计规范: 增加一张新表: 专门维护两张表之间的关系
老师表

T_ID(P)	姓名	性别
1	A	男
2	B	女

学生表

S_ID(P)	姓名	性别
1	张三	男
2	小芳	女

中间关系表: 老师与学生的关系

ID	T_ID(老师)	S_ID(学生)
1	1	1

2	1	2
3	2	1
4		

增加中间表之后：中间表与老师表形成了一对多的关系：而且中间表是多表,维护了能够唯一找到一表的关系；同样的,学生表与中间表也是一个一对多的关系：一对多的关系可以匹配到关联表之间的数据.

学生找老师：找出学生 id -> 中间表寻找匹配记录(多条) -> 老师表匹配(一条)
老师找学生：找出老师 id -> 中间表寻找匹配记录(多条) -> 学生表匹配(一条)

2.4 SQL 操作

在对表之间的关系了解后，来了解 sql 的操作，sql 语句按照操作来划分，可以分为三个部分，库操作、表操作、数据操作。

2.4.1 库操作

在进行数据库的操作之前，我们需要注意一下数据库服务器的编码和字符集，这经常会造成你的数据库查询中文字符是乱码。
执行命令 show variables like '%char%';

```
mysql> show variables like '%char%';
+--------------------------+----------------------------------------------------------+
| Variable_name            | Value                                                    |
+--------------------------+----------------------------------------------------------+
| character_set_client     | gbk                                                      |
| character_set_connection | gbk                                                      |
| character_set_database   | latin1                                                   |
| character_set_filesystem | binary                                                   |
| character_set_results    | gbk                                                      |
| character_set_server     | latin1                                                   |
| character_set_system     | utf8                                                     |
| character_sets_dir       | C:\Program Files\MySQL\MySQL Server 5.7\share\charsets\  |
+--------------------------+----------------------------------------------------------+
8 rows in set, 1 warning (0.04 sec)
```

我们发现默认的有些乱，为了保证后续的开发操作不出现乱码，我们设置所有的字符集为 utf8

以下操作只是临时的，也就是对当前的会话有效，如果以后不想再修改，就加上关键字 global 比如 set global character_set_client=utf8;
执行命令：

```
set character_set_client=utf8;
set character_set_connection=utf8;
set character_set_database=utf8;
set character_set_results=utf8;
set character_set_filesystem=utf8;
```

set character_set_server=utf8;

```
mysql> set character_set_client=utf8;
Query OK, 0 rows affected (0.00 sec)

mysql> set character_set_connection=utf8;
Query OK, 0 rows affected (0.00 sec)

mysql> set character_set_database=utf8;
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> set character_set_filesystem=utf8;
Query OK, 0 rows affected (0.00 sec)

mysql> set character_set_results=utf8;
Query OK, 0 rows affected (0.00 sec)

mysql> set character_set_server=utf8;
Query OK, 0 rows affected (0.00 sec)
```

设置完成后，再次查看字符集

```
mysql>  show variables like '%char%';
+--------------------------+----------------------------------------------------------+
| Variable_name            | Value                                                    |
+--------------------------+----------------------------------------------------------+
| character_set_client     | utf8                                                     |
| character_set_connection | utf8                                                     |
| character_set_database   | utf8                                                     |
| character_set_filesystem | utf8                                                     |
| character_set_results    | utf8                                                     |
| character_set_server     | utf8                                                     |
| character_set_system     | utf8                                                     |
| character_sets_dir       | C:\Program Files\MySQL\MySQL Server 5.7\share\charsets\  |
+--------------------------+----------------------------------------------------------+
8 rows in set, 1 warning (0.00 sec)
```

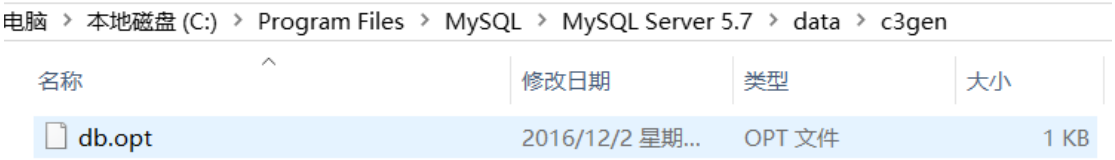
准备工作做完了，然后开始库操作。

i. 创建数据库

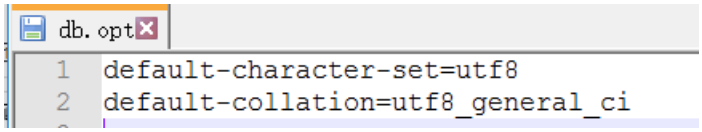
create database c3gen;

```
mysql> create database c3gen;
Query OK, 1 row affected (0.00 sec)
```

创建完成后，可以在 data 目录下发现这个创建的数据库。



进入后发现 有个 db.opt 文件，这个文件保存了数据库的基本设置。用 notepad++打开查看。



default-character-set 表示数据库的字符集。

default-collation 表示校对集，也就是校对的字符集。

****校对集: 数据比较的方式

校对集有三种格式

_bin: binary,二进制比较，取出二进制位,一位一位的比较，区分大小写

_cs: case sensitive,大小写敏感，区分大小写

_ci: case insensitice,大小写不敏感,不区分大小写

查看数据库所支持的校对集: show collation;

```
mysql> show collation;
+--------------------------+----------+-----+---------+----------+---------+
| Collation                | Charset  | Id  | Default | Compiled | Sortlen |
+--------------------------+----------+-----+---------+----------+---------+
| big5_chinese_ci          | big5     |   1 | Yes     | Yes      |       1 |
| big5_bin                 | big5     |  84 |         | Yes      |       1 |
| dec8_swedish_ci          | dec8     |   3 | Yes     | Yes      |       1 |
| dec8_bin                 | dec8     |  69 |         | Yes      |       1 |
| cp850_general_ci         | cp850    |   4 | Yes     | Yes      |       1 |
| cp850_bin                | cp850    |  80 |         | Yes      |       1 |
| hp8_english_ci           | hp8      |   6 | Yes     | Yes      |       1 |
| hp8_bin                  | hp8      |  72 |         | Yes      |       1 |
| koi8r_general_ci         | koi8r    |   7 | Yes     | Yes      |       1 |
| koi8r_bin                | koi8r    |  74 |         | Yes      |       1 |
| latin1_german1_ci        | latin1   |   5 |         | Yes      |       1 |
| latin1_swedish_ci        | latin1   |   8 | Yes     | Yes      |       1 |
| latin1_danish_ci         | latin1   |  15 |         | Yes      |       1 |
| latin1_german2_ci        | latin1   |  31 |         | Yes      |       2 |
| latin1_bin               | latin1   |  47 |         | Yes      |       1 |
| latin1_general_ci        | latin1   |  48 |         | Yes      |       1 |
| latin1_general_cs        | latin1   |  49 |         | Yes      |       1 |
| latin1_spanish_ci        | latin1   |  94 |         | Yes      |       1 |
| latin2_czech_cs          | latin2   |   2 |         | Yes      |       4 |
| latin2_general_ci        | latin2   |   9 | Yes     | Yes      |       1 |
| latin2_hungarian_ci      | latin2   |  21 |         | Yes      |       1 |
| latin2_croatian_ci       | latin2   |  27 |         | Yes      |       1 |
```

ii. 查看数据库

i. 查看所有的数据库

show databases;

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| c3gen              |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.00 sec)
```

ii. 查看指定部分的数据库

show databases like “%sch%”;

```
mysql> show databases like "%sch%";
+--------------------+
| Database (%sch%)   |
+--------------------+
| information_schema |
| performance_schema |
+--------------------+
2 rows in set (0.00 sec)
```

查看数据库的创建 sql 语句

show create database c3gen;

```
mysql> show create database c3gen;
+----------+-----------------------------------------------------------------+
| Database | Create Database                                                 |
+----------+-----------------------------------------------------------------+
| c3gen    | CREATE DATABASE `c3gen` /*!40100 DEFAULT CHARACTER SET utf8 */ |
+----------+-----------------------------------------------------------------+
1 row in set (0.00 sec)
```

iii. 更新数据库

***数据库的名称是不可更改的选项。 如果要更新数据库名，一般做法是新建新库，然后复制库的表和数据(后面再讲怎么操作)。

更新数据库字符集

alter databse c3gen charset GBK;

```
mysql> alter database c3gen charset GBK;
Query OK, 1 row affected (0.00 sec)
```

查看 数据库字符集

show variables like 'character_set_%';

```
mysql> show variables like 'character_set_%';
+--------------------------+-----------------------------------------------------------+
| Variable_name            | Value                                                     |
+--------------------------+-----------------------------------------------------------+
| character_set_client     | gbk                                                       |
| character_set_connection | gbk                                                       |
| character_set_database   | latin1                                                    |
| character_set_filesystem | binary                                                    |
| character_set_results    | gbk                                                       |
| character_set_server     | latin1                                                    |
| character_set_system     | utf8                                                      |
| character_sets_dir       | C:\Program Files\MySQL\MySQL Server 5.7\share\charsets\ |
+--------------------------+-----------------------------------------------------------+
```

这样的编码集是不符合我们的开发要求的，需要修改，怎么修改，上面的命令教过。

```
mysql> set character_set_client=utf8;
Query OK, 0 rows affected (0.00 sec)

mysql> set character_set_connection=utf8;
Query OK, 0 rows affected (0.00 sec)

mysql> set character_set_database=utf8;
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> set character_set_results=utf8;
Query OK, 0 rows affected (0.00 sec)

mysql>  set character_set_server=utf8;
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> set character_set_filesystem=utf8;
Query OK, 0 rows affected (0.00 sec)

mysql> show variables like 'character_set_%';
+--------------------------+----------------------------------------------------------+
| Variable_name            | Value                                                    |
+--------------------------+----------------------------------------------------------+
| character_set_client     | utf8                                                     |
| character_set_connection | utf8                                                     |
| character_set_database   | utf8                                                     |
| character_set_filesystem | utf8                                                     |
| character_set_results    | utf8                                                     |
| character_set_server     | utf8                                                     |
| character_set_system     | utf8                                                     |
| character_sets_dir       | C:\Program Files\MySQL\MySQL Server 5.7\share\charsets\  |
+--------------------------+----------------------------------------------------------+
8 rows in set, 1 warning (0.00 sec)
```

iv. 删除数据库

删除数据库的指令通过 drop

```
mysql> create database test1;
Query OK, 1 row affected (0.00 sec)

mysql> drop database test1;
Query OK, 0 rows affected (0.00 sec)
```

注意！！！删除数据是不可逆的，删除的过程是级联的操作，本地的数据库文件夹也会一起被删除，所以删除数据库的时候，一定要备份。怎么备份我在最后面写。

2.4.2 表操作

i. 创建表

create table 表名 () charset utf8; 带上字符集是个好习惯。后面的例子我基本上都会带上。

```
mysql> use c3gen
Database changed
mysql> create table if not exists student(
    -> stu_id int primary key,
    -> stu_name varchar(25),
    -> stu_age int(2),
    -> stu_score int(2));
Query OK, 0 rows affected (0.24 sec)
```

If not exists: 如果表名不存在,那么就创建,否则不执行创建代码: 检查功能。
创建之前记得指定数据库，不然会报错。No database selected;
当然也可以这样创建 数据库名.表名。

```
mysql> create table if not exists c3gen.stu(
    -> stu_id int,
    -> stu_name varchar(20));
Query OK, 0 rows affected, 1 warning (0.00 sec)
```

primary key 表示主键的意思。我们在创建的时候最好指定一个主键。
创建完成后，查看下本地的数据文件。

其中 .frm 是数据结构文件(这跟数据库引擎相关默认 innoDB，以后再讲)。

ii. 查看表

查看当前数据库下所有表。

```
mysql> show tables;
+-----------------+
| Tables_in_c3gen |
+-----------------+
| boss            |
| stu             |
| student         |
| test2           |
+-----------------+
4 rows in set (0.00 sec)
```

模糊查询当前数据库下的表。

```
mysql> show tables like '%ss';
+-----------------------+
| Tables_in_c3gen (%ss) |
+-----------------------+
| boss                  |
+-----------------------+
1 row in set (0.00 sec)
```

查看创建当前表的 sql 语句

```
mysql> show create table stu;
+-------+-----------------------------------------------------------------
-----------------+
| Table | Create Table
                 |
+-------+-----------------------------------------------------------------
-----------------+
| stu   | CREATE TABLE `stu` (
  `stu_id` int(11) DEFAULT NULL,
  `stu_name` varchar(20) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=gbk |
+-------+-----------------------------------------------------------------
-----------------+
1 row in set (0.00 sec)
```

查看表结构 desc/describe/show columns from 表名;

```
mysql> desc stu;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| stu_id   | int(11)     | YES  |     | NULL    |       |
| stu_name | varchar(20) | YES  |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)

mysql> show columns from stu;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| stu_id   | int(11)     | YES  |     | NULL    |       |
| stu_name | varchar(20) | YES  |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

iii. 更新表

更新表，包括更新表名和表字段两种操作。
更新表名
更新之前 查看一下当前数据库下的表名。

rename table 表名 to 新表名;

```
mysql> show tables;
+----------------+
| Tables_in_c3gen |
+----------------+
| boss           |
| stu            |
| student        |
| test2          |
+----------------+
4 rows in set (0.00 sec)

mysql> rename table stu to stu2;
Query OK, 0 rows affected (0.20 sec)

mysql> show tables;
+----------------+
| Tables_in_c3gen |
+----------------+
| boss           |
| stu2           |
| student        |
| test2          |
+----------------+
4 rows in set (0.00 sec)
```

表的字符集 alter table 表名 charset = 字符集;

```
mysql> show create table stu2;
+-------+-------------------------------------
------------------+
| Table | Create Table
                  |
+-------+-------------------------------------
------------------+
| stu2  | CREATE TABLE `stu2` (
  `stu_id` int(11) DEFAULT NULL,
  `stu_name` varchar(20) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=gbk |
+-------+-------------------------------------
------------------+
1 row in set (0.00 sec)

mysql> alter table stu2 charset=utf8;
Query OK, 0 rows affected (0.07 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

更新完再查看一下

show create table stu2;

```
mysql> show create table stu2;
+-------+-------------------------------------
------------------------------------+
| Table | Create Table
                                    |
+-------+-------------------------------------
------------------------------------+
| stu2  | CREATE TABLE `stu2` (
  `stu_id` int(11) DEFAULT NULL,
  `stu_name` varchar(20) CHARACTER SET gbk DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8 |
+-------+-------------------------------------
------------------------------------+
1 row in set (0.00 sec)
```

新表字段　可以灵活的使用 after，first 等关键字。

更新字段　modify 修改字段属性(数据类型等)　change 修改字段名

```
mysql> desc stu2;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| stu_id   | int(11)     | YES  |     | NULL    |       |
| stu_name | varchar(20) | YES  |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)

mysql> alter table stu2
    -> modify stu_name varchar(30);
Query OK, 0 rows affected (0.50 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc stu2;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| stu_id   | int(11)     | YES  |     | NULL    |       |
| stu_name | varchar(30) | YES  |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

```
mysql> alter table stu2
    -> change stu_name stu_name2 varchar(20);
Query OK, 0 rows affected (0.45 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc stu2;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| stu_id    | int(11)     | YES  |     | NULL    |       |
| stu_age   | int(11)     | YES  |     | NULL    |       |
| stu_name2 | varchar(20) | YES  |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

新增字段　可以用 after 指定　位置。

```
mysql> alter table stu2
    -> add column stu_age int after stu_id;
Query OK, 0 rows affected (0.40 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc stu2;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| stu_id   | int(11)     | YES  |     | NULL    |       |
| stu_age  | int(11)     | YES  |     | NULL    |       |
| stu_name | varchar(30) | YES  |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

删除字段　drop 字段名；　***在数据库中的所有删除 drop 操作均不可逆。

```
mysql> desc stu2;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| stu_id    | int(11)     | YES  |     | NULL    |       |
| stu_age   | int(11)     | YES  |     | NULL    |       |
| stu_name2 | varchar(20) | YES  |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)

mysql> alter table stu2 drop stu_name2;
Query OK, 0 rows affected (0.36 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc stu2;
+---------+---------+------+-----+---------+-------+
| Field   | Type    | Null | Key | Default | Extra |
+---------+---------+------+-----+---------+-------+
| stu_id  | int(11) | YES  |     | NULL    |       |
| stu_age | int(11) | YES  |     | NULL    |       |
+---------+---------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

iv. 删除表

删除表比较简单， drop table 表名..... 可以指定多张表。***在数据库中的所有删除操作均不可逆。

删除操作之后对应的数据库文件也会被删除，并且不可恢复。

```
mysql> drop table boss;
Query OK, 0 rows affected (0.10 sec)

mysql> show tables;
+----------------+
| Tables_in_c3gen |
+----------------+
| stu2           |
| student        |
| test2          |
+----------------+
3 rows in set (0.00 sec)
```

2.4.3 数据操作

i. 插入数据

插入数据分为全部插入和部分插入

```
mysql> insert into stu2 values(1,20);
Query OK, 1 row affected (0.09 sec)

mysql> insert into stu2(stu_id) values(2);
Query OK, 1 row affected (0.06 sec)

mysql> select * from stu2;
+--------+---------+
| stu_id | stu_age |
+--------+---------+
|      1 |      20 |
|      2 |    NULL |
+--------+---------+
2 rows in set (0.00 sec)
```

ii. 查看数据

查看 分为全部查看和部分查看

```
mysql> select * from stu2;
+--------+---------+
| stu_id | stu_age |
+--------+---------+
|      1 |      20 |
|      2 |    NULL |
+--------+---------+
2 rows in set (0.00 sec)

mysql> select stu_age from stu2 ;
+---------+
| stu_age |
+---------+
|      20 |
|    NULL |
+---------+
2 rows in set (0.00 sec)

mysql> select * from stu2 where stu_id = 1;
+--------+---------+
| stu_id | stu_age |
+--------+---------+
|      1 |      20 |
+--------+---------+
1 row in set (0.00 sec)
```

iii. 更新数据

更新的简单操作为 update 表名 set 字段 = 值 可带 where 条件；没有 where 就是全部更新。但是带条件的更新不一定成功，如果条件不存在，那么更新就不会成功。

```
mysql> update stu2 set stu_age = 50;
Query OK, 2 rows affected (0.07 sec)
Rows matched: 2  Changed: 2  Warnings: 0

mysql> select * from stu2;
+--------+---------+
| stu_id | stu_age |
+--------+---------+
|      1 |      50 |
|      2 |      50 |
+--------+---------+
2 rows in set (0.00 sec)
```

```
mysql> update stu2 set stu_age = 10 where stu_id = 1;
Query OK, 1 row affected (0.05 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from stu2;
+--------+---------+
| stu_id | stu_age |
+--------+---------+
|      1 |      10 |
|      2 |      50 |
+--------+---------+
2 rows in set (0.00 sec)
```

iv. 删除数据

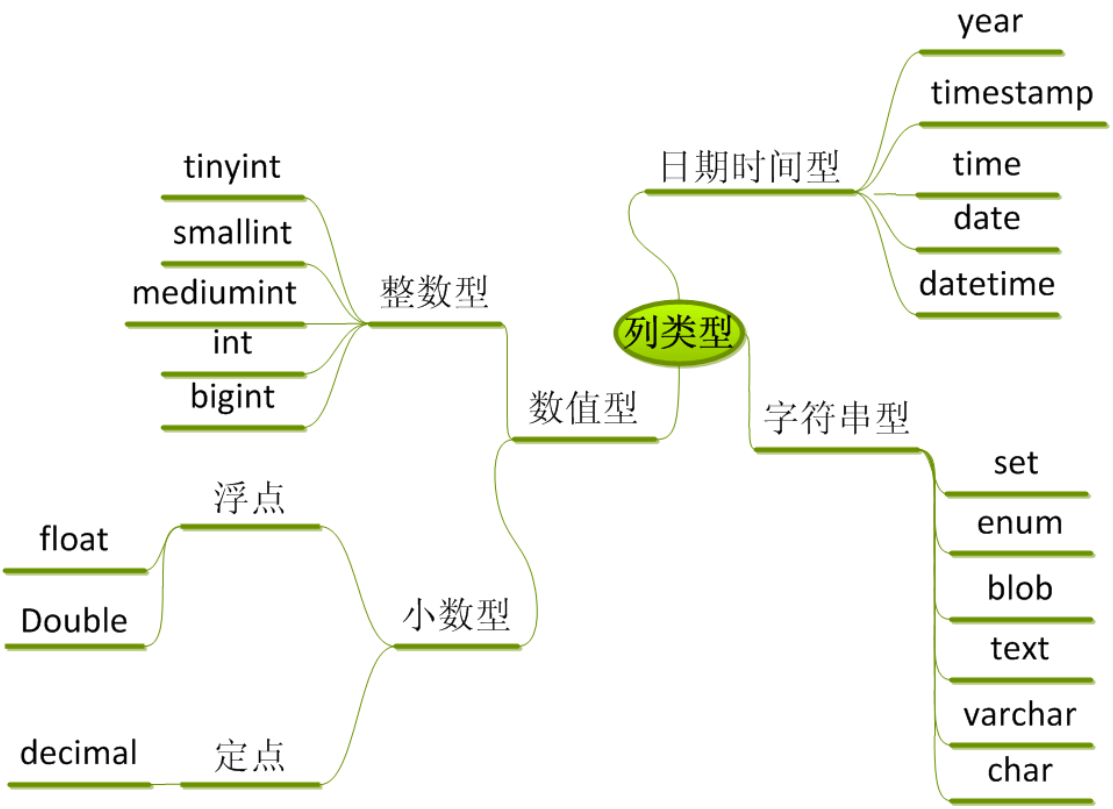
删除数据语法为 delete from 表名 可带 where， 如果不指定 where 就是全部。

```
mysql> insert into stu2 values(3,30);
Query OK, 1 row affected (0.06 sec)

mysql> select * from stu2;
+--------+---------+
| stu_id | stu_age |
+--------+---------+
|      1 |      10 |
|      2 |      50 |
|      1 |      20 |
|      3 |      30 |
+--------+---------+
4 rows in set (0.00 sec)

mysql> delete from stu2 where stu_id = 1;
Query OK, 2 rows affected (0.05 sec)

mysql> select * from stu2;
+--------+---------+
| stu_id | stu_age |
+--------+---------+
|      2 |      50 |
|      3 |      30 |
+--------+---------+
2 rows in set (0.00 sec)
```

2.5 mysql 基本语法

2.5.1 数据类型

基本数据类型：从系统的角度来设计，以更好的利用资源。
SQL 的三大基本类型：数值类型、字符类型和时间类型。



说明如下：

a) 整数型：

Tinyint: 迷你整型,使用一个字节存储, 表示的状态最多为 256 种(常用)
Smallint: 小整型,使用 2 个字节存储,表示的状态最多为 65536 种
Mediumint: 中整型, 使用 3 个字节存储
Int: 标准整型, 使用 4 个字节存储(常用)
Bigint: 大整型,使用 8 个字节存储

类型	字节	最小值（有符号/无符号）	最大值（有符号/无符号）
TINYINT	1	-128/0	127/255
SMALLINT	2	-32768/0	32767/65535
MEDIUMINT	3	-8388608/0	8388607/16777215
INT/INTEGE	4	-2147483648/0	2147483647 /4294967295
BIGINT	8	-9223372036854775808/0	9223372036854775807 /18446744073709551615

```
mysql> create table test_int(
    -> int_1 tinyint,
    -> int_2 smallint,
    -> int_3 int,
    -> int_4 bigint) charset utf8;
Query OK, 0 rows affected (0.22 sec)

mysql> insert into test_int values("a","b","c","d");
ERROR 1366 (HY000): Incorrect integer value: 'a' for column 'int_1' at row 1 数据类型错误
mysql> insert into test_int values(255,10000,1000000,1000000000);
ERROR 1264 (22003): Out of range value for column 'int_1' at row 1 超过数据范围
```

b) 浮点型

浮点型数据是一种精度型数据：因为超出指定范围之后，会丢失精度(自动四舍五入)

浮点型：理论分为两种精度

Float：单精度，占用 4 个字节存储数据，精度范围大概为 7 位左右

Double：双精度,占用 8 个字节存储数据，精度方位大概为 15 位左右

类型	存储空间（字节）	最小值（理论）	最大值（理论）
FLOAT	4	-3.402823466E+38	3.402823466E+38
DOUBLE	8	-1.7976931348623157E+308	1.7976931348623157E+308

```
mysql> create table test_float(
    -> f_1 float,
    -> f_2 float(10,2))charset utf8;
Query OK, 0 rows affected (0.28 sec)

mysql> desc test_float;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| f_1   | float       | YES  |     | NULL    |       |
| f_2   | float(10,2) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)

mysql> insert into test_float values(12,12.2);
Query OK, 1 row affected (0.06 sec)

mysql> insert into test_float values(12,12.2555555);
Query OK, 1 row affected (0.06 sec)

mysql> insert into test_float values(12,1555555555555555555555555555552.2555555);
ERROR 1264 (22003): Out of range value for column 'f_2' at row 1
```

c) 定点型: 绝对的保证整数部分不会被四舍五入(不会丢失精度),小数部分有可能(理论小数部分也不会丢失精度)

DECIMAL	变长，大致是每9个数字，采用4个字节存储。整数和小数分开计算	M，最大是65 D，最大是30 默认是10，2 -（65个9）	（+65个9）

```
mysql> create table test_dec(
    -> dec_1 decimal(10,2))charset utf8;
Query OK, 0 rows affected (0.31 sec)

mysql> insert into test_dec values(1.22);
Query OK, 1 row affected (0.08 sec)

mysql> insert into test_dec values(1.225555555);
Query OK, 1 row affected, 1 warning (0.06 sec)
```

```
mysql> select * from test_dec;
+-------+
| dec_1 |
+-------+
|  1.22 |
|  1.23 |
+-------+
2 rows in set (0.00 sec)
```

d) 时间类型

Datetime: 时间日期, 格式是 YYYY-mm-dd HH:ii:ss,表示的范围是从 1000 到 9999 年,有 0 值: 0000-00-00 00:00:00

Date: 日期,就是 datetime 中的 date 部分

Time: 时间(段), 指定的某个区间之间, -时间到+时间

Timestamp: 时间戳, 并不是时间戳,只是从 1970 年开始的 YYYY-mm-dd HH:ii:ss 格式与 datetime 完全一致

Year: 年份,两种形式, year(2)和 year(4): 1901-2156

类型	显示格式	取值	存储空间	零值
DATETIME	YYYY-MM-DD HH:MM:SS	'1000-01-01 00:00:00'到'9999-12-31 23:59:59'	8	0000-00-00 00:00:00
TIMESTAMP	YYYY-MM-DD HH:MM:SS	是'1970-01-01 00:00:00'到2038-01-19 03:14:07	4	0000-00-00 00:00:00
DATE	YYYY-MM-DD	'1000-01-01'到'9999-12-31	3	0000-00-00
TIME	HH:MM:SS	-838:59:59'到'838:59:59'	3	00:00:00
YEAR	YYYY	1901到2155	1	0000

```
mysql> create table test_date(
    -> da_1 datetime,
    -> da_2 date,
    -> da_3 timestamp,
    -> da_4 time,
    -> da_5 year)charset utf8;
Query OK, 0 rows affected (0.20 sec)

mysql> desc test_date;
+-------+-----------+------+-----+-------------------+-----------------------------+
| Field | Type      | Null | Key | Default           | Extra                       |
+-------+-----------+------+-----+-------------------+-----------------------------+
| da_1  | datetime  | YES  |     | NULL              |                             |
| da_2  | date      | YES  |     | NULL              |                             |
| da_3  | timestamp | NO   |     | CURRENT_TIMESTAMP | on update CURRENT_TIMESTAMP |
| da_4  | time      | YES  |     | NULL              |                             |
| da_5  | year(4)   | YES  |     | NULL              |                             |
+-------+-----------+------+-----+-------------------+-----------------------------+
5 rows in set (0.00 sec)

mysql> insert into test_date values('2016-12-5 15:31:22','2016-12-5','2016-12-5 15:31:22','15:31:22',50);
Query OK, 1 row affected (0.06 sec)

mysql> insert into test_date values('2016-12-15 15:31:22','2016-12-15','2016-12-5 15:31:22','-15:31:22',80);
Query OK, 1 row affected (0.05 sec)
```

time 是可以插入负值的，

```
mysql> select * from test_date;
+---------------------+------------+---------------------+-----------+------+
| da_1                | da_2       | da_3                | da_4      | da_5 |
+---------------------+------------+---------------------+-----------+------+
| 2016-12-05 15:31:22 | 2016-12-05 | 2016-12-05 15:31:22 | 15:31:22  | 2050 |
| 2016-12-15 15:31:22 | 2016-12-15 | 2016-12-05 15:31:22 | -15:31:22 | 1980 |
+---------------------+------------+---------------------+-----------+------+
2 rows in set (0.00 sec)
```

当时间发送改变时，timestamp 会跟随系统时间发生改变。

```
mysql> update test_date set da_1 ='2016-12-6 10:00:00' where da_5 =2050;
Query OK, 1 row affected (0.05 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from test_date;
+---------------------+------------+---------------------+-----------+------+
| da_1                | da_2       | da_3                | da_4      | da_5 |
+---------------------+------------+---------------------+-----------+------+
| 2016-12-06 10:00:00 | 2016-12-05 | 2016-12-05 15:36:42 | 15:31:22  | 2050 |
| 2016-12-15 15:31:22 | 2016-12-15 | 2016-12-05 15:31:22 | -15:31:22 | 1980 |
+---------------------+------------+---------------------+-----------+------+
2 rows in set (0.00 sec)
```

e) 字符类型

在 mysql 中，字符共 六种：char,varchar,text , blob, enum 和 set。

其中 char 为定长字符,varchar 为可变字符,text 为文本字符,blob 为二进制文本字符(基本不会使用),enum 为枚举类型, set 为集合字符(通常存储的是数值，类似枚举)

类型	最大长度	备注
char	255	Char(M),M字符数
varchar	65535，但需要1-2个保存信息，同时由于记录的限制，因此最大为65532	编码不同字符数不同： Gbk<=32767 Utf8<=21845
tinyText，text，mediumText，longtext	L + n。L为最大长度 2^8+1, 2^16+2, 2^24+3, 2^32+4	定义时，通常不用指定长度，可以自己计算。

```
mysql> create table test_char(
    -> c_1 char(2),
    -> c_2 varchar(2),
    -> c_3 text(255),
    -> c_4 enum('男','女','中性'),
    -> c_5 set('武汉','鄂州','黄冈'))charset utf8;
Query OK, 0 rows affected (0.21 sec)

mysql> desc test_char;
+-------+---------------------------+------+-----+---------+-------+
| Field | Type                      | Null | Key | Default | Extra |
+-------+---------------------------+------+-----+---------+-------+
| c_1   | char(2)                   | YES  |     | NULL    |       |
| c_2   | varchar(2)                | YES  |     | NULL    |       |
| c_3   | text                      | YES  |     | NULL    |       |
| c_4   | enum('男','女','中性')    | YES  |     | NULL    |       |
| c_5   | set('武汉','鄂州','黄冈') | YES  |     | NULL    |       |
+-------+---------------------------+------+-----+---------+-------+
5 rows in set (0.00 sec)

mysql> insert into test_char values('测','测试','测试文本',(1),(2));
Query OK, 1 row affected (0.06 sec)

mysql> insert into test_char values('测','测试','测试文本',(1),'宜昌');
ERROR 1265 (01000): Data truncated for column 'c_5' at row 1
```

```
mysql> select * from test_char;
+------+------+----------+------+------+
| c_1  | c_2  | c_3      | c_4  | c_5  |
+------+------+----------+------+------+
| 测   | 测试 | 测试文本 | 男   | 鄂州 |
+------+------+----------+------+------+
1 row in set (0.00 sec)
```

观察下面的代码：

```
mysql> insert into test_char values('测','测试','测试文本',(1),'武汉,鄂州');
Query OK, 1 row affected (0.08 sec)

mysql> insert into test_char values('测','测试','测试文本',(1),'武汉,黄冈');
Query OK, 1 row affected (0.06 sec)

mysql> select * from test_char;
+------+------+----------+------+-----------+
| c_1  | c_2  | c_3      | c_4  | c_5       |
+------+------+----------+------+-----------+
| 测   | 测试 | 测试文本 | 男   | 鄂州      |
| 测   | 测试 | 测试文本 | 男   | 武汉,鄂州 |
| 测   | 测试 | 测试文本 | 男   | 武汉,黄冈 |
+------+------+----------+------+-----------+
3 rows in set (0.00 sec)

mysql> select c_5+0,c_5 from test_char;
+-------+-----------+
| c_5+0 | c_5       |
+-------+-----------+
|     2 | 鄂州      |
|     3 | 武汉,鄂州 |
|     5 | 武汉,黄冈 |
+-------+-----------+
3 rows in set (0.00 sec)
```

```
----c_5 set('武汉','鄂州','黄冈')
----集合中的每一个元素都对应一个二进制位，被选中的为1，没选中的为0,最后反过来
----比如 insert into  ('武汉,黄冈')
--------------1     0       1
--------------101二进制转为十进制就是5
select c_5+0 就可以查看集合的数据
```

2.5.2 列属性

列属性是为了增加字段的额外约束(除了数据类型外的约束)。
常见的有 null/not null, default, primary key, unique key, auto_increment,comment

- NULL 属性
  null 是很常见的一个关键字，它表示空，在字段的描述用 Null 表述时表述字段是空数据，空数据是没有意义 ，一般不会让一个字段的数据是空值。

- 列描述
  列描述，也就是字段的描述文本，前期开发或者自己设计数据库可能觉得意义不大，但是后期的开发，或者给其他人使用，列描述就非常重要。

```
mysql> create table test_col(
    -> id int not null comment 'id字段',
    -> name varchar(20) not null comment '姓名字段')charset utf8;
Query OK, 0 rows affected (0.33 sec)

mysql> desc test_col;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| id    | int(11)     | NO   |     | NULL    |       |
| name  | varchar(20) | NO   |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

- 默认值
  为了让一些常用的字段有意义，可以给字段默认值。这样在 insert 数据的时候可以不给默认值的字段数据。

```
mysql> create table test_col2(
    -> id int not null default 1,
    -> name varchar(20) not null comment '姓名字段')charset utf8;
Query OK, 0 rows affected (0.26 sec)

mysql> desc test_col2;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| id    | int(11)     | NO   |     | 1       |       |
| name  | varchar(20) | NO   |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

```
mysql> insert into test_col2(name) values('张三');
Query OK, 1 row affected (0.06 sec)

mysql> insert into test_col2(id,name) values(2,'李四');
Query OK, 1 row affected (0.06 sec)

mysql> select * from test_col2;
+----+------+
| id | name |
+----+------+
|  1 | 张三 |
|  2 | 李四 |
+----+------+
2 rows in set (0.00 sec)
```

2.5.3 字段属性

这里单独把字段属性拿出来，主要是为了说明一下重要性，包括主键、自增长、唯一键、索引。

i. 主键

primary key 主键， 一张表只能有一个主键，是唯一用来约束表中的数据的字段，不可为空，不能重复。
常见的主键设置方式有：
a) 创建表的时候指定

```
mysql> create table test_pk(
    -> id int primary key ,
    -> name varchar(20) not null,
    -> sex enum('男','女') default '男')charset utf8;
Query OK, 0 rows affected (0.23 sec)

mysql> desc test_pk;
+-------+-----------------+------+-----+---------+-------+
| Field | Type            | Null | Key | Default | Extra |
+-------+-----------------+------+-----+---------+-------+
| id    | int(11)         | NO   | PRI | NULL    |       |
| name  | varchar(20)     | NO   |     | NULL    |       |
| sex   | enum('男','女') | YES  |     | 男      |       |
+-------+-----------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

使用 primary key 来指定复合主键。

```
mysql> create table test_pk2(
    -> id int not null,
    -> col_id int not null,
    -> name varchar(20) not null,
    -> primary key(id,col_id))charset utf8;
Query OK, 0 rows affected (0.22 sec)

mysql> desc test_pk2;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| id     | int(11)     | NO   | PRI | NULL    |       |
| col_id | int(11)     | NO   | PRI | NULL    |       |
| name   | varchar(20) | NO   |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

使用 alter 修改语句来增加主键。

```
mysql> create table test_pk3(
    -> id int not null,
    -> name varchar(20))charset utf8;
Query OK, 0 rows affected (0.25 sec)

mysql> alter table test_pk3 add primary key(id);
Query OK, 0 rows affected (0.32 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc test_pk3;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| id    | int(11)     | NO   | PRI | NULL    |       |
| name  | varchar(20) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

b) 删除后新增

主键是没有办法做更新操作的，只能先删除再新增。

```
mysql> create table test(
    -> old_id int primary key,
    -> name varchar(20))charset utf8;
Query OK, 0 rows affected (0.20 sec)

mysql> desc test;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| old_id | int(11)     | NO   | PRI | NULL    |       |
| name   | varchar(20) | YES  |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

在这个表的基础上新增一个字段 new_id，用来替换成新的主键。

```
mysql> alter table test add new_id int not null;
Query OK, 0 rows affected (0.34 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc test;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| old_id | int(11)     | NO   | PRI | NULL    |       |
| name   | varchar(20) | YES  |     | NULL    |       |
| new_id | int(11)     | NO   |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)

mysql> alter table test drop primary key;
Query OK, 0 rows affected (0.48 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc test;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| old_id | int(11)     | NO   |     | NULL    |       |
| name   | varchar(20) | YES  |     | NULL    |       |
| new_id | int(11)     | NO   |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

然后新增主键。

```
mysql> alter table test add primary key(new_id);
Query OK, 0 rows affected (0.28 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc test;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| old_id | int(11)     | NO   |     | NULL    |       |
| name   | varchar(20) | YES  |     | NULL    |       |
| new_id | int(11)     | NO   | PRI | NULL    |       |
+--------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

c) 逻辑主键

很多时候，我们并不会把一个表的字段如学生编码作为主键，这个时候需要一个逻辑字段来作为主键，这种主键实际是没有任何含义的，可以是任意唯一标识符。

```
mysql> create table stu_test(
    -> id int primary key auto_increment comment '逻辑主键，自增长',
    -> stu_no char(11) not null comment '学生学号',
    -> stu_name varchar(20) not null )charset utf8;
Query OK, 0 rows affected (0.23 sec)
```

ii. 自增长

自增长，顾名思义就是自动增长的字段，当对应的字段,不给值,或者说给默认值,或者给 NULL 的时候，会自动的被系统触发，系统会从当前字段中已有的最大值再进行+1 操作,得到一个新的在不同的字段.

a) 自增长的字段，必须是一个索引值，如下：

```
mysql> create table auto_test1(
    -> id int auto_increment ,
    -> name varchar(20))charset utf8;
ERROR 1075 (42000): Incorrect table definition; there can be only one auto column and it must be defined as a key
```

b) 自增长的字段，必须是整型的数据类型，如下：

```
mysql> create table auto_test1(
    -> id char(10) auto_increment ,
    -> name varchar(20))charset utf8;
ERROR 1063 (42000): Incorrect column specifier for column 'id'
```

c) 自增长的字段，必须唯一，也就是说一张表只能有一个自增长的字段，如下：

```
mysql> create table auto_test1(
    -> id int auto_increment primary key ,
    -> at_id int auto_increment,
    -> name varchar(20))charset utf8;
ERROR 1075 (42000): Incorrect table definition; there can be only one auto column and it must be defined as a key
```

d) 正确的自增长创建如下：

```
mysql> create table auto_test1(
    -> id int auto_increment primary key ,
    -> name varchar(20))charset utf8;
Query OK, 0 rows affected (0.25 sec)
```

e) 自增长 insert 用 default , null,或者不指定值来触发自增长。

```
mysql> insert into auto_test1 values(default,'张三');
Query OK, 1 row affected (0.08 sec)

mysql> insert into auto_test1 values(null,'李四');
Query OK, 1 row affected (0.03 sec)

mysql> insert into auto_test1(name) values('王五');
Query OK, 1 row affected (0.08 sec)

mysql> select * from auto_test1;
+----+------+
| id | name |
+----+------+
|  1 | 张三 |
|  2 | 李四 |
|  3 | 王五 |
+----+------+
3 rows in set (0.00 sec)
```

*****需要注意的是，当指定的了自增长的值或者删除了自增长的数据的时候，再次新增的数据会从自增长的最大值+1 开始。

```
mysql> insert into auto_test1 values(5,'李四1');
Query OK, 1 row affected (0.07 sec)

mysql> insert into auto_test1 values(null,'李四2');
Query OK, 1 row affected (0.05 sec)
```

```
mysql> select * from auto_test1;
+----+-------+
| id | name  |
+----+-------+
|  1 | 张三  |
|  2 | 李四  |
|  3 | 王五  |
|  5 | 李四1 |
|  6 | 李四2 |
+----+-------+
5 rows in set (0.00 sec)
```

*****查看下一个自增长的值是多少？　可以使用查看表的 create table 语句。

```
mysql> show create table auto_test1;
+------------+------------------------------------------------------------
---------------------------------------------------------------+
| Table      | Create Table
                                                               |
+------------+------------------------------------------------------------
---------------------------------------------------------------+
| auto_test1 | CREATE TABLE `auto_test1` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(20) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=7 DEFAULT CHARSET=utf8 |
+------------+------------------------------------------------------------
---------------------------------------------------------------+
1 row in set (0.00 sec)
```

f) 自增长 alter 修改

修改自增长和修改主键一样，都需要先删除，然后再新增，所以操作过程是一样的。那种删除新增其他字段为主键的方法为：

alter table 表名 drop 字段；----alter table 表名 add 字段；

下面介绍，怎么修改自增长的值 和 增长的量。

✓ 自增长的值

修改当前自增长已经存在的值：修改只能比当前已有的自增长的最大值大,不能小(小不生效)。 语法为： alter table 表名 auto_increment = 值；

```
mysql> alter table auto_test1 auto_increment = 10;
Query OK, 0 rows affected (0.05 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> show create table auto_test1;
+------------+-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------+
| Table      | Create Table
                                                                                                          |
+------------+-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------+
| auto_test1 | CREATE TABLE `auto_test1` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(20) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=10 DEFAULT CHARSET=utf8 |
+------------+-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

这时候发现自增长的值已经从 10 开始了。

✓ 自增长的量

有时候，我们会觉得自增长的量太小，为什么一直都是 1 呢？这是由系统的默认变量来控制的。

查看 show variables like ‘auto_increment%’; increment 为增长的量，offset 为起始值。

```
mysql> show variables like 'auto_increment%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| auto_increment_increment | 1     |
| auto_increment_offset    | 1     |
+--------------------------+-------+
2 rows in set, 1 warning (0.20 sec)
```

获取到这两个值，就可以设置这两个值；比如设置增量为 5；

set auto_increment_increment=5; 注意修改的值只对当前的会话有效，也就是关闭 cmd 窗口后值会失效。

```
mysql> set auto_increment_increment=5;
Query OK, 0 rows affected (0.03 sec)
```

```
mysql> create table auto_test2(
    -> id int auto_increment primary key,
    -> name varchar(20))charset utf8;
Query OK, 0 rows affected (0.22 sec)

mysql> insert into auto_test2 values(null,'张三');
Query OK, 1 row affected (0.07 sec)

mysql> insert into auto_test2 values(null,'张三1');
Query OK, 1 row affected (0.07 sec)

mysql> select * from auto_test2;
+----+-------+
| id | name  |
+----+-------+
|  4 | 张三  |
|  9 | 张三1 |
+----+-------+
2 rows in set (0.00 sec)
```

g) 自增长 alter 删除

删除自增长，其实只要修改自增长的值，去掉 auto_increment 就可以了。

```
mysql> alter table auto_test2 modify id int;
Query OK, 2 rows affected (0.48 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> desc auto_test2;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| id    | int(11)     | NO   | PRI | NULL    |       |
| name  | varchar(20) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

iii. 唯一键

一张表往往有很多字段需要具有唯一性,数据不能重复: 但是一张表中只能有一个主键: 唯一键(unique key)就可以解决表中有多个字段需要唯一性约束的问题。

唯一键的本质与主键差不多: 唯一键默认的允许自动为空,而且可以多个为空(空字段不参与唯一性比较)。

a) 创建唯一键的方式很多，比如在创建的时候指定字段 unique,
b) 当表没有设置主键，并且 unique 的字段不为空的时候，该字段会升级成为主键。
c) 当表创建完成后，需要增加唯一键，就使用 alter 来修改增加。

```
mysql> create table uni_test(
    -> stu_id int unique comment '学号唯一值，可以为空',
    -> stu_name varchar(20))charset utf8;
Query OK, 0 rows affected (0.48 sec)

mysql> desc uni_test;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| stu_id   | int(11)     | YES  | UNI | NULL    |       |
| stu_name | varchar(20) | YES  |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

```
mysql> create table uni_test2(
    -> id int not null,
    -> name varchar(20),
    -> unique key(id))charset utf8;
Query OK, 0 rows affected (0.28 sec)

mysql> desc uni_test2;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| id    | int(11)     | NO   | PRI | NULL    |       |
| name  | varchar(20) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

```
mysql> create table uni_test3(
    -> id int not null,
    -> name varchar(20))charset utf8;
Query OK, 0 rows affected (0.27 sec)

mysql> alter table uni_test3 add unique key(id);
Query OK, 0 rows affected (0.45 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc uni_test3;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| id    | int(11)     | NO   | PRI | NULL    |       |
| name  | varchar(20) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

d) 更新和删除唯一键

操作和之前的一样，如果要更新这个键，就必须先删除，然后再新增。

当我们直接使用 drop unique key 来删除唯一键的时候回报错，因为唯一键可以有多个。

```
mysql> alter table uni_test drop unique key(stu_id);
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version
 for the right syntax to use near 'unique key(stu_id)' at line 1
```

所以正确的做法是通过删除索引来删除唯一键，如下：

```
mysql> alter table uni_test drop index stu_id;
Query OK, 0 rows affected (0.17 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc uni_test;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| stu_id   | int(11)     | YES  |     | NULL    |       |
| stu_name | varchar(20) | YES  |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

iv. 索引

索引: 系统根据某种算法, 将已有的数据(未来可能新增的数据),单独建立一个文件: 文件能够实现快速的匹配数据, 并且能够快速的找到对应表中的记录. 索引是一种特殊的文件(innodb 数据表上的索引是表空间的一个组成部分), 它们包含着对数据表里所有记录的引用指针。更通俗的说，数据库索引好比是一本书前面的目录，能加快数据库的查询速度。

索引的意义
1. 提升查询数据的效率
2. 约束数据的有效性(唯一性等)

增加索引的前提条件: 索引本身会产生索引文件(有时候有可能比数据文件还大) ,会非常耗费磁盘空间.

如果某个字段需要作为查询的条件经常使用, 那么可以使用索引(一定会想办法增加);
如果某个字段需要进行数据的有效性约束, 也可能使用索引(主键,唯一键)

mysql 中提供了多种索引
1) 主键索引: primary key
2) 唯一索引: unique key. 上面写到删除唯一键的做法就是通过删除索引来实现。
3) 全文索引: fulltext index
4) 普通索引: index

全文索引: 针对文章内部的关键字进行索引，但是限定数据类型为 char/varchar/text
全文索引最大的问题: 在于如何确定关键字

英文很容易: 英文单词与单词之间有空格
中文很难: 没有空格, 而且中文可以各种随意组合(分词: sphinx)

下面简单介绍下普通索引，唯一索引和全文索引的使用，其他的索引优化 sql，以后在第二版 。
a) 普通索引
这是最基本的索引，它没有任何限制，比如上文中为 title 字段创建的索引就是一个普通索

引，MyIASM 中默认的 BTREE 类型的索引，也是我们大多数情况下用到的索引。

注意创建索引的字段一定是字符串 char varchar 和 text.

```
mysql> desc student;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| stu_id    | int(11)     | NO   | PRI | NULL    |       |
| stu_name  | varchar(25) | YES  |     | NULL    |       |
| stu_age   | int(2)      | YES  |     | NULL    |       |
| stu_score | int(2)      | YES  |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

```
mysql> create index indx_age on student(stu_age(20));
ERROR 1089 (HY000): Incorrect prefix key; the used key part isn't a string, the used length is longer than the key part,
 or the storage engine doesn't support unique prefix keys
mysql> create index indx_name on student(stu_name(20));
Query OK, 0 rows affected (0.29 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

- 直接创建

  `create index index_name on table(column(length));`

- 修改表结构的方式添加索引

  `alter table table_name add index index_name on (column(length));`

- 创建表的时候就创建索引

```
mysql> create table index_test(
    -> id int not null,
    -> name varchar(20),
    -> primary key(id),
    -> index index_name(name(20)))charset utf8;
Query OK, 0 rows affected (0.25 sec)

mysql> desc index_test;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| id    | int(11)     | NO   | PRI | NULL    |       |
| name  | varchar(20) | YES  | MUL | NULL    |       |
+-------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

- 删除索引

  `drop index index_name on table;`

```
mysql> drop index index_name on index_test;
Query OK, 0 rows affected (0.11 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc index_test;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| id    | int(11)     | NO   | PRI | NULL    |       |
| name  | varchar(20) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

b) 唯一索引

与普通索引类似，不同的就是：索引列的值必须唯一，但允许有空值（注意和主键不同）。如果是组合索引，则列值的组合必须唯一，创建方法和普通索引类似。

- 创建唯一索引

```
create unique index index_name  on table(column(length));
```

- 修改表结构创建索引

```
alter table table_name add unique index_name on (column(length));
```

- 创建表的时候创建索引

```
mysql> create table ind_test(
    -> id int not null,
    -> name varchar(20),
    -> sex char(2),
    -> primary key(id),
    -> unique ind_name (name(20)))charset utf8;
Query OK, 0 rows affected (0.28 sec)

mysql> desc ind_test;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| id    | int(11)     | NO   | PRI | NULL    |       |
| name  | varchar(20) | YES  | UNI | NULL    |       |
| sex   | char(2)     | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

- 删除索引

```
drop index index_name on table;
```

c) 全文索引

FULLTEXT 索引仅可用于 MyISAM 表；他们可以从 CHAR、VARCHAR 或 TEXT 列中作为 CREATE TABLE 语句的一部分被创建，或是随后使用 ALTER TABLE 或 CREATE INDEX 被添加。****对于较大的数据集，将你的资料输入一个没有 FULLTEXT 索引的表中，然后创建索引，其速度比把资料输入现有 FULLTEXT 索引的速度更为快。不过切记对于大容量的数据表，生成全文索引是一个非常消耗时间非常消耗硬盘空间的做法。

- 直接创建索引

```
create fulltext index index_content on article(content);
```

- 修改表结构创建索引

```
alter table article  add fulltext index_content(content);
```

- 创建表的时候创建索引

```
mysql> create table full_test(
    -> id int not null,
    -> title varchar(25),
    -> content text,
    -> primary key(id),
    -> fulltext(content))charset utf8;
Query OK, 0 rows affected (1.82 sec)

mysql> desc full_test;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| id      | int(11)     | NO   | PRI | NULL    |       |
| title   | varchar(25) | YES  |     | NULL    |       |
| content | text        | YES  | MUL | NULL    |       |
+---------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

****索引的缺点：

虽然索引大大提高了查询速度，同时却会降低更新表的速度，如对表进行 insert、update 和 delete。因为更新表时，mysql 不仅要保存数据，还要保存一下索引文件。建立索引会占用磁盘空间的索引文件。一般情况这个问题不太严重，但如果你在一个大表上创建了多种组合索引，索引文件的会膨胀很快。索引只是提高效率的一个因素，如果你的 mysql 有大数据量的表，就需要花时间研究建立最优秀的索引，或优化查询语句

****常见的使用误区：

例如：select * from users where YEAR(adddate)<2017，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<’2017-01-01′。关于这一点可以围观：一个单引号引发的 MYSQL 性能损失，也就是不要在列上进行运算。

又如 like ‘%op%’,这样写的 like 是不会让索引生效的，但是 like’op%’却可以，所以一般情况下不鼓励使用 like 操作，如果非使用不可，如何使用也是一个问题。##############################索引属于 mysql 的优化部分以后再探究##################

```
mysql> select * from v_stu;
+--------+----------+---------+-----------+
| stu_id | stu_name | stu_age | stu_score |
+--------+----------+---------+-----------+
|      1 | 张三     |      20 |        60 |
|      2 | 张三2    |      30 |        70 |
|      3 | 李四     |      24 |        70 |
|      4 | 李四2    |      19 |        70 |
+--------+----------+---------+-----------+
4 rows in set (0.00 sec)

mysql> update v_stu set stu_name ='张三丰' where stu_id=1;
Query OK, 1 row affected (0.06 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from v_stu;
+--------+----------+---------+-----------+
| stu_id | stu_name | stu_age | stu_score |
+--------+----------+---------+-----------+
|      1 | 张三丰   |      20 |        60 |
|      2 | 张三2    |      30 |        70 |
|      3 | 李四     |      24 |        70 |
|      4 | 李四2    |      19 |        70 |
+--------+----------+---------+-----------+
4 rows in set (0.00 sec)
```

2.5.4 常用的查询子句

select 选项

select 选项: select 对查出来的结果的处理方式
all: 默认的,保留所有的结果
distinct: 去重, 查出来的结果,将重复给去除(所有字段都相同)
新建一个测试表并新增如下数据：

```
mysql> create table tb_t(
    -> name char(2))charset utf8;
Query OK, 0 rows affected (0.29 sec)

mysql> insert into tb_t values('a');
Query OK, 1 row affected (0.08 sec)

mysql> insert into tb_t values('a');
Query OK, 1 row affected (0.09 sec)

mysql> insert into tb_t values('A');
Query OK, 1 row affected (0.07 sec)

mysql> insert into tb_t values('A');
Query OK, 1 row affected (0.06 sec)

mysql> insert into tb_t values('B');
Query OK, 1 row affected (0.05 sec)

mysql> insert into tb_t values('c');
Query OK, 1 row affected (0.11 sec)

mysql> insert into tb_t values('c');
Query OK, 1 row affected (0.09 sec)

mysql> insert into tb_t values('D');
Query OK, 1 row affected (0.11 sec)
```

a) select * from /select all * from 匹配所有结果;

b) select distinct * from 去重匹配

```
mysql> select * from tb_t;
+------+
| name |
+------+
| a    |
| a    |
| A    |
| A    |
| B    |
| c    |
| c    |
| D    |
+------+
8 rows in set (0.00 sec)

mysql> select all * from tb_t;
+------+
| name |
+------+
| a    |
| a    |
| A    |
| A    |
| B    |
| c    |
| c    |
| D    |
+------+
8 rows in set (0.00 sec)
```

```
mysql> select distinct * from tb_t;
+------+
| name |
+------+
| a    |
| B    |
| c    |
| D    |
+------+
4 rows in set (0.00 sec)
```

select 别名

别名分为字段别名和表别名，
字段别名: 当数据进行查询出来的时候, 有时候名字并一定就满足需求(多表查询的时候, 会有同名字段). 需要对字段名进行重命名: 别名
表别名：当进行联合查询的时候，有时候表的名字太长，我们可以用别名来指代表名。
语法
字段名/表名 as 可选 别名;

测试，创建两张表，tb_t1 和 tb_t2 其中 t1 的 id 是 t2 的外键。

```
mysql> create table tb_t1(
    -> id int primary key auto_increment,
    -> name varchar(20),
    -> age int)charset utf8;
Query OK, 0 rows affected (0.23 sec)
```

```
mysql> create table tb_t2(
    -> id int primary key auto_increment,
    -> name varchar(20),
    -> t_id int not null,
    -> foreign key(t_id) references tb_t1(id))charset utf8;
Query OK, 0 rows affected (0.32 sec)
```

插入测试数据：

```
mysql> insert into tb_t1 values(null,'张三',20);
Query OK, 1 row affected (0.08 sec)

mysql> insert into tb_t1 values(null,'张三1',22);
Query OK, 1 row affected (0.10 sec)

mysql> insert into tb_t1 values(null,'张三2',23);
Query OK, 1 row affected (0.06 sec)

mysql> insert into tb_t2 values(null,'语文',1);
Query OK, 1 row affected (0.07 sec)

mysql> insert into tb_t2 values(null,'数学',1);
Query OK, 1 row affected (0.07 sec)

mysql> insert into tb_t2 values(null,'语文',2);
Query OK, 1 row affected (0.03 sec)
```

如下，给字段取别名：

```
mysql> select id,
    -> name as 姓名,
    -> age as 年龄 from tb_t1;
+----+-------+------+
| id | 姓名  | 年龄 |
+----+-------+------+
|  1 | 张三  |   20 |
|  2 | 张三1 |   22 |
|  3 | 张三2 |   23 |
+----+-------+------+
3 rows in set (0.05 sec)
```

如下，给表取别名：

```
mysql> select a.id,a.name,b.name from tb_t1 as a
    -> left join tb_t2 as b on
    -> a.id=b.t_id;
+----+-------+------+
| id | name  | name |
+----+-------+------+
|  1 | 张三  | 语文 |
|  1 | 张三  | 数学 |
|  2 | 张三1 | 语文 |
|  3 | 张三2 | NULL |
+----+-------+------+
4 rows in set (0.06 sec)
```

select 数据源

数据源: 数据的来源, 关系型数据库的来源都是数据表: 本质上只要保证数据类似二维表,最终都可以作为数据源.
数据源分为多种: 单表数据源, 多表数据源, 查询语句

a) 单表数据源: select * from 表名;

```
mysql> select * from tb_t1;
+----+-------+------+
| id | name  | age  |
+----+-------+------+
|  1 | 张三  |   20 |
|  2 | 张三1 |   22 |
|  3 | 张三2 |   23 |
+----+-------+------+
3 rows in set (0.00 sec)
```

多表数据源: select* from 表名 1,表名 2...;

```
mysql> select * from tb_t1,tb_t2;
+----+-------+------+----+------+------+
| id | name  | age  | id | name | t_id |
+----+-------+------+----+------+------+
|  1 | 张三  |   20 |  1 | 语文 |    1 |
|  2 | 张三1 |   22 |  1 | 语文 |    1 |
|  3 | 张三2 |   23 |  1 | 语文 |    1 |
|  1 | 张三  |   20 |  2 | 数学 |    1 |
|  2 | 张三1 |   22 |  2 | 数学 |    1 |
|  3 | 张三2 |   23 |  2 | 数学 |    1 |
|  1 | 张三  |   20 |  3 | 语文 |    2 |
|  2 | 张三1 |   22 |  3 | 语文 |    2 |
|  3 | 张三2 |   23 |  3 | 语文 |    2 |
+----+-------+------+----+------+------+
9 rows in set (0.00 sec)
```

从一张表中取出一条记录,去另外一张表中匹配所有记录,而且全部保留:(记录数和字段数),将这种结果成为: 笛卡尔积(交叉连接): 笛卡尔积没什么卵用, 所以应该尽量避免.

子查询: 数据的来源是一条查询语句(查询语句的结果是二维表)
select * from (select 语句) as[这里的 as 不可以省略] 表名;

```
mysql> select * from (select * from tb_t1);
ERROR 1248 (42000): Every derived table must have its own alias
mysql> select * from (select * from tb_t1) as s;
+----+-------+------+
| id | name  | age  |
+----+-------+------+
|  1 | 张三  |   20 |
|  2 | 张三1 |   22 |
|  3 | 张三2 |   23 |
+----+-------+------+
3 rows in set (0.00 sec)
```

子查询是一种高明的方法，可以让你的查询结果更清晰。

where 子句

where 子句: 用来判断数据,筛选数据.
where 子句返回结果: 0 或者 1, 0 代表 false,1 代表 true.

判断条件:

比较运算符: >, <, >=, <= ,!= ,<>, =, like, between and, in/not in
逻辑运算符: &&(and), ||(or), !(not)

where 原理: where 是唯一一个直接从磁盘获取数据的时候就开始判断的条件: 从磁盘取出一条记录, 开始进行 where 判断: 判断的结果如果成立保存到内存;如果失败直接放弃.

条件查询 1: 要求找出学生 id 为 1 或者 3 或者 5 的学生 多条件筛选用 in

```
mysql> select * from tb_t1 where id=1;
+----+------+------+
| id | name | age  |
+----+------+------+
|  1 | 张三 |   20 |
+----+------+------+
1 row in set (0.00 sec)

mysql> select * from tb_t1 where id in(1,2,3);
+----+-------+------+
| id | name  | age  |
+----+-------+------+
|  1 | 张三  |   20 |
|  2 | 张三1 |   22 |
|  3 | 张三2 |   23 |
+----+-------+------+
3 rows in set (0.00 sec)
```

条件查询 2: 查出 age 区间落在 20,22 之间的数据:

```
mysql> select * from tb_t1 where age >=20 and age<=22;
+----+-------+------+
| id | name  | age  |
+----+-------+------+
|  1 | 张三  |   20 |
|  2 | 张三1 |   22 |
+----+-------+------+
2 rows in set (0.04 sec)

mysql> select * from tb_t1 where age between 20 and 22;
+----+-------+------+
| id | name  | age  |
+----+-------+------+
|  1 | 张三  |   20 |
|  2 | 张三1 |   22 |
+----+-------+------+
2 rows in set (0.00 sec)
```

between 本身是闭区间; between 左边的值必须小于或者等于右边的值，所以一定要注意区间，一般情况下多使用第一种方式来查询，因为更加可控，比如查询日期 在哪一天的订单。

group by 子句

group by:分组的意思, 根据某个字段进行分组(相同的放一组,不同的分到不同的组)

基本语法: group by 字段名;
测试，在 tb_t1 的基础上增加 sex 性别字段，插入一条女性数据,并根据性别来分组

```
mysql> alter table tb_t1 add sex char(2);
Query OK, 0 rows affected (0.66 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> update tb_t1 set sex ='男' where name like '张三%';
Query OK, 3 rows affected (0.05 sec)
Rows matched: 3  Changed: 3  Warnings: 0

mysql> insert into tb_t1 values(null,'翠花',18,'女');
Query OK, 1 row affected (0.06 sec)

mysql> select * from tb_t1 group by sex;
+----+------+------+------+
| id | name | age  | sex  |
+----+------+------+------+
|  4 | 翠花 |   18 | 女   |
|  1 | 张三 |   20 | 男   |
+----+------+------+------+
2 rows in set (0.00 sec)
```

分组的意思: 是为了统计数据(按组统计: 按分组字段进行数据统计)
sql 提供了一系列统计函数
count(): 统计分组后的记录数: 每一组有多少记录
max(): 统计每组中最大的值

min(): 统计最小值
avg(): 统计平均值
sum(): 统计和

```
mysql> select sex,count(*),max(age),min(age),avg(age),sum(age) from tb_t1 group by sex;
+------+----------+----------+----------+----------+----------+
| sex  | count(*) | max(age) | min(age) | avg(age) | sum(age) |
+------+----------+----------+----------+----------+----------+
| 女   |        1 |       18 |       18 |  18.0000 |       18 |
| 男   |        3 |       23 |       20 |  21.6667 |       65 |
+------+----------+----------+----------+----------+----------+
2 rows in set (0.03 sec)
```

****count 函数: 里面可以使用两种参数: *代表统计记录,字段名代表统计对应的字段(null 值不统计)
测试之前的准备如下：

```
mysql> insert into tb_t1 values(null,'小红',19,null);
Query OK, 1 row affected (0.09 sec)

mysql> select * from tb_t1;
+----+-------+------+------+
| id | name  | age  | sex  |
+----+-------+------+------+
|  1 | 张三  |   20 | 男   |
|  2 | 张三1 |   22 | 男   |
|  3 | 张三2 |   23 | 男   |
|  4 | 翠花  |   18 | 女   |
|  5 | 小红  |   19 | NULL |
+----+-------+------+------+
5 rows in set (0.01 sec)
```

可以看到 count() 并不会统计 null 值的字段。 count()比较常用的地方有用来检测用户是否存在，传入参数用户名，然后查询，如果 count(用户名)=0 就说明用户名是不存在的，反之存在。

```
mysql> select count(sex) from tb_t1;
+------------+
| count(sex) |
+------------+
|          4 |
+------------+
1 row in set (0.00 sec)
```

****分组会自动排序: 根据分组字段:默认升序
group by 字段 [asc|desc]; -- 对分组的结果然后合并之后的整个结果进行排序

```
mysql> select * from tb_t1 group by sex desc;
+----+------+------+------+
| id | name | age  | sex  |
+----+------+------+------+
|  1 | 张三 |   20 | 男   |
|  4 | 翠花 |   18 | 女   |
|  5 | 小红 |   19 | NULL |
+----+------+------+------+
3 rows in set (0.00 sec)

mysql> select * from tb_t1 group by sex asc;
+----+------+------+------+
| id | name | age  | sex  |
+----+------+------+------+
|  5 | 小红 |   19 | NULL |
|  4 | 翠花 |   18 | 女   |
|  1 | 张三 |   20 | 男   |
+----+------+------+------+
3 rows in set (0.00 sec)
```

多字段分组: 先根据一个字段进行分组,然后对分组后的结果再次按照其他字段进行分组

```
mysql> select * from tb_t1 group by sex,age;
+----+-------+------+------+
| id | name  | age  | sex  |
+----+-------+------+------+
|  5 | 小红  |   19 | NULL |
|  4 | 翠花  |   18 | 女   |
|  1 | 张三  |   20 | 男   |
|  2 | 张三1 |   22 | 男   |
|  3 | 张三2 |   23 | 男   |
+----+-------+------+------+
5 rows in set (0.00 sec)
```

group_concat: 可以对分组的结果中的某个字段进行字符串连接(保留该组所有的某个字段): group_concat(字段); 这是个很灵活的字段，通常用来过滤一对多的情况，

为了更好的体现，我们创建一个 tb_menu,然后插入测试数据;

```
mysql> create table tb_menu(
    -> parent_id int,
    -> child_id int)charset utf8;
Query OK, 0 rows affected (0.23 sec)
```

```
mysql> insert into tb_menu values(1,1),(1,2),(1,3),(2,1),(2,2),(2,3),(3,1);
Query OK, 7 rows affected (0.03 sec)
Records: 7  Duplicates: 0  Warnings: 0

mysql> select * from tb_menu;
+-----------+----------+
| parent_id | child_id |
+-----------+----------+
|         1 |        1 |
|         1 |        2 |
|         1 |        3 |
|         2 |        1 |
|         2 |        2 |
|         2 |        3 |
|         3 |        1 |
+-----------+----------+
7 rows in set (0.00 sec)
```

使用 group_concat 配合 group by 来分组，来得到实际需要的结果：

```
mysql> select parent_id,group_concat(distinct child_id) as c_id
    -> from tb_menu
    -> group by parent_id;
+-----------+-------+
| parent_id | c_id  |
+-----------+-------+
|         1 | 1,2,3 |
|         2 | 1,2,3 |
|         3 | 1     |
+-----------+-------+
3 rows in set (0.00 sec)
```

回溯统计: with rollup: 任何一个分组后都会有一个小组, 最后都需要向上级分组进行汇报统计: 根据当前分组的字段. 这就是回溯统计: 回溯统计的时候会将分组字段置空.

```
mysql> select *,count(*) from tb_menu group by parent_id;
+-----------+----------+----------+
| parent_id | child_id | count(*) |
+-----------+----------+----------+
|         1 |        1 |        3 |
|         2 |        1 |        3 |
|         3 |        1 |        1 |
+-----------+----------+----------+
3 rows in set (0.00 sec)

mysql> select *,count(*) from tb_menu group by parent_id with rollup;
+-----------+----------+----------+
| parent_id | child_id | count(*) |
+-----------+----------+----------+
|         1 |        1 |        3 |
|         2 |        1 |        3 |
|         3 |        1 |        1 |
|      NULL |        1 |        7 |
+-----------+----------+----------+
4 rows in set (0.00 sec)
```

这是个很强大的函数功能，所以 我打算再详细的探讨一下。
比如，我们有个账单表，记录了最近一天消费的信息。

```
mysql> create table bill(
    -> id int primary key auto_increment,
    -> item_name varchar(20) not null,
    -> item_price double,
    -> item_num int)charset utf8;
Query OK, 0 rows affected (0.27 sec)
```

里面的数据表示我最近一天的消费。

```
mysql> select * from bill;
+----+-----------+------------+----------+
| id | item_name | item_price | item_num |
+----+-----------+------------+----------+
|  1 | 早餐面条  |          5 |        1 |
|  2 | 早餐牛奶  |          5 |        1 |
|  3 | 午餐套餐  |         15 |        1 |
|  4 | 晚餐套餐  |         12 |        1 |
+----+-----------+------------+----------+
4 rows in set (0.00 sec)
```

用 sum 统计单项的消费如下：

```
mysql> update bill set item_num = 5  where id=3;
Query OK, 1 row affected (0.05 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select *,sum(item_price*item_num) from bill group by id;
+----+-----------+------------+----------+--------------------------+
| id | item_name | item_price | item_num | sum(item_price*item_num) |
+----+-----------+------------+----------+--------------------------+
|  1 | 早餐面条  |          5 |        1 |                        5 |
|  2 | 早餐牛奶  |          5 |        1 |                        5 |
|  3 | 午餐套餐  |         15 |        5 |                       75 |
|  4 | 晚餐套餐  |         12 |        1 |                       12 |
+----+-----------+------------+----------+--------------------------+
4 rows in set (0.00 sec)
```

使用 with rollup 进行最后的汇总如下：

```
mysql> select *,sum(item_price*item_num) from bill group by id with rollup;
+----+-----------+------------+----------+--------------------------+
| id | item_name | item_price | item_num | sum(item_price*item_num) |
+----+-----------+------------+----------+--------------------------+
|  1 | 早餐面条  |          5 |        1 |                        5 |
|  2 | 早餐牛奶  |          5 |        1 |                        5 |
|  3 | 午餐套餐  |         15 |        5 |                       75 |
|  4 | 晚餐套餐  |         12 |        1 |                       12 |
| NULL | 晚餐套餐 |         12 |        1 |                       97 |
+----+-----------+------------+----------+--------------------------+
5 rows in set (0.00 sec)
```

多字段回溯: 考虑第一层分组会有此回溯: 第二次分组要看第一次分组的组数, 组数是多少, 回溯就是多少,然后加上第一层回溯即可.

```
mysql> select * ,sum(item_price*item_num) as cost from bill group by id,item_name;
+----+-----------+------------+----------+------+
| id | item_name | item_price | item_num | cost |
+----+-----------+------------+----------+------+
|  1 | 早餐面条  |          5 |        1 |    5 |
|  2 | 早餐牛奶  |          5 |        1 |    5 |
|  3 | 午餐套餐  |         15 |        5 |   75 |
|  4 | 晚餐套餐  |         12 |        1 |   12 |
+----+-----------+------------+----------+------+
4 rows in set (0.00 sec)

mysql> select * ,sum(item_price*item_num) as cost from bill group by id,item_name with rollup;
+----+-----------+------------+----------+------+
| id | item_name | item_price | item_num | cost |
+----+-----------+------------+----------+------+
|  1 | 早餐面条  |          5 |        1 |    5 |
|  1 | NULL      |          5 |        1 |    5 |
|  2 | 早餐牛奶  |          5 |        1 |    5 |
|  2 | NULL      |          5 |        1 |    5 |
|  3 | 午餐套餐  |         15 |        5 |   75 |
|  3 | NULL      |         15 |        5 |   75 |
|  4 | 晚餐套餐  |         12 |        1 |   12 |
|  4 | NULL      |         12 |        1 |   12 |
| NULL | NULL    |         12 |        1 |   97 |
+----+-----------+------------+----------+------+
9 rows in set (0.00 sec)
```

having 子句

having 子句: 与 where 子句一样: 进行条件判断的.

where 是针对磁盘数据进行判断: 进入到内存之后,会进行分组操作: 分组结果就需要 having 来处理.

having 能做 where 能做的几乎所有事情, 但是 where 却不能做 having 能做的很多事情.
having 子句在查询过程中慢于聚合语句(sum,min,max,avg,count).而 where 子句在查询过程中则快于聚合语句(sum,min,max,avg,count)。

```
mysql> select *,sum(item_price*item_num) from bill where item_price >=12;
+----+-----------+------------+----------+---------------------------+
| id | item_name | item_price | item_num | sum(item_price*item_num)  |
+----+-----------+------------+----------+---------------------------+
|  3 | 午餐套餐  |         15 |        5 |                        87 |
+----+-----------+------------+----------+---------------------------+
1 row in set (0.00 sec)
```

比如上面，先走的是 where 条件，判断 item_price>=12 然后再执行 sum。

1. 分组统计的结果或者说统计函数都只有 having 能够使用.

```
mysql> select * from bill group by id having avg(item_price*item_num) >=12;
+----+-----------+------------+----------+
| id | item_name | item_price | item_num |
+----+-----------+------------+----------+
|  3 | 午餐套餐  |         15 |        5 |
|  4 | 晚餐套餐  |         12 |        1 |
+----+-----------+------------+----------+
2 rows in set (0.00 sec)
```

where 却不行。

```
mysql> select * from bill where avg(item_price*item_num) >=12 group by id;
ERROR 1111 (HY000): Invalid use of group function
```

2. having 能够使用字段别名: where 不能: where 是从磁盘取数据,而名字只可能是字段名:

别名是在字段进入到内存后才会产生.这也能很好的解释他们的执行时机。

```
mysql> select *,sum(item_price*item_num) as cost from bill
    -> group by id having cost>=12;
+----+-----------+------------+----------+------+
| id | item_name | item_price | item_num | cost |
+----+-----------+------------+----------+------+
|  3 | 午餐套餐  |         15 |        5 |   75 |
|  4 | 晚餐套餐  |         12 |        1 |   12 |
+----+-----------+------------+----------+------+
2 rows in set (0.00 sec)

mysql> select *,sum(item_price*item_num) as cost from bill
    -> where cost>=12 group by id;
ERROR 1054 (42S22): Unknown column 'cost' in 'where clause'
```

结论：统计分组数据时用到聚合语句，对分组数据再次判断时要用 having。如果不用这些关系就不存在使用 having。直接使用 where 就行了。
having 就是来弥补 where 在分组数据判断时的不足。因为 where 要快于聚合语句。

order by 子句

order by: 排序, 根据某个字段进行升序或者降序排序, 依赖校对集.

使用基本语法
order by 字段名 [asc|desc]; -- asc 是升序(默认的),desc 是降序

```
mysql> select * from bill order by id;
+----+-----------+------------+----------+
| id | item_name | item_price | item_num |
+----+-----------+------------+----------+
|  1 | 早餐面条  |          5 |        1 |
|  2 | 早餐牛奶  |          5 |        1 |
|  3 | 午餐套餐  |         15 |        5 |
|  4 | 晚餐套餐  |         12 |        1 |
|  7 | 下午茶    |         10 |        2 |
|  8 | 夜宵      |         20 |        5 |
+----+-----------+------------+----------+
6 rows in set (0.00 sec)

mysql> select * from bill order by id desc;
+----+-----------+------------+----------+
| id | item_name | item_price | item_num |
+----+-----------+------------+----------+
|  8 | 夜宵      |         20 |        5 |
|  7 | 下午茶    |         10 |        2 |
|  4 | 晚餐套餐  |         12 |        1 |
|  3 | 午餐套餐  |         15 |        5 |
|  2 | 早餐牛奶  |          5 |        1 |
|  1 | 早餐面条  |          5 |        1 |
+----+-----------+------------+----------+
6 rows in set (0.00 sec)
```

排序可以进行多字段排序: 先根据某个字段进行排序, 然后排序好的内部,再按照某个数据进行再次排序:

```
mysql> select * from bill order by id desc;
+----+-----------+------------+----------+
| id | item_name | item_price | item_num |
+----+-----------+------------+----------+
|  8 | 夜宵      |         20 |        5 |
|  7 | 下午茶    |         10 |        2 |
|  4 | 晚餐套餐  |         12 |        1 |
|  3 | 午餐套餐  |         15 |        5 |
|  2 | 早餐牛奶  |          5 |        1 |
|  1 | 早餐面条  |          5 |        1 |
+----+-----------+------------+----------+
6 rows in set (0.00 sec)

mysql> select * from bill order by id ,item_num desc;
+----+-----------+------------+----------+
| id | item_name | item_price | item_num |
+----+-----------+------------+----------+
|  1 | 早餐面条  |          5 |        1 |
|  2 | 早餐牛奶  |          5 |        1 |
|  3 | 午餐套餐  |         15 |        5 |
|  4 | 晚餐套餐  |         12 |        1 |
|  7 | 下午茶    |         10 |        2 |
|  8 | 夜宵      |         20 |        5 |
+----+-----------+------------+----------+
6 rows in set (0.00 sec)

mysql> select * from bill order by id desc,item_num desc;
+----+-----------+------------+----------+
| id | item_name | item_price | item_num |
+----+-----------+------------+----------+
|  8 | 夜宵      |         20 |        5 |
|  7 | 下午茶    |         10 |        2 |
|  4 | 晚餐套餐  |         12 |        1 |
|  3 | 午餐套餐  |         15 |        5 |
|  2 | 早餐牛奶  |          5 |        1 |
|  1 | 早餐面条  |          5 |        1 |
+----+-----------+------------+----------+
6 rows in set (0.00 sec)
```

注意，desc/asc 是紧跟在排序的字段后面的。

limit 子句

limit 子句是一种限制结果的语句: 限制数量.

limit 有两种使用方式

方案 1: 只用来限制长度(数据量): limit 数据量;
limit 默认取前 num 条数据记录：

```
mysql> select * from bill limit 3;
+----+-----------+------------+----------+
| id | item_name | item_price | item_num |
+----+-----------+------------+----------+
|  1 | 早餐面条  |          5 |        1 |
|  2 | 早餐牛奶  |          5 |        1 |
|  3 | 午餐套餐  |         15 |        5 |
+----+-----------+------------+----------+
3 rows in set (0.00 sec)

mysql> select * from bill;
+----+-----------+------------+----------+
| id | item_name | item_price | item_num |
+----+-----------+------------+----------+
|  1 | 早餐面条  |          5 |        1 |
|  2 | 早餐牛奶  |          5 |        1 |
|  3 | 午餐套餐  |         15 |        5 |
|  4 | 晚餐套餐  |         12 |        1 |
|  7 | 下午茶    |         10 |        2 |
|  8 | 夜宵      |         20 |        5 |
+----+-----------+------------+----------+
6 rows in set (0.00 sec)
```

方案 2: 限制起始位置,限制数量: limit 起始位置,长度;

```
mysql> select * from bill limit 0,2;
+----+-----------+------------+----------+
| id | item_name | item_price | item_num |
+----+-----------+------------+----------+
|  1 | 早餐面条  |          5 |        1 |
|  2 | 早餐牛奶  |          5 |        1 |
+----+-----------+------------+----------+
2 rows in set (0.00 sec)

mysql> select * from bill limit 2,2;
+----+-----------+------------+----------+
| id | item_name | item_price | item_num |
+----+-----------+------------+----------+
|  3 | 午餐套餐  |         15 |        5 |
|  4 | 晚餐套餐  |         12 |        1 |
+----+-----------+------------+----------+
2 rows in set (0.00 sec)

mysql> select * from bill limit 3,2;
+----+-----------+------------+----------+
| id | item_name | item_price | item_num |
+----+-----------+------------+----------+
|  4 | 晚餐套餐  |         12 |        1 |
|  7 | 下午茶    |         10 |        2 |
+----+-----------+------------+----------+
2 rows in set (0.00 sec)
```

limit 方案 2 主要用来实现数据的分页: 为用户节省时间,提交服务器的响应效率, 减少资源

的浪费.
对于用户来讲: 可以点击的分页按钮: 1,2,3,4
对于服务器来讲: 根据用户选择的页码来获取不同的数据: limit offset,length;

length: 每页显示的数据量: 基本不变
offset: offset = (页码 - 1) * 每页显示量

这个在以后的程序里再写个例子。

exists 子查询

exists: 是否存在的意思, exists 子查询就是用来判断某些条件是否满足(跨表), exists 是接在 where 之后: exists 返回的结果只有 0 和 1.

a) 比如，我们在最开始创建数据库和表的时候使用的 if not exists 语句，这可以判断是否存在数据库或者表，存在就删除。

```
mysql> create table if not exists tb_test(
    -> id int)charset utf8;
Query OK, 0 rows affected (0.48 sec)
```

b) 判断 select 语句的结果是否存在
先判断数据源，然后再判断是否满足条件：

```
mysql> select * from bill where
    -> exists(select * from bill where id=1);
+----+-----------+------------+----------+
| id | item_name | item_price | item_num |
+----+-----------+------------+----------+
|  1 | 早餐面条  |          5 |        1 |
|  2 | 早餐牛奶  |          5 |        1 |
|  3 | 午餐套餐  |         15 |        5 |
|  4 | 晚餐套餐  |         12 |        1 |
|  7 | 下午茶    |         10 |        2 |
|  8 | 夜宵      |         20 |        5 |
+----+-----------+------------+----------+
6 rows in set (0.00 sec)
```

c) exists 和 in
这是两个功能相似的关键字，在面试的过程中很频繁的被问到，
直接看使用， 这错误很明显的告诉我们，in 对应的是一条字段。

```
mysql> select * from bill where id
    -> in (select * from bill where id=1);
ERROR 1241 (21000): Operand should contain 1 column(s)
```

```
mysql> select * from bill where id
    -> in (select id from bill);
+----+-----------+------------+----------+
| id | item_name | item_price | item_num |
+----+-----------+------------+----------+
|  1 | 早餐面条  |          5 |        1 |
|  2 | 早餐牛奶  |          5 |        1 |
|  3 | 午餐套餐  |         15 |        5 |
|  4 | 晚餐套餐  |         12 |        1 |
|  7 | 下午茶    |         10 |        2 |
|  8 | 夜宵      |         20 |        5 |
+----+-----------+------------+----------+
6 rows in set (0.00 sec)
```

```
mysql> explain select * from bill where id
    -> in (select id from bill);
+----+-------------+-------+------------+--------+---------------+---------+---------+---------------+------+----------+-------------+
| id | select_type | table | partitions | type   | possible_keys | key     | key_len | ref           | rows | filtered | Extra       |
+----+-------------+-------+------------+--------+---------------+---------+---------+---------------+------+----------+-------------+
|  1 | SIMPLE      | bill  | NULL       | ALL    | PRIMARY       | NULL    | NULL    | NULL          |    6 |   100.00 | NULL        |
|  1 | SIMPLE      | bill  | NULL       | eq_ref | PRIMARY       | PRIMARY | 4       | c3gen.bill.id |    1 |   100.00 | Using index |
+----+-------------+-------+------------+--------+---------------+---------+---------+---------------+------+----------+-------------+
2 rows in set, 1 warning (0.00 sec)
```

```
mysql> explain select * from bill where
    -> exists(select * from bill where id=1);
+----+-------------+-------+------------+-------+---------------+---------+---------+-------+------+----------+-------------+
| id | select_type | table | partitions | type  | possible_keys | key     | key_len | ref   | rows | filtered | Extra       |
+----+-------------+-------+------------+-------+---------------+---------+---------+-------+------+----------+-------------+
|  1 | PRIMARY     | bill  | NULL       | ALL   | NULL          | NULL    | NULL    | NULL  |    6 |   100.00 | NULL        |
|  2 | SUBQUERY    | bill  | NULL       | const | PRIMARY       | PRIMARY | 4       | const |    1 |   100.00 | Using index |
+----+-------------+-------+------------+-------+---------------+---------+---------+-------+------+----------+-------------+
2 rows in set, 1 warning (0.00 sec)
```

对比一下两个 sql 的执行，其中 eq_ref 表示对于每个来自于前面的表的行组合，从该表中读取一行，const 表示表最多有一个匹配行，它将在查询开始时被读取。因为仅有一行，在这行的列值可被优化器剩余部分认为是常数。

所以 可以推断 exists 执行是一种循环操作，而 in 则是一种连接操作。所以当两个表的数据差不多大的时候，这两者的区别不大，但是数据量大的时候 exists 是要优于 in。

2.5.5 联合查询 union

联合查询: 将多次查询(多条 select 语句), 在记录上进行拼接(字段不会增加)。

看下面的例子：

```
mysql> select * from bill
    -> union
    -> select * from bill;
+----+-----------+------------+----------+
| id | item_name | item_price | item_num |
+----+-----------+------------+----------+
|  1 | 早餐面条  |          5 |        1 |
|  2 | 早餐牛奶  |          5 |        1 |
|  3 | 午餐套餐  |         15 |        5 |
|  4 | 晚餐套餐  |         12 |        1 |
|  7 | 下午茶    |         10 |        2 |
|  8 | 夜宵      |         20 |        5 |
+----+-----------+------------+----------+
6 rows in set (0.00 sec)

mysql> select * from bill
    -> union all
    -> select * from bill;
+----+-----------+------------+----------+
| id | item_name | item_price | item_num |
+----+-----------+------------+----------+
|  1 | 早餐面条  |          5 |        1 |
|  2 | 早餐牛奶  |          5 |        1 |
|  3 | 午餐套餐  |         15 |        5 |
|  4 | 晚餐套餐  |         12 |        1 |
|  7 | 下午茶    |         10 |        2 |
|  8 | 夜宵      |         20 |        5 |
|  1 | 早餐面条  |          5 |        1 |
|  2 | 早餐牛奶  |          5 |        1 |
|  3 | 午餐套餐  |         15 |        5 |
|  4 | 晚餐套餐  |         12 |        1 |
|  7 | 下午茶    |         10 |        2 |
|  8 | 夜宵      |         20 |        5 |
+----+-----------+------------+----------+
12 rows in set (0.00 sec)
```

union 选项: 与 select 选项一样有两个

all: 保留所有(不管重复)

distinct: 去重(整个重复): 默认的

****联合查询只要求字段一样, 跟数据类型无关

为此，我们创建一个新表 bill2，如下：

```
mysql> create table bill2(
    -> id bigint primary key auto_increment,
    -> item_name char(4),
    -> item_cost double)charset utf8;
Query OK, 0 rows affected (0.20 sec)

mysql> insert into bill2 values(null,'b2',20);
Query OK, 1 row affected (0.05 sec)

mysql> insert into bill2 values(null,'下午茶',10);
Query OK, 1 row affected (0.07 sec)
```

将两张表做联合查询，如下：

```
mysql> select * from bill
    -> union all
    -> select * from bill2;
ERROR 1222 (21000): The used SELECT statements have a different number of columns
mysql> alter table bill2 add item_notes varchar(20);
Query OK, 0 rows affected (0.48 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> select * from bill
    -> union all
    -> select * from bill2;
+----+-----------+------------+----------+
| id | item_name | item_price | item_num |
+----+-----------+------------+----------+
|  1 | 早餐面条  |          5 | 1        |
|  2 | 早餐牛奶  |          5 | 1        |
|  3 | 午餐套餐  |         15 | 5        |
|  4 | 晚餐套餐  |         12 | 1        |
|  7 | 下午茶    |         10 | 2        |
|  8 | 夜宵      |         20 | 5        |
|  1 | b2        |         20 | NULL     |
|  2 | 下午茶    |         10 | NULL     |
+----+-----------+------------+----------+
8 rows in set (0.00 sec)
```

应用场景：

1. 查询同一张表,但是需求不同: 如查询账单 Bill 信息, 价格升序, 数量降序.
2. 多表查询: 多张表的结构是完全一样的,保存的数据(结构)也是一样的.

例子：对账单表的价格升序、 数量降序

union 的联合是两个整体 ，所以一定要带括号：

```
mysql> (select * from bill order by item_price asc)
    -> union
    -> (select * from bill order by item_num desc)
    -> ;
+----+-----------+------------+----------+
| id | item_name | item_price | item_num |
+----+-----------+------------+----------+
|  1 | 早餐面条  |          5 |        1 |
|  2 | 早餐牛奶  |          5 |        1 |
|  3 | 午餐套餐  |         15 |        5 |
|  4 | 晚餐套餐  |         12 |        1 |
|  7 | 下午茶    |         10 |        2 |
|  8 | 夜宵      |         20 |        5 |
+----+-----------+------------+----------+
6 rows in set (0.00 sec)

mysql> select * from bill;
+----+-----------+------------+----------+
| id | item_name | item_price | item_num |
+----+-----------+------------+----------+
|  1 | 早餐面条  |          5 |        1 |
|  2 | 早餐牛奶  |          5 |        1 |
|  3 | 午餐套餐  |         15 |        5 |
|  4 | 晚餐套餐  |         12 |        1 |
|  7 | 下午茶    |         10 |        2 |
|  8 | 夜宵      |         20 |        5 |
+----+-----------+------------+----------+
6 rows in set (0.00 sec)
```

会发现好像并没有达到想要的效果，怎么办？？ 使用 limit 并且给最大值：

```
mysql> (select * from bill order by item_price asc limit 9999999)
    -> union
    -> (select * from bill order by item_num desc limit 9999999);
+----+-----------+------------+----------+
| id | item_name | item_price | item_num |
+----+-----------+------------+----------+
|  1 | 早餐面条  |          5 |        1 |
|  2 | 早餐牛奶  |          5 |        1 |
|  7 | 下午茶    |         10 |        2 |
|  4 | 晚餐套餐  |         12 |        1 |
|  3 | 午餐套餐  |         15 |        5 |
|  8 | 夜宵      |         20 |        5 |
+----+-----------+------------+----------+
6 rows in set (0.00 sec)
```

2.5.5 连接查询 join

这是 sql 语句中最常用的部分，所以在把一些基础的东西说完后，就重点写一下连接查询。SQL 中将连接查询分成四类: 内连接,外连接,自然连接和交叉连接。

先准备两张表，一张 t1，一张 t2，如下：

```
mysql> create table t2(
    -> p_id int primary key,
    -> u_id int not null,
    -> foreign key(u_id) references t1(u_id))charset utf8;
Query OK, 0 rows affected (0.28 sec)

mysql> insert into t1 values(1,'测试1');
Query OK, 1 row affected (0.08 sec)

mysql> insert into t1 values(2,'测试2');
Query OK, 1 row affected (0.06 sec)

mysql> insert into t2 values(1,1),(2,1),(3,2);
Query OK, 3 rows affected (0.07 sec)
Records: 3  Duplicates: 0  Warnings: 0
```

i. 交叉连接 cross join

交叉连接: cross join, 从一张表中循环取出每一条记录, 每条记录都去另外一张表进行匹配: 匹配一定保留(没有条件匹配), 而连接本身字段就会增加(保留),最终形成的结果叫做: 笛卡尔积.

基本语法: 左表 cross join 右表; ===== from 左表,右表;

```
mysql> select * from t1 cross join t2;
+------+-------+------+------+
| u_id | name  | p_id | u_id |
+------+-------+------+------+
|    1 | 测试1 |    1 |    1 |
|    2 | 测试2 |    1 |    1 |
|    1 | 测试1 |    2 |    1 |
|    2 | 测试2 |    2 |    1 |
|    1 | 测试1 |    3 |    2 |
|    2 | 测试2 |    3 |    2 |
+------+-------+------+------+
6 rows in set (0.00 sec)
```

笛卡尔积没有意义: 应该尽量避免(交叉连接没用)，我们在实际中，可以使用 CROSS JOIN 生成大量的测试数据。
交叉连接存在的价值: 保证连接这种结构的完整性

ii. 内连接 inner join

内连接: [inner] join, 从左表中取出每一条记录,去右表中与所有的记录进行匹配: 匹配必须是某个条件在左表中与右表中相同最终才会保留结果,否则不保留.

基本语法
左表 [inner] join 右表 on 左表.字段 = 右表.字段; on 表示连接条件: 条件字段就是代表相同的业务含义(如 a.id 和 b.cus_id),通常情况下会使用表的别名，这样查询就看起来更简洁。

```
mysql> select * from t1 as a
    -> inner join t2 as b
    -> on a.u_id = b.u_id;
+------+-------+------+------+
| u_id | name  | p_id | u_id |
+------+-------+------+------+
|    1 | 测试1 |    1 |    1 |
|    1 | 测试1 |    2 |    1 |
|    2 | 测试2 |    3 |    2 |
+------+-------+------+------+
3 rows in set (0.00 sec)
```

内连接可以没有连接条件: 没有 on 之后的内容,这个时候系统会保留所有结果(笛卡尔积)

```
mysql> select * from t1 as a
    -> inner join t2 as b;
+------+-------+------+------+
| u_id | name  | p_id | u_id |
+------+-------+------+------+
|    1 | 测试1 |    1 |    1 |
|    2 | 测试2 |    1 |    1 |
|    1 | 测试1 |    2 |    1 |
|    2 | 测试2 |    2 |    1 |
|    1 | 测试1 |    3 |    2 |
|    2 | 测试2 |    3 |    2 |
+------+-------+------+------+
6 rows in set (0.00 sec)
```

内连接还可以使用 where 代替 on 关键字，在 csdn 上看到帖子上 where 的性能比 on 低，这个实在没有测试，一般也很少用 where 来代替 on。

```
mysql> select * from t1 as a
    -> inner join t2 as b
    -> where a.u_id =b.u_id;
+------+-------+------+------+
| u_id | name  | p_id | u_id |
+------+-------+------+------+
|    1 | 测试1 |    1 |    1 |
|    1 | 测试1 |    2 |    1 |
|    2 | 测试2 |    3 |    2 |
+------+-------+------+------+
3 rows in set (0.00 sec)
```

****当关联的表的条件字段是一样的时候，可以使用 using 来简写 on 条件

```
mysql> select * from t1 inner join t2 using(u_id);
+------+-------+------+
| u_id | name  | p_id |
+------+-------+------+
|    1 | 测试1 |    1 |
|    1 | 测试1 |    2 |
|    2 | 测试2 |    3 |
+------+-------+------+
3 rows in set (0.00 sec)
```

iii. 外连接 outer join

外连接: outer join, 以某张表为主,取出里面的所有记录, 然后每条与另外一张表进行连接: 不管能不能匹配上条件,最终都会保留: 能比配,正确保留; 不能匹配,其他表的字段都置空 null.

外连接分为两种: 是以某张表为主: 有主表

left join: 左外连接(左连接), 以左表为主表

right join: 右外连接(右连接), 以右表为主表

基本语法: 左表 left/right join 右表 on 左表.字段 = 右表.字段;

左连接：左表为主数据表，最终的记录不少于左表的记录：当副表没有匹配的记录时就为 null。

```
mysql> insert into t1 values(3,'其他');
Query OK, 1 row affected (0.07 sec)

mysql> select a.u_id,a.name,b.p_id from
    -> t1 as a left join t2 as b
    -> on a.u_id=b.u_id;
+------+-------+------+
| u_id | name  | p_id |
+------+-------+------+
|    1 | 测试1 |    1 |
|    1 | 测试1 |    2 |
|    2 | 测试2 |    3 |
|    3 | 其他  | NULL |
+------+-------+------+
4 rows in set (0.00 sec)
```

右连接：右表为主数据表，最终的记录不少于右表的记录：

```
mysql> select a.u_id,a.name,b.p_id from
    -> t1 as a right join t2 as b
    -> on a.u_id=b.u_id;
+------+-------+------+
| u_id | name  | p_id |
+------+-------+------+
|    1 | 测试1 |    1 |
|    1 | 测试1 |    2 |
|    2 | 测试2 |    3 |
+------+-------+------+
3 rows in set (0.00 sec)
```

虽然左连接和右连接有主表差异, 但是显示的结果: 左表的数据在左边,右表数据在右边.

左连接和右连接可以互转.

****关于内外连接的差别：
在 sql 逻辑查询语句执行的前三步中，inner join 会执行第一步和第二步；即没有第三步，不添加外部行，这是 inner join 和接下来要说的 outer join 的最大区别之一。

通过 outer join，我们可以按照一些过滤条件来匹配表之间的数据。outer join 的结果集等于 inner join 的结果集加上外部行；也就是说，在使用 outer join 时，sql 逻辑查询语句执行的前三步，都会执行一遍。

*****下面是 SELECT 语句的逻辑执行顺序：

1.from
2.on
3.join
4.where
5.group by
6.with cube or with rollup
7.having
8.select
9.distinct
10.order by
11.top

这部分同索引一样都可以优化 sql 查询，以后再探讨。

iv. 自然连接 natural join

自然连接: natural join, 自然连接, 就是自动匹配连接条件: 系统以字段名字作为匹配模式(同名字段就作为条件, 多个同名字段都作为条件).

natural join 等同于 inner(outer) join 与 using 的组合，它隐含的作用是将两个表中具有相同名称的列进行匹配。同样的，natural left(right) join 等同于 left(right) join 与 using 的组合。比如：

```
mysql> select * from t1 join t2 using(u_id);
+------+-------+------+
| u_id | name  | p_id |
+------+-------+------+
|    1 | 测试1 |    1 |
|    1 | 测试1 |    2 |
|    2 | 测试2 |    3 |
+------+-------+------+
3 rows in set (0.00 sec)

mysql> select * from t1 natural join t2;
+------+-------+------+
| u_id | name  | p_id |
+------+-------+------+
|    1 | 测试1 |    1 |
|    1 | 测试1 |    2 |
|    2 | 测试2 |    3 |
+------+-------+------+
3 rows in set (0.00 sec)
```

```
mysql> select * from t1 left join t2 using(u_id);
+------+-------+------+
| u_id | name  | p_id |
+------+-------+------+
|    1 | 测试1 |    1 |
|    1 | 测试1 |    2 |
|    2 | 测试2 |    3 |
|    3 | 其他  | NULL |
+------+-------+------+
4 rows in set (0.00 sec)

mysql> select * from t1 natural left join t2;
+------+-------+------+
| u_id | name  | p_id |
+------+-------+------+
|    1 | 测试1 |    1 |
|    1 | 测试1 |    2 |
|    2 | 测试2 |    3 |
|    3 | 其他  | NULL |
+------+-------+------+
4 rows in set (0.00 sec)
```

v. 直连接 straight_join

这是官方 api 之外的一种连接方式，是在长期的使用中积累而来，可以优化 sql 的执行。

看例子： 使用 explain 来查看 sql 的执行：

```
mysql> explain select * from t1 join t2 using(u_id);
+----+-------------+-------+------------+-------+---------------+------+---------+------+------+----------+-------------+
| id | select_type | table | partitions | type  | possible_keys | key  | key_len | ref  | rows | filtered | Extra       |
+----+-------------+-------+------------+-------+---------------+------+---------+------+------+----------+-------------+
|  1 | SIMPLE      | t2    | NULL       | index | u_id          | u_id | 4       | NULL |    3 |   100.00 | Using index |
|  1 | SIMPLE      | t1    | NULL       | ALL   | PRIMARY       | NULL | NULL    | NULL |    3 |    33.33 | Using where |
+----+-------------+-------+------------+-------+---------------+------+---------+------+------+----------+-------------+
2 rows in set, 1 warning (0.00 sec)
```

可以很清楚的看到，mysql 是先选择的 t2 表，然后再进行的匹配。
下面我们使用 straight_join 来改变这种选择：

```
mysql> explain select * from t1 straight_join t2 on t1.u_id=t2.u_id;
+----+-------------+-------+------------+------+---------------+------+---------+--------------+------+----------+-------------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref          | rows | filtered | Extra       |
+----+-------------+-------+------------+------+---------------+------+---------+--------------+------+----------+-------------+
|  1 | SIMPLE      | t1    | NULL       | ALL  | PRIMARY       | NULL | NULL    | NULL         |    3 |   100.00 | NULL        |
|  1 | SIMPLE      | t2    | NULL       | ref  | u_id          | u_id | 4       | c3gen.t1.u_id |    1 |   100.00 | Using index |
+----+-------------+-------+------------+------+---------------+------+---------+--------------+------+----------+-------------+
2 rows in set, 1 warning (0.00 sec)
```

当指定 straight_join 方式以后，mysql 就会先选择 t1 表，然后再进行的匹配。
这样做的好处是，当大量的数据需要处理时，我们可以精确的控制 mysql 的选择。

多表连接: A 表 inner join B 表 on 条件 left join C 表 on 条件 ...
执行顺序: A 表内连接 B 表,得到一个二维表, 左连接 C 表形成二维表...

第三章 sql 高级部分

3.1 变量

变量分为两种: 系统变量和自定义变量

3.1.1 系统变量

系统定义好的变量: 大部分的时候用户根本不需要使用系统变量: 系统变量是用来控制服务器的表现的: 如 autocommit, auto_increment_increment 等

查看系统变量

show variables; -- 查看所有系统变量

```
| version_comment                                           | MySQL Community Server (GPL)
| version_compile_machine                                   | x86_64
| version_compile_os                                        | Win64
| wait_timeout                                              | 28800
| warning_count                                             | 0
+-----------------------------------------------------------+-----------------------------------
-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
504 rows in set, 1 warning (0.00 sec)
```

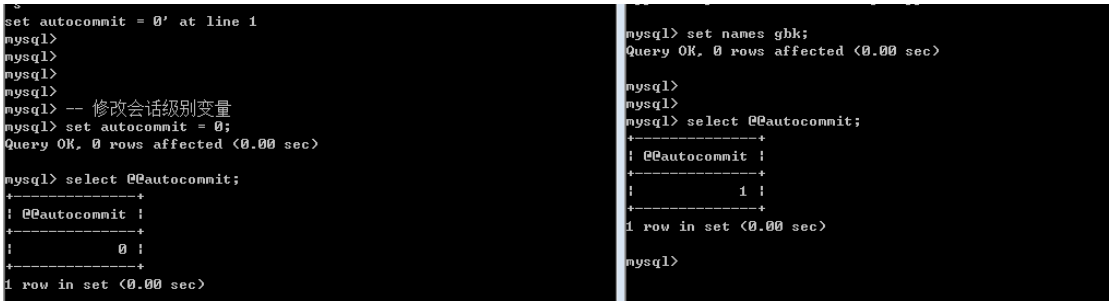
查看具体变量值: 任何一个有数据返回的内容都是由 select 查看
select @@变量名;

```
| wait_timeout                                              | 28800
| warning_count                                             | 0
+-----------------------------------------------------------+-----------------------------------
-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
504 rows in set, 1 warning (0.00 sec)

mysql> select @@wait_timeout;
+----------------+
| @@wait_timeout |
+----------------+
|          28800 |
+----------------+
1 row in set (0.00 sec)
```

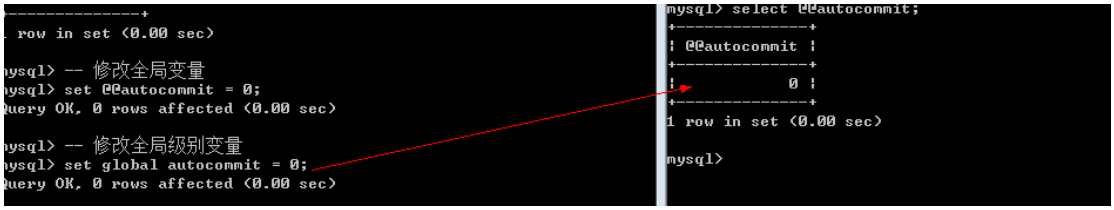
修改系统变量

修改系统变量分为两种方式: 会话级别和全局级别。系统变量尽量少修改，要修改也多是会话级别的。

****会话级别: 临时修改, 当前客户端当次连接有效
set 变量名 = 值;/set @@变量名 = 值;



全局级别: 一次修改,永久生效(对所有客户端都生效)
set global 变量名 = 值;



如果对方(其他)客户端当前已经连上服务器,那么当次修改无效,要退出重新登录才会生效

3.1.2 自定义变量

定义变量

系统为了区分系统变量, 规定用户自定义变量必须使用一个@符号
set @变量名 = 值;

```
mysql> set @c_name = 'c3gen';
Query OK, 0 rows affected (0.00 sec)
```

查看变量

自定义变量也是类似系统变量查看
select @变量名;

```
mysql> select @c_name;
+----------+
| @c_name  |
+----------+
| c3gen    |
+----------+
1 row in set (0.00 sec)
```

在 mysql 中, “=”会默认的当做比较符号处理(很多地方), mysql 为了区分比较和赋值的概念: 重新定义了一个新的的赋值符号: :=

```
mysql> set @c_age := 18;
Query OK, 0 rows affected (0.00 sec)

mysql> select @c_age;
+--------+
| @c_age |
+--------+
|     18 |
+--------+
1 row in set (0.00 sec)
```

mysql 允许从数据表中获取数据,然后赋值给变量: 两种方式

方案 1: 边赋值,变查看结果

select @变量名 := 字段名 from 数据源; -- 从字段中取值赋值给变量名, 如果使用=会变成比较。如果 select 返回是多条记录，那么会匹配最后一条记录：

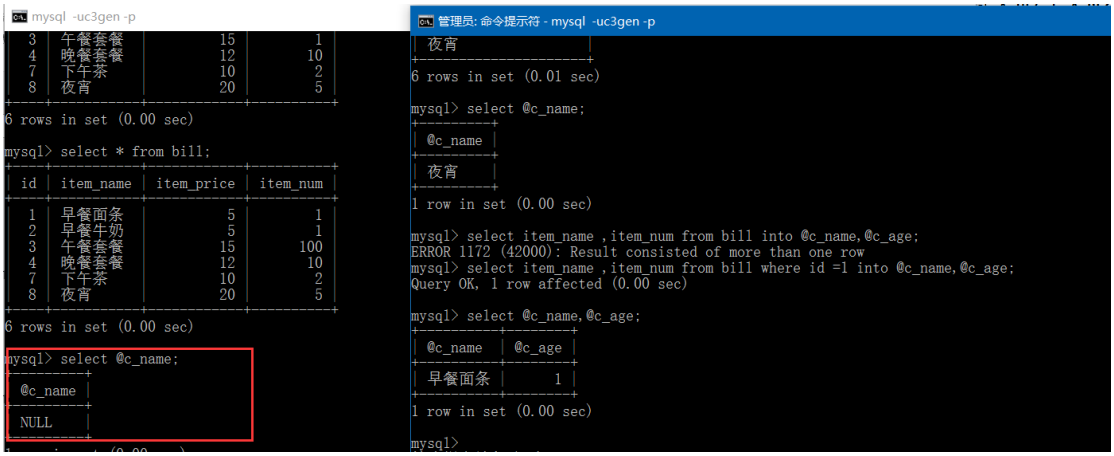
```
mysql> select @c_name :=item_name from bill;
+----------------------+
| @c_name :=item_name  |
+----------------------+
| 早餐面条             |
| 早餐牛奶             |
| 午餐套餐             |
| 晚餐套餐             |
| 下午茶               |
| 夜宵                 |
+----------------------+
6 rows in set (0.01 sec)

mysql> select @c_name;
+---------+
| @c_name |
+---------+
| 夜宵    |
+---------+
1 row in set (0.00 sec)
```

方案 2: 只有赋值不看结果: 要求很严格: 数据记录最多只允许获取一条,这时候并不会自动的给你匹配最后一条记录: 因为 mysql 不支持数组。

select 字段列表 from 表名 into 变量列表;

```
mysql> select item_name ,item_num from bill into @c_name,@c_age;
ERROR 1172 (42000): Result consisted of more than one row
mysql> select item_name ,item_num from bill where id =1 into @c_name,@c_age;
Query OK, 1 row affected (0.00 sec)

mysql> select @c_name,@c_age;
+----------+--------+
| @c_name  | @c_age |
+----------+--------+
| 早餐面条 |      1 |
+----------+--------+
1 row in set (0.00 sec)
```

所有自定义的变量都是会话级别: 当前客户端当次连接有效
所有自定义变量不区分数据库(用户级别)



3.2 函数

函数: 将一段代码块封装到一个结构中, 在需要执行代码块的时候, 调用结构执行即可.(代码复用)

函数分为两类: 系统函数和自定义函数

3.2.1 系统函数

系统定义好的函数, 直接调用即可.mysql 预定义了非常丰富的函数，在实际使用非常多的就是操作字符串。 更多的系统函数，可以查看我的博客
https://my.oschina.net/c3gen/blog/802184
任何函数都有返回值, 因此函数的调用是通过 select 调用.

mysql 中,字符串的基本操作单位(最常见的是字符)
substring: 字符串截取(字符为单位)

例子， 还是用之前创建的变量来试验：

```
mysql> select @c_name;
+---------+
| @c_name |
+---------+
| NULL    |
+---------+
1 row in set (0.00 sec)

mysql> set @c_name='c3gen';
Query OK, 0 rows affected (0.00 sec)

mysql> select @c_name;
+---------+
| @c_name |
+---------+
| c3gen   |
+---------+
1 row in set (0.00 sec)
```

然后 字符串截取： substring(index1,i); index1 表示下标，mysql 的下标是从 1 开始，i 表示截图的位数。

```
mysql> set @c_name='c3gen小胖';
Query OK, 0 rows affected (0.01 sec)

mysql> select @c_name;
+-----------+
| @c_name   |
+-----------+
| c3gen小胖 |
+-----------+
1 row in set (0.00 sec)

mysql> select substring(@c_name,6,2);
+------------------------+
| substring(@c_name,6,2) |
+------------------------+
| 小胖                   |
+------------------------+
1 row in set (0.00 sec)

mysql> select substring(@c_name,1,1);
+------------------------+
| substring(@c_name,1,1) |
+------------------------+
| c                      |
+------------------------+
1 row in set (0.00 sec)
```

char_length: 字符长度

```
mysql> select char_length(@c_name);
+----------------------+
| char_length(@c_name) |
+----------------------+
|                    7 |
+----------------------+
1 row in set (0.11 sec)
```

length: 字节长度

```
mysql> select length(@c_name);
+-----------------+
| length(@c_name) |
+-----------------+
|               9 |
+-----------------+
1 row in set (0.00 sec)
```

instr: 判断字符串是否在某个具体的字符串中存在, 存在返回位置。返回 0 表示没有找到。

```
mysql> select instr(@c_name,'天');
+---------------------+
| instr(@c_name,'天') |
+---------------------+
|                   0 |
+---------------------+
1 row in set (0.01 sec)

mysql> select instr(@c_name,'小');
+---------------------+
| instr(@c_name,'小') |
+---------------------+
|                   6 |
+---------------------+
1 row in set (0.00 sec)
```

insert: 替换,找到目标位置,指定长度的字符串,替换成目标字符串

```
mysql> select @c_name;
+-----------+
| @c_name   |
+-----------+
| c3gen小胖 |
+-----------+
1 row in set (0.00 sec)
```

```
mysql> select insert(@c_name,6,2,'菜鸟');
+----------------------------+
| insert(@c_name,6,2,'菜鸟') |
+----------------------------+
| c3gen菜鸟                  |
+----------------------------+
1 row in set (0.07 sec)
```

strcmp: compare,字符串比较，返回值 -1 表示小，0 表示等于，1 表示大于：

可以发现，mysql 是不区分大小写的。

```
mysql> set @a='hello';
Query OK, 0 rows affected (0.00 sec)

mysql> set @b='HELLO';
Query OK, 0 rows affected (0.00 sec)

mysql> set @c='helloworld';
Query OK, 0 rows affected (0.00 sec)

mysql> select strcmp(@a,@b),strcmp(@a,@c),strcmp(@b,@c);
+---------------+---------------+---------------+
| strcmp(@a,@b) | strcmp(@a,@c) | strcmp(@b,@c) |
+---------------+---------------+---------------+
|             0 |            -1 |            -1 |
+---------------+---------------+---------------+
1 row in set (0.04 sec)
```

3.2.2 自定义函数

函数要素: 函数名, 参数列表(形参和实参), 返回值, 函数体(作用域)

创建函数

创建语法

create function 函数名([形参列表]) returns 数据类型 -- 规定要返回的数据类型
begin
    -- 函数体
    -- 返回值: return 类型(指定数据类型);
end

定义函数

```
mysql> create function addr() returns varchar(20)
    -> return '武汉市';
Query OK, 0 rows affected (0.09 sec)
```

自定义函数与系统函数的调用方式是一样: select 函数名([实参列表]);

```
mysql> select addr();
+--------+
| addr() |
+--------+
| 武汉市 |
+--------+
1 row in set (0.12 sec)
```

查看函数

查看所有函数: show function status [like 'pattern']; 最好是指定函数名，不然会查询所有的函数：

```
mysql> show function status like 'addr'\G;
*************************** 1. row ***************************
                  Db: c3gen
                Name: addr
                Type: FUNCTION
             Definer: c3gen@%
            Modified: 2016-12-08 15:57:45
             Created: 2016-12-08 15:57:45
       Security_type: DEFINER
             Comment:
character_set_client: gbk
collation_connection: gbk_chinese_ci
  Database Collation: utf8_general_ci
1 row in set (0.00 sec)
```

查看函数的创建语句: show create function 函数名;

```
mysql> show create function addr\G;
*************************** 1. row ***************************
           Function: addr
           sql_mode: STRICT_TRANS_TABLES,NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION
    Create Function: CREATE DEFINER=`c3gen`@`%` FUNCTION `addr`() RETURNS varchar(20) CHARSET utf8
return '武汉市'
character_set_client: gbk
collation_connection: gbk_chinese_ci
  Database Collation: utf8_general_ci
1 row in set (0.00 sec)
```

修改函数&删除函数

函数只能先删除后新增,不能修改.

drop function 函数名;

```
mysql> drop function addr;
Query OK, 0 rows affected (0.02 sec)
```

```
mysql> show function status like 'addr'\G;
Empty set (0.00 sec)
```

函数参数

参数分为两种: 定义时的参数叫形参, 调用时的参数叫实参(实参可以是数值也可以是变量)
形参: 要求必须指定数据类型
function 函数名(形参名字 字段类型) returns 数据类型

注意下面的小细节， 我在创建变量 i 和 result 的时候 ：

```
mysql> delimiter ##
mysql> create function tosum(a int) returns int
    -> begin
    -> set @i =1 ;
    -> set @result =0;
    -> while @i <= a do
    -> set @result = @result +@i;
    -> set @i =@i+1;
    -> end while;
    -> return @result;
    -> end
    -> ##
Query OK, 0 rows affected (0.05 sec)

mysql> delimiter ;
```

在函数内部使用 set @定义的变量在函数外部也可以访问，但是在函数没有被调用的时候，参数的值是空的，当函数调用后，参数的值就会被赋予：

```
mysql> select @result;
+---------+
| @result |
+---------+
| NULL    |
+---------+
1 row in set (0.00 sec)

mysql> select tosum(10);
+-----------+
| tosum(10) |
+-----------+
|        55 |
+-----------+
1 row in set (0.00 sec)

mysql> select @result;
+---------+
| @result |
+---------+
|      55 |
+---------+
1 row in set (0.00 sec)
```

作用域

mysql 中的作用域与 js 中的作用域完全一样

全局变量可以在任何地方使用; 局部变量只能在函数内部使用.

全局变量: 使用 set 关键字定义, 使用@符号标志

局部变量: 使用 declare 关键字声明, 没有@符号: 所有的局部变量的声明,必须在函数体开始之前

继续用上面的例子，我改写一下：

```
mysql> delimiter ##
mysql> create function tosum2(a int) returns int
    -> begin
    -> declare i int default 1;
    -> set @result=0;
    -> while i< a do
    -> set @result =@result +@i;
    -> set i = i+1;
    -> end while;
    -> return @result;
    -> end
    -> ##
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter ;
mysql> select tosum2(10);
+------------+
| tosum2(10) |
+------------+
|         99 |
+------------+
1 row in set (0.00 sec)
```

然后 我分别访问 变量 i 和 result ：

结果会怎么样？？？

3.3 代码执行结构

其实 mysql 中，也可以编码，就像前面的触发器、定义变量、事务。同样的，mysql 也有控制流程结构。
代码执行结构有三种: 顺序结构, 分支结构和循环结构

分支结构

分支结构: 实现准备多个代码块, 按照条件选择性执行某段代码.

在 mysql 中只有 if 分支

基本语法

```
if  条件判断  then
    -- 满足条件要执行的代码;
else
    -- 不满足条件要执行的代码;
```

end if;

触发器结合 if 分支: 判断库存是否足够,不够不能继续信息商品信息：

```
mysql> delimiter ##
mysql> create trigger befo_check before insert on goods for each row
    -> begin
    -> select sku_num from stock where id =1 into @sku_num;
    -> if @sku_num <new.sku_num then
    -> insert into abc values (aaaa);
    -> end if;
    -> end
    -> ##
Query OK, 0 rows affected (0.09 sec)

mysql> delimiter ;
```

由于 mysql 没有阻止事件或者方法，于是我们随便写个错误的 sql，就会让它报错退出。
效果

```
mysql> select * from stock;
+----+---------+
| id | sku_num |
+----+---------+
|  1 |      79 |
+----+---------+
1 row in set (0.00 sec)

mysql> insert into goods values(4,'榴莲',100);
ERROR 1146 (42S02): Table 'c3gen.abc' doesn't exist
mysql> select * from stock;
+----+---------+
| id | sku_num |
+----+---------+
|  1 |      79 |
+----+---------+
1 row in set (0.00 sec)

mysql> select * from goods;
+-------+----------+---------+
| go_id | sku_name | sku_num |
+-------+----------+---------+
|     2 | 橘子     |       1 |
|     3 | 苹果     |      20 |
+-------+----------+---------+
2 rows in set (0.00 sec)
```

循环结构

循环结构: 某段代码在指定条件执行重复执行.

while 循环(没有 for 循环)

```
while 条件判断 do
    -- 满足条件要执行的代码
    -- 变更循环条件
end while;
```

循环控制: 在循环内部进行循环判断和控制
mysql 中没有对应 continue 和 break. 但是有替代品.
iterate: 迭代 , 类似 continue, 后面的代码不执行, 循环重新来过
leave: 离开, 类似 break,整个循环接收

使用方式: itrate/leave 循环名字;

```
-- 定义循环名字
循环名字:while 条件 do
    -- 循环体
    -- 循环控制
    leave/iterate 循环名字;
end while;
```

例子， 可以参照上面的 if 语句，在很多情况下 if 和 while 可以互换。
问题：判断库存是否足够,不够不能继续信息商品信息。

3.4 视图

视图 view 是一种虚拟存在的表，对于使用视图的用户来说基本是透明的，视图并不在数据库中实际存在，行和列数据来自定义视图的查询使用的表，并且是在使用视图时动态生成。

视图相对于真实存在的数据表的优点：

- 简单
  使用视图的时候，不再关心对应的表结构、关联条件。
- 安全
  使用视图可以管理到数据显示的列，这样基础数据就更安全。
- 数据独立
  视图是独立于数据真实表的，当数据真实表新增字段时，视图不会产生变化，如果需要则更新视图就可以。

创建视图

单表的视图创建如下：

```
mysql> create view v_stu as
    -> select * from student;
Query OK, 0 rows affected (0.05 sec)
```

视图的实际使用中，多是多表之间的查询，如下：

```
mysql> desc student;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| stu_id    | int(11)     | NO   | PRI | NULL    |       |
| stu_name  | varchar(25) | YES  | MUL | NULL    |       |
| stu_age   | int(2)      | YES  |     | NULL    |       |
| stu_score | int(2)      | YES  |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
4 rows in set (0.00 sec)

mysql> create table score(
    -> sco_id int not null,
    -> sco_rel int,
    -> primary key(sco_id),
    -> stu_id int,
    -> foreign key(stu_id) references student(stu_id))charset utf8;
Query OK, 0 rows affected (0.33 sec)

mysql> create view v_sco as
    -> select s.*,c.sco_rel from student as s
    -> left join score as c
    -> on s.stu_id = c.stu_id;
Query OK, 0 rows affected (0.06 sec)
```

视图 v_sco 的目的是为了获取所有学生的考试成绩，于是将 student 表和 score 表左关联，就可以得到查询的结果。

查看视图

查看视图结构：desc 视图名;

```
mysql> desc v_sco;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| stu_id    | int(11)     | NO   |     | NULL    |       |
| stu_name  | varchar(25) | YES  |     | NULL    |       |
| stu_age   | int(2)      | YES  |     | NULL    |       |
| stu_score | int(2)      | YES  |     | NULL    |       |
| sco_rel   | int(11)     | YES  |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
5 rows in set (0.07 sec)
```

查看创建视图的语句 show create view 视图名；

```
mysql> show create view v_sco;
+-------+----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+----------------------+----------------------+
| View  | Create View                                                                                                                                                                                                                                                    | character_set_client | collation_connection |
+-------+----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+----------------------+----------------------+
| v_sco | CREATE ALGORITHM=UNDEFINED DEFINER=`root`@`localhost` SQL SECURITY DEFINER VIEW `v_sco` AS select `s`.`stu_id` AS `stu_id`,`s`.`stu_name` AS `stu_name`,`s`.`stu_age` AS `stu_age`,`s`.`stu_score` AS `stu_score`,`c`.`sco_rel` AS `sco_rel` from (`student` `s` left join `score` `c` on((`s`.`stu_id` = `c`.`stu_id`))) | gbk                  | gbk_chinese_ci       |
+-------+----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+----------------------+----------------------+
1 row in set (0.00 sec)
```

使用视图

视图的使用，其实就是执行 select 语句。

```
mysql> select * from v_sco;
+--------+----------+---------+-----------+---------+
| stu_id | stu_name | stu_age | stu_score | sco_rel |
+--------+----------+---------+-----------+---------+
|      1 | 张三     |      20 |        60 | NULL    |
|      2 | 张三2    |      30 |        70 | NULL    |
|      3 | 李四     |      24 |        70 | NULL    |
|      4 | 李四2    |      19 |        70 | NULL    |
+--------+----------+---------+-----------+---------+
4 rows in set (0.05 sec)

mysql> select stu_name ,stu_score  from v_sco;
+----------+-----------+
| stu_name | stu_score |
+----------+-----------+
| 张三     |        60 |
| 张三2    |        70 |
| 李四     |        70 |
| 李四2    |        70 |
+----------+-----------+
4 rows in set (0.00 sec)
```

修改视图

视图本身不可修改，但是视图的来源是可以修改的.

修改视图：修改视图本身的来源语句(select 语句)

alter view 视图名字 as 新的 select 语句;

```
mysql> alter view v_sco as
    -> select s.stu_name,c.sco_rel from student s
    -> left join score c
    -> on s.stu_id = c.stu_id;
Query OK, 0 rows affected (0.05 sec)

mysql> desc v_sco;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| stu_name | varchar(25) | YES  |     | NULL    |       |
| sco_rel  | int(11)     | YES  |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

删除视图

视图的删除语句 drop view 视图名;

视图数据操作 insert,update,delete

数据新增就是直接对视图进行数据新增.

1) 多表的视图不能新增数据

```
mysql> insert into v_sco values('李四',20);
ERROR 1471 (HY000): The target table v_sco of the INSERT is not insertable-into
```

2) 可以向单表插入数据，但是视图包含的表的字段不可为空。

```
mysql> select * from v_stu;
+--------+----------+---------+-----------+
| stu_id | stu_name | stu_age | stu_score |
+--------+----------+---------+-----------+
|      1 | 张三     |      20 |        60 |
|      2 | 张三2    |      30 |        70 |
|      3 | 李四     |      24 |        70 |
|      4 | 李四2    |      19 |        70 |
+--------+----------+---------+-----------+
4 rows in set (0.00 sec)

mysql> insert into v_stu values(6,'视图1',23,80);
Query OK, 1 row affected (0.06 sec)

mysql> select * from v_stu;
+--------+----------+---------+-----------+
| stu_id | stu_name | stu_age | stu_score |
+--------+----------+---------+-----------+
|      1 | 张三     |      20 |        60 |
|      2 | 张三2    |      30 |        70 |
|      3 | 李四     |      24 |        70 |
|      4 | 李四2    |      19 |        70 |
|      6 | 视图1    |      23 |        80 |
+--------+----------+---------+-----------+
5 rows in set (0.00 sec)
```

同时视图包含的表的数据也会新增。

```
mysql> select * from student;
+--------+----------+---------+-----------+
| stu_id | stu_name | stu_age | stu_score |
+--------+----------+---------+-----------+
|      1 | 张三     |      20 |        60 |
|      2 | 张三2    |      30 |        70 |
|      3 | 李四     |      24 |        70 |
|      4 | 李四2    |      19 |        70 |
|      6 | 视图1    |      23 |        80 |
+--------+----------+---------+-----------+
5 rows in set (0.00 sec)
```

3) 多表视图的数据是不允许删除

```
mysql> select * from v_sco;
+----------+---------+
| stu_name | sco_rel |
+----------+---------+
| 张三     |    NULL |
| 张三2    |    NULL |
| 李四     |    NULL |
| 李四2    |    NULL |
| 视图1    |    NULL |
+----------+---------+
5 rows in set (0.00 sec)

mysql> delete from v_sco where stu_name='视图1';
ERROR 1288 (HY000): The target table v_sco of the DELETE is not updatable
```

4) 单表视图的数据可以删除，同时视图包含的表的数据也会被删除。

```
mysql> select * from v_stu;
+--------+----------+---------+-----------+
| stu_id | stu_name | stu_age | stu_score |
+--------+----------+---------+-----------+
|      1 | 张三     |      20 |        60 |
|      2 | 张三2    |      30 |        70 |
|      3 | 李四     |      24 |        70 |
|      4 | 李四2    |      19 |        70 |
|      6 | 视图1    |      23 |        80 |
+--------+----------+---------+-----------+
5 rows in set (0.00 sec)

mysql> delete from v_stu where stu_id=6;
Query OK, 1 row affected (0.09 sec)

mysql> select * from v_stu;
+--------+----------+---------+-----------+
| stu_id | stu_name | stu_age | stu_score |
+--------+----------+---------+-----------+
|      1 | 张三     |      20 |        60 |
|      2 | 张三2    |      30 |        70 |
|      3 | 李四     |      24 |        70 |
|      4 | 李四2    |      19 |        70 |
+--------+----------+---------+-----------+
4 rows in set (0.00 sec)
```

5) 更新数据，单表视图可以更新

3.5 事务和锁

事务: transaction, 一系列要发生的连续的操作，一个事务是一个连续的一组数据库操作，就好像它是一个单一的工作单元进行。换句话说，永远不会是完整的事务，除非该组内的每个单独的操作是成功的。如果在事务的任何操作失败，则整个事务将失败。

mysql 的事务是跟存储引擎相关的，不同的引擎对事务的支持是不一样的：
1.MyISAM/memory：表级锁定，用于只读程序提高性能
2.InnoDB：行级锁定，支持 ACID 事务、行级锁、并发
3.Berkeley DB：页级锁定
在默认情况下，表锁和行锁是自动获得的，不需要额外的命令，但是实际的开发过程中，我们需要明确的进行锁表和行事务控制，以保证事务的完整性。

本文档我们主要说 innoDB 的事务，也就是 mysql5.7 默认的存储引擎，锁机制以后写优化的时候再说。

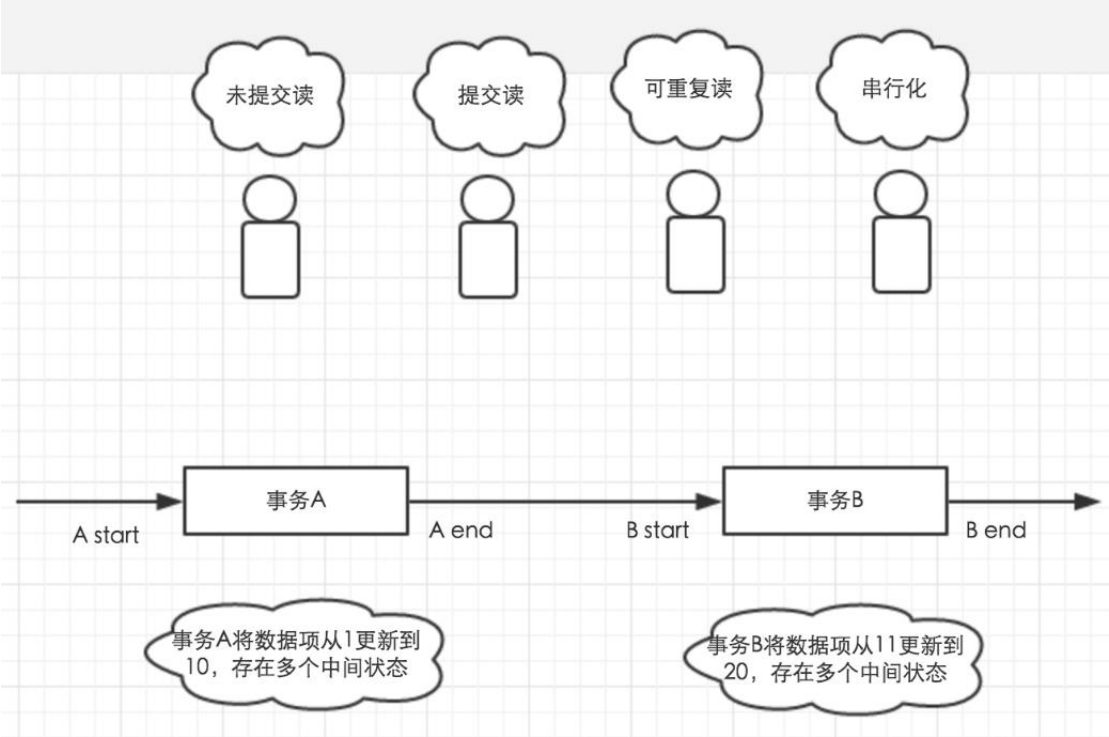
****ACID，也就是事务的四个特性，任何支持事务的数据库都必须遵守的原则：
原子性(Atomicity): 确保工作单元内的所有操作都成功完成，否则事务将被中止在故障点，和以前的操作将回滚到以前的状态。事务是应用中不可再分的最小逻辑执行体。

一致性(Consistency): 确保数据库正确地改变状态后，成功提交的事务。

隔离性(Isolation): 使事务操作彼此独立的和透明的。

持久性(Durability): 确保提交的事务的结果或效果的系统出现故障的情况下仍然存在。

关于事务，举个栗子：事务 A 和事务 B 先后开启，并对数据 1 进行多次更新。四个小人在不同的时刻开启事务，可能看到数据 1 的哪些值呢？



解读：第一个小人，可能读到 1-20 之间的任何一个。因为未提交读的隔离级别下，其他事务对数据的修改也是对当前事务可见的。第二个小人可能读到 1，10 和 20，他只能读到其他事务已经提交了的数据。第三个小人读到的数据取决于自身事务开启的时间点。在事务开启时，读到的是多少，那么在事务提交之前读到的值就是多少。第四个小人，只有在 A end 到 B start 之间开启，才有可能读到数据，而在事务 A 和事务 B 执行的期间是读不到数据的。因为第四小人读数据是需要加锁的，事务 A 和 B 执行期间，会占用数据的写锁，导致第四个小人等待锁。

事务安全: 一种保护连续操作同时满足(实现)的一种机制

事务安全的意义: 保证数据操作的完整性

本部分的内容，依旧使用 bill 表来测试，表结构和现有数据如下：

```
mysql> desc bill;
+------------+-------------+------+-----+---------+----------------+
| Field      | Type        | Null | Key | Default | Extra          |
+------------+-------------+------+-----+---------+----------------+
| id         | int(11)     | NO   | PRI | NULL    | auto_increment |
| item_name  | varchar(20) | NO   |     | NULL    |                |
| item_price | double      | YES  |     | NULL    |                |
| item_num   | int(11)     | YES  |     | NULL    |                |
+------------+-------------+------+-----+---------+----------------+
4 rows in set (0.13 sec)

mysql> select * from bill;
+----+-----------+------------+----------+
| id | item_name | item_price | item_num |
+----+-----------+------------+----------+
|  1 | 早餐面条  |          5 |        1 |
|  2 | 早餐牛奶  |          5 |        1 |
|  3 | 午餐套餐  |         15 |        5 |
|  4 | 晚餐套餐  |         12 |        1 |
|  7 | 下午茶    |         10 |        2 |
|  8 | 夜宵      |         20 |        5 |
+----+-----------+------------+----------+
6 rows in set (0.06 sec)
```

锁(un)lock table

这是一种很常用的做法，锁表，lock table 可以锁定当前线程的表，如果当前表被其他线程锁定，当前线程会等待，一直到可以获得所有锁定为止。对应的 unlock table 就可以释放锁。

我们锁定 bill 表只读。

```
mysql> lock table bill read;
Query OK, 0 rows affected (0.00 sec)
```

当前 session， 也就是当前当前的 cmd 窗口会话：

```
mysql> select * from bill;
+----+-----------+------------+----------+
| id | item_name | item_price | item_num |
+----+-----------+------------+----------+
|  1 | 早餐面条  |          5 |        1 |
|  2 | 早餐牛奶  |          5 |        1 |
|  3 | 午餐套餐  |         15 |        5 |
|  4 | 晚餐套餐  |         12 |        1 |
|  7 | 下午茶    |         10 |        2 |
|  8 | 夜宵      |         20 |        5 |
+----+-----------+------------+----------+
6 rows in set (0.00 sec)
```

再打开一个 cmd 窗口，其他会话窗口也可以执行 read：

```
mysql> use c3gen;
Database changed
mysql> select * from bill;
+----+-----------+------------+----------+
| id | item_name | item_price | item_num |
+----+-----------+------------+----------+
|  1 | 早餐面条  |          5 |        1 |
|  2 | 早餐牛奶  |          5 |        1 |
|  3 | 午餐套餐  |         15 |        5 |
|  4 | 晚餐套餐  |         12 |        1 |
|  7 | 下午茶    |         10 |        2 |
|  8 | 夜宵      |         20 |        5 |
+----+-----------+------------+----------+
6 rows in set (0.00 sec)
```

这个时候，我们执行更新操作：

```
mysql> update bill set item_num = 10 where id=4;
ERROR 1099 (HY000): Table 'bill' was locked with a READ lock and can't be updated
```

会发现，更新是失败的，因为 bill 表被锁定，且为只读。

释放锁，再更新：

```
mysql> unlock table;
Query OK, 0 rows affected (0.00 sec)

mysql> update bill set item_num= 10 where id=4;
Query OK, 1 row affected (0.06 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

事务操作

事务操作分为两种: 自动事务(默认的), 手动事务

手动事务: 操作流程

1. 开启事务: 告诉系统以下所有的操作(写)不要直接写入到数据表, 先存放到事务日志
start transaction 或者 begin;

```
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)
```

2. 进行事务操作: 一系列操作
    a) 修改午餐的 num

```
mysql> update bill set item_num= 10 where id=3;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from bill;
+----+-----------+------------+-----------+
| id | item_name | item_price | item_num  |
+----+-----------+------------+-----------+
|  1 | 早餐面条  |          5 |         1 |
|  2 | 早餐牛奶  |          5 |         1 |
|  3 | 午餐套餐  |         15 |        10 |
|  4 | 晚餐套餐  |         12 |        10 |
|  7 | 下午茶    |         10 |         2 |
|  8 | 夜宵      |         20 |         5 |
+----+-----------+------------+-----------+
6 rows in set (0.00 sec)
```

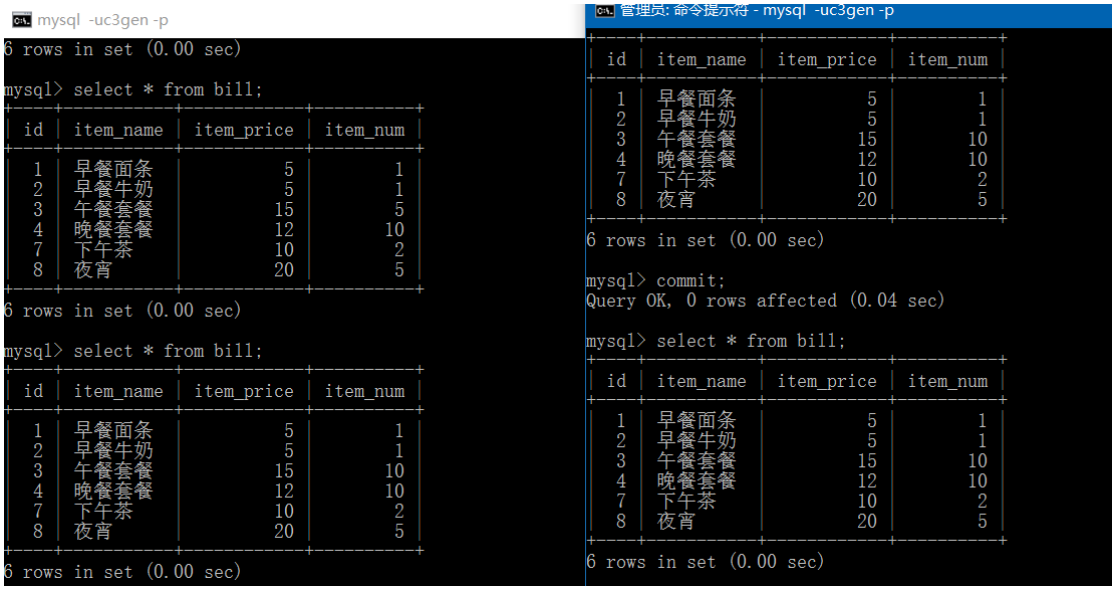
b) 午餐数据增加，但是我并没有 commit 提交，所以在另外一个窗口执行 select 是看不到我更新的数据的：

```
mysql -uc3gen -p
mysql> use c3gen;
Database changed
mysql> select * from bill;
+----+-----------+------------+-----------+
| id | item_name | item_price | item_num  |
+----+-----------+------------+-----------+
|  1 | 早餐面条  |          5 |         1 |
|  2 | 早餐牛奶  |          5 |         1 |
|  3 | 午餐套餐  |         15 |         5 |
|  4 | 晚餐套餐  |         12 |         1 |
|  7 | 下午茶    |         10 |         2 |
|  8 | 夜宵      |         20 |         5 |
+----+-----------+------------+-----------+
6 rows in set (0.00 sec)

mysql> select * from bill;
+----+-----------+------------+-----------+
| id | item_name | item_price | item_num  |
+----+-----------+------------+-----------+
|  1 | 早餐面条  |          5 |         1 |
|  2 | 早餐牛奶  |          5 |         1 |
|  3 | 午餐套餐  |         15 |         5 |
|  4 | 晚餐套餐  |         12 |        10 |
|  7 | 下午茶    |         10 |         2 |
|  8 | 夜宵      |         20 |         5 |
+----+-----------+------------+-----------+
6 rows in set (0.00 sec)
```

```
管理员: 命令提示符 - mysql -uc3gen -p
 for the right syntax to use near 'bill' at line 1
mysql> unlock table;
Query OK, 0 rows affected (0.00 sec)

mysql> update bill set item_num= 10 where id=4;
Query OK, 1 row affected (0.06 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> update bill set item_num= 10 where id=3;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from bill;
+----+-----------+------------+-----------+
| id | item_name | item_price | item_num  |
+----+-----------+------------+-----------+
|  1 | 早餐面条  |          5 |         1 |
|  2 | 早餐牛奶  |          5 |         1 |
|  3 | 午餐套餐  |         15 |        10 |
|  4 | 晚餐套餐  |         12 |        10 |
|  7 | 下午茶    |         10 |         2 |
|  8 | 夜宵      |         20 |         5 |
+----+-----------+------------+-----------+
6 rows in set (0.00 sec)
```

3. 关闭事务: 选择性的将日志文件中操作的结果保存到数据表(同步)或者说直接清空事务日志(原来操作全部清空)

a) 提交事务: 同步数据表(操作成功): commit;

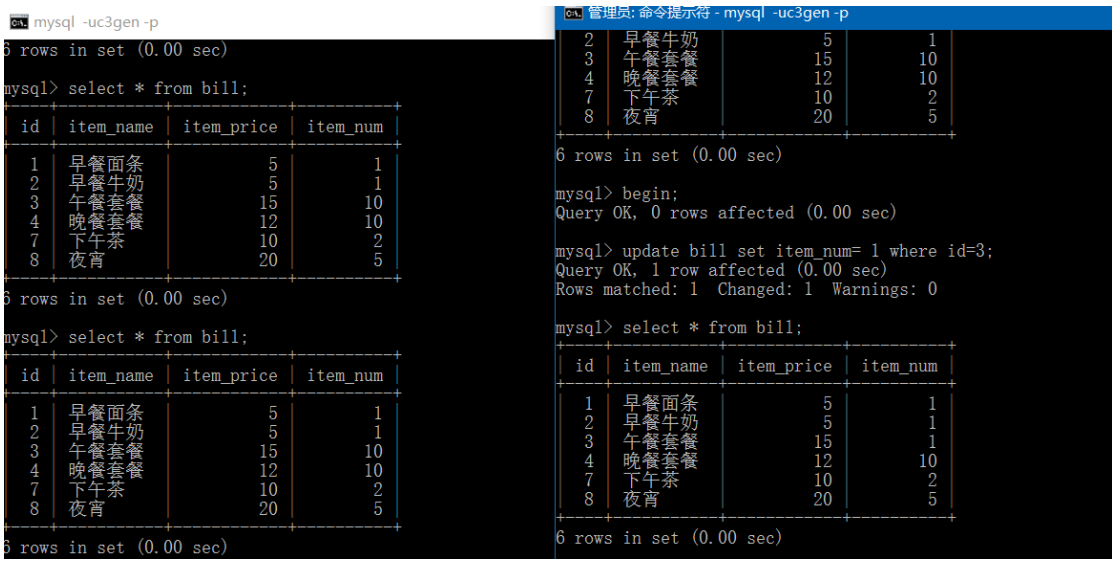


```
mysql -uc3gen -p
6 rows in set (0.00 sec)

mysql> select * from bill;
+----+-----------+------------+----------+
| id | item_name | item_price | item_num |
+----+-----------+------------+----------+
|  1 | 早餐面条  |          5 |        1 |
|  2 | 早餐牛奶  |          5 |        1 |
|  3 | 午餐套餐  |         15 |        5 |
|  4 | 晚餐套餐  |         12 |       10 |
|  7 | 下午茶    |         10 |        2 |
|  8 | 夜宵      |         20 |        5 |
+----+-----------+------------+----------+
6 rows in set (0.00 sec)

mysql> select * from bill;
+----+-----------+------------+----------+
| id | item_name | item_price | item_num |
+----+-----------+------------+----------+
|  1 | 早餐面条  |          5 |        1 |
|  2 | 早餐牛奶  |          5 |        1 |
|  3 | 午餐套餐  |         15 |       10 |
|  4 | 晚餐套餐  |         12 |       10 |
|  7 | 下午茶    |         10 |        2 |
|  8 | 夜宵      |         20 |        5 |
+----+-----------+------------+----------+
6 rows in set (0.00 sec)
```

```
管理员: 命令提示符 - mysql -uc3gen -p
+----+-----------+------------+----------+
| id | item_name | item_price | item_num |
+----+-----------+------------+----------+
|  1 | 早餐面条  |          5 |        1 |
|  2 | 早餐牛奶  |          5 |        1 |
|  3 | 午餐套餐  |         15 |       10 |
|  4 | 晚餐套餐  |         12 |       10 |
|  7 | 下午茶    |         10 |        2 |
|  8 | 夜宵      |         20 |        5 |
+----+-----------+------------+----------+
6 rows in set (0.00 sec)

mysql> commit;
Query OK, 0 rows affected (0.04 sec)

mysql> select * from bill;
+----+-----------+------------+----------+
| id | item_name | item_price | item_num |
+----+-----------+------------+----------+
|  1 | 早餐面条  |          5 |        1 |
|  2 | 早餐牛奶  |          5 |        1 |
|  3 | 午餐套餐  |         15 |       10 |
|  4 | 晚餐套餐  |         12 |       10 |
|  7 | 下午茶    |         10 |        2 |
|  8 | 夜宵      |         20 |        5 |
+----+-----------+------------+----------+
6 rows in set (0.00 sec)
```

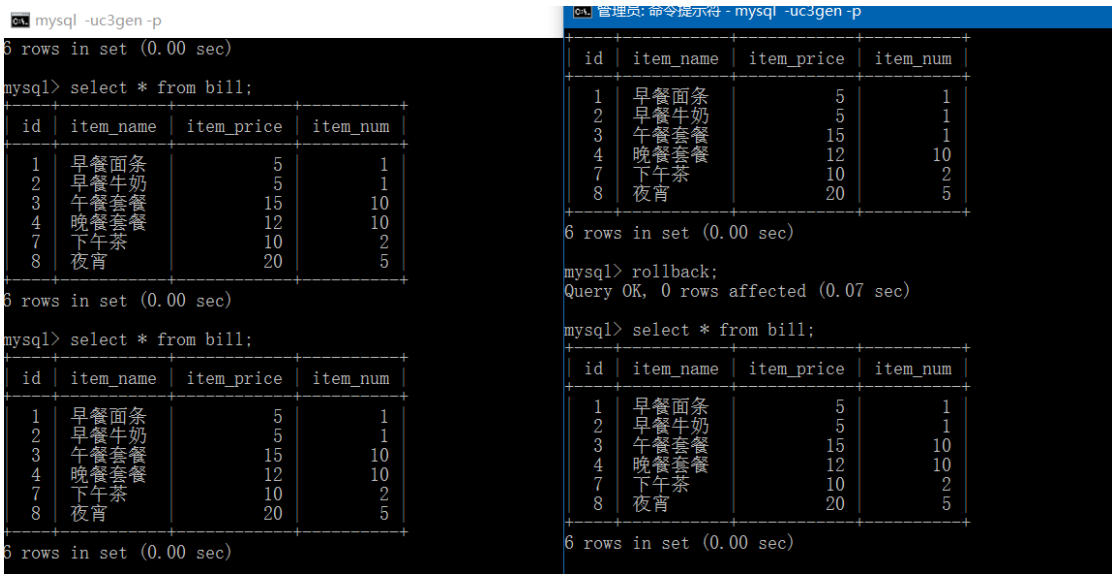
b) 回滚事务：直接清空日志表(操作失败)：rollback;

再次开启事务，进行更新操作，更新了午餐的 num: 然后进行事务回滚：



```
mysql -uc3gen -p
6 rows in set (0.00 sec)

mysql> select * from bill;
+----+-----------+------------+----------+
| id | item_name | item_price | item_num |
+----+-----------+------------+----------+
|  1 | 早餐面条  |          5 |        1 |
|  2 | 早餐牛奶  |          5 |        1 |
|  3 | 午餐套餐  |         15 |       10 |
|  4 | 晚餐套餐  |         12 |       10 |
|  7 | 下午茶    |         10 |        2 |
|  8 | 夜宵      |         20 |        5 |
+----+-----------+------------+----------+
6 rows in set (0.00 sec)

mysql> select * from bill;
+----+-----------+------------+----------+
| id | item_name | item_price | item_num |
+----+-----------+------------+----------+
|  1 | 早餐面条  |          5 |        1 |
|  2 | 早餐牛奶  |          5 |        1 |
|  3 | 午餐套餐  |         15 |       10 |
|  4 | 晚餐套餐  |         12 |       10 |
|  7 | 下午茶    |         10 |        2 |
|  8 | 夜宵      |         20 |        5 |
+----+-----------+------------+----------+
6 rows in set (0.00 sec)
```

```
管理员: 命令提示符 - mysql -uc3gen -p
|  2 | 早餐牛奶  |          5 |        1 |
|  3 | 午餐套餐  |         15 |       10 |
|  4 | 晚餐套餐  |         12 |       10 |
|  7 | 下午茶    |         10 |        2 |
|  8 | 夜宵      |         20 |        5 |
+----+-----------+------------+----------+
6 rows in set (0.00 sec)

mysql> begin;
Query OK, 0 rows affected (0.00 sec)

mysql> update bill set item_num= 1 where id=3;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from bill;
+----+-----------+------------+----------+
| id | item_name | item_price | item_num |
+----+-----------+------------+----------+
|  1 | 早餐面条  |          5 |        1 |
|  2 | 早餐牛奶  |          5 |        1 |
|  3 | 午餐套餐  |         15 |        1 |
|  4 | 晚餐套餐  |         12 |       10 |
|  7 | 下午茶    |         10 |        2 |
|  8 | 夜宵      |         20 |        5 |
+----+-----------+------------+----------+
6 rows in set (0.00 sec)
```

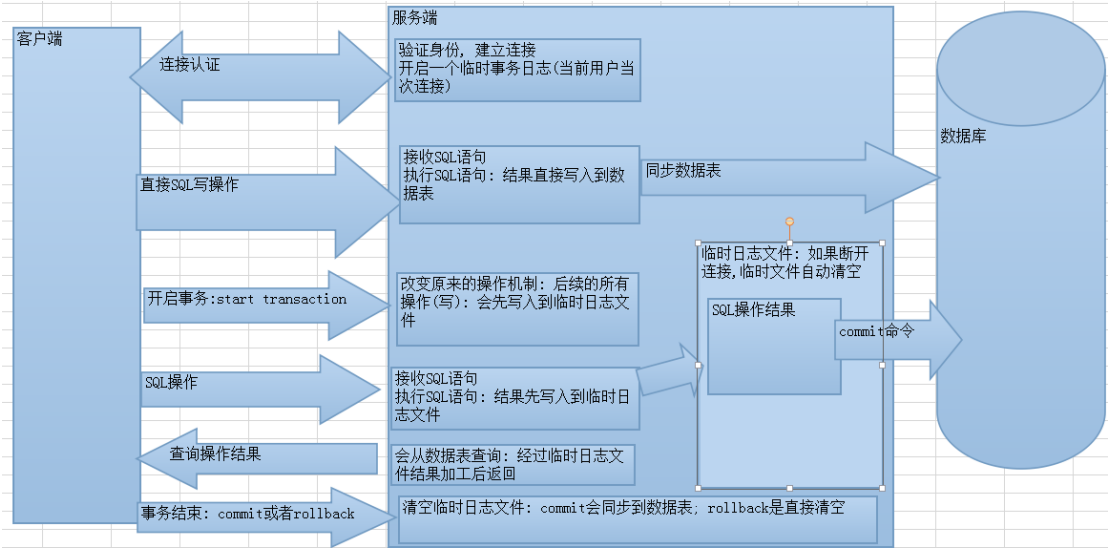


```
mysql -uc3gen -p
6 rows in set (0.00 sec)

mysql> select * from bill;
+----+-----------+------------+----------+
| id | item_name | item_price | item_num |
+----+-----------+------------+----------+
|  1 | 早餐面条  |          5 |        1 |
|  2 | 早餐牛奶  |          5 |        1 |
|  3 | 午餐套餐  |         15 |       10 |
|  4 | 晚餐套餐  |         12 |       10 |
|  7 | 下午茶    |         10 |        2 |
|  8 | 夜宵      |         20 |        5 |
+----+-----------+------------+----------+
6 rows in set (0.00 sec)

mysql> select * from bill;
+----+-----------+------------+----------+
| id | item_name | item_price | item_num |
+----+-----------+------------+----------+
|  1 | 早餐面条  |          5 |        1 |
|  2 | 早餐牛奶  |          5 |        1 |
|  3 | 午餐套餐  |         15 |       10 |
|  4 | 晚餐套餐  |         12 |       10 |
|  7 | 下午茶    |         10 |        2 |
|  8 | 夜宵      |         20 |        5 |
+----+-----------+------------+----------+
6 rows in set (0.00 sec)
```

```
管理员: 命令提示符 - mysql -uc3gen -p
+----+-----------+------------+----------+
| id | item_name | item_price | item_num |
+----+-----------+------------+----------+
|  1 | 早餐面条  |          5 |        1 |
|  2 | 早餐牛奶  |          5 |        1 |
|  3 | 午餐套餐  |         15 |        1 |
|  4 | 晚餐套餐  |         12 |       10 |
|  7 | 下午茶    |         10 |        2 |
|  8 | 夜宵      |         20 |        5 |
+----+-----------+------------+----------+
6 rows in set (0.00 sec)

mysql> rollback;
Query OK, 0 rows affected (0.07 sec)

mysql> select * from bill;
+----+-----------+------------+----------+
| id | item_name | item_price | item_num |
+----+-----------+------------+----------+
|  1 | 早餐面条  |          5 |        1 |
|  2 | 早餐牛奶  |          5 |        1 |
|  3 | 午餐套餐  |         15 |       10 |
|  4 | 晚餐套餐  |         12 |       10 |
|  7 | 下午茶    |         10 |        2 |
|  8 | 夜宵      |         20 |        5 |
+----+-----------+------------+----------+
6 rows in set (0.00 sec)
```

事务原理

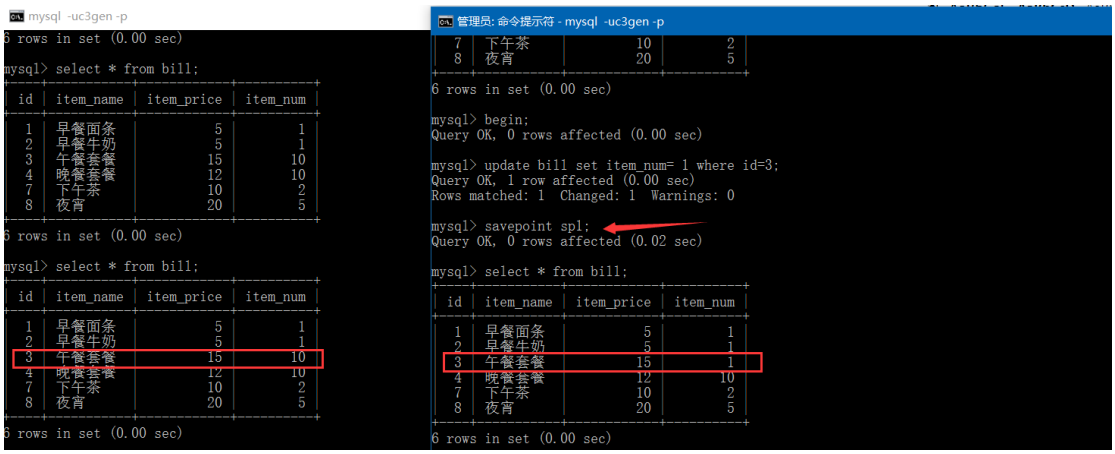
事务操作原理：事务开启之后，所有的操作都会临时保存到事务日志，事务日志只有在得到 commit 命令才会同步到数据表,其他任何情况都会清空(rollback，断电，断开连接)



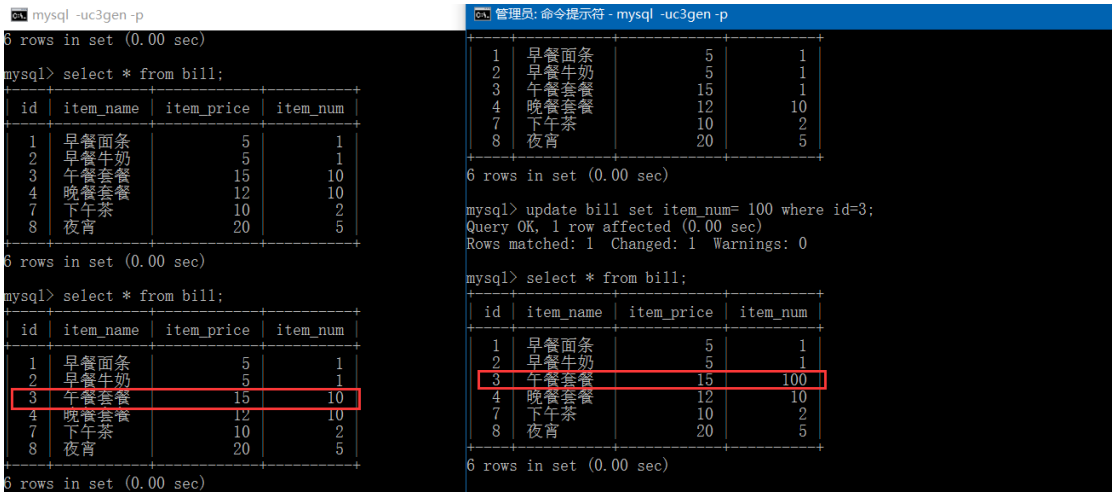
回滚点

回滚点：在某个成功的操作完成之后，后续的操作有可能成功有可能失败，但是不管成功还是失败,前面操作都已经成功：可以在当前成功的位置，设置一个点：可以供后续失败操作返回到该位置，而不是返回所有操作，这个点称之为回滚点.

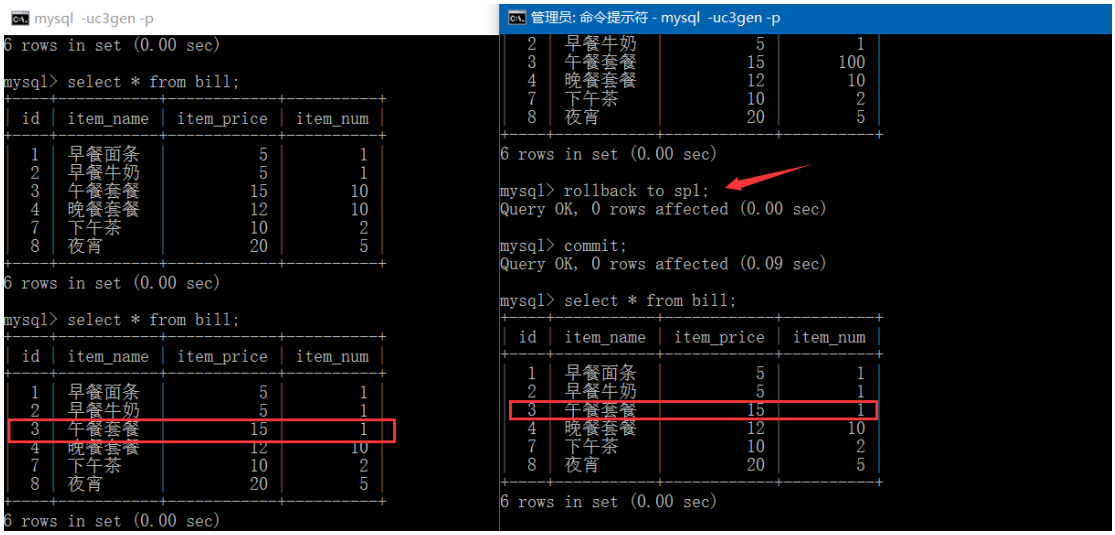
设置回滚点语法: savepoint 回滚点名字;



然后我继续更新操作：



回到回滚点语法: rollback to 回滚点名字 ; 然后提交事务:

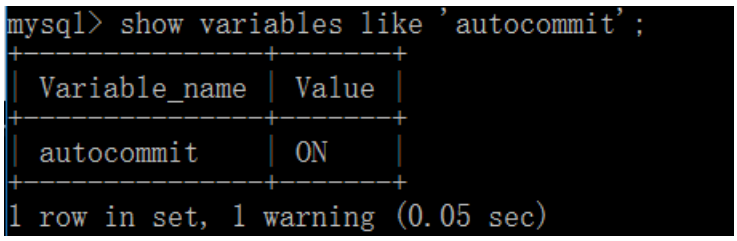


自动事务处理

在 mysql 中: 默认的都是自动事务处理, 用户操作完会立即同步到数据表中.

自动事务: 系统通过 autocommit 变量控制

show variables like 'autocommit';



关闭自动提交: set autocommit = off/0;

```
mysql> set autocommit=0;
Query OK, 0 rows affected (0.00 sec)

mysql> show variables like 'autocommit';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| autocommit    | OFF   |
+---------------+-------+
1 row in set, 1 warning (0.00 sec)
```

再次直接写操作

mysql -uc3gen -p

```
6 rows in set (0.00 sec)

mysql> select * from bill;
+----+-----------+------------+----------+
| id | item_name | item_price | item_num |
+----+-----------+------------+----------+
|  1 | 早餐面条  |          5 |        1 |
|  2 | 早餐牛奶  |          5 |        1 |
|  3 | 午餐套餐  |         15 |        1 |
|  4 | 晚餐套餐  |         12 |       10 |
|  7 | 下午茶    |         10 |        2 |
|  8 | 夜宵      |         20 |        5 |
+----+-----------+------------+----------+
6 rows in set (0.00 sec)

mysql> select * from bill;
+----+-----------+------------+----------+
| id | item_name | item_price | item_num |
+----+-----------+------------+----------+
|  1 | 早餐面条  |          5 |        1 |
|  2 | 早餐牛奶  |          5 |        1 |
|  3 | 午餐套餐  |         15 |        1 |
|  4 | 晚餐套餐  |         12 |       10 |
|  7 | 下午茶    |         10 |        2 |
|  8 | 夜宵      |         20 |        5 |
+----+-----------+------------+----------+
6 rows in set (0.00 sec)
```

管理员: 命令提示符 - mysql -uc3gen -p

```
Query OK, 0 rows affected (0.00 sec)

mysql> show variables like 'autocommit';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| autocommit    | OFF   |
+---------------+-------+
1 row in set, 1 warning (0.00 sec)

mysql> update bill set item_num= 100 where id=3;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from bill;
+----+-----------+------------+----------+
| id | item_name | item_price | item_num |
+----+-----------+------------+----------+
|  1 | 早餐面条  |          5 |        1 |
|  2 | 早餐牛奶  |          5 |        1 |
|  3 | 午餐套餐  |         15 |      100 |
|  4 | 晚餐套餐  |         12 |       10 |
|  7 | 下午茶    |         10 |        2 |
|  8 | 夜宵      |         20 |        5 |
+----+-----------+------------+----------+
6 rows in set (0.00 sec)
```

自动关闭之后,需要手动来选择处理: commit 提交, rollback 回滚

mysql -uc3gen -p

```
6 rows in set (0.00 sec)

mysql> select * from bill;
+----+-----------+------------+----------+
| id | item_name | item_price | item_num |
+----+-----------+------------+----------+
|  1 | 早餐面条  |          5 |        1 |
|  2 | 早餐牛奶  |          5 |        1 |
|  3 | 午餐套餐  |         15 |        1 |
|  4 | 晚餐套餐  |         12 |       10 |
|  7 | 下午茶    |         10 |        2 |
|  8 | 夜宵      |         20 |        5 |
+----+-----------+------------+----------+
6 rows in set (0.00 sec)

mysql> select * from bill;
+----+-----------+------------+----------+
| id | item_name | item_price | item_num |
+----+-----------+------------+----------+
|  1 | 早餐面条  |          5 |        1 |
|  2 | 早餐牛奶  |          5 |        1 |
|  3 | 午餐套餐  |         15 |      100 |
|  4 | 晚餐套餐  |         12 |       10 |
|  7 | 下午茶    |         10 |        2 |
|  8 | 夜宵      |         20 |        5 |
+----+-----------+------------+----------+
6 rows in set (0.00 sec)
```

管理员: 命令提示符 - mysql -uc3gen -p

```
| id | item_name | item_price | item_num |
+----+-----------+------------+----------+
|  1 | 早餐面条  |          5 |        1 |
|  2 | 早餐牛奶  |          5 |        1 |
|  3 | 午餐套餐  |         15 |      100 |
|  4 | 晚餐套餐  |         12 |       10 |
|  7 | 下午茶    |         10 |        2 |
|  8 | 夜宵      |         20 |        5 |
+----+-----------+------------+----------+
6 rows in set (0.00 sec)

mysql> commit;
Query OK, 0 rows affected (0.03 sec)

mysql> select * from bill;
+----+-----------+------------+----------+
| id | item_name | item_price | item_num |
+----+-----------+------------+----------+
|  1 | 早餐面条  |          5 |        1 |
|  2 | 早餐牛奶  |          5 |        1 |
|  3 | 午餐套餐  |         15 |      100 |
|  4 | 晚餐套餐  |         12 |       10 |
|  7 | 下午茶    |         10 |        2 |
|  8 | 夜宵      |         20 |        5 |
+----+-----------+------------+----------+
6 rows in set (0.00 sec)
```

注意: 通常都会使用自动事务

3.6 存储过程

存储过程简称过程,procedure, 是一种用来处理数据的方式.
存储过程是一种没有返回值的函数.

创建过程

create procedure 过程名字([参数列表])
begin
    -- 过程体
end
创建如下： 查询所有 bill 记录

```
mysql> create procedure test_pro1()
    -> select * from bill;
Query OK, 0 rows affected (0.00 sec)
```

查看过程

函数的查看方式和函数是相似的: 关键字换成 procedure

查看所有过程: show procedure status [like 'pattern'];

```
mysql> show procedure status like 'test_pro%'\G;
*************************** 1. row ***************************
                  Db: c3gen
                Name: test_pro1
                Type: PROCEDURE
             Definer: c3gen@%
            Modified: 2016-12-08 16:33:38
             Created: 2016-12-08 16:33:38
       Security_type: DEFINER
             Comment:
character_set_client: gbk
collation_connection: gbk_chinese_ci
  Database Collation: utf8_general_ci
1 row in set (0.00 sec)
```

查看过程创建语句也和函数相似: show create procedure 过程名;

```
mysql> show create procedure test_pro1;
+-----------+-----------------------------------------------------------------+----------------------+----------------------+--------------------+
| Procedure | sql_mode                                                        | Create Procedure
                        | character_set_client | collation_connection | Database Collation |
+-----------+-----------------------------------------------------------------+----------------------+----------------------+--------------------+
| test_pro1 | STRICT_TRANS_TABLES,NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION | CREATE DEFINER=`c3gen`@`%` PROCEDURE `tes
t_pro1`()
select * from bill | gbk                  | gbk_chinese_ci       | utf8_general_ci    |
+-----------+-----------------------------------------------------------------+----------------------+----------------------+--------------------+
1 row in set (0.00 sec)
```

调用存储过程

存储过程没有返回值: select 是不能访问的. 所有 要记住 在 mysql 中，有返回值的才能 select:

```
mysql> select test_pro1();
ERROR 1305 (42000): FUNCTION c3gen.test_pro1 does not exist
```

过程有一个专门的调用关键字: call

```
mysql> call test_pro1;
+----+-----------+------------+-----------+
| id | item_name | item_price | item_num  |
+----+-----------+------------+-----------+
|  1 | 早餐面条  |          5 |         1 |
|  2 | 早餐牛奶  |          5 |         1 |
|  3 | 午餐套餐  |         15 |       100 |
|  4 | 晚餐套餐  |         12 |        10 |
|  7 | 下午茶    |         10 |         2 |
|  8 | 夜宵      |         20 |         5 |
+----+-----------+------------+-----------+
6 rows in set (0.00 sec)

Query OK, 0 rows affected (0.02 sec)
```

修改存储过程&删除存储过程

存储过程只能先删除,后新增，这和函数是一样的：

drop procedure 过程名;

```
mysql> drop procedure test_pro1;
Query OK, 0 rows affected (0.00 sec)

mysql> show procedure status like 'test_pro1';
Empty set (0.00 sec)
```

过程参数

函数的参数需要数据类型指定，存储过程比函数更严格.

存储过程还有自己的类型限定：三种类型

in：数据只是从外部传入给内部使用(值传递)：可以是数值也可以是变量

out：只允许过程内部使用(不用外部数据)，给外部使用的.(引用传递：外部的数据会被先清空才会进入到内部)：只能是变量

inout：外部可以在内部使用,内部修改也可以给外部使用：典型的引用传递：只能传变量

基本使用

create procedure 过程名(in 形参名字 数据类型, out 形参名字 数据类型, inout 形参名字 数据类型)

举个栗子如下： 查看入参 a,出参 b 以及 c 的值：

```
mysql> delimiter ##
mysql> create procedure pro1(in a int,out b int ,inout c int)
    -> begin
    -> select a,b,c;
    -> end
    -> ##
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter ;
```

调用: out 和 inout 类型的参数必须传入变量,而不能是数值

```
mysql> call pro1(1,2,3);
ERROR 1414 (42000): OUT or INOUT argument 2 for routine c3gen.pro1 is not a variable or NEW pseudo-variable in BEFORE trigger
```

正确调用：传入变量，预定义三个变量，然后传入：

```
mysql> set @a=1,@b=2,@c=3;
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> call pro1(@a,@b,@c);
+------+------+------+
| a    | b    | c    |
+------+------+------+
|    1 | NULL |    3 |
+------+------+------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)
```

会发现 出参是 null，为什么？？？？

存储过程对于变量的操作(返回)是滞后的：是在存储过程调用结束的时候,才会重新将内部修改的值赋值给外部传入的全局变量.下面的注释肯定无法编译的，写上面告诉你我在干嘛。

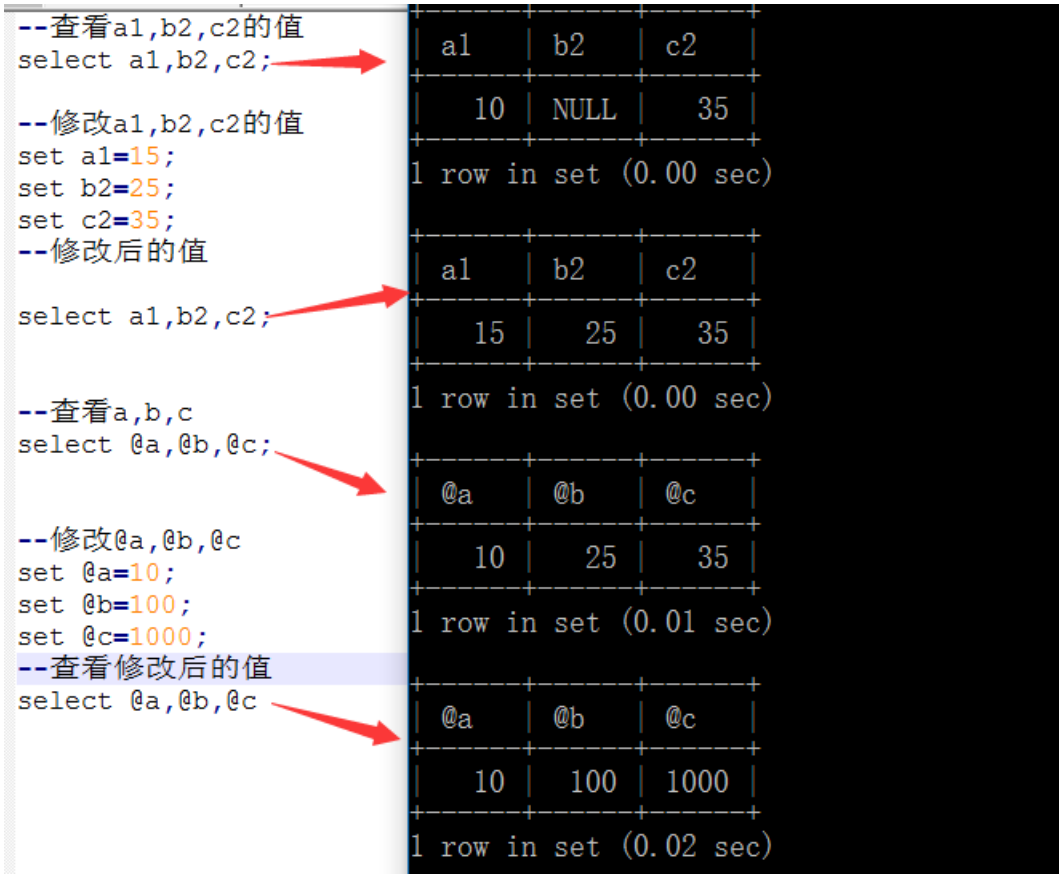
```
mysql> create procedure pro2(in a1,out b2,inout c2)
    -> begin
    -> -- 查看a2,b2,c2的值
    -> select a1,b2,c2;
    -> --b2的值肯定是null,因为这是个局部变量
    -> --修改局部变量的值
    -> set a1= 15;
    -> set b2= 25;
    -> set c2= 35;
    -> --再查看局部变量a1 b2 c2
    -> select a1,b2,c2;
    -> --查看定义好的全局变量a,b,c
    -> select @a,@b,@c;
    -> --修改全局变量a,b,c
    -> set @a=10;
    -> set @b=100;
    -> set @c=1000;
    -> --再查看全局变量
    -> select @a,@b,@c;
    -> end
    -> ##
```

--设置全局变量 a,b,c 的值

```
set @a=1,@b=2,@c=3;
```

```
delimiter ##
create procedure pro2(in a1 int,out b2 int ,inout c2 int)
begin
select a1,b2,c2;
set a1=15;
set b2=25;
set c2=35;
select a1,b2,c2;
select @a,@b,@c;
set @a=10;
set @b=100;
set @c=1000;
select @a,@b,@c;
select a1,b2,c2;
end
##
delimiter ;

show procedure status like 'pro2';
```

测试: 传入数据 1,2,3: 说明局部变量与全局变量无关



最后: 在存储过程调用结束之后，系统会将局部变量重复返回给全局变量(out 和 inout)
@a,@b,@c 的值最开始设置的是 1,2,3
可以看到 a,b,c 的值都已经被修改了：

```
mysql> select @a,@b,@c;
+------+------+------+
| @a   | @b   | @c   |
+------+------+------+
|   10 |   25 |   35 |
+------+------+------+
1 row in set (0.00 sec)
```

3.7 触发器

触发器: trigger，与表有关的数据库对象，在满足定义的条件的时候触发，并执行触发的定义的语句的集合。事先为某张表绑定好一段代码 ,当表中的某些内容发生改变的时候(增删改)系统会自动触发代码,执行。或者说 触发器是特殊的存储过程。

触发器: 事件类型，触发时间，触发对象

事件类型: 增删改，三种类型 insert,delete 和 update

触发时间: 前后: before 和 after

触发对象: 表中的每一条记录(行)

一张表中只能拥有一种触发时间的一种类型的触发器: 最多一张表能有 6 个触发器

创建触发器

在 mysql 高级结构中: 没有大括号, 都是用对应的字符符号代替

触发器基本语法
-- 临时修改语句结束符
delimiter 自定义符号: 后续代码中只有碰到自定义符号才算结束

create trigger 触发器名字 触发时间 事件类型 on 表名 for each row
begin -- 代表左大括号: 开始
-- 里面就是触发器的内容: 每行内容都必须使用语句结束符: 分号
end -- 代表右带括号: 结束
-- 语句结束符
自定义符号
-- 将临时修改修正过来
delimiter ;
举个栗子，很常见的场景，当一个商品被购买后，对应的库存表库存减少。
于是创建商品表和库存表：

```
mysql> create table stock(
    -> id int not null,
    -> sku_num int )charset utf8;
Query OK, 0 rows affected (0.21 sec)

mysql> insert into stock values(1,100);
Query OK, 1 row affected (0.08 sec)
```

```
mysql> create table goods(
    -> go_id int primary key auto_increment,
    -> sku_name varchar(20) not null,
    -> sku_num int)charset utf8;
Query OK, 0 rows affected (0.30 sec)
```

```
mysql> delimiter ##
mysql> create trigger af_sto after insert on goods for each row
    -> begin
    -> update stock set sku_num =sku_num-1 where id =1;
    -> end
    -> ##
Query OK, 0 rows affected (0.13 sec)

mysql> delimiter ;
mysql>
```

查看触发器

查看所有触发器或者模糊匹配

show triggers [like ‘pattern’];

```
mysql> show triggers \G;
*************************** 1. row ***************************
             Trigger: af_sto
               Event: INSERT
               Table: goods
           Statement: begin
update stock set sku_num =sku_num-1 where id =1;
end
              Timing: AFTER
             Created: 2016-12-08 11:35:04.63
            sql_mode: STRICT_TRANS_TABLES,NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION
             Definer: c3gen@%
character_set_client: gbk
collation_connection: gbk_chinese_ci
  Database Collation: utf8_general_ci
1 row in set (0.00 sec)

ERROR:
No query specified
```

可以查看触发器创建语句

show create trigger 触发器名字;

```
mysql> show create trigger af_sto\G;
*************************** 1. row ***************************
               Trigger: af_sto
              sql_mode: STRICT_TRANS_TABLES,NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION
SQL Original Statement: CREATE DEFINER=`c3gen`@`%` trigger af_sto after insert on goods for each row
begin
update stock set sku_num =sku_num-1 where id =1;
end
  character_set_client: gbk
  collation_connection: gbk_chinese_ci
    Database Collation: utf8_general_ci
               Created: 2016-12-08 11:35:04.63
1 row in set (0.00 sec)
```

所有的触发器都会保存一张表中: Information_schema.triggers

```
mysql> select * from information_schema.triggers\G
*************************** 1. row ***************************
           TRIGGER_CATALOG: def
            TRIGGER_SCHEMA: c3gen
              TRIGGER_NAME: af_sto
        EVENT_MANIPULATION: INSERT
      EVENT_OBJECT_CATALOG: def
       EVENT_OBJECT_SCHEMA: c3gen
        EVENT_OBJECT_TABLE: goods
              ACTION_ORDER: 1
          ACTION_CONDITION: NULL
          ACTION_STATEMENT: begin
update stock set sku_num =sku_num-1 where id =1;
end
        ACTION_ORIENTATION: ROW
             ACTION_TIMING: AFTER
ACTION_REFERENCE_OLD_TABLE: NULL
ACTION_REFERENCE_NEW_TABLE: NULL
  ACTION_REFERENCE_OLD_ROW: OLD
  ACTION_REFERENCE_NEW_ROW: NEW
                   CREATED: 2016-12-08 11:35:04.63
                  SQL_MODE: STRICT_TRANS_TABLES,NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION
                   DEFINER: c3gen@%
      CHARACTER_SET_CLIENT: gbk
      COLLATION_CONNECTION: gbk_chinese_ci
        DATABASE_COLLATION: utf8_general_ci
*************************** 2. row ***************************
           TRIGGER_CATALOG: def
            TRIGGER_SCHEMA: sys
              TRIGGER_NAME: sys_config_insert_set_user
        EVENT_MANIPULATION: INSERT
      EVENT_OBJECT_CATALOG: def
       EVENT_OBJECT_SCHEMA: sys
        EVENT_OBJECT_TABLE: sys_config
              ACTION_ORDER: 1
          ACTION_CONDITION: NULL
          ACTION_STATEMENT: BEGIN IF @sys.ignore_sys_config_triggers != true AND NEW.set_by IS NULL THEN SET NEW.set_by
= USER(); END IF; END
```

使用触发器

触发器: 不需要手动调用, 而是当某种情况发生时会自动触发.(goods 里面插入记录之后)

```
mysql> select * from stock;
+----+---------+
| id | sku_num |
+----+---------+
|  1 |     100 |
+----+---------+
1 row in set (0.00 sec)

mysql> insert into goods values(null,'橘子',1);
Query OK, 1 row affected (0.09 sec)

mysql> select * from stock;
+----+---------+
| id | sku_num |
+----+---------+
|  1 |      99 |
+----+---------+
1 row in set (0.00 sec)
```

修改触发器&删除触发器

触发器不能修改,只能先删除,后新增.

Drop trigger 触发器名字;

```
mysql> drop trigger af_sto;
Query OK, 0 rows affected (0.00 sec)
```

*****修改触发的 name: 可以通过 if not exists 来实现。

```
mysql> delimiter ##
mysql> drop trigger if exists my_tri ##
Query OK, 0 rows affected, 1 warning (0.04 sec)

mysql> create trigger my_tri after update on goods for each row
    -> begin
    -> update stock set sku_num=sku_num -1 where id= 1;
    -> end ##
Query OK, 0 rows affected (0.08 sec)

mysql> delimiter ;
```

触发器记录

触发器记录: 不管触发器是否触发了,只要当某种操作准备执行, 系统就会将当前要操作的记录的当前状态和即将执行之后新的状态给分别保留下来, 供触发器使用: 其中, 要操作的当前状态保存到 old 中, 操作之后的可能形态保存给 new.

Old 代表的是旧记录,new 代表的是新记录

删除的时候是没有 new 的; 插入的时候是没有 old

Old 和 new 都是代表记录本身: 任何一条记录除了有数据, 还有字段名字.
使用方式: old.字段名 / new.字段名(new 代表的是假设发生之后的结果)

因为我是 insert 之后的 update，所以没有 old 记录，只有 new 记录：

```
mysql> delimiter ##
mysql> create trigger asyn_stock after insert on goods for each row
    -> begin
    -> update stock set sku_num = sku_num - new.sku_num where id=1;
    -> end
    -> ##
Query OK, 0 rows affected (0.10 sec)
```

查看触发器的效果

```
mysql> select * from stock;
+----+---------+
| id | sku_num |
+----+---------+
|  1 |      99 |
+----+---------+
1 row in set (0.00 sec)
```

```
mysql> insert into goods values(3,'苹果',20);
Query OK, 1 row affected (0.08 sec)

mysql> select * from stock;
+----+---------+
| id | sku_num |
+----+---------+
|  1 |      79 |
+----+---------+
1 row in set (0.00 sec)
```

如果触发器内部只有一条要执行的 sql 指令, 可以省略大括号(begin 和 end)
create trigger 触发器名字 触发时间 事件类型 on 表名 for each row
一条sql 指令;

触发器: 可以很好的协调表内部的数据处理顺序和关系. 但是从 PHP 角度出发, 触发器会增加数据库维护的难度, 所以较少使用触发器.

3.8 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?
1. 防止数据丢失: 被盗, 误操作
2. 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

```
mysql> create table test_isam(
    -> id int)charset utf8 engine=myisam;
Query OK, 0 rows affected (0.07 sec)
```

创建后的目录如下：frm 是数据结构，myd 是数据，myi 是索引：

对比看下 我们用 innodb 引擎创建的文件：frm 是数据结构，ibd 是数据，trg 是触发器：

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. 我们从 c3gen 数据库目录下复制文件到 test 数据库下：

使用 use 来切换数据库

```
mysql> use test;
Database changed
mysql> select * from test_isam;
Empty set (0.00 sec)

mysql> show create table test_isam\G;
*************************** 1. row ***************************
       Table: test_isam
Create Table: CREATE TABLE `test_isam` (
  `id` int(11) DEFAULT NULL
) ENGINE=MyISAM DEFAULT CHARSET=utf8
1 row in set (0.00 sec)
```

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

```
mysql> select * from bill into outfile 'd:/bill.txt' ;
ERROR 1290 (HY000): The MySQL server is running with the --secure-file-priv option so it cannot execute this statement
```

如果你出现这样的错误。
找到 mysql 的 my.ini 文件(如果是解压缩版是没有这个错误的，但是安装版就会出现这个错误，因为这个选项默认是开启的，位置在 C:\ProgramData\MySQL\MySQL Server 5.7) 找到 这个 secure-file-priv 选项注释掉， 然后重启服务：

my.ini

```
long_query_time=10

# Binary Logging.
# log-bin

# Error Logging.
log-error="C3GEN.err"

# Server Id.
server-id=1

# Secure File Priv.
#secure-file-priv="C:/ProgramData/MySQL/MySQL Server 5.7/Uploads"

# The maximum amount of concurrent sessions the MySQL server will
# allow. One of these connections will be reserved for a user with
# SUPER privileges to allow the administrator to login even if the
# connection limit has been reached.
max_connections=151
```

```
mysql> use c3gen;
Database changed
mysql> select * from bill into outfile 'd:/bill.txt';
Query OK, 6 rows affected (0.06 sec)
```

再执行备份，发现执行成功：D 盘根目录下

打开这个备份的文件

bill.txt

```
1	早餐面条	5	1
2	早餐牛奶	5	1
3	午餐套餐	15	100
4	晚餐套餐	12	10
7	下午茶	10	2
8	夜宵	20	5
```

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

```
mysql> select * from bill into outfile 'd:/bill2.txt'
    -> fields terminated by ',';
Query OK, 6 rows affected (0.00 sec)
```

bill2.txt

```
1,早餐面条,5,1
2,早餐牛奶,5,1
3,午餐套餐,15,100
4,晚餐套餐,12,10
7,下午茶,10,2
8,夜宵,20,5
```

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

为了测试还原，我们清空 bill 数据表的数据，使用 truncate table 表名：

```
mysql> truncate table bill;
Query OK, 0 rows affected (0.22 sec)

mysql> select * from bill;
Empty set (0.00 sec)

mysql> load data infile 'd:/bill2.txt' into table bill
    -> fields terminated by ',';
Query OK, 6 rows affected (0.08 sec)
Records: 6  Deleted: 0  Skipped: 0  Warnings: 0

mysql> select * from bill;
+----+-----------+------------+----------+
| id | item_name | item_price | item_num |
+----+-----------+------------+----------+
|  1 | 早餐面条  |          5 |        1 |
|  2 | 早餐牛奶  |          5 |        1 |
|  3 | 午餐套餐  |         15 |      100 |
|  4 | 晚餐套餐  |         12 |       10 |
|  7 | 下午茶    |         10 |        2 |
|  8 | 夜宵      |         20 |        5 |
+----+-----------+------------+----------+
6 rows in set (0.00 sec)
```

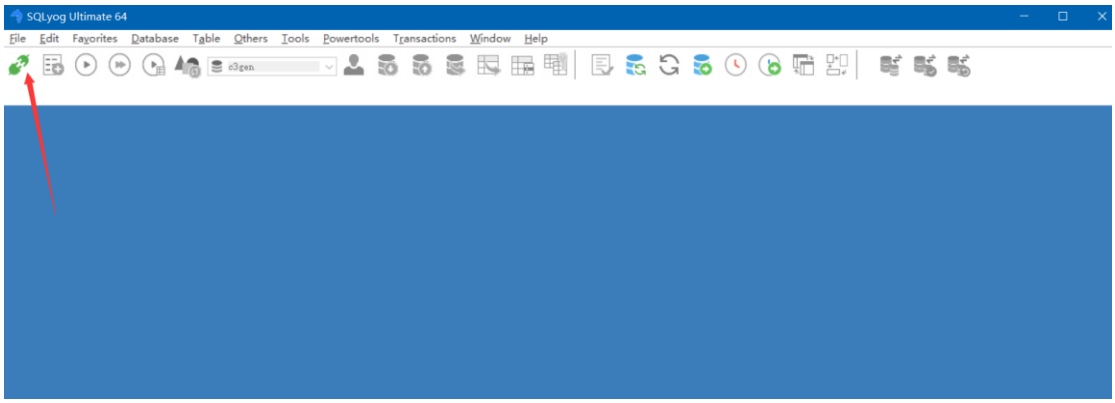
SQL 备份

备份

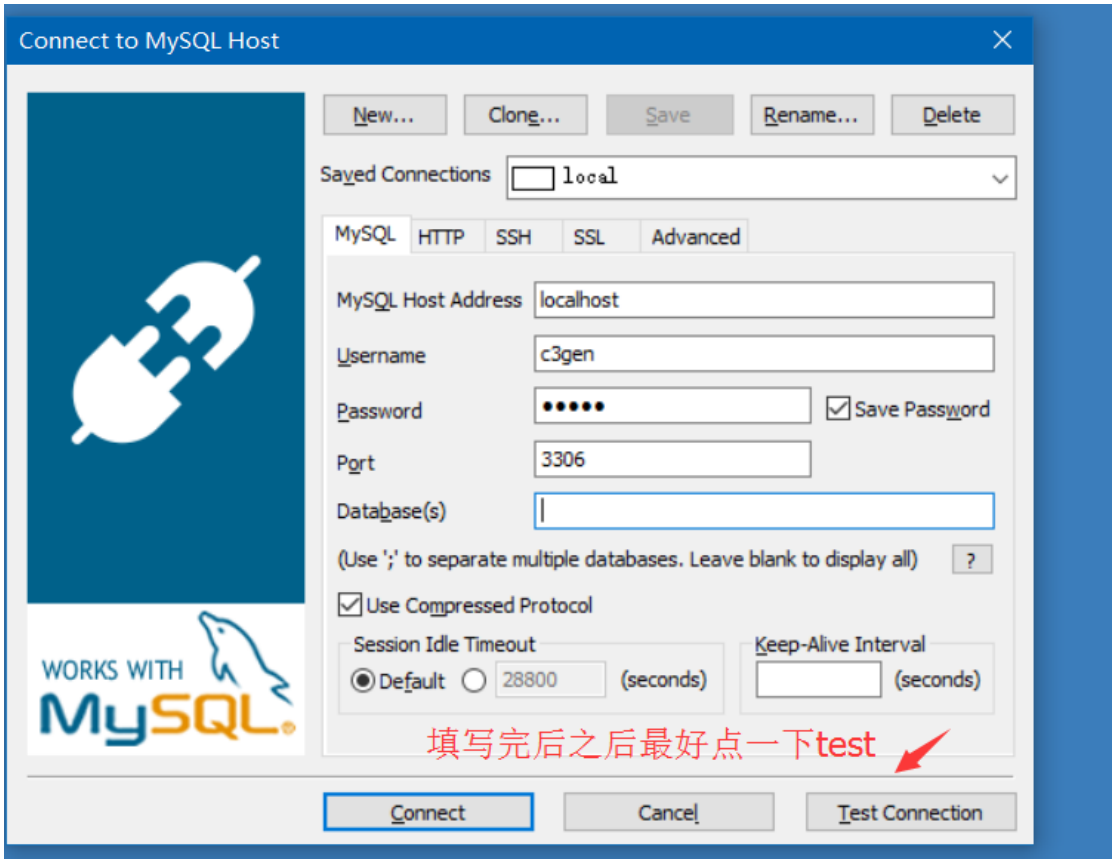
备份的是 SQL 语句: 系统会对表结构以及数据进行处理,变成对应的 SQL 语句, 然后进行备份: 还原的时候只要执行 SQL 指令即可.(主要就是针对表结构)

备份: mysql 没有提供备份指令: 需要利用自己的客户端或者第三方软件:比如 navicat for mysql、sqlyog，
我个人喜欢使用 sqlyog：
下面我简单写一下 sqlyog 怎么玩：

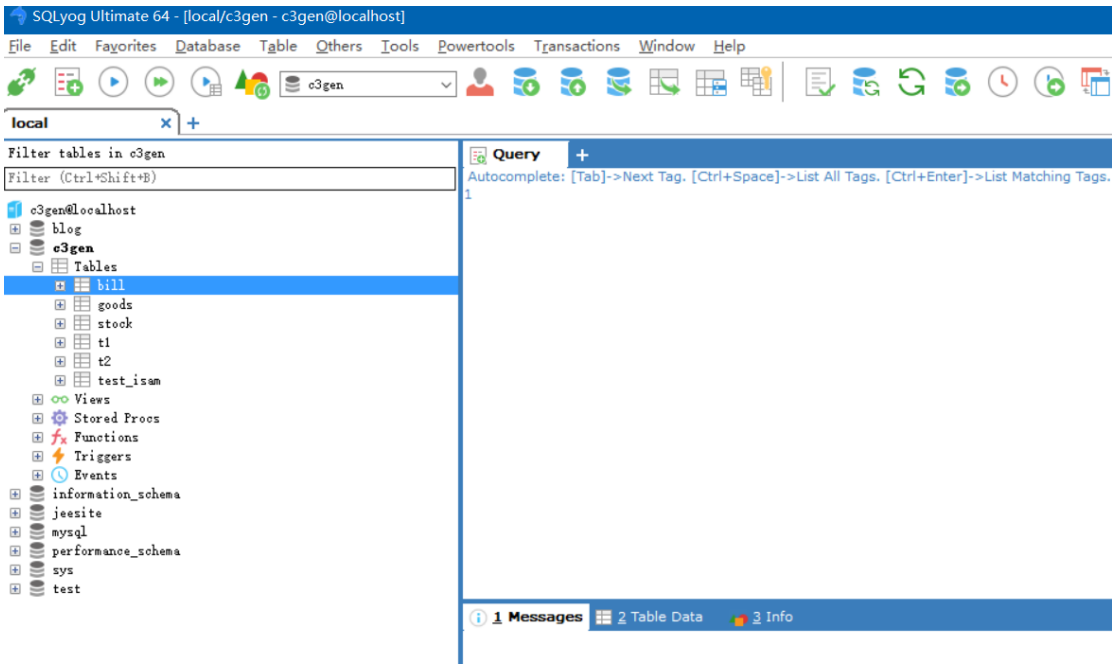
sqlyog 打开的界面是这样的。



点击这个插头一样的图标，创建会话连接：

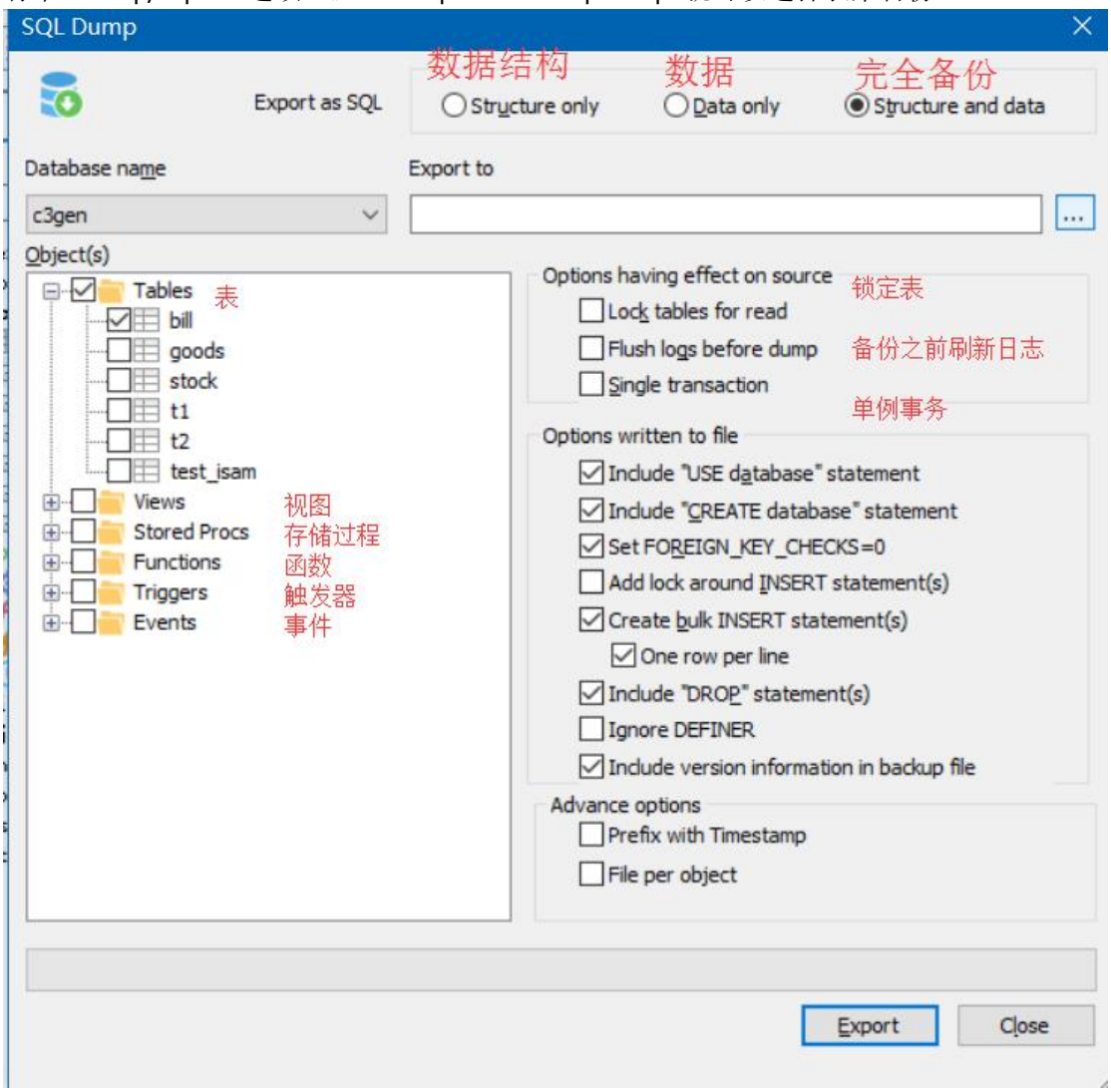


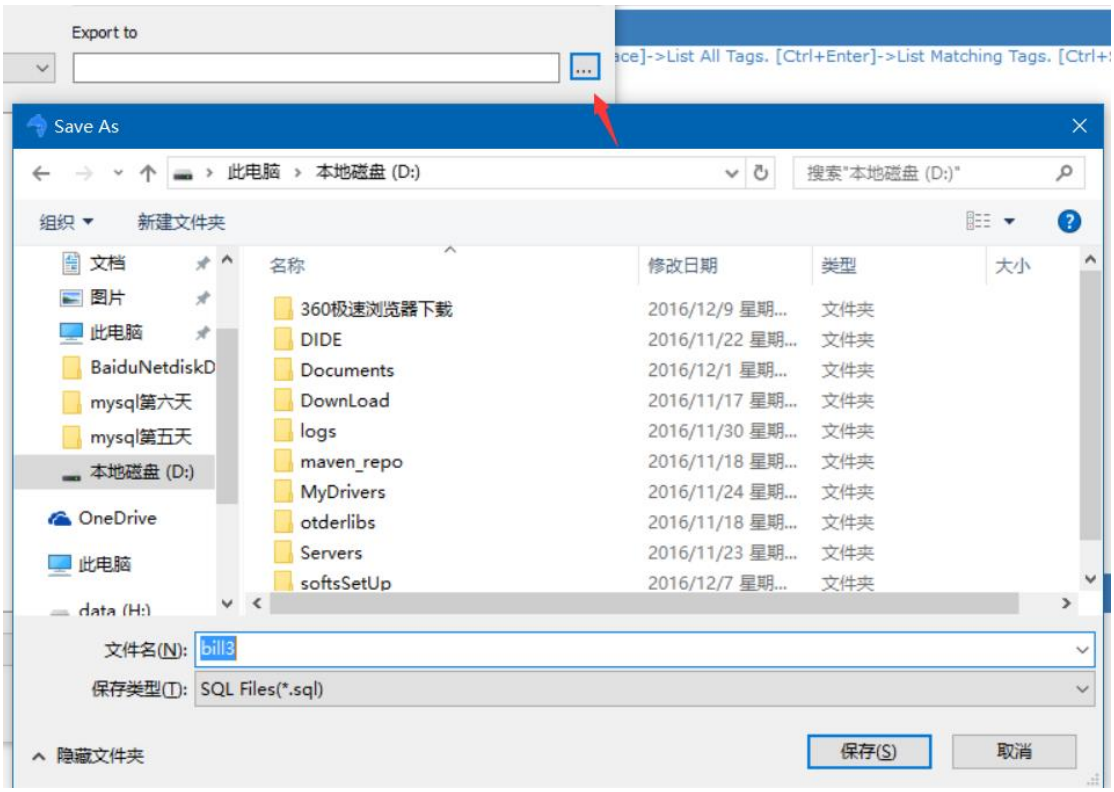
登录后：



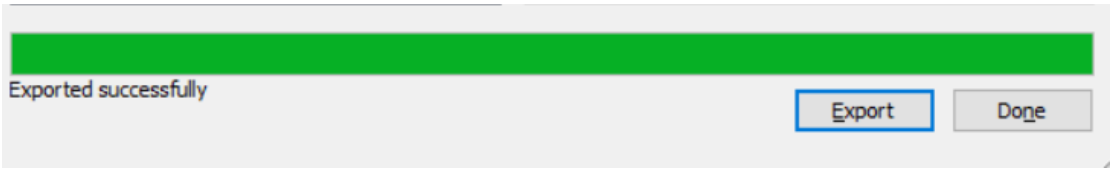
点到 bill 名，鼠标右键，打开菜单

有个 backup/export 选项 –》 backup tables as sql dump 就可以选择表来备份

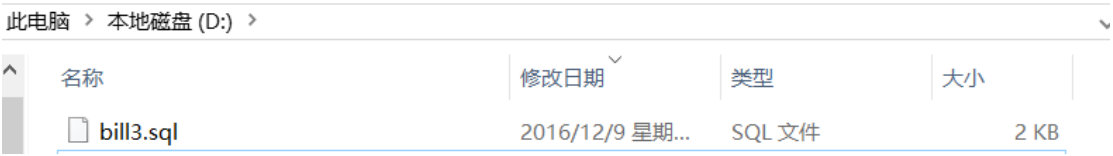




点最先的 export 导出 sql



查看备份的文件：



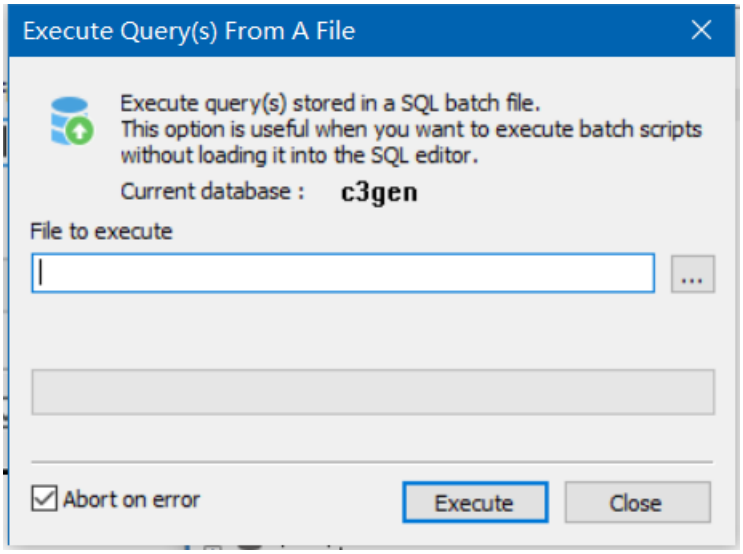
bill3.sql

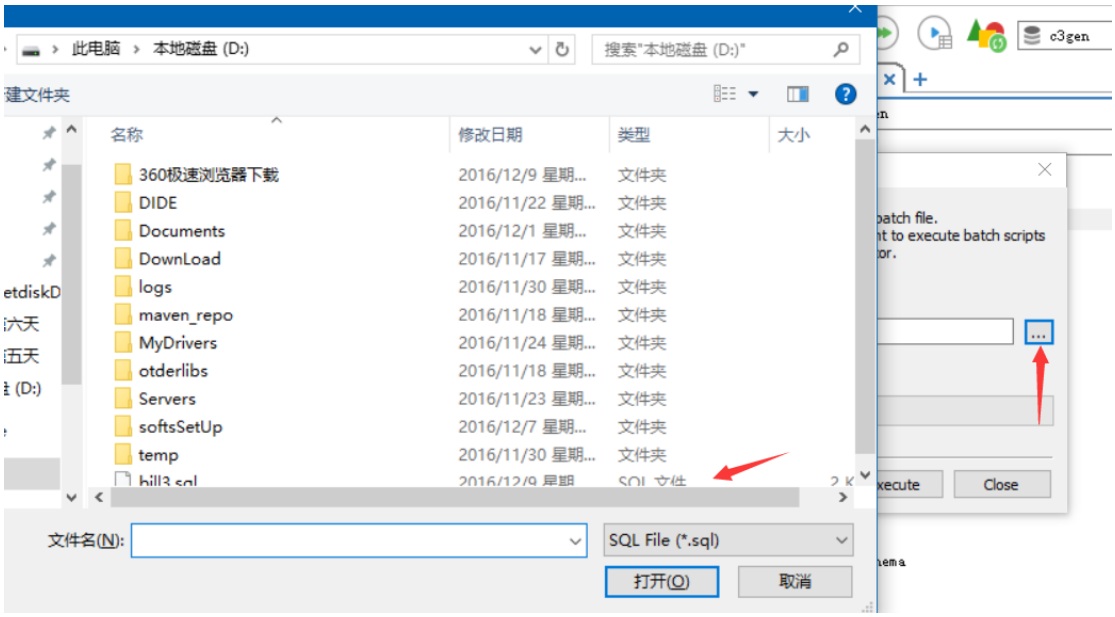
```sql
/*Table structure for table `bill` */

DROP TABLE IF EXISTS `bill`;

CREATE TABLE `bill` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `item_name` varchar(20) NOT NULL,
  `item_price` double DEFAULT NULL,
  `item_num` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=9 DEFAULT CHARSET=utf8;

/*Data for the table `bill` */

insert  into `bill`(`id`,`item_name`,`item_price`,`item_num`) values

(1,'早餐面条',5,1),

(2,'早餐牛奶',5,1),

(3,'午餐套餐',15,100),

(4,'晚餐套餐',12,10),

(7,'下午茶',10,2),
```

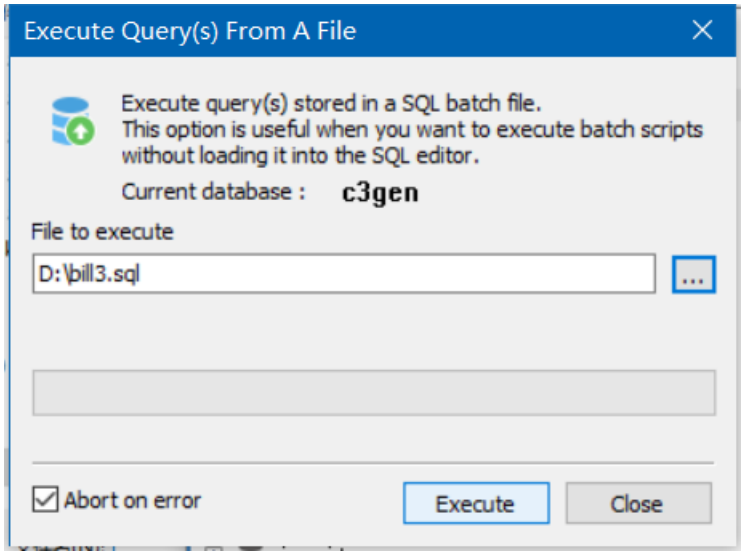
这样备份就完成了。

还原

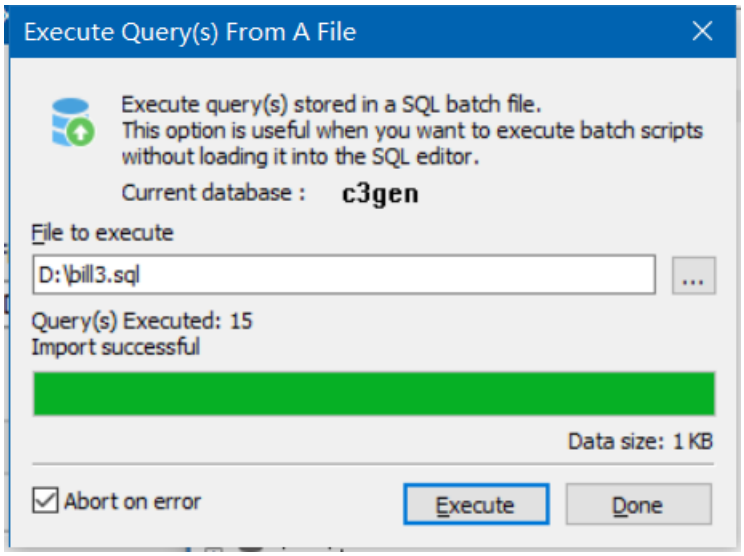
SQL 还原数据: 两种方式还原
方案 1: 使用客户端还原
使用 sqlyog, 鼠标指向数据库 c3gen,这是要还原的数据库名:
鼠标右键 import –》execute sql script ：







然后 execute 执行 sql



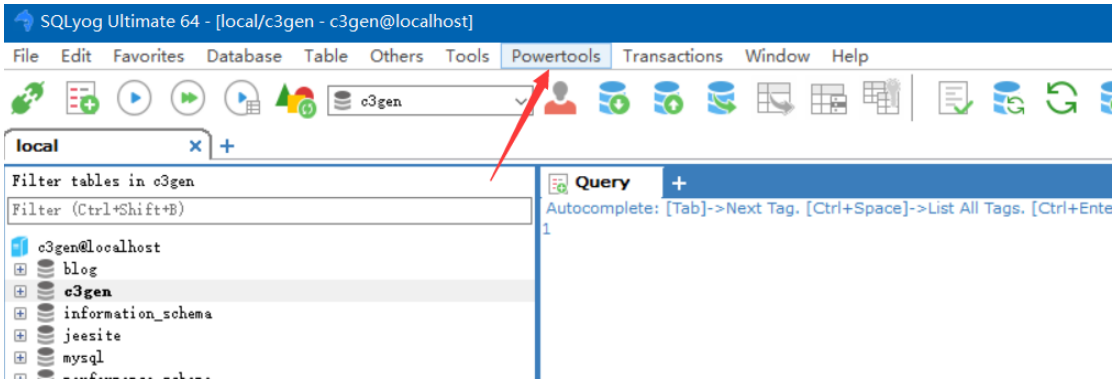
这样就还原了。
或者 更粗暴的，直接执行 sql 语句，把备份文件用 sql 打开，然后选中 f9 执行：
方案 2: 使用 SQL 指令还原
source 备份文件所在路径;
很多时候 我们会使用 source 命令来还原备份文件

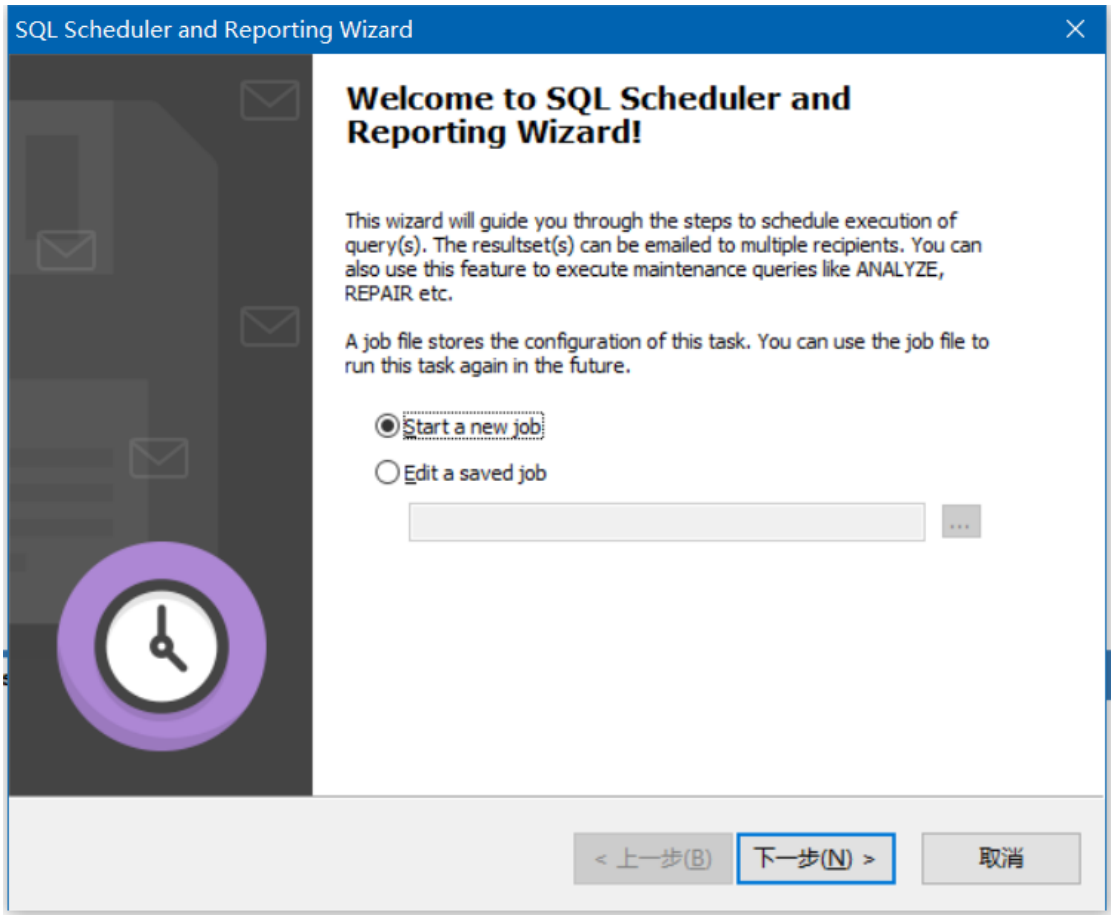
```
mysql> source d:/bill3.sql;
Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected, 1 warning (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

Query OK, 1 row affected (0.00 sec)
```

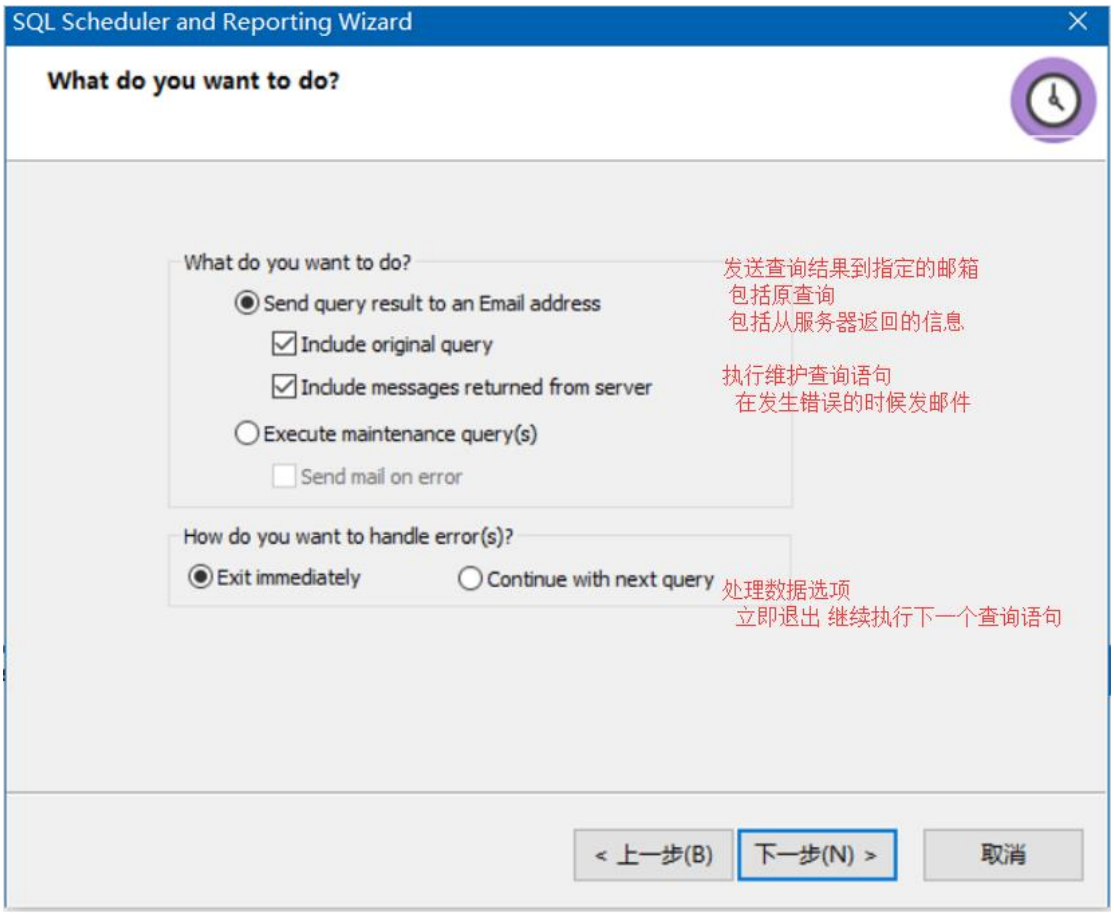
SQL 备份优缺点
1. 优点: 可以备份结构
2. 缺点: 会浪费空间(额外的增加 SQL 指令)

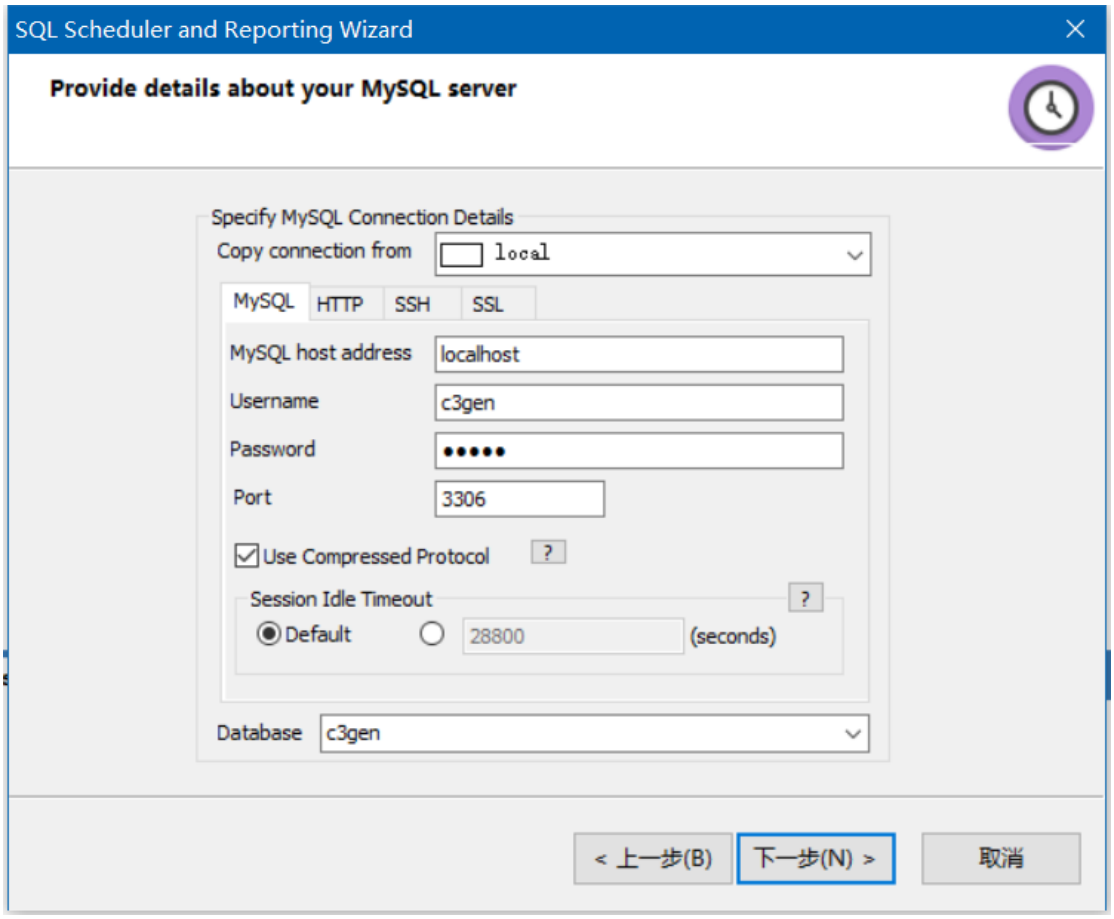
定时任务备份

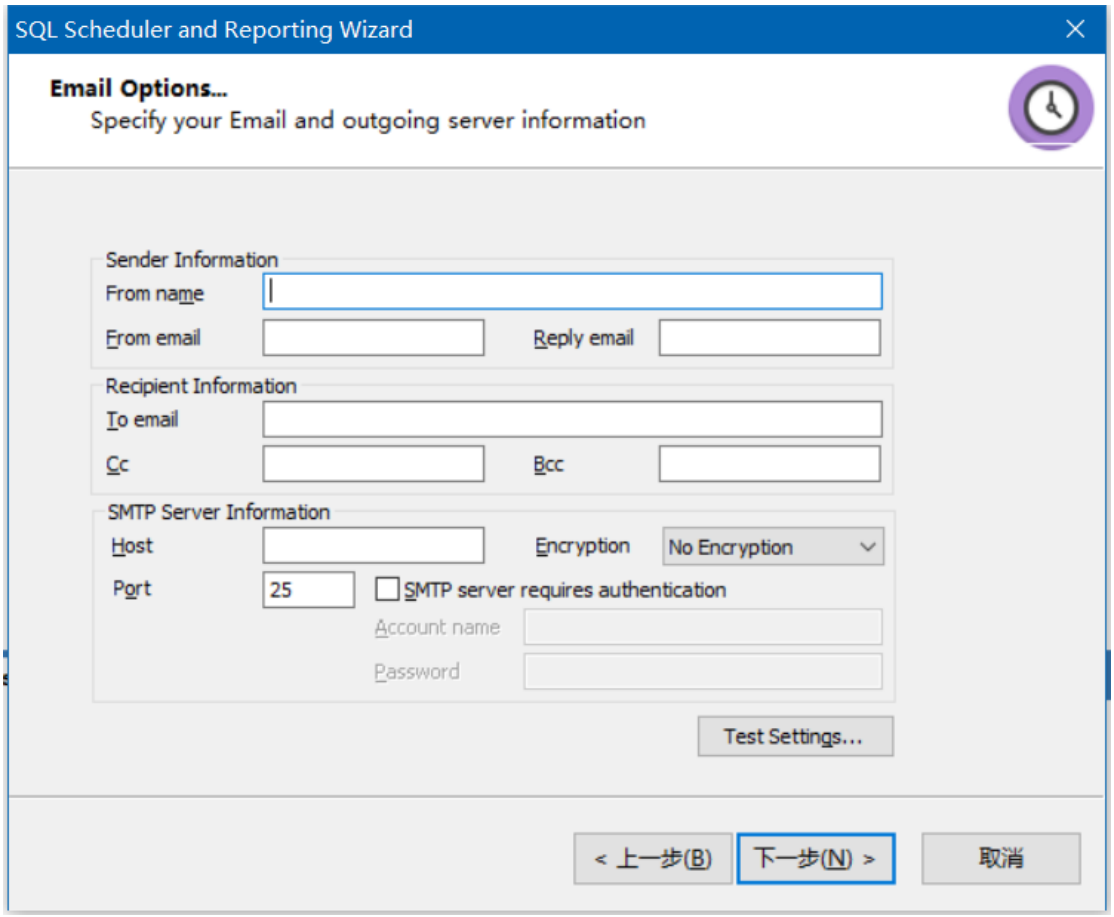
在很多时候，我们不可能一直守在电脑或者服务器旁边，这就需要定时任务来实现了。
还是使用我们的工具 sqlyog， 看到菜单栏有个 powertools，找到日历图标的选项
sql scheduler and reporting wizard 这个就是用来定时备份的任务计划：




SQL Scheduler and Reporting Wizard
Welcome to SQL Scheduler and Reporting Wizard!
This wizard will guide you through the steps to schedule execution of query(s). The resultset(s) can be emailed to multiple recipients. You can also use this feature to execute maintenance queries like ANALYZE, REPAIR etc.
A job file stores the configuration of this task. You can use the job file to run this task again in the future.
Start a new job
Edit a saved job
...
< 上一步(B)
下一步(N) >
取消


SQL Scheduler and Reporting Wizard
What do you want to do?
What do you want to do?
Send query result to an Email address
Include original query
Include messages returned from server
Execute maintenance query(s)
Send mail on error
How do you want to handle error(s)?
Exit immediately
Continue with next query
发送查询结果到指定的邮箱
包括原查询
包括从服务器返回的信息
执行维护查询语句
在发生错误的时候发邮件
处理数据选项
立即退出 继续执行下一个查询语句
< 上一步(B)
下一步(N) >
取消


SQL Scheduler and Reporting Wizard
Provide details about your MySQL server
Specify MySQL Connection Details
Copy connection from
local
MySQL
HTTP
SSH
SSL
MySQL host address
localhost
Username
c3gen
Password
Port
3306
Use Compressed Protocol
Session Idle Timeout
Default
28800
(seconds)
Database
c3gen
< 上一步(B)
下一步(N) >
取消



配置完成后，就可以开始定时任务了。

3.9 常见请求编码乱码(重要)

一般的动态网站都是有三部分构成，浏览器、服务器、数据库，这三部分各自独立，所以串联在一起的时候回出现乱码。 浏览器端实际是不可控的，所以程序控制很重要。