1、数据库介绍篇

1.1什么是数据库

数据库：保存数据的仓库。它体现我们电脑中，就是一个文件系统。然后把数据都保存这些特殊的文件中，并且需要使用固定的语言（**SQL**语言）去操作文件中的数据。

技术定义：

数据库(Database)是按照数据结构来组织、[存储和管理数据的建立在计算机存储设备上的仓库。

1.2数据库介绍

我们开发应用程序的时候，程序中的所有数据，最后都需要保存到专业软件中。这些专业的保存数据的软件我们称为数据库。

我们学习数据库，并不是学习如何去开发一个数据库软件，我们学习的是如何使用数据库以及数据库中的数据记录的操作。而数据库软件是由第三方公司研发。

1.3数据库的分类

关系型、非关系型的数据库

常见的数据库软件：

Oracle：它是Oracle公司的大型关系型数据库，它是收费的。

DB2：IBM公司的数据库，它是收费的。

SqlServer：微软数据库。收费

Sybase：Sybase公司的。 工具PowerDesign 数据库建模工具。

MySql：早期瑞典一个公司发明，后期被sun公司收购，后期被Oracle。

Java开发应用程序主要使用的数据库：

MySQL（5.5）、Oracle、DB2。

1.4什么是关系型数据库

在开发软件的时候，软件中的数据之间必然会有一定的关系存在，需要把这些数据保存在数据库中，同时也要维护数据之间的关系，这时就可以直接使用上述的那些数据库。而上述的所有数据库都属于关系型数据库。

描述数据之间的关系，并保存在数据库中，同时学习如果根据这些关系查询数据库中的数据，

关系型数据：设计数据库的时候，需要使用E-R图来描述。实体关系

E-R：实体关系图。

实体：可以理解成我们Java程序中的一个对象。在E-R图中使用 矩形(长方形) 表示。

针对一个实体中的属性，我们称为这个实体的数据，在E-R图中使用 椭圆表示。

实体和实体之间的关系：在E-R图中使用菱形表示。

2、mysql在linux-安装篇

2.1、vmware中安装linux注意事项

2.1.1、记得关闭防火墙

```
service iptables stop
chkconfig iptables off（关闭开机自启：所谓的永久关闭防火墙）
```

2.1.2、创建统一的管理目录

```
mkdir -p /exprot/software

mkdir -p /export/servers
```

2.1.3软件环境

```
VMware、crt、centos6.9
```

2.1.4安装环境

```
1、VMware软件安装

        2、构建虚拟机

        3、需要配置Linux（ip,mac地址，hostname，防火墙），就可以通过crt这个客户端连接进行操作

        4、在linux操作系统进行安装msyql-5.6

        说明：因为在linux操作系统中，安装软件的方式主要有3种：1、源码安装（redis）2、rpm安装

3、yum在线安装（安装MySQL为例）---linux联网（）
```

2.2、centos6.9安装mysql

2.2.1、检查是否有自带的mysql

```
[root@hadoop-01 servers]# rpm -qa |grep mysql
mysql-libs-5.1.73-8.el6_8.x86_64
```

2.2.2、卸载自带的mysql

```
[root@hadoop-01 servers]# rpm -e --nodeps mysql-libs-5.1.73-8.el6_8.x86_64
[root@hadoop-01 servers]#
```

2.2.3、下载mysql安装包

2.2.4、上传安装包到linux服务器

```
rz 上传文件到指定的目录（yum install lrzsz）
/export/software/mysql
```

2.2.5、安装

```
rpm -ivh *.rpm
```

2.2.6、查看初始化密码

```
A RANDOM PASSWORD HAS BEEN SET FOR THE MySQL root USER !
You will find that password in '/root/.mysql_secret'.
```

```
[root@mysql ~]# cat /root/.mysql_secret
# The random password set for the root user at Wed Aug  8 22:19:00 2018 (local time):
xQkcU3kbyuZby1_V

[root@mysql ~]#
```

2.2.7、启动mysql并登录

```
#启动mysql
service mysql start
```

```
#登录mysql
mysql -uroot -p
(粘贴密码：xQkcU3kbyuZby1_V)
```

2.2.8、修改密码

```
set PASSWORD=PASSWORD('123456');
```

2.2.9、退出mysql客户端

```
mysql>quit
```

2.2.10、用新密码进行登录

```
mysql -uroot -p
123456（新密码）
```

2.2.11、远程授权

```
grant all privileges on *.* to 'root' @'%' identified by '123456';
flush privileges;
```

2.2.12、验证远程授权是否成功

```
通过windows的mysql客户端工具连接，是否能连接上，能连接上就授权成功，没有连接上，说明没有授权成功！
```

3、mysql-基础操作篇

3.1、登录mysql

```
mysql -uroot -p
123456
```

3.2、退出mysql

```
mysql>quit
```

3.3、输入查询

- 查看当前mysql的版本号及当前时间

```
SELECT VERSION(), CURRENT_DATE;
```

```
mysql> SELECT VERSION(), CURRENT_DATE;
+-----------+--------------+
| VERSION() | CURRENT_DATE |
+-----------+--------------+
| 5.6.25    | 2018-08-08   |
+-----------+--------------+
1 row in set (0.32 sec)
```

- mysql中sql语句不区分大小写

```
mysql> SELECT VERSION(), CURRENT_DATE;
mysql> select version(), current_date;
mysql> SeLeCt vErSiOn(), current_DATE;
```

```
mysql> SELECT VERSION(), CURRENT_DATE;
+-----------+--------------+
| VERSION() | CURRENT_DATE |
+-----------+--------------+
| 5.6.25    | 2018-08-08   |
+-----------+--------------+
1 row in set (0.00 sec)

mysql> select version(), current_date;
+-----------+--------------+
| version() | current_date |
+-----------+--------------+
| 5.6.25    | 2018-08-08   |
+-----------+--------------+
1 row in set (0.00 sec)

mysql> SeLeCt vErSiOn(), current_DATE;
+-----------+--------------+
| vErSiOn() | current_DATE |
+-----------+--------------+
| 5.6.25    | 2018-08-08   |
+-----------+--------------+
1 row in set (0.00 sec)

mysql>
```

- 可以进行简单的计算（如下所示）

```
mysql>SELECT SIN(PI()/4), (4+1)*5;

mysql> SELECT SIN(PI()/4), (4+1)*5;
+--------------------+---------+
| SIN(PI()/4)        | (4+1)*5 |
+--------------------+---------+
| 0.7071067811865475 |      25 |
+--------------------+---------+
1 row in set (0.34 sec)
```

- 多条语句比较短，可以写在一行

```
mysql>SELECT VERSION(); SELECT NOW();

mysql> SELECT VERSION(); SELECT NOW();
+-----------+
| VERSION() |
+-----------+
| 5.6.25    |
+-----------+
1 row in set (0.00 sec)

+---------------------+
| NOW()               |
+---------------------+
| 2018-08-08 23:11:11 |
+---------------------+
1 row in set (0.00 sec)
```

- 多个字段之间可以用逗号分隔，多行组成一条语句结束以分号结束

```
mysql> SELECT
-> USER()
-> ,
-> CURRENT_DATE;
+---------------+--------------+
| USER() | CURRENT_DATE |
+---------------+--------------+
| jon@localhost | 2010-08-06 |
+---------------+--------------+
```

- sql语句写了一半，又不想执行可以在语句末尾加上'\c'

```
mysql> select
    -> user()
    -> \c
mysql>
```

3.4、创建和使用数据库

- 查看当前有哪些数据库

```
mysql>show databases;

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| test               |
+--------------------+
4 rows in set (0.07 sec)
```

- 创建数据库

```
mysql> CREATE DATABASE menagerie;
```

- 使用及切换数据库

```
mysql> USE menagerie
Database changed
```

3.5、创建表及使用

- 查看当前数据库有哪些表

```
mysql>show tables;
```

- 创建一个表

```
mysql> CREATE TABLE pet (name VARCHAR(20), owner VARCHAR(20),
    -> species VARCHAR(20), sex CHAR(1), birth DATE, death DATE);
```

- 校验创建表语句是否和执行的一致

```
mysql>show create table pet;

+-------+---------------------------------
| Table | Create Table
+-------+---------------------------------
| pet   | CREATE TABLE `pet` (
  `name` varchar(20) DEFAULT NULL,
  `owner` varchar(20) DEFAULT NULL,
  `species` varchar(20) DEFAULT NULL,
  `sex` char(1) DEFAULT NULL,
  `birth` date DEFAULT NULL,
  `death` date DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1 |
+-------+---------------------------------
```

- 查看表详情

```
mysql> desc pet;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| name    | varchar(20) | YES  |     | NULL    |       |
| owner   | varchar(20) | YES  |     | NULL    |       |
| species | varchar(20) | YES  |     | NULL    |       |
| sex     | char(1)     | YES  |     | NULL    |       |
| birth   | date        | YES  |     | NULL    |       |
| death   | date        | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
```

- 准备数据

```
Fluffy Harold cat f 1993-02-04
Claws Gwen cat m 1994-03-17
Buffy Harold dog f 1989-05-13
Fang Benny dog m 1990-08-27
Bowser Diane dog m 1979-08-31 1995-07-29
Chirpy Gwen bird f 1998-09-11
Whistler Gwen bird 1997-12-09
Slim Benny snake m 1996-04-29
```

3.6、表中导入数据

在表中导入数据的方式有两种

- 第一种：将以上数据整理成SQL语句，insert into pet....
- 第二种：通过加载文件的方式将数据导入到表中

1、创建一个pet.txt的文件（注：每个字段中用tab键隔开，字段没有值得记录用\N代替）

```
Fluffy  Harold  cat     f       1993-02-04
Claws   Gwen    cat     m       1994-03-17
Buffy   Harold  dog     f       1989-05-13
Fang    Benny   dog     m       1990-08-27
Bowser  Diane   dog     m       1979-08-31      1995-07-29
Chirpy  Gwen    bird    f       1998-09-11
Whistler        Gwen    bird    \N      1997-12-09      \N
Slim    Benny   snake   m       1996-04-29
```

2、加载数据

```
mysql> load data local infile '/root/data/pet.txt' into table pet;
Query OK, 8 rows affected, 6 warnings (0.06 sec)
Records: 8  Deleted: 0  Skipped: 0  Warnings: 6
```

3、校验是否加载进去

```
mysql> select *from pet;
+----------+--------+---------+------+------------+------------+
| name     | owner  | species | sex  | birth      | death      |
+----------+--------+---------+------+------------+------------+
| Fluffy   | Harold | cat     | f    | 1993-02-04 | NULL       |
| Claws    | Gwen   | cat     | m    | 1994-03-17 | NULL       |
| Buffy    | Harold | dog     | f    | 1989-05-13 | NULL       |
| Fang     | Benny  | dog     | m    | 1990-08-27 | NULL       |
| Bowser   | Diane  | dog     | m    | 1979-08-31 | 1995-07-29 |
| Chirpy   | Gwen   | bird    | f    | 1998-09-11 | NULL       |
| Whistler | Gwen   | bird    | NULL | 1997-12-09 | NULL       |
| Slim     | Benny  | snake   | m    | 1996-04-29 | NULL       |
+----------+--------+---------+------+------------+------------+
8 rows in set (0.01 sec)
```

3.7、数据检索部分

3.7.1、检索全部数据

```
mysql> select *from pet;
+----------+--------+---------+------+------------+------------+
| name     | owner  | species | sex  | birth      | death      |
+----------+--------+---------+------+------------+------------+
| Fluffy   | Harold | cat     | f    | 1993-02-04 | NULL       |
| Claws    | Gwen   | cat     | m    | 1994-03-17 | NULL       |
| Buffy    | Harold | dog     | f    | 1989-05-13 | NULL       |
| Fang     | Benny  | dog     | m    | 1990-08-27 | NULL       |
| Bowser   | Diane  | dog     | m    | 1979-08-31 | 1995-07-29 |
| Chirpy   | Gwen   | bird    | f    | 1998-09-11 | NULL       |
| Whistler | Gwen   | bird    | NULL | 1997-12-09 | NULL       |
| Slim     | Benny  | snake   | m    | 1996-04-29 | NULL       |
+----------+--------+---------+------+------------+------------+
8 rows in set (0.01 sec)
```

3.7.2、删除表中全部数据

```
mysql> DELETE FROM pet;
mysql> LOAD DATA LOCAL INFILE '/path/pet.txt' INTO TABLE pet;
```

3.7.3、更新表中特定记录的数据

- 更新表中名字为Bowser的生日

```
mysql> UPDATE pet SET birth = '1989-08-31' WHERE name = 'Bowser';
```

3.7.4、查询特定的行

- 查询名字为Bowser的记录

```
mysql> SELECT * FROM pet WHERE name = 'Bowser';
+--------+-------+---------+------+------------+------------+
| name | owner | species | sex | birth | death |
+--------+-------+---------+------+------------+------------+
| Bowser | Diane | dog | m | 1989-08-31 | 1995-07-29 |
+--------+-------+---------+------+------------+------------+
```

说明：字符串比较不区分大小写！如下所示：

```
mysql> SELECT * FROM pet WHERE name = 'Bowser';
+--------+-------+---------+------+------------+------------+
| name   | owner | species | sex  | birth      | death      |
+--------+-------+---------+------+------------+------------+
| Bowser | Diane | dog     | m    | 1979-08-31 | 1995-07-29 |
+--------+-------+---------+------+------------+------------+
1 row in set (0.00 sec)

mysql> SELECT * FROM pet WHERE name = 'BowsEr';
+--------+-------+---------+------+------------+------------+
| name   | owner | species | sex  | birth      | death      |
+--------+-------+---------+------+------------+------------+
| Bowser | Diane | dog     | m    | 1979-08-31 | 1995-07-29 |
+--------+-------+---------+------+------------+------------+
1 row in set (0.00 sec)

mysql> SELECT * FROM pet WHERE name = 'BOWSER';
+--------+-------+---------+------+------------+------------+
| name   | owner | species | sex  | birth      | death      |
+--------+-------+---------+------+------------+------------+
| Bowser | Diane | dog     | m    | 1979-08-31 | 1995-07-29 |
+--------+-------+---------+------+------------+------------+
1 row in set (0.00 sec)
```

3.7.4.1、查找生日在1998年以后的特定查询

```
mysql> SELECT * FROM pet WHERE birth >= '1998-1-1';
+----------+-------+---------+------+------------+-------+
| name | owner | species | sex | birth | death |
+----------+-------+---------+------+------------+-------+
| Chirpy | Gwen | bird | f | 1998-09-11 | NULL |
| Puffball | Diane | hamster | f | 1999-03-30 | NULL |
+----------+-------+---------+------+------------+-------+
```

3.7.4.2、多条件查询（and | or）

```
mysql> SELECT * FROM pet WHERE species = 'dog' AND sex = 'f';
+-------+--------+---------+------+------------+-------+
| name | owner | species | sex | birth | death |
+-------+--------+---------+------+------------+-------+
| Buffy | Harold | dog | f | 1989-05-13 | NULL |
+-------+--------+---------+------+------------+-------+
```

```
mysql> SELECT * FROM pet WHERE species = 'snake' OR species = 'bird';
+----------+-------+---------+------+------------+-------+
| name | owner | species | sex | birth | death |
+----------+-------+---------+------+------------+-------+
| Chirpy | Gwen | bird | f | 1998-09-11 | NULL |
| Whistler | Gwen | bird | NULL | 1997-12-09 | NULL |
| Slim | Benny | snake | m | 1996-04-29 | NULL |
+----------+-------+---------+------+------------+-------+
```

- 优先执行括号中的逻辑

```
mysql> SELECT * FROM pet WHERE (species = 'cat' AND sex = 'm')
-> OR (species = 'dog' AND sex = 'f');
+-------+--------+---------+------+------------+-------+
| name | owner | species | sex | birth | death |
+-------+--------+---------+------+------------+-------+
| Claws | Gwen | cat | m | 1994-03-17 | NULL |
| Buffy | Harold | dog | f | 1989-05-13 | NULL |
+-------+--------+---------+------+------------+-------+
```

3.7.5、检索特定的列

```
mysql> SELECT name, birth FROM pet;
+----------+------------+
| name | birth |
+----------+------------+
| Fluffy | 1993-02-04 |
| Claws | 1994-03-17 |
| Buffy | 1989-05-13 |
| Fang | 1990-08-27 |
| Bowser | 1989-08-31 |
| Chirpy | 1998-09-11 |
| Whistler | 1997-12-09 |
| Slim | 1996-04-29 |
| Puffball | 1999-03-30 |
+----------+------------+
```

- 查询不重复的字段要使用关键词DISTINCT

```
mysql> SELECT DISTINCT owner FROM pet;
+--------+
| owner |
+--------+
| Benny |
| Diane |
| Gwen |
| Harold |
+--------+
```

- 可以使用组合条件查询特定的列

```
mysql> SELECT name, species, birth FROM pet
-> WHERE species = 'dog' OR species = 'cat';
+--------+---------+------------+
| name | species | birth |
+--------+---------+------------+
| Fluffy | cat | 1993-02-04 |
| Claws | cat | 1994-03-17 |
| Buffy | dog | 1989-05-13 |
| Fang | dog | 1990-08-27 |
| Bowser | dog | 1989-08-31 |
+--------+---------+------------+
```

3.7.6、排序

- 根据某个字段进行排序（关键词：ORDER BY ）

```
mysql> SELECT name, birth FROM pet ORDER BY birth;
+----------+------------+
| name | birth |
+----------+------------+
| Buffy | 1989-05-13 |
| Bowser | 1989-08-31 |
| Fang | 1990-08-27 |
| Fluffy | 1993-02-04 |
| Claws | 1994-03-17 |
| Slim | 1996-04-29 |
| Whistler | 1997-12-09 |
| Chirpy | 1998-09-11 |
| Puffball | 1999-03-30 |
+----------+------------+
```

- 升降序排列（desc：降序；asc：升序）

```
mysql> SELECT name, birth FROM pet ORDER BY birth desc;//降序排列
mysql> SELECT name, birth FROM pet ORDER BY birth asc ;//升序排列
```

- 多列排序

  根据species字段升序排列，根据birth字段降序排列

  注： **ORDER BY species** 中无**asc**，**desc**，默认为升序排列

```
mysql> SELECT name, species, birth FROM pet
-> ORDER BY species, birth DESC;
+----------+---------+------------+
| name | species | birth |
+----------+---------+------------+
| Chirpy | bird | 1998-09-11 |
| Whistler | bird | 1997-12-09 |
| Claws | cat | 1994-03-17 |
| Fluffy | cat | 1993-02-04 |
| Fang | dog | 1990-08-27 |
| Bowser | dog | 1989-08-31 |
| Buffy | dog | 1989-05-13 |
| Puffball | hamster | 1999-03-30 |
| Slim | snake | 1996-04-29 |
+----------+---------+------------+
```

3.7.7、日期计算

查看宠物多少岁，就可以使用计算日期的函数TIMESTAMPDIFF()

```
#查询当前的日期
mysql> select curdate() from pet;
+------------+
| curdate()  |
+------------+
| 2018-08-09 |
+------------+

#获取当年的年
mysql> select YEAR('2018-02-05') AS YEARS from pet;
+-------+
| YEARS |
+-------+
|  2018 |
+-------+

#获取当年的月
mysql> select month('2018-02-05') AS YEARS from pet;
+-------+
| YEARS |
+-------+
|     2 |
+-------+

#获取当年的日
mysql> select day('2018-02-05') AS YEARS from pet;
+-------+
| YEARS |
+-------+
|     5 |
+-------+
```

```
mysql> SELECT name, birth, CURDATE(),
-> TIMESTAMPDIFF(YEAR,birth,CURDATE()) AS age
-> FROM pet;
```

3.7.8、null和not null值

对一些字段类型要进行检查，判断某些字段是否为NULL，或者 non-NULL

```
mysql> SELECT name, birth, death,
-> TIMESTAMPDIFF(YEAR,birth,death) AS age
-> FROM pet WHERE death IS NOT NULL ORDER BY age;
+--------+------------+------------+------+
| name | birth | death | age |
+--------+------------+------------+------+
| Bowser | 1989-08-31 | 1995-07-29 | 5 |
+--------+------------+------------+------+
```

4、实例

以下是如何解决MySQL的一些常见问题的示例。

4.1、首先创建一个表，并且导入数据

```
CREATE TABLE shop (
article INT(4) UNSIGNED ZEROFILL DEFAULT '0000' NOT NULL,
dealer CHAR(20) DEFAULT '' NOT NULL,
price DOUBLE(16,2) DEFAULT '0.00' NOT NULL,
PRIMARY KEY(article, dealer));


INSERT INTO shop VALUES
(1,'A',3.45),(1,'B',3.99),(2,'A',10.99),(3,'B',1.45),
(3,'C',1.69),(3,'D',1.25),(4,'D',19.95);
```

4.2、检索表中的全部数据

```
select * from shop;

+---------+--------+-------+
| article | dealer | price |
+---------+--------+-------+
|    0001 | A      |  3.45 |
|    0001 | B      |  3.99 |
|    0002 | A      | 10.99 |
|    0003 | B      |  1.45 |
|    0003 | C      |  1.69 |
|    0003 | D      |  1.25 |
|    0004 | D      | 19.95 |
+---------+--------+-------+
```

4.3、求某一列的最大值或者 最小值

```
SELECT MAX(article) AS article FROM shop;

+---------+
| article |
+---------+
|       4 |
+---------+

//求某一列的最小值
 select min(price) as article from shop;

 +---------+
| article |
+---------+
|    1.25 |
+---------+
```

4.4、过滤出某个字段值最大的整条记录数据-涉及到子查询

```
SELECT article, dealer, price
FROM shop
WHERE price=(SELECT MAX(price) FROM shop);

+---------+--------+-------+
| article | dealer | price |
+---------+--------+-------+
|    0004 | D      | 19.95 |
+---------+--------+-------+
```

4.4、也可以通过关联查询来进行检索

```
SELECT s1.article, s1.dealer, s1.price
FROM shop s1
LEFT JOIN shop s2 ON s1.price < s2.price
WHERE s2.article IS NULL;

SELECT article, dealer, price
FROM shop
ORDER BY price DESC
LIMIT 1;
```

4.5、求出每一列的最大值，并且根据某一个字段进行分组--分组**topn**求法

```
SELECT article, MAX(price) AS price
FROM shop
GROUP BY article;

+---------+-------+
| article | price |
+---------+-------+
|    0001 |  3.99 |
|    0002 | 10.99 |
|    0003 |  1.69 |
|    0004 | 19.95 |
+---------+-------+
```

4.5的另一种写法

```
SELECT article, dealer, price
FROM shop s1
WHERE price=(SELECT MAX(s2.price)
FROM shop s2
WHERE s1.article = s2.article);

+---------+--------+-------+
| article | dealer | price |
+---------+--------+-------+
|    0001 | B      |  3.99 |
|    0002 | A      | 10.99 |
|    0003 | C      |  1.69 |
|    0004 | D      | 19.95 |
+---------+--------+-------+
```

5、SQL中的聚合函数

SQL语言中定义了部分的函数，可以帮助我们完成对查询结果的计算操作：

1.count 统计个数（行数）

2.sum函数：求和

3.avg函数：求平均值

4.max、min 求最大值和最小值

5.1、count函数

语法：**select count(*)|count(**列名) from表名

注意： count在根据指定的列统计的时候，如果这一列中有null 不会被统计在其中。

```
mysql> select * from pet;
+----------+--------+---------+------+------------+------------+
| name     | owner  | species | sex  | birth      | death      |
+----------+--------+---------+------+------------+------------+
| Fluffy   | Harold | cat     | f    | 1993-02-04 | NULL       |
| Claws    | Gwen   | cat     | m    | 1994-03-17 | NULL       |
| Buffy    | Harold | dog     | f    | 1989-05-13 | NULL       |
| Fang     | Benny  | dog     | m    | 1990-08-27 | NULL       |
| Bowser   | Diane  | dog     | m    | 1989-08-31 | 1995-07-29 |
| Chirpy   | Gwen   | bird    | f    | 1998-09-11 | NULL       |
| Whistler | Gwen   | bird    | NULL | 1997-12-09 | NULL       |
| Slim     | Benny  | snake   | m    | 1996-04-29 | NULL       |
+----------+--------+---------+------+------------+------------+
8 rows in set (0.00 sec)

mysql> select count(sex) from pet;
+------------+
| count(sex) |
+------------+
|          7 |
+------------+
1 row in set (0.00 sec)

mysql> select count(owner) from pet;
+--------------+
| count(owner) |
+--------------+
|            8 |
+--------------+
1 row in set (0.00 sec)

mysql> select count(death) from pet;
+--------------+
| count(death) |
+--------------+
|            1 |
+--------------+
```

5.2、sum函数

语法：**select sum(**列名) from 表名;

注意事项：

1、如果使用sum 多列进行求和的时候，如果某一列中的有null，这一列所在的行中的其他数据不会被加到总和。

2、可以使用mysql 数据库提供的函数 ifnull(列名,值)

3、在数据库中定义double类型数据，是一个近似值，需要确定准确的位数，这时可以把这一列设计成numeric类型。numeric(数据的总列数,小数位数)

numericdouble float

```
mysql> select sum(price) from shop;
+------------+
| sum(price) |
+------------+
|      42.77 |
+------------+
```

5.3、avg函数

语法：select avg(列名) from 表名;

```
mysql> select avg(price) from shop;
+------------+
| avg(price) |
+------------+
|   6.110000 |
+------------+
```

5.4、max函数

语法：select max(列名) from 表名;

```
mysql> select max(price) from shop;
+------------+
| max(price) |
+------------+
|      19.95 |
+------------+
```

5.5、min函数

语法：select min(列名) from 表名;

```
mysql> select min(price) from shop;
+------------+
| min(price) |
+------------+
|       1.25 |
+------------+
```

6、SQL分类

6.1、DDL（数据定义问题）

数据定义语言 - Data Definition Language

用来定义数据库的对象，如数据表、视图、索引等

```
创建数据库：create database test;
创建视图：create  view  test;
创建索引：create index test;
创建表：create table test1;
```

6.2、DML（数据操纵问题）

数据处理语言 - Data Manipulation Language

在数据库表中更新，增加和删除记录

如 update， insert， delete

```
update tableName set age='18' where name='lisi'

insert into tableName value('1','2','3');

drop table tableName //删除表操作
```

6.3、DCL（数据控制问题）

数据控制语言 – Data Control Language

指用于设置用户权限和控制事务语句

如grant，revoke，if...else，while，begintransaction

6.4、DQL（数据查询问题）

数据查询语言 – Data Query Language

select

6.5、小结

```
1、创建数据库：create database  itcast;
2、使用数据库：use itcast;
3、查看当前数据库中的所有表：show tables ;
4、查看所有的数据库：show databases;

5、删除数据库：drop database itcast;
6、删除数据库中的表：drop table t1;
```

7、数据库的备份与恢复

7.1、备份命令

在mysql的安装目录的bin目录下有mysqldump命令，可以完成对数据库的备份。

语法：mysqldump -u 用户名 -p 数据库名> 磁盘SQL文件路径

由于mysqldump命令不是sql命令，需要在dos窗口下使用。

注意：在备份数据的时候，数据库不会被删除。可以手动删除数据库。同时在恢复数据的时候，不会自动的给我们创建数据库，仅仅只会恢复数据库中的表和表中的数据。

```
mysqldump -uroot -p123456 menagerie >/root/data/menagerie.sql

//备份的文件
-rw-r--r--. 1 root root 3118 Oct 20 04:04 menagerie.sql
```

7.2、恢复命令

恢复数据库，需要手动的先创建数据库：

create database heima2;

语法：mysql -u 用户名-p 导入库名< 硬盘SQL文件绝对路径

需求：

1、创建heima8数据库。

2、重新开启一个新的dos窗口。

3、将mydb2备份的数据表和表数据 恢复到mydb6中。

```
//恢复命令
mysql -uroot -p123456 itcast</root/data/menagerie.sql

//恢复校验
```

8、多表查询

8.1、笛卡尔积介绍

笛卡尔乘积是指在数学中，两个集合X和Y的笛卡尔积（Cartesian product），又称直积，表示为X × Y，第一个对象是X的成员而第二个对象是Y的所有可能有序对的其中一个成员

准备数据：

```
create table A(
  A_ID int primary key auto_increment,
  A_NAME varchar(20) not null
);
insert into A values(1,'apple');
insert into A values(2,'orange');
insert into A values(3,'banana');

create table B(
   A_ID int primary key auto_increment,
   B_PRICE double
);
insert into B values(1,2.30);
insert into B values(2,3.50);
insert into B values(4,null);
```

展示效果：

```
mysql> select * from A,B;
+------+--------+------+---------+
| A_ID | A_NAME | A_ID | B_PRICE |
+------+--------+------+---------+
|    1 | apple  |    1 |     2.3 |
|    2 | orange |    1 |     2.3 |
|    3 | banana |    1 |     2.3 |
|    1 | apple  |    2 |     3.5 |
|    2 | orange |    2 |     3.5 |
|    3 | banana |    2 |     3.5 |
|    1 | apple  |    4 |    NULL |
|    2 | orange |    4 |    NULL |
|    3 | banana |    4 |    NULL |
+------+--------+------+---------+
```

作用：笛卡尔积的数据，对程序是没有意义的，我们需要对笛卡尔积中的数据再次进行过滤。

对于多表查询操作，需要过滤出满足条件的数据，需要把多个表进行连接，连接之后需要加上过滤的条件。

```
mysql> select * from A,B where B.A_ID=1;
+------+--------+------+---------+
| A_ID | A_NAME | A_ID | B_PRICE |
+------+--------+------+---------+
|    1 | apple  |    1 |     2.3 |
|    2 | orange |    1 |     2.3 |
|    3 | banana |    1 |     2.3 |
+------+--------+------+---------+
3 rows in set (0.00 sec)

mysql> select * from A,B where B.A_ID=1 and A.A_ID=1;
+------+--------+------+---------+
| A_ID | A_NAME | A_ID | B_PRICE |
+------+--------+------+---------+
|    1 | apple  |    1 |     2.3 |
+------+--------+------+---------+
```

8.2、内连接

内连接：
语法一：
select 列名 , 列名 .... from 表名1,表名2 where 表名1.列名 = 表名2.列名;

语法二：

select * from 表名1 inner join 表名2 on 条件

```
mysql> select * from A inner join B on A.A_ID=B.A_ID;
+------+--------+------+---------+
| A_ID | A_NAME | A_ID | B_PRICE |
+------+--------+------+---------+
|    1 | apple  |    1 |     2.3 |
|    2 | orange |    2 |     3.5 |
+------+--------+------+---------+
```

8.3、左外连接

外链接：左外连接、右外连接、全连接、自连接。

左外连接：用左边表去右边表中查询对应记录，不管是否找到，都将显示左边表中全部记录。

即：虽然右表没有香蕉对应的价格，也要把他查询出来。

语法：select * from 表1 left outer join 表2 on 条件;

```
mysql> select * from A left join B on A.A_ID=B.A_ID;
+------+--------+------+---------+
| A_ID | A_NAME | A_ID | B_PRICE |
+------+--------+------+---------+
|    1 | apple  |    1 |     2.3 |
|    2 | orange |    2 |     3.5 |
|    3 | banana | NULL |    NULL |
+------+--------+------+---------+
```

8.4、右外连接

用右边表去左边表查询对应记录，不管是否找到，右边表全部记录都将显示。

即：不管左方能够找到右方价格对应的水果，都要把左方的价格显示出来。

语法：select * from 表1 right outer join 表2 on 条件;

```
mysql> select * from A right join B on A.A_ID=B.A_ID;
+------+--------+------+---------+
| A_ID | A_NAME | A_ID | B_PRICE |
+------+--------+------+---------+
|    1 | apple  |    1 |     2.3 |
|    2 | orange |    2 |     3.5 |
| NULL | NULL   |    4 |    NULL |
+------+--------+------+---------+
```

8.5、全外连接

全外连接：左外连接和右外连接的结果合并，单会去掉重复的记录。
select * from 表1 full outer join 表2 on 条件
select * from a full outer join b on a.A_ID = b.A_ID; 但是mysql数据库不支持此语法。

8.6、关联子查询

子查询：把一个sql的查询结果作为另外一个查询的参数存在。

8.6.1、in和exist关键词的用法

关联子查询其他的关键字使用：

回忆：age=23 or age=24 等价于 age in (23,24)

in 表示条件应该是在多个列值中。

in：使用在where后面，经常表示是一个列表中的数据，只要被查询的数据在这个列表中存在即可。

```
mysql> select * from A where A_ID in(1,2,3);
+------+--------+
| A_ID | A_NAME |
+------+--------+
|    1 | apple  |
|    2 | orange |
|    3 | banana |
+------+--------+
3 rows in set (0.00 sec)

mysql> select * from A where A_ID =1 or A_ID =2 or A_ID =3;
+------+--------+
| A_ID | A_NAME |
+------+--------+
|    1 | apple  |
|    2 | orange |
|    3 | banana |
+------+--------+
//not in
mysql> select * from A where A_ID not in (1,2,3,4);
Empty set (0.00 sec)

mysql> select * from A where A_ID not in (3,4);
+------+--------+
| A_ID | A_NAME |
+------+--------+
|    1 | apple  |
|    2 | orange |
+------+--------+
2 rows in set (0.00 sec)
```

exists：

exists：表示存在，当子查询的结果存在，就会显示主查询中的所有数据。

使用exists完成：

```
mysql> select * from A where exists(select A_ID from B);
+------+--------+
| A_ID | A_NAME |
+------+--------+
|    1 | apple  |
|    2 | orange |
|    3 | banana |
+------+--------+

mysql> select * from A where not exists(select A_ID from B);
Empty set (0.00 sec)
```

8.6.2、union 和union all使用法

UNION 语句：用于将不同表中相同列中查询的数据展示出来；（不包括重复数据）

UNION ALL 语句：用于将不同表中相同列中查询的数据展示出来；（包括重复数据）

```
mysql> select * from A  union   select * from B;
+------+--------+
| A_ID | A_NAME |
+------+--------+
|    1 | apple  |
|    2 | orange |
|    3 | banana |
|    1 | 2.3    |
|    2 | 3.5    |
|    4 | NULL   |
+------+--------+
6 rows in set (0.00 sec)

mysql> select * from A  union all  select * from B;
+------+--------+
| A_ID | A_NAME |
+------+--------+
|    1 | apple  |
|    2 | orange |
|    3 | banana |
|    1 | 2.3    |
|    2 | 3.5    |
|    4 | NULL   |
+------+--------+
```

8.6.3、case when 语句

case when 语句语法结构：

```
CASE sex
WHEN '1' THEN '男'
WHEN '2' THEN '女'
ELSE '其他' END
```

准备数据

```
//创建表
create table employee(
empid                   int   ,
deptid                  int   ,
sex                     varchar(20)  ,
salary                  double
);

//加载数据
1       10      female  5500.0
2       10      male    4500.0
3       20      female  1900.0
4       20      male    4800.0
5       40      female  6500.0
6       40      female  14500.0
7       40      male    44500.0
8       50      male    6500.0
9       50      male    7500.0

load data local infile '/root/data/emp.txt' into table employee ;
```

```
select *,
case
when salary < 5000 then "低等收入"
when salary>= 5000 and salary < 10000 then "中等收入"
when salary > 10000 then "高等收入"
end  as level,
case sex
when "female" then 1
when "male" then 0
end as flag
from employee;
```

9、MySQL 数据类型

MySQL中定义数据字段的类型对你数据库的优化是非常重要的。

MySQL支持多种类型，大致可以分为三类：数值、日期/时间和字符串(字符)类型。

9.1、数值类型

MySQL支持所有标准SQL数值数据类型。

这些类型包括严格数值数据类型(INTEGER、SMALLINT、DECIMAL和NUMERIC)，以及近似数值数据类型(FLOAT、REAL和DOUBLE PRECISION)。

关键字INT是INTEGER的同义词，关键字DEC是DECIMAL的同义词。

BIT数据类型保存位字段值，并且支持MyISAM、MEMORY、InnoDB和BDB表。

作为SQL标准的扩展，MySQL也支持整数类型TINYINT、MEDIUMINT和BIGINT。下面的表显示了需要的每个整数类型的存储和范围。

类型	大小	范围（有符号）	范围（无符号）	用途
TINYINT	1 字节	(-128，127)	(0，255)	小整数值
SMALLINT	2 字节	(-32 768，32 767)	(0，65 535)	大整数值
MEDIUMINT	3 字节	(-8 388 608，8 388 607)	(0，16 777 215)	大整数值
INT或INTEGER	4 字节	(-2 147 483 648，2 147 483 647)	(0，4 294 967 295)	大整数值
BIGINT	8 字节	(-9 233 372 036 854 775 808，9 223 372 036 854 775 807)	(0，18 446 744 073 709 551 615)	极大整数值
FLOAT	4 字节	(-3.402 823 466 E+38，-1.175 494 351 E-38)，0，(1.175 494 351 E-38，3.402 823 466 351 E+38)	0，(1.175 494 351 E-38，3.402 823 466 E+38)	单精度浮点数值
DOUBLE	8 字节	(-1.797 693 134 862 315 7 E+308，-2.225 073 858 507 201 4 E-308)，0，(2.225 073 858 507 201 4 E-308，1.797 693 134 862 315 7 E+308)	0，(2.225 073 858 507 201 4 E-308，1.797 693 134 862 315 7 E+308)	双精度浮点数值
DECIMAL	对DECIMAL(M,D)，如果M>D，为M+2否则为D+2	依赖于M和D的值	依赖于M和D的值	小数值

9.2、日期和时间类型

表示时间值的日期和时间类型为DATETIME、DATE、TIMESTAMP、TIME和YEAR。

每个时间类型有一个有效值范围和一个"零"值，当指定不合法的MySQL不能表示的值时使用"零"值。

TIMESTAMP类型有专有的自动更新特性，将在后面描述。

类型	大小(字节)	范围	格式	用途
DATE	3	1000-01-01/9999-12-31	YYYY-MM-DD	日期值
TIME	3	'-838:59:59'/'838:59:59'	HH:MM:SS	时间值或持续时间
YEAR	1	1901/2155	YYYY	年份值
DATETIME	8	1000-01-01 00:00:00/9999-12-31 23:59:59	YYYY-MM-DD HH:MM:SS	混合日期和时间值
TIMESTAMP	4	1970-01-01 00:00:00/2038结束时间是第 **2147483647** 秒，北京时间 **2038-1-19 11:14:07**，格林尼治时间 2038年1月19日 凌晨 03:14:07	YYYYMMDD HHMMSS	混合日期和时间值，时间戳

9.3、字符串类型

字符串类型指CHAR、VARCHAR、BINARY、VARBINARY、BLOB、TEXT、ENUM和SET。该节描述了这些类型如何工作以及如何在查询中使用这些类型。

类型	大小	用途
CHAR	0-255字节	定长字符串
VARCHAR	0-65535 字节	变长字符串
TINYBLOB	0-255字节	不超过 255 个字符的二进制字符串
TINYTEXT	0-255字节	短文本字符串
BLOB	0-65 535字节	二进制形式的长文本数据
TEXT	0-65 535字节	长文本数据
MEDIUMBLOB	0-16 777 215字节	二进制形式的中等长度文本数据
MEDIUMTEXT	0-16 777 215字节	中等长度文本数据
LONGBLOB	0-4 294 967 295字节	二进制形式的极大文本数据
LONGTEXT	0-4 294 967 295字节	极大文本数据

CHAR 和 VARCHAR 类型类似，但它们保存和检索的方式不同。它们的最大长度和是否尾部空格被保留等方面也不同。在存储或检索过程中不进行大小写转换。

BINARY 和 VARBINARY 类似于 CHAR 和 VARCHAR，不同的是它们包含二进制字符串而不要非二进制字符串。也就是说，它们包含字节字符串而不是字符字符串。这说明它们没有字符集，并且排序和比较基于列值字节的数值值。

BLOB 是一个二进制大对象，可以容纳可变数量的数据。有 4 种 BLOB 类型：TINYBLOB、BLOB、MEDIUMBLOB 和 LONGBLOB。它们区别在于可容纳存储范围不同。

有 4 种 TEXT 类型：TINYTEXT、TEXT、MEDIUMTEXT 和 LONGTEXT。对应的这 4 种 BLOB 类型，可存储的最大长度不同，可根据实际情况选择。

MySQL 5.0 以上的版本：

1、一个汉字占多少长度与编码有关：

UTF－8：一个汉字＝3个字节

GBK：一个汉字＝2个字节

2、varchar(n) 表示 n 个字符，无论汉字和英文，Mysql 都能存入 n 个字符，仅是实际字节长度有所区别

3、MySQL 检查长度，可用 SQL 语言来查看：

```
select LENGTH(fieldname) from tablename
```

1、整型

MySQL数据类型	含义（有符号）
tinyint(m)	1个字节 范围(-128~127)
smallint(m)	2个字节 范围(-32768~32767)
mediumint(m)	3个字节 范围(-8388608~8388607)
int(m)	4个字节 范围(-2147483648~2147483647)
bigint(m)	8个字节 范围(+-9.22*10的18次方)

取值范围如果加了 unsigned，则最大值翻倍，如 tinyint unsigned 的取值范围为(0~256)。

int(m) 里的 m 是表示 SELECT 查询结果集中的显示宽度，并不影响实际的取值范围，没有影响到显示的宽度，不知道这个 m 有什么用。

2、浮点型**(float** 和 **double)**

MySQL数据类型	含义
float(m,d)	单精度浮点型 8位精度(4字节) m总个数，d小数位
double(m,d)	双精度浮点型 16位精度(8字节) m总个数，d小数位

设一个字段定义为 float(5,3)，如果插入一个数 123.45678,实际数据库里存的是 123.457，但总个数还以实际为准，即 6 位。

3、定点数

浮点型在数据库中存放的是近似值，而定点类型在数据库中存放的是精确值。

decimal(m,d) 参数 m<65 是总个数，d<30 且 d<m 是小数位。

4、字符串**(char,varchar,_text)**

MySQL数据类型	含义
char(n)	固定长度，最多255个字符
varchar(n)	固定长度，最多65535个字符
tinytext	可变长度，最多255个字符
text	可变长度，最多65535个字符
mediumtext	可变长度，最多2的24次方-1个字符
longtext	可变长度，最多2的32次方-1个字符

char 和 varchar：

- ** 1.char(n) 若存入字符数小于n，则以空格补于其后，查询之时再将空格去掉。所以 char 类型存储

的字符串末尾不能有空格，varchar 不限于此。

- ** 2.char(n) 固定长度，char(4) 不管是存入几个字符，都将占用 4 个字节，varchar 是存入的实际字符数 +1 个字节（n<=255）或2个字节(n>255)，所以 varchar(4),存入 3 个字符将占用 4 个字节。
- ** 3.char 类型的字符串检索速度要比 varchar 类型的快。

varchar 和 text：

- ** 1.varchar 可指定 n，text 不能指定，内部存储 varchar 是存入的实际字符数 +1 个字节（n<=255）或 2 个字节(n>255)，text 是实际字符数 +2 个字节。
- ** 2.text 类型不能有默认值。
- ** 3.varchar 可直接创建索引，text 创建索引要指定前多少个字符。varchar 查询速度快于 text, 在都创建索引的情况下，text 的索引似乎不起作用。

5.二进制数据**(_Blob)**

- ** 1.*BLOB*和text存储方式不同，*TEXT*以文本方式存储，英文存储区分大小写，而Blob是以二进制方式存储，不分大小写。
- ** 2._BLOB存储的数据只能整体读出。
- ** 3.*TEXT*可以指定字符集，BLOB不用指定字符集。

6.日期时间类型

MySQL数据类型	含义
date	日期 '2008-12-2'
time	时间 '12:25:36'
datetime	日期时间 '2008-12-2 22:06:44'
timestamp	自动存储记录修改时间

若定义一个字段为timestamp，这个字段里的时间数据会随其他字段修改的时候自动刷新，所以这个数据类型的字段可以存放这条记录最后被修改的时间。

数据类型的属性

MySQL关键字	含义
NULL	数据列可包含NULL值
NOT NULL	数据列不允许包含NULL值
DEFAULT	默认值
PRIMARY KEY	主键
AUTO_INCREMENT	自动递增，适用于整数类型
UNSIGNED	无符号
CHARACTER SET name	指定一个字符集

10、MySQL GROUP BY 语句

GROUP BY 语句根据一个或多个列对结果集进行分组。

在分组的列上我们可以使用 COUNT, SUM, AVG,等函数。

语法结构：

```
SELECT column_name, function(column_name)
FROM table_name
WHERE column_name operator value
GROUP BY column_name;
```

准备数据：

```
CREATE TABLE `employee_tbl` (
  `id` int(11) NOT NULL,
  `name` char(10) NOT NULL DEFAULT '',
  `date` datetime NOT NULL,
  `singin` tinyint(4) NOT NULL DEFAULT '0' COMMENT '登录次数',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- ----------------------------

INSERT INTO `employee_tbl` VALUES ('1', '小明', '2016-04-22 15:25:33', '1'), ('2', '小王',
'2016-04-20 15:25:47', '3'), ('3', '小丽', '2016-04-19 15:26:02', '2'), ('4', '小王',
'2016-04-07 15:26:14', '4'), ('5', '小明', '2016-04-11 15:26:40', '4'), ('6', '小明',
'2016-04-04 15:26:54', '2');
```

```
mysql> SELECT name, COUNT(*) FROM   employee_tbl GROUP BY name;
+--------+----------+
| name   | COUNT(*) |
+--------+----------+
| 小丽 |        1 |
| 小明 |        3 |
| 小王 |        2 |
+--------+----------+

mysql> select * from employee_tbl;
+----+------+---------------------+--------+
| id | name | date                | singin |
+----+------+---------------------+--------+
|  1 | ??   | 2016-04-22 15:25:33 |      1 |
|  2 | ??   | 2016-04-20 15:25:47 |      3 |
|  3 | ??   | 2016-04-19 15:26:02 |      2 |
|  4 | ??   | 2016-04-07 15:26:14 |      4 |
|  5 | ??   | 2016-04-11 15:26:40 |      4 |
|  6 | ??   | 2016-04-04 15:26:54 |      2 |
+----+------+---------------------+--------+

mysql> select * from employee_tbl group by singin;
+----+------+---------------------+--------+
| id | name | date                | singin |
+----+------+---------------------+--------+
|  1 | ??   | 2016-04-22 15:25:33 |      1 |
|  3 | ??   | 2016-04-19 15:26:02 |      2 |
|  2 | ??   | 2016-04-20 15:25:47 |      3 |
|  4 | ??   | 2016-04-07 15:26:14 |      4 |
+----+------+---------------------+--------+
```

注意：

1、group by 可以实现一个最简单的去重查询，假设想看下有哪些员工，除了用 distinct,还可以用：

```
SELECT name FROM employee_tbl GROUP BY name;
```

返回的结果集就是所有员工的名字。

2、分组后的条件使用 HAVING 来限定，WHERE 是对原始数据进行条件限制。几个关键字的使用顺序为 where 、group by 、having、order by ，例如：

```
SELECT name ,sum(*)  FROM employee_tbl WHERE id<>1 GROUP BY name  HAVING sum(*)>5 ORDER
BY sum(*) DESC;
```

11、MySQL LIKE 子句

我们知道在 MySQL 中使用 SQL SELECT 命令来读取数据， 同时我们可以在 SELECT 语句中使用 WHERE 子句来获取指定的记录。

WHERE 子句中可以使用等号 = 来设定获取数据的条件，如 "company = 'itcast"。

但是有时候我们需要获取 company 字段含有 "it" 字符的所有记录，这时我们就需要在 WHERE 子句中使用 SQL LIKE 子句。

SQL LIKE 子句中使用百分号 %字符来表示任意字符，类似于UNIX或正则表达式中的星号 *。

如果没有使用百分号 %, LIKE 子句与等号 = 的效果是一样的。

语法：

以下是 SQL SELECT 语句使用 LIKE 子句从数据表中读取数据的通用语法：

```
SELECT field1, field2,...fieldN
FROM table_name
WHERE field1 LIKE condition1 [AND [OR]] filed2 = 'somevalue'
```

- 你可以在 WHERE 子句中指定任何条件。
- 你可以在 WHERE 子句中使用LIKE子句。
- 你可以使用LIKE子句代替等号 =。
- LIKE 通常与 % 一同使用，类似于一个元字符的搜索。
- 你可以使用 AND 或者 OR 指定一个或多个条件。
- 你可以在 DELETE 或 UPDATE 命令中使用 WHERE...LIKE 子句来指定条件。

```
mysql> select * from pet where species like '%d%';
+----------+--------+---------+------+------------+------------+
| name     | owner  | species | sex  | birth      | death      |
+----------+--------+---------+------+------------+------------+
| Buffy    | Harold | dog     | f    | 1989-05-13 | NULL       |
| Fang     | Benny  | dog     | m    | 1990-08-27 | NULL       |
| Bowser   | Diane  | dog     | m    | 1989-08-31 | 1995-07-29 |
| Chirpy   | Gwen   | bird    | f    | 1998-09-11 | NULL       |
| Whistler | Gwen   | bird    | NULL | 1997-12-09 | NULL       |
+----------+--------+---------+------+------------+------------+
```

12、MySQL NULL 值处理

我们已经知道 MySQL 使用 SQL SELECT 命令及 WHERE 子句来读取数据表中的数据,但是当提供的查询条件字段为 NULL 时， 该命令可能就无法正常工作。

为了处理这种情况，MySQL提供了三大运算符:

- **IS NULL:** 当列的值是 NULL,此运算符返回 true。
- **IS NOT NULL:** 当列的值不为 NULL, 运算符返回 true。
- **<=>:** 比较操作符（不同于=运算符），当比较的的两个值为 NULL 时返回 true。

关于 NULL 的条件比较运算是比较特殊的。你不能使用 = NULL 或 != NULL 在列中查找 NULL 值 。

在 MySQL 中，NULL 值与任何其它值的比较（即使是 NULL）永远返回 false，即 NULL = NULL 返回 false 。

MySQL 中处理 NULL 使用 IS NULL 和 IS NOT NULL 运算符。

13、MySQL 元数据

你可能想知道MySQL以下三种信息：

- 查询结果信息： SELECT, UPDATE 或 DELETE语句影响的记录数。
- 数据库和数据表的信息： 包含了数据库及数据表的结构信息。
- **MySQL**服务器信息： 包含了数据库服务器的当前状态，版本号等。

在MySQL的命令提示符中，我们可以很容易的获取以上服务器信息。

获取服务器元数据

以下命令语句可以在 MySQL 的命令提示符使用，也可以在脚本中 使用，如PHP脚本。

命令	描述
SELECT VERSION( )	服务器版本信息
SELECT DATABASE( )	当前数据库名 (或者返回空)
SELECT USER( )	当前用户名
SHOW STATUS	服务器状态
SHOW VARIABLES	服务器配置变量

14、MySQL ALTER命令

当我们需要修改数据表名或者修改数据表字段时，就需要使用到MySQL ALTER命令。

14.1、删除、添加或修改表字段

如下命令使用了 ALTER 命令及 DROP 子句来删除以上创建表的 i 字段：

```
mysql> ALTER TABLE testalter_tbl  DROP i;
```

如果数据表中只剩余一个字段则无法使用DROP来删除字段。

MySQL 中使用 ADD 子句来向数据表中添加列，如下实例在表 testalter_tbl 中添加 i 字段，并定义数据类型:

```
mysql> ALTER TABLE testalter_tbl ADD i INT;
```

执行以上命令后，i 字段会自动添加到数据表字段的末尾

14.2、修改字段类型及名称

如果需要修改字段类型及名称, 你可以在ALTER命令中使用 MODIFY 或 CHANGE 子句 。

例如，把字段 c 的类型从 CHAR(1) 改为 CHAR(10)，可以执行以下命令:

```
mysql> ALTER TABLE testalter_tbl MODIFY c CHAR(10);
```

14.3、修改表名

如果需要修改数据表的名称，可以在 ALTER TABLE 语句中使用 RENAME 子句来实现。

尝试以下实例将数据表 testalter_tbl 重命名为 alter_tbl：

```
mysql> ALTER TABLE testalter_tbl RENAME TO alter_tbl;
```

15、MySQL 函数

MySQL 有很多内置的函数，以下列出了这些函数的说明。

15.1、MySQL 字符串函数

函数	描述	实例
ASCII(s)	返回字符串 s 的第一个字符的 ASCII 码。	返回 CustomerName 字段第一个字母的 ASCII 码：`SELECT ASCII(CustomerName) AS NumCodeOfFirstCharFROM Customers;`
CHAR_LENGTH(s)	返回字符串 s 的字符数	返回字符串 itcast 的字符数`SELECT CHAR_LENGTH("itcast") AS LengthOfString;`
CHARACTER_LENGTH(s)	返回字符串 s 的字符数	返回字符串 itcast 的字符数`SELECT CHARACTER_LENGTH("itcast") AS LengthOfString;`
CONCAT(s1,s2...sn)	字符串 s1,s2 等多个字符串合并为一个字符串	合并多个字符串`SELECT CONCAT("SQL ", "itcast ", "Gooogle ", "Facebook") AS ConcatenatedString;`
CONCAT_WS(x, s1,s2...sn)	同 CONCAT(s1,s2,...) 函数，但是每个字符串直接要加上 x，x 可以是分隔符	合并多个字符串，并添加分隔符：`SELECT CONCAT_WS("-", "SQL", "Tutorial", "is", "fun!")AS ConcatenatedString;`
FIELD(s,s1,s2...)	返回第一个字符串 s 在字符串列表(s1,s2...)中的位置	返回字符串 c 在列表值中的位置：`SELECT FIELD("c", "a", "b", "c", "d", "e");`
FIND_IN_SET(s1,s2)	返回在字符串s2中与s1匹配的字符串的位置	返回字符串 c 在指定字符串中的位置：`SELECT FIND_IN_SET("c", "a,b,c,d,e");`
FORMAT(x,n)	函数可以将数字 x 进行格式化 "#,###.##", 将 x 保留到小数点后 n 位，最后一位四舍五入。	格式化数字 "#,###.##" 形式：`SELECT FORMAT(250500.5634, 2); -- 输出 250,500.56`
INSERT(s1,x,len,s2)	字符串 s2 替换 s1 的 x 位置开始长度为 len 的字符串	从字符串第一个位置开始的 6 个字符替换为 itcast：`SELECT INSERT("google.com", 1, 6, "runnob");  -- 输出：itcast.com`
LOCATE(s1,s)	从字符串 s 中获取 s1 的开始位置	获取 b 在字符串 abc 中的位置：`SELECT INSTR('abc','b') -- 2`
LCASE(s)	将字符串 s 的所有字母变成小写字母	字符串 itcast 转换为小写：`SELECT LOWER('itcast') -- itcast`
LEFT(s,n)	返回字符串 s 的前 n 个字符	返回字符串 itcast 中的前两个字符：`SELECT LEFT('itcast',2) -- it`
LEFT(s,n)	返回字符串 s 的前 n 个字符	返回字符串 abcde 的前两个字符：`SELECT LEFT('abcde',2) -- ab`
LOCATE(s1,s)	从字符串 s 中获取 s1 的开始位置	返回字符串 abc 中 b 的位置：`SELECT LOCATE('b', 'abc') -- 2`
LOWER(s)	将字符串 s 的所有字母变成小写字母	字符串 itcast 转换为小写：`SELECT LOWER('itcast') -- itcast`
LPAD(s1,len,s2)	在字符串 s1 的开始处填充字符串 s2，使字符串长度达到 len	将字符串 xx 填充到 abc 字符串的开始处：`SELECT LPAD('abc',5,'xx') -- xxabc`
LTRIM(s)	去掉字符串 s 开始处的空格	去掉字符串 itcast开始处的空格：`SELECT LTRIM("    itcast") AS LeftTrimmedString;-- itcast`

MID(s,n,len)	从字符串 s 的 start 位置截取长度为 length 的子字符串，同 SUBSTRING(s,n,len)	从字符串 itcast 中的第 2 个位置截取 3个 字符：`SELECT MID("itcast", 2, 3) AS ExtractString; -- UNO`
POSITION(s1 IN s)	从字符串 s 中获取 s1 的开始位置	返回字符串 abc 中 b 的位置：`SELECT POSITION('b' in 'abc') -- 2`
REPEAT(s,n)	将字符串 s 重复 n 次	将字符串 itcast 重复三次：`SELECT REPEAT('itcast',3) -- itcastitcastitcast`
REPLACE(s,s1,s2)	将字符串 s2 替代字符串 s 中的字符串 s1	将字符串 abc 中的字符 a 替换为字符 x：`SELECT REPLACE('abc','a','x') --xbc`
REVERSE(s)	将字符串s的顺序反过来	将字符串 abc 的顺序反过来：`SELECT REVERSE('abc') -- cba`
RIGHT(s,n)	返回字符串 s 的后 n 个字符	返回字符串 itcast 的后两个字符：`SELECT RIGHT('itcast',2) -- ob`
RPAD(s1,len,s2)	在字符串 s1 的结尾处添加字符串 s1，使字符串的长度达到 len	将字符串 xx 填充到 abc 字符串的结尾处：`SELECT RPAD('abc',5,'xx') -- abcxx`
RTRIM(s)	去掉字符串 s 结尾处的空格	去掉字符串 itcast 的末尾空格：`SELECT RTRIM("itcast  ") AS RightTrimmedString;  -- itcast`
SPACE(n)	返回 n 个空格	返回 10 个空格：`SELECT SPACE(10);`
STRCMP(s1,s2)	比较字符串 s1 和 s2，如果 s1 与 s2 相等返回 0 ，如果 s1>s2 返回 1，如果 s1<s2 返回 -1	比较字符串：`SELECT STRCMP("itcast", "itcast");  -- 0`
SUBSTR(s, start, length)	从字符串 s 的 start 位置截取长度为 length 的子字符串	从字符串 itcast 中的第 2 个位置截取 3个 字符：`SELECT SUBSTR("itcast", 2, 3) AS ExtractString; -- UNO`
SUBSTRING(s, start, length)	从字符串 s 的 start 位置截取长度为 length 的子字符串	从字符串 itcast 中的第 2 个位置截取 3个 字符：`SELECT SUBSTRING("itcast", 2, 3) AS ExtractString; -- UNO`
SUBSTRING_INDEX(s, delimiter, number)	返回从字符串 s 的第 number 个出现的分隔符 delimiter 之后的子串。如果 number 是正数，返回第 number 个字符左边的字符串。如果 number 是负数，返回第(number 的绝对值(从右边数))个字符右边的字符串。	`SELECT SUBSTRING_INDEX('a*b','*',1) -- aSELECT SUBSTRING_INDEX('a*b','*',-1)  -- bSELECT SUBSTRING_INDEX(SUBSTRING_INDEX('a*b*c*d*e','*',3),'*',-1)  -- c`
TRIM(s)	去掉字符串 s 开始和结尾处的空格	去掉字符串 itcast 的首尾空格：`SELECT TRIM('  itcast  ') AS TrimmedString;`
UCASE(s)	将字符串转换为大写	将字符串 itcast 转换为大写：`SELECT UCASE("itcast"); -- itcast`
UPPER(s)	将字符串转换为大写	将字符串 itcast 转换为大写：`SELECT UPPER("itcast"); -- itcast`

15.2、MySQL 数字函数

函数名	描述	实例
ABS(x)	返回 x 的绝对值	返回 -1 的绝对值：`SELECT ABS(-1) -- 返回1`
ACOS(x)	求 x 的反余弦值(参数是弧度)	`SELECT ACOS(0.25);`
ASIN(x)	求反正弦值(参数是弧度)	`SELECT ASIN(0.25);`
ATAN(x)	求反正切值(参数是弧度)	`SELECT ATAN(2.5);`
ATAN2(n, m)	求反正切值(参数是弧度)	`SELECT ATAN2(-0.8, 2);`
AVG(expression)	返回一个表达式的平均值，expression 是一个字段	返回 Products 表中Price 字段的平均值：`SELECT AVG(Price) AS AveragePrice FROM Products;`
CEIL(x)	返回大于或等于 x 的最小整数	`SELECT CEIL(1.5) -- 返回2`
CEILING(x)	返回大于或等于 x 的最小整数	`SELECT CEIL(1.5) -- 返回2`
COS(x)	求余弦值(参数是弧度)	`SELECT COS(2);`
COT(x)	求余切值(参数是弧度)	`SELECT COT(6);`
COUNT(expression)	返回查询的记录总数，expression 参数是一个字段或者 * 号	返回 Products 表中 products 字段总共有多少条记录：`SELECT COUNT(ProductID) AS NumberOfProducts FROM Products;`
DEGREES(x)	将弧度转换为角度	`SELECT DEGREES(3.1415926535898) -- 180`
n DIV m	整除，n 为被除数，m 为除数	计算 10 除于 5：`SELECT 10 DIV 5;  -- 2`
EXP(x)	返回 e 的 x 次方	计算 e 的三次方：`SELECT EXP(3) -- 20.085536923188`
FLOOR(x)	返回小于或等于 x 的最大整数	小于或等于 1.5 的整数：`SELECT FLOOR(1.5) -- 返回1`
GREATEST(expr1, expr2, expr3, ...)	返回列表中的最大值	返回以下数字列表中的最大值：`SELECT GREATEST(3, 12, 34, 8, 25); -- 34` 返回以下字符串列表中的最大值：`SELECT GREATEST("Google", "itcast", "Apple");  -- itcast`
LEAST(expr1, expr2, expr3, ...)	返回列表中的最小值	返回以下数字列表中的最小值：`SELECT LEAST(3, 12, 34, 8, 25); -- 3` 返回以下字符串列表中的最小值：`SELECT LEAST("Google", "itcast", "Apple");  -- Apple`
LN	返回数字的自然对数	返回 2 的自然对数：`SELECT LN(2);  -- 0.6931471805599453`
LOG(x)	返回自然对数(以 e 为底的对数)	`SELECT LOG(20.085536923188) -- 3`
LOG10(x)	返回以 10 为底的对数	`SELECT LOG10(100) -- 2`
LOG2(x)	返回以 2 为底的对数	返回以 2 为底 6 的对数：`SELECT LOG2(6);  -- 2.584962500721156`
		返回数据表 Products 中字段 Price 的最大

MAX(expression)	返回字段 expression 中的最大值	值：`SELECT MAX(Price) AS LargestPrice FROM Products;`
MIN(expression)	返回字段 expression 中的最小值	返回数据表 Products 中字段 Price 的最小值：`SELECT MIN(Price) AS LargestPrice FROM Products;`
MOD(x,y)	返回 x 除以 y 以后的余数	5 除于 2 的余数：`SELECT MOD(5,2) -- 1`
PI()	返回圆周率(3.141593）	`SELECT PI() --3.141593`
POW(x,y)	返回 x 的 y 次方	2 的 3 次方：`SELECT POW(2,3) -- 8`
POWER(x,y)	返回 x 的 y 次方	2 的 3 次方：`SELECT POWER(2,3) -- 8`
RADIANS(x)	将角度转换为弧度	180 度转换为弧度：`SELECT RADIANS(180) -- 3.1415926535898`
RAND()	返回 0 到 1 的随机数	`SELECT RAND() --0.93099315644334`
ROUND(x)	返回离 x 最近的整数	`SELECT ROUND(1.23456) --1`
SIGN(x)	返回 x 的符号，x 是负数、0、正数分别返回 -1、0 和 1	`SELECT SIGN(-10) -- (-1)`
SIN(x)	求正弦值(参数是弧度)	`SELECT SIN(RADIANS(30)) -- 0.5`
SQRT(x)	返回x的平方根	25 的平方根：`SELECT SQRT(25) -- 5`
SUM(expression)	返回指定字段的总和	计算 OrderDetails 表中字段 Quantity 的总和：`SELECT SUM(Quantity) AS TotalItemsOrdered FROM OrderDetails;`
TAN(x)	求正切值(参数是弧度)	`SELECT TAN(1.75);  -- -5.52037992250933`
TRUNCATE(x,y)	返回数值 x 保留到小数点后 y 位的值（与 ROUND 最大的区别是不会进行四舍五入）	`SELECT TRUNCATE(1.23456,3) -- 1.234`

15.3、MySQL 日期函数

函数名	描述
ADDDATE(d,n)	计算其实日期 d 加上 n 天的日期
ADDTIME(t,n)	时间 t 加上 n 秒的时间
CURDATE()	返回当前日期
CURRENT_DATE()	返回当前日期
CURRENT_TIME	返回当前时间
CURRENT_TIMESTAMP()	返回当前日期和时间
CURTIME()	返回当前时间
DATE()	从日期或日期时间表达式中提取日期值
DATEDIFF(d1,d2)	计算日期 d1->d2 之间相隔的天数
DATE_ADD(d，INTERVAL expr type)	计算起始日期 d 加上一个时间段后的日期
DATE_FORMAT(d,f)	按表达式 f的要求显示日期 d
DATE_SUB(date,INTERVAL expr type)	函数从日期减去指定的时间间隔。
DAY(d)	返回日期值 d 的日期部分
DAYNAME(d)	返回日期 d 是星期几，如 Monday,Tuesday
DAYOFMONTH(d)	计算日期 d 是本月的第几天
DAYOFWEEK(d)	日期 d 今天是星期几，1 星期日，2 星期一，以此类推
DAYOFYEAR(d)	计算日期 d 是本年的第几天
EXTRACT(type FROM d)	从日期 d 中获取指定的值，type 指定返回的值。 type可取值为： MICROSECONDSECONDMINUTEHOURDAYWEEKMONTHQUARTERYEARSECOND_MICROSECON
ROM_DAYS(n)	计算从 0000 年 1 月 1 日开始 n 天后的日期
HOUR(t)	返回 t 中的小时值
LAST_DAY(d)	返回给给定日期的那一月份的最后一天

LOCALTIME()	返回当前日期和时间
LOCALTIMESTAMP()	返回当前日期和时间
MAKEDATE(year, day-of-year)	基于给定参数年份 year 和所在年中的天数序号 day-of-year 返回一个日期
MAKETIME(hour, minute, second)	组合时间，参数分别为小时、分钟、秒
MICROSECOND(date)	返回日期参数所对应的毫秒数
MINUTE(t)	返回 t 中的分钟值
MONTHNAME(d)	返回日期当中的月份名称，如 Janyary
MONTH(d)	返回日期d中的月份值，1 到 12
NOW()	返回当前日期和时间
PERIOD_ADD(period, number)	为 年-月 组合日期添加一个时段
PERIOD_DIFF(period1, period2)	返回两个时段之间的月份差值
QUARTER(d)	返回日期d是第几季节，返回 1 到 4
SECOND(t)	返回 t 中的秒钟值
SEC_TO_TIME(s)	将以秒为单位的时间 s 转换为时分秒的格式
STR_TO_DATE(string, format_mask)	将字符串转变为日期
SUBDATE(d,n)	日期 d 减去 n 天后的日期
SUBTIME(t,n)	时间 t 减去 n 秒的时间
SYSDATE()	返回当前日期和时间
TIME(expression)	提取传入表达式的时间部分
TIME_FORMAT(t,f)	按表达式 f 的要求显示时间 t
TIME_TO_SEC(t)	将时间 t 转换为秒
TIMEDIFF(time1, time2)	计算时间差值
TIMESTAMP(expression, interval)	单个参数时，函数返回日期或日期时间表达式；有2个参数时，将参数加和

TO_DAYS(d)	计算日期 d 距离 0000 年 1 月 1 日的天数
WEEK(d)	计算日期 d 是本年的第几个星期，范围是 0 到 53
WEEKDAY(d)	日期 d 是星期几，0 表示星期一，1 表示星期二
WEEKOFYEAR(d)	计算日期 d 是本年的第几个星期，范围是 0 到 53
YEAR(d)	返回年份
YEARWEEK(date, mode)	返回年份及第几周（0到53），mode 中 0 表示周天，1表示周一，以此类推

15.4、MySQL 高级函数

函数名	描述	实例
BIN(x)	返回 x 的二进制编码	15 的 2 进制编码:`SELECT BIN(15); -- 1111`
BINARY(s)	将字符串 s 转换为二进制字符串	`SELECT BINARY "itcast";-> itcast`
`CASE expression WHEN condition1 THEN result1 WHEN condition2 THEN result2 ... WHEN conditionN THEN resultN ELSE resultEND`	CASE 表示函数开始，END 表示函数结束。如果 condition1 成立，则返回 result1, 如果 condition2 成立，则返回 result2，当全部不成立则返回 result，而当有一个成立之后，后面的就不执行了。	`SELECT CASE WHEN 1 > 0 THEN '1 > 0' WHEN 2 > 0 THEN '2 > 0' ELSE '3 > 0' END->1 > 0`
CAST(x AS type)	转换数据类型	字符串日期转换为日期: `SELECT CAST("2017-08-29" AS DATE);-> 2017-08-29`
COALESCE(expr1, expr2, ..., expr_n)	返回参数中的第一个非空表达式（从左向右）	`SELECT COALESCE(NULL, NULL, NULL, 'itcast.com', NULL, 'google.com');-> itcast.com`
CONNECTION_ID()	返回服务器的连接数	`SELECT CONNECTION_ID();-> 4292835`
CONV(x,f1,f2)	返回 f1 进制数变成 f2 进制数	`SELECT CONV(15, 10, 2);-> 1111`
CONVERT(s USING cs)	函数将字符串 s 的字符集变成 cs	`SELECT CHARSET('ABC')->utf-8 SELECT CHARSET(CONVERT('ABC' USING gbk))->gbk`
CURRENT_USER()	返回当前用户	`SELECT CURRENT_USER();-> guest@%`
DATABASE()	返回当前数据库名	`SELECT DATABASE(); -> itcast`
IF(expr,v1,v2)	如果表达式 expr 成立，返回结果 v1；否则，返回结果 v2。	`SELECT IF(1 > 0,'正确','错误') ->正确`
IFNULL(v1,v2)	如果 v1 的值不为 NULL，则返回 v1，否则返回 v2。	`SELECT IFNULL(null,'Hello Word')->Hello Word`
ISNULL(expression)	判断表达式是否为空	`SELECT ISNULL(NULL);->1`
LAST_INSERT_ID()	返回最近生成的 AUTO_INCREMENT 值	`SELECT LAST_INSERT_ID();->6`
NULLIF(expr1, expr2)	比较两个字符串，如果字符串 expr1 与 expr2 相等 返回 NULL，否则返回 expr1	`SELECT NULLIF(25, 25);->`
		`SELECT`

SESSION_USER()	返回当前用户	`SESSION_USER();-> guest@%`
SYSTEM_USER()	返回当前用户	`SELECT SYSTEM_USER();-> guest@%`
USER()	返回当前用户	`SELECT USER();-> guest@%`
VERSION()	返回数据库的版本号	`SELECT VERSION()-> 5.6.34`

16、MySQL 索引

MySQL索引的建立对于MySQL的高效运行是很重要的，索引可以大大提高MySQL的检索速度。

索引分单列索引和组合索引。单列索引，即一个索引只包含单个列，一个表可以有多个单列索引，但这不是组合索引。组合索引，即一个索引包含多个列。

创建索引时，你需要确保该索引是应用在 SQL 查询语句的条件(一般作为 WHERE 子句的条件)。

实际上，索引也是一张表，该表保存了主键与索引字段，并指向实体表的记录。

上面都在说使用索引的好处，但过多的使用索引将会造成滥用。因此索引也会有它的缺点：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。

16.1、普通索引

16.1.1、创建索引

这是最基本的索引，它没有任何限制。它有以下几种创建方式：

```
CREATE INDEX indexName ON mytable(username(length));

//创建索引
 create index id on B(A_ID);
```

如果是CHAR，VARCHAR类型，length可以小于字段实际长度；如果是BLOB和TEXT类型，必须指定length。

16.1.2、修改表结构(添加索引)

```
ALTER table tableName ADD INDEX indexName(columnName)
```

16.1.3、创建表的时候直接指定

```
CREATE TABLE mytable(

ID INT NOT NULL,

username VARCHAR(16) NOT NULL,

INDEX [indexName] (username(length))

);
```

16.1.4、删除索引的语法

```
DROP INDEX [indexName] ON mytable;
```

16.2、唯一索引

它与前面的普通索引类似，不同的就是：索引列的值必须唯一，但允许有空值。如果是组合索引，则列值的组合必须唯一。它有以下几种创建方式：

16.2.1、创建索引

```
CREATE UNIQUE INDEX indexName ON mytable(username(length))
```

16.2.2、修改表结构

```
ALTER table mytable ADD UNIQUE [indexName] (username(length))
```

16.2.3、创建表的时候直接指定

```
CREATE TABLE mytable(

ID INT NOT NULL,

username VARCHAR(16) NOT NULL,

UNIQUE [indexName] (username(length))

);
```

16.3、使用ALTER 命令添加和删除索引

有四种方式来添加数据表的索引：

- **ALTER TABLE tbl_name ADD PRIMARY KEY (column_list):** 该语句添加一个主键，这意味着索引值必须是唯一的，且不能为NULL。
- **ALTER TABLE tbl_name ADD UNIQUE index_name (column_list):** 这条语句创建索引的值必须是唯一的（除了NULL外，NULL可能会出现多次）。
- **ALTER TABLE tbl_name ADD INDEX index_name (column_list):** 添加普通索引，索引值可出现多次。
- **ALTER TABLE tbl_name ADD FULLTEXT index_name (column_list):**该语句指定了索引为 FULLTEXT ，用于全文索引。

以下实例为在表中添加索引。

```
mysql> ALTER TABLE testalter_tbl ADD INDEX (c);
```

你还可以在 ALTER 命令中使用 DROP 子句来删除索引。尝试以下实例删除索引:

```
mysql> ALTER TABLE testalter_tbl DROP INDEX c;
```

16.4、使用 ALTER 命令添加和删除主键

主键只能作用于一个列上，添加主键索引时，你需要确保该主键默认不为空（NOT NULL）。实例如下：

```
mysql> ALTER TABLE testalter_tbl MODIFY itcast INT NOT NULL;
mysql> ALTER TABLE testalter_tbl ADD PRIMARY KEY (itcast);
```

你也可以使用 ALTER 命令删除主键：

```
mysql> ALTER TABLE testalter_tbl DROP PRIMARY KEY;
```

删除主键时只需指定PRIMARY KEY，但在删除索引时，你必须知道索引名。

16.5、显示索引信息

你可以使用 SHOW INDEX 命令来列出表中的相关的索引信息。可以通过添加 \G 来格式化输出信息。

尝试以下实例:

```
mysql> SHOW INDEX FROM table_name; \G
```

mysql> show index from B;

```
+-------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
| Table | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment |
+-------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
| B     |          0 | PRIMARY  |            1 | A_ID        | A         |           3 |     NULL | NULL   |      | BTREE      |         |               |
+-------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
```

17、MySQL 事务

MySQL 事务主要用于处理操作量大，复杂度高的数据。比如说，在人员管理系统中，你删除一个人员，你即需要删除人员的基本资料，也要删除和该人员相关的信息，如信箱，文章等等，这样，这些数据库操作语句就构成一个事务！

- 在 MySQL 中只有使用了 Innodb 数据库引擎的数据库或表才支持事务。
- 事务处理可以用来维护数据库的完整性，保证成批的 SQL 语句要么全部执行，要么全部不执行。
- 事务用来管理 insert,update,delete 语句

一般来说，事务是必须满足4个条件（ACID）：：原子性（**A**tomicity，或称不可分割性）、一致性（**C**onsistency）、隔离性（**I**solation，又称独立性）、持久性（**D**urability）。

- 原子性：一个事务（transaction）中的所有操作，要么全部完成，要么全部不完成，在中间某个环节不会结束。事务在执行过程中发生错误，会被回滚（Rollback）到事务开始前的状态，就像这个事务从来没有执行过一样。
- 一致性：在事务开始之前和事务结束以后，数据库的完整性没有被破坏。这表示写入的资料必须完全符合所有的预设规则，这包含资料的精确度、串联性以及后续数据库可以自发性地完成预定的工作。
- 隔离性：数据库允许多个并发事务同时对其数据进行读写和修改的能力，隔离性可以防止多个事务并发执行时由于交叉执行而导致数据的不一致。事务隔离分为不同级别，包括读未提交（Read uncommitted）、读提交（read committed）、可重复读（repeatable read）和串行化（Serializable）。
- 持久性：事务处理结束后，对数据的修改就是永久的，即便系统故障也不会丢失。

17.1、事务控制语句：

- BEGIN或START TRANSACTION；显式地开启一个事务；
- COMMIT；也可以使用COMMIT WORK，不过二者是等价的。COMMIT会提交事务，并使已对数据库进行的所有修改成为永久性的；
- ROLLBACK；有可以使用ROLLBACK WORK，不过二者是等价的。回滚会结束用户的事务，并撤销正在进行的所有未提交的修改；
- SAVEPOINT identifier；SAVEPOINT允许在事务中创建一个保存点，一个事务中可以有多个SAVEPOINT；
- RELEASE SAVEPOINT identifier；删除一个事务的保存点，当没有指定的保存点时，执行该语句会抛出一个异常；
- ROLLBACK TO identifier；把事务回滚到标记点；
- SET TRANSACTION；用来设置事务的隔离级别。InnoDB存储引擎提供事务的隔离级别有READ UNCOMMITTED、READ COMMITTED、REPEATABLE READ和SERIALIZABLE。

17.2、MYSQL 事务处理主要有两种方法：

1、用 BEGIN, ROLLBACK, COMMIT来实现

- **BEGIN** 开始一个事务
- **ROLLBACK** 事务回滚
- **COMMIT** 事务确认

2、直接用 SET 来改变 MySQL 的自动提交模式:

- **SET AUTOCOMMIT=0** 禁止自动提交
- **SET AUTOCOMMIT=1** 开启自动提交