MySQL快速入门

MySQL快速入门

1 什么是数据库？

- 数据库

  数据库（Database）是按照数据结构来组织、存储和管理数据的仓库，每个数据库都有一个或多个不同的API用于创建，访问，管理，搜索和复制所保存的数据。

  我们也可以将数据存储在文件中，但是在文件中读写数据速度相对较慢。所以，现在我们使用关系型数据库管理系统（RDBMS：Relational Database Management System）来存储和管理的大数据量。所谓的关系型数据库，是建立在关系模型基础上的数据库，借助于集合代数等数学概念和方法来处理数据库中的数据。

  RDBMS即关系数据库管理系统(Relational Database Management System)的特点：

1. 数据以表格的形式出现
2. 每行为各种记录名称
3. 每列为记录名称所对应的数据域
4. 许多的行和列组成一张表单
5. 若干的表单组成database

- 常见关系型数据库

数据库	所属公司	特点
Oracle	Oracle	运行稳定，可移植性高，功能齐全，性能超群！适用于大型企业领域。
DB2	IBM	速度快、可靠性好，适于海量数据，恢复性极强。适用于大中型企业领域。
SQL Server	MS	全面，效率高，界面友好，操作容易，但是不跨平台。适用于于中小型企业领域。
MySQL	Oracle	开源，体积小，速度快。适用于于中小型企业领域。

- RDBMS 术语

  在我们开始学习MySQL 数据库前，让我们先了解下RDBMS的一些术语：

```
数据库：#数据库是一些关联表的集合。

数据表：#表是数据的矩阵。在一个数据库中的表看起来像一个简单的电子表格。

列：#一列(数据元素) 包含了相同的数据，例如邮政编码的数据。

行：#一行（=元组，或记录）是一组相关的数据，例如一条用户订阅的数据。

冗余：#存储两倍数据，冗余可以使系统速度更快。

主键：#主键是唯一的。一个数据表中只能包含一个主键。你可以使用主键来查询数据。

外键：#外键用于关联两个表。

复合键：#复合键（组合键）将多个列作为一个索引键，一般用于复合索引。

索引：#使用索引可快速访问数据库表中的特定信息。索引是对数据库表中一列或多列的值进行排序的一种结构。类似于书籍的目录。
```

- MySQL数据库

  MySQL是一个关系型数据库管理系统，由瑞典MySQL AB公司开发，目前属于Oracle公司。MySQL是一种关联数据库管理系统，关联数据库将数据保存在不同的表中，而不是将所有数据放在一个大仓库内，这样就增加了速度并提高了灵活性。

```
1. #Mysql是开源的，所以你不需要支付额外的费用。
2. #Mysql支持大型的数据库。可以处理拥有上千万条记录的大型数据库。
3. #MySQL使用标准的SQL数据语言形式。
4. #Mysql可以允许于多个系统上，并且支持多种语言。这些编程语言包括C、C++、Python、Java、Perl、PHP、Eiffel、Ruby和Tcl等。
5. #MySQL支持大型数据库，支持5000万条记录的数据仓库，32位系统表文件最大可支持4GB，64位系统支持最大的表文件为8TB。
6. #Mysql是可以定制的，采用了GPL协议，你可以修改源码来开发自己的Mysql系统。
```

2 MySQL 安装

所有平台的Mysql下载地址为： MySQL 下载. 挑选你需要的 *MySQL Community Server* 版本及对应的平台。

2.1 ubuntu上安装MySQL

```
sudo apt-get install mysql-server
sudo apt-get install mysql-client
sudo apt-get install libmysqlclient-dev
```

- 查看mysql是否启动

```
where@ubuntu:~$ ps -aux |grep mysqld
mysql    19069  0.2  7.2 1115908 147304 ?      Ssl  23:19   0:00 /usr/sbin/mysqld
where    19342  0.0  0.0  15984   980 pts/4    S+   23:22   0:00 grep --color=auto mysqld
where@ubuntu:~$ sudo netstat -anp |grep 3306
tcp        0      0 127.0.0.1:3306          0.0.0.0:*               LISTEN      19069/mysqld
where@ubuntu:~$
```

- 启动、重启、关闭mysql服务器

```
sudo /etc/init.d/mysqld start
sudo /etc/init.d/mysqld restart
sudo /etc/init.d/mysqld stop
```

- 配置文件在

```
 vi /etc/mysql/mysql.conf.d/mysqld.cnf
```

- 主要配置内容如下：

```
[mysqld]
datadir=/var/lib/mysql    #数据库文件存放地址
port=3306                 #监听的端口
[mysqld_safe]
log-error=/var/log/mysqld.log       #错误日志存放地址
pid-file=/var/run/mysqld/mysqld.pid #mysqld的进程ID
```

2.2 Window上安装Mysql

Window上安装Mysql相对来说会较为简单，你只需要载 MySQL 下载中下载window版本的mysql安装包，并解压安装包。

如果报错误2502或者2503，请按如下操作：

```
1、WIN+X键->命令提示符（管理员）；
2、msiexec /package   ‘msi文件路径’
```

双击 setup.exe 文件，接下来你只需要安装默认的配置点击"next"即可. 所需要的工具都在`C:\Program Files\MySQL\MySQL Server 5.7\bin`目录中。

2.3 验证Mysql安装

```
mysqladmin --version
```

2.4 密码修改

```
[root@host] mysqladmin -u用户名 -p旧密码 password 新密码


[root@host] mysqladmin -uroot -p111111 password 123456
```

2.5 MySQL 登录

```
[root@host] mysql -h数据库服务器安装的主机 -P数据库端口 -u账户 -p密码

[root@host] mysql -h127.0.0.1 -P3306 -uroot -p123456

#若连接的数据库服务器在本机上，并且端口是3306。
#则可以简写：
[root@host] mysql -uroot -p123456
#或
[root@host] mysql -u root -p
Enter password:******
```

在登录成功后会出现 mysql> 命令提示窗口，你可以在上面执行任何 SQL 语句。

MySQL默认情况下只运行本地登录，如果使用原厂登录则：

```
mysql> GRANT ALL PRIVILEGES ON *.* TO 'root'@'%' IDENTIFIED BY 'passwd' WITH GRANT OPTION;
mysql> FLUSH PRIVILEGES; /*授权远程访问，并执行。*/
```

设置完成后，远端主机可以通过root用户以及passwd来登录。如果还是不能登录，设置my.ini中的`bind-addr = *`

2.6 命令行提示符的意义

提示符	含义
mysql>	准备好接受新命令。
->	等待多行命令的下一行。
'>	等待下一行，等待以单引号(''')开始的字符串的结束。
">	等待下一行，等待以双引号(""")开始的字符串的结束。
`>	等待下一行，等待以反斜点(``)开始的识别符的结束。
/*>	等待下一行，等待以/*开始的注释的结束。

3 数据库基础

3.1 数据库操作

```
/*查看数据库服务器存在哪些数据库：*/
SHOW  DATABASES;

/*使用指定的数据库：*/
USE database_name;

/*查看指定的数据库中有哪些数据表：*/
SHOW TABLES;
```

```
/*创建指定名称的数据库：*/
CREATE DATABASE database_name;

/*删除数据库：*/
DROP DATABASE database_name;
```

3.2 SQL

3.2.1 What SQL

SQL：结构化查询语言(Structured Query Language)。是关系型数据库标准语言。特点：简单，灵活，功能强大。

3.2.2 SQL分类

分类	语意
DQL 数据查询语言	其语句，也称为"数据检索语句"，用以从表中获得数据，确定数据怎样在应用程序给出。保留字SELECT是DQL（也是所有SQL）用得最多的动词，其他DQL常用的保留字有WHERE，ORDER BY，GROUP BY和HAVING。这些DQL保留字常与其他类型的SQL语句一起使用。
DML 数据操作语言	其语句包括动词INSERT，UPDATE和DELETE。它们分别用于添加，修改和删除表中的行。也称为动作查询语言。
TPL 事务处理语言	它的语句能确保被DML语句影响的表的所有行及时得以更新。TPL语句包括BEGIN TRANSACTION，COMMIT和ROLLBACK。
DCL 数据控制语言	它的语句通过GRANT或REVOKE获得许可，确定单个用户和用户组对数据库对象的访问。某些RDBMS可用GRANT或REVOKE控制对表单个列的访问。
DDL 数据定义语言	其语句包括动词CREATE和DROP。在数据库中创建新表或删除表（CREAT TABLE 或 DROP TABLE）；为表加入索引等。DDL包括许多与人数据库目录中获得数据有关的保留字。它也是动作查询的一部分。
CCL 指针控制语言	它的语句，像DECLARE CURSOR，FETCH INTO和UPDATE WHERE CURRENT用于对一个或多个表单独行的操作。

3.2.3 书写规则

1.在MySQL数据库中，SQL语句大小写不敏感

2.SQL语句可单行或多行书写

3.在SQL语句中，关键字不能跨多行或缩写

4.为了提高可读性，一般**关键字大写，其他小写**

5.空格和缩进使程序易读

3.3 表

我们说MySQL是一种关系型数据库。关系数据库最重要的概念就是表。表具有固定的列数和任意的行数，在数学上称为"关系"。

二维表是同类实体的各种属性的集合，每个实体对应于表中的一行，在关系中称为元组，相当于通常的一条记录；表中的列表示属性，称为Field，相当于通常记录中的一个数据项，也叫列、字段。

3.3.1 创建表

```
CREATE TABLE 表名(
   列名1    列的类型   [约束],
   列名2    列的类型   [约束],
   ....
   列名N    列的类型   [约束]
);
/*注意:最后一行没有逗号*/
```

若在建表中使用到了数据库的关键字.

比如新建一张订单表:(order),但是order是数据库中的关键字(排序使用)。表名:t_order,若费用使用order这个单词，此时使用反引号(`)括起来,`order`。

3.3.2 查看表结构

```
DESC table_name;
```

3.3.3 查看表的详细定义

```
SHOW CREATE TABLE table_name;
```

3.3.4 删除表

```
DROP TABLE table_name;
```

3.3.5 示例

```
mysql> CREATE TABLE stu(num INT,sex CHAR,score DOUBLE);
Query OK, 0 rows affected (0.01 sec)

mysql> DESC stu;
+-------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+-------+---------+------+-----+---------+-------+
| num   | int(11) | YES  |     | NULL    |       |
| sex   | char(1) | YES  |     | NULL    |       |
| score | double  | YES  |     | NULL    |       |
+-------+---------+------+-----+---------+-------+
3 rows in set (0.00 sec)

mysql> DROP TABLE stu;
Query OK, 0 rows affected (0.00 sec)
```

3.4 列

3.4.1 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。

1. 数值类型

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

类型	大小	范围（有符号）	范围（无符号）	用途
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,例如DECIMAL(5, 2)范围为-999.99到999.99，占用7个字节，小数点与符号各占用一个字节	依赖于M和D的值	小数值

2. 日期和时间类型

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 00:00:00/2037 年某时	YYYYMMDD HHMMSS	混合日期和时间值，时间戳

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。

```
/*DATE*/
mysql> CREATE TABLE test(id INT,_date DATE);
mysql> INSERT INTO test  VALUES(1,'2008-12-2');

/*TIME*/
mysql> CREATE TABLE test2(id INT,_time TIME);
mysql> INSERT test2 VALUES(2,'11:12:2');


/*TIMESTAMP*/
mysql> CREATE TABLE  test3(id INT,_timestamp TIMESTAMP);
mysql> DESC test3;
+------------+-----------+------+-----+-------------------+-----------------------------+
| Field      | Type      | Null | Key | Default           | Extra                       |
+------------+-----------+------+-----+-------------------+-----------------------------+
| id         | int(11)   | YES  |     | NULL              |                             |
| _timestamp | timestamp | NO   |     | CURRENT_TIMESTAMP | on update CURRENT_TIMESTAMP |
```

```
+------------+-----------+------+-----+-------------------+-----------------------------+

mysql> INSERT INTO test3(id) VALUES(2);
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM test3;
+------+---------------------+
| id   | _timestamp          |
+------+---------------------+
|    2 | 2017-08-06 22:30:42 |
+------+---------------------+
1 row in set (0.00 sec)



mysql> CREATE TABLE  test4(id INT,_datetime DATETIME);
mysql> INSERT INTO test4(id, _datetime) VALUES(1, '1990-02-10');
```

使用函数填充：

```
mysql> SELECT NOW();
+---------------------+
| NOW()               |
+---------------------+
| 2017-08-06 22:43:26 |
+---------------------+
1 row in set (0.00 sec)

mysql> SELECT CURRENT_DATE();
+----------------+
| CURRENT_DATE() |
+----------------+
| 2017-08-06     |
+----------------+
1 row in set (0.00 sec)

mysql> SELECT CURRENT_TIME();
+----------------+
| CURRENT_TIME() |
+----------------+
| 22:43:51       |
+----------------+
1 row in set (0.00 sec)
```

DATE可以通过CURDATE()来赋值当前的日期，

TIME可以通过CURTIME()来赋值当前的时间，

DATETIME与TIMESTAMP都可以通过函数NOW()来赋值当前的时间日期。

3. 字符串类型

类型	大小	用途
CHAR(n)	0-255字节	定长字符串，知道固定长度的时候用CHAR
VARCHAR(n)	0-65535 字节	变长字符串，经常变化的字段用VARCHAR
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，能用varchar的地方不用text
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

超过CHAR和VARCHAR的n设置后，字符串会被截断。

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。

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。它们只是可容纳值的最大长度不同。

有4种TEXT类型：TINYTEXT、TEXT、MEDIUMTEXT和LONGTEXT。这些对应4种BLOB类型，有相同的最大长度和存储需求。

Value	`CHAR(4)`	*Storage Required*	`VARCHAR(4)`	Storage Required
`''`	`'  '`	4 bytes	`''`	1 byte
`'ab'`	`'ab '`	4 bytes	`'ab'`	3 bytes
`'abcd'`	`'abcd'`	4 bytes	`'abcd'`	5 bytes
`'abcdefgh'`	`'abcd'`	4 bytes	`'abcd'`	5 bytes

(1)、char，存定长，速度快，存在空间浪费的可能，会处理尾部空格，上限255。

(2)、varchar，存变长，速度慢，不存在空间浪费，不处理尾部空格，上限65535，但是有存储长度实际65532最大可用。

(3)、text，存变长大数据，速度慢，不存在空间浪费，不处理尾部空格，上限65535，会用额外空间存放数据长度，顾可以全部使用65535。

从官方文档中我们可以得知当varchar大于某些数值的时候，其会自动转换为text，大概规则如下：

- 大于varchar（255）变为 tinytext
- 大于varchar（500）变为 text
- 大于varchar（20000）变为 mediumtext

3.4.2 数据类型的属性约束

MySQL关键字	含义
NULL	数据列可包含NULL值 ，默认不填即为NULL。
NOT NULL	数据列不允许包含NULL值 ，在操作数据库时如果输入该字段的数据为**NULL** ，就会报错。
DEFAULT	默认值，DATE，TIME不能使用函数默认值。DATETIME与TIMESTAMP可以使用NOW()函数默认值。
PRIMARY KEY	主键 ，您可以使用多列来定义主键，列间以逗号分隔。主键不管有没NOT NULL修饰，都不能为NULL，主键值不能重复。主键可以由多个字段组成。例如：PRIMARY KEY (id, name)
AUTO_INCREMENT	定义列为自增的属性，数值会自动加1, 默认初始值为0。一个表只能有一个自增字段，并且该字段必须是主键或者索引。
UNSIGNED	无符号
CHARACTER SET name	指定一个字符集

3.4.3 给表添加列

```
ALTER TABLE table_name ADD column_name datatype
```

例如：

```
mysql> CREATE TABLE stu(num INT, sex CHAR(1), score DOUBLE);
Query OK, 0 rows affected (0.04 sec)
mysql> ALTER TABLE stu ADD addr varchar(50);
Query OK, 0 rows affected
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc stu;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| num   | int(11)     | YES  |     | NULL    |       |
| sex   | char(1)     | YES  |     | NULL    |       |
| score | double      | YES  |     | NULL    |       |
| addr  | varchar(50) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
4 rows in set
```

3.4.4 修改列数据类型

```
ALTER TABLE table_name MODIFY column_name column_type;
```

例如：

```
mysql> ALTER TABLE stu MODIFY sex CHAR(2);
Query OK, 0 rows affected
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESC stu;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| num   | int(11)     | YES  |     | NULL    |       |
| sex   | char(2)     | YES  |     | NULL    |       |
| score | double      | YES  |     | NULL    |       |
| addr  | varchar(50) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
4 rows in set
```

3.4.5 删除列

```
ALTER TABLE table_name DROP column_name;
```

例如：

```
mysql> ALTER TABLE stu DROP addr;
Query OK, 0 rows affected
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESC stu;
+-------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+-------+---------+------+-----+---------+-------+
| num   | int(11) | YES  |     | NULL    |       |
| sex   | char(2) | YES  |     | NULL    |       |
| score | double  | YES  |     | NULL    |       |
+-------+---------+------+-----+---------+-------+
3 rows in set
```

3.5 列约束

3.5.1 列默认值DEFAULT

在未指定默认值的情况下，系统提供default null这样的约束。只有列中提供了default，在插入时，可以省略。

```
mysql> CREATE TABLE stu(id INT , sex char(2));
Query OK, 0 rows affected (0.00 sec)

mysql> DESC stu;
-------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+-------+---------+------+-----+---------+-------+
| id    | int(11) | YES  |     | NULL    |       |
| sex   | char(2) | YES  |     | NULL    |       |
+-------+---------+------+-----+---------+-------+
2 rows in set (0.01 sec)

mysql> SHOW CREATE TABLE stu;
+-------+-------------------------------------------------+
| Table | Create Table
                                |
+-------+-------------------------------------------------+
| stu   | CREATE TABLE `stu` (

  `id` int(11) DEFAULT NULL,
```

```
  `sex` char(2) DEFAULT NULL
) ENGINE=MyISAM DEFAULT CHARSET=latin1 |
+-------+-------------------------------------------------------------------------------------------
-------------------------------+
1 row in set (0.00 sec)
mysql> INSERT INTO stu  values();
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO stu(id,sex)  values();
ERROR 1136 (21S01): Column count doesnot match value count at row 1

mysql> INSERT INTO stu(id,sex)  values(1,'x');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM stu;
+------+------+
| id   | sex  |
+------+------+
| NULL | NULL |
|    1 | x    |
+------+------+
2 rows in set (0.00 sec)
mysql> DROP TABLE stu;
mysql> CREATE TABLE stu(id INT , sex char(2) DEFAULT 'm');
mysql> INSERT INTO stu  values();
```

3.5.2 非空约束NOT NULL

NULL存在的意义在于标志。

NULL类型特征:所有的类型的值都可以是null，包括int、float等数据类型,空字符串是不等于null，0也不等于null。

非空约束用于确保当前列的值不为空值，非空约束只能出现在表对象的列上。

```
mysql> CREATE TABLE stu2 (
  id INT DEFAULT NULL,
  sex CHAR(2) NOT NULL
)
1 row in set (0.00 sec)

mysql> DESC stu2;
+-------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+-------+---------+------+-----+---------+-------+
| id    | int(11) | YES  |     | NULL    |       |
| sex   | char(2) | NO   |     | NULL    |       |
+-------+---------+------+-----+---------+-------+

2 rows in set (0.00 sec)
```

```
mysql> INSERT INTO stu2   VALUES();
ERROR 1364 (HY000): Field 'sex' doesn't have a default value

mysql> INSERT INTO stu2(sex) VALUES('m');
Query OK, 1 row affected (0.00 sec)
mysql> select * from stu2;
+------+-----+
| id   | sex |
+------+-----+
| NULL | m   |
+------+-----+
1 row in set (0.00 sec)
```

3.5.3 唯一约束UNIQUE KEY

唯一约束是指定table的列或列组合不能重复，保证数据的唯一性。虽然唯一约束不允许出现重复的值，但是可以为多个null，同一个**表可以有多个唯一约束**，多个列组合的约束。

在创建唯一约束的时候，如果不给唯一约束名称，就默认和列名相同。 MySQL会给唯一约束的列上默认创建一个唯一索引.

```
mysql> CREATE  TABLE stu3(id INT UNIQUE, sex CHAR(2));
Query OK, 0 rows affected (0.00 sec)

mysql> DESC stu3;
+-------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+-------+---------+------+-----+---------+-------+
| id    | int(11) | YES  | UNI | NULL    |       |
| sex   | char(2) | YES  |     | NULL    |       |
+-------+---------+------+-----+---------+-------+
2 rows in set (0.00 sec)

mysql> INSERT INTO  stu3 VALUES(1,'f');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO  stu3 VALUES(1,'f');
ERROR 1062 (23000): Duplicate entry '1' for key 'id'

mysql> DROP TABLE stu3;
mysql> CREATE TABLE stu3(id INT, sex CHAR(2), UNIQUE(id)); /*第二种写法*/
```

3.5.4 主键约束PRIMARY KEY

主键约束相当于唯一约束+非空约束的组合，主键约束列不允许重复，也不允许出现空值；

如果有多列组合的主键约束，那么这些列都不允许为空值，并且组合的值不允许重复。

每个表最多只允许一个主键，建立主键约束可以在列级别创建，也可以在表级别上创建。

MySQL的主键名总是PRIMARY KEY， 当创建主键约束时，系统默认会在所在的列或列组合上建立对应的唯一索引。

```
mysql> CREATE TABLE stu4(id INT PRIMARY KEY, sex CHAR(2));
Query OK, 0 rows affected (0.00 sec)

mysql> DESC stu4;
+-------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+-------+---------+------+-----+---------+-------+
| id    | int(11) | NO   | PRI | NULL    |       |
| sex   | char(2) | YES  |     | NULL    |       |
+-------+---------+------+-----+---------+-------+
2 rows in set (0.00 sec)

mysql> SHOW CREATE TABLE stu4;
+-------+-----------------------------------------+
| Table | Create Table
                                                  |
+-------+-----------------------------------------+
| stu4  | CREATE TABLE `stu4` (
  `id` int(11) NOT NULL,
  `sex` char(2) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1 |
+-------+-----------------------------------------+
1 row in set (0.00 sec)

mysql> INSERT INTO stu4  values();
ERROR 1364 (HY000): Field 'id' doesn't have a default value

mysql> INSERT INTO stu4(id,sex) VALUES(1,'f');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO stu4(id,sex) VALUES(1,'f');
ERROR 1062 (23000): Duplicate entry '1' for key 'PRIMARY'
mysql>

/*第二种方式创建主键*/
mysql> CREATE TABLE stu4(id INT, sex CHAR(2), PRIMARY KEY(id));
Query OK, 0 rows affected (0.03 sec)
```

3.5.5 自增约束AUTO_INCREMENT

MySQL的中AUTO_INCREMENT类型的属性用于为一个表中记录自动生成ID功能。一个表只能有一个自增字段，并且该字段必须是主键或者索引。

```
mysql> CREATE TABLE stu5(id INT AUTO_INCREMENT,sex CHAR(2));
ERROR 1075 (42000): Incorrect table definition; there can be only one auto column and it must be
defined as a key

mysql> CREATE TABLE stu5(id INT PRIMARY KEY AUTO_INCREMENT ,sex CHAR(2));
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE TABLE stu6(id INT UNIQUE KEY AUTO_INCREMENT ,sex CHAR(2));
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO stu5 values();
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO stu5(id, sex) VALUES(1,'f');
ERROR 1062 (23000): Duplicate entry '1' for key 'PRIMARY'
mysql> INSERT INTO stu5(id, sex) VALUES(2,'f');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO stu5(id, sex) VALUES(3,'f');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO stu5(sex) VALUES('f');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM stu5;
+----+------+
| id | sex  |
+----+------+
|  1 | NULL |
|  2 | f    |
|  3 | f    |
|  4 | f    |
+----+------+
4 rows in set (0.00 sec)

mysql> INSERT INTO stu5(id, sex) VALUES(100,'f');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO stu5(sex) VALUES('f');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM stu5;
+-----+------+
| id  | sex  |
+-----+------+
|   1 | NULL |
|   2 | f    |
|   3 | f    |
|   4 | f    |
| 100 | f    |
| 101 | f    |
+-----+------+
6 rows in set (0.00 sec)
```

3.6 行操作

3.6.1 行增

```
INSERT INTO table_name ( field1, field2,...fieldN )
VALUES
( value1, value2,...valueN ), (value1, value2,...valueN);
```

示例：

```
INSERT INTO product VALUES(21,"罗技M911",4,101,"罗技","罗技",0.6,40);
INSERT INTO
product(product_name,product_type, sale_price,supplier, brand,cutoff,cost_price)
VALUES("罗技M911",4,101,"罗技","罗技",0.6,40);
```

3.6.2 行删

```
delete from <表名> [where <删除条件>]
```

示例：

```
delete  from product  where _id = 21;
delete  from product;
```

注意：若不跟删除条件，则整表被删除。此时等价于truncate table <表名>。

3.6.3 行改

```
UPDATE <表名> SET <列名=更新值> [WHERE <更新条件>]
```

示例：

```
update  product  SET sale_price = 100 WHERE product_name = "罗技M90";

UPDATE  product
set sale_price = 100, product_type = 4, cutoff = 0.9
WHERE product_name = "罗技M90";
```

4 行查询

4.1 先导入数据

```
mysql> CREATE DATABASE testdb;
mysql> USE testdb;
mysql> source  C:\Users\Administrator\Desktop\oracleTest.sql;

mysql> show tables;
+------------------+
| Tables_in_testdb |
+------------------+
| dept             |
| emp              |
| salgrade         |
+------------------+
3 rows in set (0.00 sec)

mysql>
```

一共导入三张表，分别是职员信息表emp，部门信息表dept，薪水等级表salgrade.

4.2 全列查询/投影查询

```
SELECT column_name0,column_name1...
FROM table_name0,table_name1...
[WHERE clause]
[LIMIT N] [OFFSET M ]
```

练习：

```
/*1-查询所有员工信息*/
SELECT   * FROM  emp;
/*2-查询每个员工的编号、姓名、职位*/

/*3-查询部门所有信息*/
```

4.3 消除重复

distinct关键字可以用于一列，也可以用于多列.

```
SELECT distinct column_name0,column_name1...
FROM table_name0,table_name1...
[WHERE clause]
[LIMIT N] [OFFSET M ]
```

练习：

```
/*1-查询公司有哪些岗位*/

/*2-查询所有员工的部门编号*/
```

4.4 算术操作符

对NUMBER型数据可以使用算数操作符创建表达式（+ - * / ）

对DATE型数据可以使用算数操作符创建表达式（+ - ）

练习：

```
/*1-查询所有员工的月薪 */

/*2-查询所有员工的年薪((月薪+奖金)*12 使用别名)*/

/*3-查询每月都有500元的餐补和200元交通补助并且年底多发一个月工资的年薪*/

/*4-演示date类型数据的运算:查询员工的雇佣日期加上10日（+ INTERVAL 10 DAY）*/
```

4.5 空值判断IS NULL/IS NOT NULL

1、空值是指不可用、未分配的值,也就是没有值。

2、空值不等于零或空格

3、任意类型都可以支持空值，也就是说任何类型的字段都可以允许空值作为值的存在

4、空字符串和字符串为null的区别

5、包括空值的任何算术表达式都等于空，使用IFNULL(expr1,expr2)来处理, expr1为NULL就用expr2替代。

练习：

```
/*1-查询有奖金的员工信息 */
mysql> SELECT * FROM emp WHERE comm IS NOT NULL;
/*2-查询公司的老板*/
mysql> SELECT * FROM emp WHERE mgr IS NULL;
/*3-计算员工的年薪（月薪+补贴）*12*/
mysql> SELECT (sal +  ifnull(comm,0))*12 FROM emp;
```

4.6 比较运算符

比较运算符	表达式	示例
等于，不等于，大于，小于	= , != , <> , > , >= , < , <=	WHERE `picsad` >= 1113 and `picsad`<=1122
在两值之间 (包含开始和结尾)	BETWEEN ... AND ...	WHERE `id` between 1113 and 1122
不在两者之间	NOT BETWEEN ... AND ...	WHERE `id` NOT between 1113 and 1122
匹配列出的值	IN (list)	IN('aaa',bbb','ccc','ddd','eee',ffff'');
不在列出的值	NOT IN (list)	NOT IN('aaa',bbb','ccc','ddd','eee',ffff'');
模糊匹配	LIKE	LIKE 'abc%' LIKE '%abc' LIKE '%abc%'

LIKE运算符必须使用通配符才有意义： 匹配单个字符`_` 匹配任意多个字符`%` (0、1 、2 ...)

练习：

```
/* 使用比较运算符
1-要求查询出基本工资高于1500的所有员工信息
SELECT * FROM emp WHERE sal > 1500;
2-查询年薪小于3W的员工

3-查询所有不是销售人员的员工信息

4-查询1981年之后入职的员工信息

5-查询名字叫SCOTT的员工所从事的工作

*/


/*  使用BETWEEN运算符
1-查询工资在2000-3000之间的员工信息

2-查询工资不在2000-3000之间的员工信息

*/


/*  使用IN运算符
1-查询工资为800或1600或3000的员工

2-查询工资不为800或1600或3000的员工

*/
```

```
/*  使用LIKE运算符
1.查询出所有雇员姓名是以A开头的全部雇员信息。

2.查询出雇员姓名第二个字母是M的全部雇员信息。

3.查询出雇员姓名任意位置上包含字母A的全部雇员信息。

*/
```

4.7 逻辑运算符

优先级规则： 比较运算符 > NOT > AND > OR

逻辑运算符	解释
AND	如果组合的条件都是true,返回true.
OR	如果组合的条件 之一是true,返回true.
NOT	如果下面的条件是false,返回true.

练习：

```
/*1，查询姓名中有e或者a的员工姓名*/
SELECT * FROM emp  WHERE ename LIKE '%e%' OR ename LIKE '%a%';
/*2，查询工资在1500~3000之间的全部员工信息*/

/*3，查询出职位是办事员(CLERK)或者是销售人员(SALESMAN)的全部信息,并且工资在1000以上.*/
```

4.8 结果排序

使用ORDER BY 子句将记录排序，ORDER BY 子句出现在SELECT语句的最后，ORDER BY 可以使用别名。ASC: 升序，缺省DESC: 降序。

练习：

```
/*1-查询所有员工信息，按照工资排序*/
SELECT * FROM emp ORDER BY SAL DESC;
/*2-查询所有员工信息，按照年薪降序排序；*/

/*3-查询所有员工信息，按照部门和年薪降序排序；*/
```

5 进阶

5.1 索引

5.1.1 索引介绍

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。

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。

5.1.2 查看索引

```
SHOW INDEX FROM table_name;
```

5.1.3 创建索引

- 自动创建的索引

  当在表上定义一个PRIMARY KEY时，自动创建一个对应的唯一索引。当在表上定义一个外键时，自动创建一个普通索引；

- 手动创建索引

  用户可以创建索引以加速查询，在一列或者多列上创建索引。如果多列在一起，就叫做复合索引；在很多情况下，复合索引比单个索引更好。

  ```
  CREATE INDEX index_name ON table_name (column_name);
  CREATE UNIQUE INDEX index_name ON table_name (column_name);
  ```

5.1.4 删除索引

```
ALTER TABLE table_name DROP INDEX index_name;
```

5.1.5 注意事项

```
哪些值可以创建索引？
1，经常使用的查询条件要创建索引。如果使用like '%'操作，不会使用索引。
2，索引不是越多越好
3，MySQL索引的使用，并不是所有情况下都会使用索引，只有当MySQL认为索引足够能够提升查询性能时才会使用；
```

5.1.6 EXPLAIN

检查查询语句是否使用了索引。

```
EXPLAIN select * from table_name where clause;
```

字段	含义
id	select查询的序列号
select_type	select查询的类型，主要是区别普通查询和联合查询、子查询之类的复杂查询。
table	输出的行所引用的表。
type	联合查询所使用的类型。type显示的是访问类型，是较为重要的一个指标，结果值从好到坏依次是：system > const > eq_ref > ref > fulltext > ref_or_null > index_merge > unique_subquery > index_subquery > range > index > ALL 一般来说，得保证查询至少达到range级别，最好能达到ref。
possible_keys	指出MySQL能使用哪个索引在该表中找到行。如果是空的，没有相关的索引。这时要提高性能，可通过检验WHERE子句，看是否引用某些字段，或者检查字段不是适合索引。
key	显示MySQL实际决定使用的键。如果没有索引被选择，键是NULL。
key_len	显示MySQL决定使用的键长度。如果键是NULL，长度就是NULL。
ref	显示索引的哪一列被使用了，如果可能的话，是一个常数
rows	MYSQL认为必须检查的用来返回请求数据的行数。

5.2 分页

mysql中的分页机制是采用limit实现的。

```
SELECT * FROM table  LIMIT m OFFSET n;
```

练习：

```
SELECT * FROM emp LIMIT 5;
SELECT * FROM emp LIMIT 5 OFFSET 0;
SELECT * FROM emp LIMIT 10 OFFSET 5;
```

5.3 修改表结构

5.3.1 表字段增删改

- 先创建一张表

```
mysql> CREATE TABLE good(proName varchar(30),price float);
Query OK, 0 rows affected (0.01 sec)
```

- 表结构显示

```
mysql> desc good;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| proName | varchar(30) | YES  |     | NULL    |       |
| price   | float       | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

- 修改表名

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

例如：

```
mysql> ALTER TABLE good RENAME TO goods;
Query OK, 0 rows affected (0.01 sec)

mysql>
```

- 添加字段

```
ALTER TABLE table_name ADD column_name column_type;
ALTER TABLE table_name ADD column_name column_type FIRST; /*放在第一位*/
ALTER TABLE table_name ADD column_name column_type AFTER other_column_name; /*放在某字段后面*/
```

例如：

```
mysql> ALTER TABLE  goods  ADD type INT;
Query OK, 0 rows affected (0.00 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESC goods;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| proName | varchar(30) | YES  |     | NULL    |       |
| price   | float       | YES  |     | NULL    |       |
| type    | int(11)     | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

- 删除字段

```
ALTER TABLE table_name DROP column_name;
```

例如：

```
mysql> ALTER TABLE goods DROP type;
Query OK, 0 rows affected (0.00 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESC goods;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| proName | varchar(30) | YES  |     | NULL    |       |
| price   | float       | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

- 修改字段类型

```
ALTER TABLE table_name  MODIFY column_name column_type
```

例如，把字段price 的类型从 float 改为 double，可以执行以下命令:

```
mysql> ALTER TABLE goods MODIFY price DOUBLE;
Query OK, 0 rows affected (0.00 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESC goods;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| proName | varchar(30) | YES  |     | NULL    |       |
| price   | double      | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

- 修改字段名称

```
ALTER TABLE table_name  CHANGE old_column_name new_column_name column_type
```

例如，把proName改为productName:

```
mysql> ALTER TABLE goods CHANGE proName productName VARCHAR(30);
Query OK, 0 rows affected (0.00 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> ALTER TABLE goods CHANGE price prices  float;
Query OK, 0 rows affected (0.00 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESC goods;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| productName | varchar(30) | YES  |     | NULL    |       |
| prices      | float       | YES  |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

5.3.2 表字段修改约束

- 设置与删除DEFAULT

```
ALTER TABLE table_name ALTER column_name SET DEFAULT column_value;
ALTER TABLE table_name ALTER column_name DROP DEFAULT;
```

例如，修改prices的DEFAULT的值为10，然后再删除DEFAULT。

```
mysql> ALTER TABLE goods ALTER prices SET DEFAULT 10;
Query OK, 0 rows affected (0.00 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESC goods;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| productName | varchar(30) | YES  |     | NULL    |       |
| prices      | int(11)     | NO   |     | 10      |       |
+-------------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)

mysql> ALTER TABLE goods ALTER prices DROP DEFAULT;
Query OK, 0 rows affected (0.00 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESC goods;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| productName | varchar(30) | YES  |     | NULL    |       |
| prices      | int(11)     | NO   |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

- 修改NULL/NOT NULL

  当你修改字段时，你可以指定是否为NULL或者是否设置默认值。比如，现在要将prices字段类型设置为INT，置为非空，将默认值为0。

```
mysql> ALTER TABLE goods MODIFY prices INT NOT NULL DEFAULT 0;
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESC goods;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| productName | varchar(30) | YES  |     | NULL    |       |
| prices      | int(11)     | NO   |     | 0       |       |
+-------------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

- 添加删除primary key

```
ALTER TABLE table_name ADD primary key(column_name); /*必须是不存在重复值的列*/
ALTER TABLE table_name DROP primary key;
```

例如：

```
mysql> ALTER TABLE goods ADD id INT NOT NULL DEFAULT 1 FIRST;
Query OK, 0 rows affected (0.07 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> ALTER TABLE goods ADD PRIMARY KEY(id);
Query OK, 0 rows affected (0.00 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESC goods;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| id          | int(11)     | NO   | PRI | 1       |       |
| productName | varchar(30) | YES  |     | NULL    |       |
| price       | int(11)     | NO   |     | 0       |       |
+-------------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)

mysql> ALTER TABLE goods DROP PRIMARY KEY;
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0


mysql> DESC goods;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| id          | int(11)     | NO   |     | 1       |       |
| productName | varchar(30) | YES  |     | NULL    |       |
| price       | int(11)     | NO   |     | 0       |       |
+-------------+-------------+------+-----+---------+-------+
3 rows in set (0.02 sec)
```

- 给主键添加AUTO_INCREMENT

```
ALTER TABLE table_name  CHANGE old_column_name column_name column_type AUTO_INCREMENT;
```

例如：

```
mysql> ALTER TABLE  goods CHANGE id id INT AUTO_INCREMENT;
Query OK, 0 rows affected (0.01 sec)
```

```
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESC goods;
+-------------+-------------+------+-----+---------+----------------+
| Field       | Type        | Null | Key | Default | Extra          |
+-------------+-------------+------+-----+---------+----------------+
| id          | int(11)     | NO   | PRI | NULL    | auto_increment |
| productName | varchar(30) | YES  |     | NULL    |                |
| price       | int(11)     | NO   |     | 0       |                |
+-------------+-------------+------+-----+---------+----------------+
3 rows in set (0.00 sec)

mysql> INSERT INTO goods (productName, price) VALUES ('milk', 3);
Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM goods;
+----+-------------+-------+
| id | productName | price |
+----+-------------+-------+
|  1 | milk        |     3 |
+----+-------------+-------+
1 row in set (0.00 sec)
```

注意：只能给索引添加自动增长约束，而且只能有一个字段能够约束为自增的。

- 设置AUTO_INCREMNET自增初始值

```
ALTER TABLE table_name AUTO_INCREMENT=number;
```

例如：

```
mysql> ALTER TABLE goods AUTO_INCREMENT=10;
Query OK, 0 rows affected (0.00 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> SHOW CREATE TABLE goods;
+-------+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
| Table | Create Table                                                                                                                                                                            |
+-------+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
| goods | CREATE TABLE `goods` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `productName` varchar(30) DEFAULT NULL,
  `price` int(11) NOT NULL DEFAULT '0',
```

```
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=10 DEFAULT CHARSET=utf8 |
+-------+-------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------
--------------------------------------+
1 row in set (0.00 sec)

mysql> INSERT INTO goods (productName, price) VALUES ('milk', 4);
Query OK, 1 row affected (0.02 sec)

mysql> SELECT * FROM goods;
+----+-------------+-------+
| id | productName | price |
+----+-------------+-------+
|  1 | milk        |     3 |
| 10 | milk        |     4 |
+----+-------------+-------+
2 rows in set (0.00 sec)

mysql>
```

- 添加与删除 INDEX

```
ALTER TABLE table_name ADD INDEX  index_name (column_name);
ALTER TABLE table_name ADD UNIQUE unique_index_name(column_name);
ALTER TABLE table _ name DROP INDEX index_name;
```

例如：

```
mysql> ALTER TABLE goods ADD INDEX idx_price(price);
Query OK, 0 rows affected (0.04 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> SHOW INDEX FROM goods;
+-------+------------+----------+--------------+-------------+-----------+-------------+-------
---+--------+------+------------+---------+---------------+
| Table | Non_unique | Key_name  | Seq_in_index | Column_name | Collation | Cardinality |
Sub_part | Packed | Null | Index_type | Comment | Index_comment |
+-------+------------+----------+--------------+-------------+-----------+-------------+-------
---+--------+------+------------+---------+---------------+
| goods |          0 | PRIMARY   |            1 | id          | A         |           1 |
NULL | NULL   |      | BTREE      |         |               |
| goods |          1 | idx_price |            1 | price       | A         |           1 |
NULL | NULL   |      | BTREE      |         |               |
+-------+------------+----------+--------------+-------------+-----------+-------------+-------
---+--------+------+------------+---------+---------------+
2 rows in set (0.00 sec)

mysql> ALTER TABLE goods DROP INDEX idx_price;
Query OK, 0 rows affected (0.02 sec)

Records: 0  Duplicates: 0  Warnings: 0
```

```
mysql> SHOW INDEX FROM goods;
+-------+------------+----------+--------------+-------------+-----------+-------------+--------
--+--------+------+------------+---------+---------------+
| Table | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality |
Sub_part | Packed | Null | Index_type | Comment | Index_comment |
+-------+------------+----------+--------------+-------------+-----------+-------------+--------
--+--------+------+------------+---------+---------------+
| goods |          0 | PRIMARY  |            1 | id          | A         |           1 |
NULL | NULL   |      | BTREE      |         |               |
+-------+------------+----------+--------------+-------------+-----------+-------------+--------
--+--------+------+------------+---------+---------------+
1 row in set (0.00 sec)

mysql>
```

5.4 函数

对函数进行分类，大体上可以分类两类，一类是单行函数，即一进一出，另一类是多行函数，即多进一出。 多行函数，即我们常说的聚合函数。除此之外，均为单行函数。

单行函数	多行函数
字符函数	聚合函数
数学函数	
日期函数	
转换函数	

5.4.1 字符函数

函数	声明与功能
LOWER/UPPER	字符串大小写转化
CONCAT	字符连接
CHAR_LENGTH/LENGTH	求字符串长度
LPAD/RPAD	RPAD(s1,len,s2)
LTRIM/RTRIM/TRIM	去除空格
REPLACE	REPLACE(str,from_str,to_str)
SUBSTRING	获取子串

```
SELECT  LOWER("ABC");
SELECT  UPPER("abc");
SELECT CONCAT("china","is",UPPER("greate"));

SELECT  CHAR_LENGTH("abc");
SELECT LENGTH("china");
SELECT RPAD("abc",6,"xyz"); SELECT LPAD("abc",6,"X");

SELECT  TRIM("  abc   ");
SELECT  LTRIM(" abc   ");
SELECT  RTRIM(" abc   ");

SELECT REPLACE("abc","b","BB");

SELECT  SUBSTRING("abcchina",3);
SELECT  SUBSTRING("abcchina",3,4); SELECT  SUBSTRING("abcchina",-3,4);


/*1-查询出每个员工的姓名长度*/
mysql> SELECT ename,CHAR_LENGTH(ename) NAMELEN FROM emp;

/*2-使用字符"@"替换掉姓名中的所有字母"A"*/
mysql> SELECT REPLACE(ename,'A','@') FROM emp;
```

5.4.2 数学函数

函数	声明与功能
ABS	求绝对值
MOD	求模
FLOOR/CEIL	向下/向上取整
ROUND	四舍五入 round(x,d)
TRUNCATE	按位数截断

```
SELECT  abs(-100);
SELECT  MOD(100,3);

/*都是对浮点型数字进行操作*/
SELECT  FLOOR(1.23);
SELECT  CEIL(1.23);

SELECT  ROUND(3.145);
SELECT  ROUND(3.145,2);  /*保留小数点后几位*/
SELECT  TRUNCATE(3.14,1); /*截取小数点后1位*/
```

5.4.3 日期函数

函数	声明与功能
NOW	当前时间 2017-08-06 22:33:39
YEAR	年
MONTH	月
DAY	日
HOUR	时
MINUTE	分
SECOND	秒
CURRENT_DATE	年/月/日 2017-08-06
CURRENT_TIME	时/分/秒 22:35:01
LAST_DAY	所在年份月份的最后一天
DATE_ADD/DATE_SUB	DATE_ADD(date,INTERVAL expr type)
DATEDIFF	日期差

```
SELECT  NOW();
SELECT  year(NOW());
SELECT  month(NOW());
SELECT  day(NOW());
SELECT  hour(NOW());
SELECT  minute(NOW());
SELECT  second(NOW());

SELECT CURRENT_DATE();
SELECT CURRENT_TIME();

SELECT LAST_DAY(NOW());

SELECT NOW();
SELECT DATE_ADD(NOW(),INTERVAL 2 DAY);
SELECT DATE_SUB(NOW(),INTERVAL 3 HOUR);

SELECT  DATEDIFF(
DATE_ADD(NOW(),INTERVAL 2 DAY),
DATE_SUB(NOW(),INTERVAL 2 HOUR));
```

5.4.4 转换函数

函数	声明与功能
FORMAT	数字到字符串的转化
DATE_FORMAT	DATE_FORMAT(date,format)
STR_TO_DATE	分隔符一致，年月日要一致

```
SELECT FORMAT(423423234.65434453,2);  /*小数点保留2位*/
SELECT FORMAT(423423234.65434453,4);

SELECT NOW();
SELECT DATE_FORMAT(NOW(),'%b %d %Y %h:%i %p');
SELECT DATE_FORMAT(NOW(),'%m-%d-%Y');
SELECT DATE_FORMAT(NOW(),'%d %b %y');
SELECT DATE_FORMAT(NOW(),'%d %b %Y %T');
SELECT DATE_FORMAT(NOW(),'%Y%m%d%H%i%s');

select str_to_date('2008-4-2 15:3:28','%Y-%m-%d %H:%i:%s');
select str_to_date('2008-08-09 08:9:30', '%Y-%m-%d %h:%i:%s');
```

5.4.5 多行(聚合)函数

分组函数在计算时省略列中的空值,不能在where语句中使用分组函数,用COUNT计算行数时，常使用主键。

函数	声明与功能
COUNT()	返回指定列中非NULL值的个数
AVG()	返回指定列的平均值
SUM()	返回指定列的所有值之和
MAX()	返回指定列的最大值
MIN()	返回指定列的最小值

```
/*1查询所有员工每个月支付的平均工资及总工资。*/
mysql> select sum(sal),avg(sal) from emp;
/*2查询月薪在2000以上的员工总人数*/
mysql> select count(*) from emp where sal>2000;
/*3查询员工最高工资和最低工资差距*/
mysql> select max(sal),min(sal) from emp;
mysql> select max(sal)-min(sal) from emp;
```

5.5 分组

```
SELECT [DISTINCT] function(分组字段 [别名])
FROM 表名称 [别名], [表名称 [别名] ,…]
[WHERE 条件(s)]
[GROUP BY 分组字段]
[ORDER BY 排序字段 ASC | DESC [,排序字段 ASC | DESC]];
-----------------------------------------------------
使用GROUP BY子句将表分成小组
组忽略空值,可以使用ifnull函数
结果集隐式按升序排列,如果需要改变排序方式可以使用order by 子句
```

5.5.1 GROUP BY

1出现在SELECT列表中的字段，如果出现的位置不是在聚合函数中，那么必须出现在GROUP BY子句中.

2 在GROUP BY 子句中出现的字段，可以不出现在SELECT列表中 。

例如：

```
/*分组函数单独使用：*/
SELECT COUNT(empno) FROM emp;
/*错误的使用，虽然也能执行成功，但是数据毫无意义：*/
SELECT empno,COUNT(empno) FROM emp;
/*正确做法：查询出所有职业的平均工资以及各职业人数*/
SELECT job,COUNT(empno),AVG(sal) FROM emp GROUP BY job;
```

5.5.2 HAVING

使用HAVING子句对分组的结果进行限制:

```
SELECT [DISTINCT] function(分组字段 别名)

FROM 表名称 [别名], [表名称 [别名] ,…]

[WHERE 条件(s)]

[GROUPBY 分组字段]

[HAVING 分组后的过滤条件（可以使用统计函数）]

[ORDERBY 排序字段 ASC |DESC [,排序字段 ASC |DESC]];
```

注意点：WHERE和HAVING的区别

WHERE：是在执行GROUP BY操作之前进行的过滤，表示从全部数据之中筛选出部分的数据，在WHERE之中不能使用统计函数，无法使用别名；

HAVING：是在GROUP BY分组之后的再次过滤，可以在HAVING子句中使用统计函数，可以使用别名；

适合示例：

```
/*1-按照职位分组，求出每个职位的最高和最低工资*/
mysql> select job,max(sal),min(sal) from emp  group by job;
/*2-查询出每一个部门员工的平均工资*/
mysql> select deptno,avg(sal) from emp group by deptno;
/*3-查询各个部门和岗位的平均工资*/
mysql> select deptno,job,avg(sal) from emp group by deptno,job
/*4-查询平均工资高于2000的部门和其平均工资*/
mysql> select deptno,avg(sal) from emp group by deptno having avg(sal) >2000;
/*可以使用别名*/
select deptno,avg(sal) avg_sal from emp group by deptno having avg_sal >2000;
/*5-查询在各个年份进公司多少人。*/
mysql> select year(hiredate),count(*) from emp group by year(hiredate);
```

5.6 多表查询

```
单表查询:
SELECT <selectList>
FROM  表名
-----------------------------------
多表查询(最简单的):
SELECT <selectList>
FROM  表名A , 表名B
```

- 笛卡尔积

  多表查询,如果没有连接条件,则会产生笛卡尔积，实际运行环境下，应避免使用全笛卡尔集。

  决方案: 在WHERE加入有效的连接条件---->等值连接。注意连接 n张表，至少需要 n-1个连接条件。

例如：

```
/*1-显示所有员工的部门名称*/
mysql> select  e.empno, e.ename, e.job,d.dname ,d.loc from emp e, dept d where e.deptno =
d.deptno;
```

5.7 主键与外键

主键约束(PRIMARY KEY): 约束在当前表中,指定列的值非空且唯一。外键约束(FOREIGN KEY): A表中的外键列的值必须引用于于B表中的某主键列。

为了保证数据的合理性,我们需要建立外键约束关系。

规定: emp表中的deptno列的值,应该来源于dept表中的主键列deptno,我们就把emp表中的deptno列称之为外键列。

```
ALTER TABLE `emp` ADD FOREIGN KEY (`DEPTNO`) REFERENCES `dept` (`DEPTNO`);
```

需要注意，如果建立外键后，要删除dept中数据的时候，需要先删除emp中相关的数据。

5.8 子查询

子查询指的就是在一个查询之中嵌套了其他的若干查询.

在使用select语句查询数据时,有时候会遇到这样的情况，在where查询条件中的限制条件不是一个确定的值，而是一个来自于另一个查询的结果。

子查询一般出现在FROM和WHERE子句中.

```
SELECT  <select_list>
FROM table
WHERE    expr operator (SELECT select_list FROM table );
```

例如：

```
/*1-查询大于公司平均工资的员工姓名*/
SELECT ename,sal FROM emp WHERE sal >(SELECT AVG(sal) FROM emp);

/*2-查询出工资比MARTIN还要高的全部雇员信息*/
SELECT * FROM emp WHERE sal > (SELECT sal FROM emp WHERE ename = 'MARTIN');
```

注意事项

1、子查询在主查询前执行一次

2、主查询使用子查询的结果

3、子查询要用括号括起来

4、将子查询放在比较运算符的右边(增强可读性)

5、对单行子查询使用单行运算符

6、对多行子查询使用多行运算符

5.8.1 子查询返回单行单列

单行单列子查询：只包含一个字段的查询，返回的查询结果也只包含一行数据, 看做是一个值. 使用在WHERE之后。

```
SELECT AVG(sal) FROM emp;
SELECT sal FROM emp WHERE ename = 'MARTIN';
```

5.8.2 子查询返回多行单列

多行单列子查询：只包含了一个字段，但返回的查询结果可能多行或者零行，看做是多个值，使用在WHERE之后。

```
mysql> SELECT sal FROM emp WHERE job='manager';
+------+
| sal  |
+------+
| 2975 |
| 2850 |
| 2450 |
+------+
3 rows in set
```

关键字	含义	应用
IN	与列表中的任意一个值相等	需求:查询工资等于部门经理的员工信息.
ANY	与子查询返回的任意一个值比较	1): = ANY:此时和IN操作符相同。 需求:查询工资等于任意部门经理的员工信息. 2): > ANY:大于子查询中最小的数据. 需求:查询工资大于任意部门经理的员工信息. 3): < ANY:小于子查询中最大的数据。 需求:查询工资小于任意部门经理的员工信息.
ALL	与子查询返回的每一个值比较	1): > ALL:大于子查询中最大的数据. 2): < ALL:小于子查询中最小的数据.

例如：

```
/*查询工资等于部门经理的员工信息.*/
SELECT * FROM emp WHERE sal IN (SELECT sal FROM emp WHERE JOB='manager');
SELECT * FROM emp WHERE sal = ANY (SELECT sal FROM emp WHERE JOB='manager');
/*查询工资大于任意部门经理的员工信息. */
SELECT * FROM emp WHERE sal > ANY (SELECT sal FROM emp WHERE JOB='manager');
/*查询工资小于任意部门经理的员工信息*/
SELECT * FROM emp WHERE sal < ANY (SELECT sal FROM emp WHERE JOB='manager');
/*查询工资大于所有部门经理的员工信息*/
SELECT * FROM emp WHERE sal > ALL (SELECT sal FROM emp WHERE JOB='manager');
/*查询工资小于所有部门经理的员工信息*/
SELECT * FROM emp WHERE sal < ALL (SELECT sal FROM emp WHERE JOB='manager');
```

5.8.3 子查询返回多行多列

多行多列子查询：包含多个字段的返回，查询结构可能是单行或者多行，看做是临时表，使用在FROM之后，临时表需要有别名。

例如：

```
/*查询出部门平均工资大于2000的部门*/

SELECT tmp.* FROM (SELECT deptno,AVG(sal) avg_sal FROM emp GROUP BY deptno) tmp WHERE
tmp.avg_sal > 2000;
```

6 高级

6.1 SQL语句执行顺序

- 书写顺序

  SQL 语句有一个让大部分人都感到困惑的特性，就是：SQL 语句的执行顺序跟其语句的语法顺序并不一致。SQL 语句的语法顺序是：

  ```
  SELECT [DISTINCT] [聚合函数]
  FROM
  WHERE
  GROUP BY
  HAVING
  ORDER BY
  ```

- 执行顺序

  为了方便理解，上面并没有把所有的 SQL 语法结构都列出来，但是已经足以说明 SQL 语句的语法顺序和其执行顺序完全不一样，就以上述语句为例，其执行顺序为：

```
FROM
WHERE
GROUP BY   /*可以使用别名了*/
聚合函数
HAVING
SELECT
DISTINCT
ORDER BY
```

6.2 事务处理(transaction)

在数据库中，所谓事务是指一组逻辑操作单元，使数据从一种状态变换到另一种状态。

为确保数据库中数据的一致性，数据的操纵应当是离散的成组的逻辑单元:当它全部完成时，数据的一致性可以保持，而当这个单元中的一部分操作失败，整个事务应全部视为错误，所有从起始点以后的操作应全部回退到开始状态。

事务的操作:先定义开始一个事务，然后对数据作修改操作，这时如果提交(COMMIT)，这些修改就永久地保存下来，如果回退(ROLLBACK)，数据库管理系统将放弃您所作的所有修改而回到开始事务时的状态。

6.2.1 事务的ACID属性

1. 原子性（Atomicity）原子性是指事务是一个不可分割的工作单位，事务中的操作要么都发生，要么都不发生。
2. 一致性（Consistency）事务必须使数据库从一个一致性状态变换到另外一个一致性状态。(数据不被破坏).
3. 隔离性（Isolation）事务的隔离性是指一个事务的执行不能被其他事务干扰，即一个事务内部的操作及使用的数据对并发的其他事务是隔离的，并发执行的各个事务之间不能互相干扰,每一个事务都存在一个事务空间,彼此不干扰。
4. 持久性（Durability）持久性是指一个事务一旦被提交，它对数据库中数据的改变就是永久性的，接下来的其他操作和数据库故障不应该对其有任何影响.

6.2.2 事务语句

```
begin:开启一个事务,开启一个新的事务空间.
commit:提交事务.
rollback:回滚事务.
```

例如：

```
begin;
select * from goods where id = 1 for update;
update goods set prices = prices - 1 where id = 1;
commit;
```

6.2.3 事务并发问题

数据库的事务并发问题: 存在五种问题:脏读，不可重复读，幻读，第一类丢失更新，第二类丢失更新。

为了解决上述的问题，我们提出了隔离级别的概念，不同的隔离级别可以处理的并发问题是不一样的。

使用不同的隔离级别就可以阻止自己所期望的并发问题。

隔离级别	脏读	不可重复读	幻读	第一类丢失更新	第二类丢失更新
READ UNCOMMITED	√	√	√	×	√
READ COMMITTED	×	√	√	√	√
REPREATABLE READ	×	×	√	√	√
SERIALIZABLE	×	×	×	×	×

√ 表示可能出现的情况

SQL92 推荐使用 REPREATABLE READ 以保证数据的读一致性，不过用户可以根据具体的需求选择适合的事务隔离级别。

默认情况下:MySQL不会出现幻读.
除非:select * from 表名 lock in share mode;
MySQl中锁基于索引机制,也不会出现第一类丢失更新.

- 脏读(dirty read)：

A 事务读取到 B 事务尚未提交的更改数据，并在这个数据的基础上操作。如果恰巧 B 事务回滚，那么 A 事务读到的数据根本是不被承认的。

时间	取款事务 A	转账事务 B
T1		开始事务
T2	开始事务	
T3		查询账户余额为 1000 元
T4		取出 1000 元，修改余额为 0 元
T5	查询账户余额为 0 元（脏读）	
T6		撤销事务，余额恢复为 1000 元。
T7	存入 500 元，把余额修改为 500 元。	
T8	提交事务	

在 Oracle 中不会出现脏读情况。

- 不可重复读（unrepeatable read）：

A 事务读取了 B 事务已经提交的更改数据。

时间	取款事务 A	转账事务 B
T1	开始事务	
T2		开始事务
T3	查询账户余额为 1000 元	
T4		查询账户余额为 1000 元
T5	取出 1000 元，修改余额为 0 元	
T6	提交事务	
T7		查询账户余额为 0 元（和 T4 不一致）

- 幻读（phantom read）：

A 事务读取 B 事务提交的新增数据，这是 A 事务将出现幻读问题。

时间	统计金额事务 A	转账事务 B
T1		开始事务
T2	开始事务	
T3	统计总存款为 10000 元	
T4		新增一个存款账户存款 500 元
T5		提交事务
T6	再次统计，总存款金额为 10500 元（幻读）	

一般，使用表锁机制，防止新增数据。

- 第一类丢失更新：

A 事务撤销时，把已经提交的 B 事务的更新数据覆盖了。（严重）

时间	取款事务 A	转账事务 B
T1	开始事务	
T2		开始事务
T3	查询账户余额为 1000 元	
T4		查询账户余额为 1000 元
T5		存入 500 元，把余额修改为 1500 元
T6		提交事务
T7	取出 200 元，把余额修改为 800 元	
T8	撤销事务	
T9	余额恢复为 1000 元（丢失更新）	

- 第二类丢失更新：

覆盖丢失

A 事务覆盖 B 事务已经提交的数据，造成 B 事务所做操作丢失。

时间	取款事务 A	转账事务 B
T1	开始事务	
T2		开始事务
T3	查询账户余额为 1000 元	
T4		查询账户余额为 1000 元
T5	取出 500 元，把余额修改为 500 元	
T6	提交事务	
T7		存入 500 元
T8		提交事务
T9		余额修改为 1500 元（丢失更新）

6.2.4 锁机制

mysql中使用repeatable read模式，只存在第二类丢失更新，通过加锁的方式可以避免。

```
for update /*加锁  锁释放发生在回滚和提交。*/
```

for update仅适用于InnoDB，且必须在事务块(BEGIN/COMMIT)中才能生效。在进行事务操作时，通过“for update”语句，MySQL会对查询结果集中每行数据都添加排他锁，其他线程对该记录的更新与删除操作都会阻塞。排他锁包含行锁、表锁。

- 不加锁情况

```
begin;
select * from goods where id = 1;
update goods set prices = prices - 1 where id = 1;
commit;
```

- 加锁情况

```
begin;
select * from goods where id = 1 for update;
update goods set prices = prices - 1 where id = 1;
commit;
```

6.3 备份与恢复

- 备份

```
mysqldump -u账户 -p密码 数据库名称>脚本文件存储地址

例如：
mysqldump -uroot -p123456 demoDB> C:/shop_bak.sql
```

- 恢复

```
mysql -u账户 -p密码 数据库名称< 脚本文件存储地址 （首先你要有个已经存在的数据名）

mysql -uroot -p123456 demoDB< C:/shop_bak.sql
```

7 Qt use MySQL

- Qt数据中的管理类为QSqlDataBase，没有继承任何基类，需要在pro文件中添加QT += sql。

代码如下：

```
#include <QSqlDatabase>
#include <QDebug>
#include <QCoreApplication>
int main(int argc, char**argv)
{
    QSqlDatabase db = QSqlDatabase::addDatabase("QMYSQL");
    db.setHostName("127.0.0.1");
    db.setDatabaseName("TESTDB");
    db.setUserName("root");
    db.setPassword("root");
    bool ok = db.open();
    qDebug() << "open db:" << ok << endl;
    db.close();
    return 0;

}
```

这个时候会报错：

```
QSqlDatabase: mysql driver not loaded
QSqlDatabase: available drivers: QSQLITE QMYSQL QMYSQL3 QODBC QODBC3 QPSQL QPSQL7
```

这是因为Qt默认没有安装mysql驱动，需要将libmysql.dll拷贝到Qt的安装路径中，这个libmysql.dll可以到官网 http://dev.mysql.com/downloads/connector/c/ 下载，下载32位还是64位的要根据编译工具来判断，例如32位的MinGW就下载32位的mysql驱动。

将libmysql.dll拷贝到如`C:\Qt\Qt5.7.0\5.7\mingw53_32\bin`目录下。

- SQL语句使用使用QSqlQuery

```
#include <QSqlDatabase>
#include <QDebug>
#include <QSqlQuery>
#include <QSqlError>
int main(int argc, char**argv)
{
    QSqlDatabase db = QSqlDatabase::addDatabase("QMYSQL");
    db.setHostName("127.0.0.1");
    db.setDatabaseName("TESTDB");
    db.setUserName("root");
    db.setPassword("root");
    if(!db.open())
    {
        qDebug() << db.lastError().text() << endl;
    }


    QSqlQuery sql(db);

    if(!sql.exec("SELECT * FROM emp")) //exec返回true代表执行SQL语句成功
    {
        qDebug() << sql.lastError().text() << endl;
    }

    while(sql.next()) //获取下一条记录，如果没有下一条则返回false。
    {
        qDebug() << sql.value("ENAME").toString() << endl
                 << sql.value("JOB").toString() << endl
                   << sql.value("HIREDATE").toString() << endl;
    }

    if(!sql.exec("DELETE FROM emp WHERE ename = \'SMITH\'"))
    {
        qDebug() << sql.lastError().text() << endl;
    }


    if(!sql.exec("UPDATE emp SET ename=\'ALLON\' WHERE ename = \'ALLEN\'"))
```

```
    {
        qDebug() << sql.lastError().text() << endl;
    }


    db.close();
    return 0;
}
```