mysql

一、创建数据库

```
#实例1
mysql>CREATE DATABASE IF NOT EXISTS test_db;
```

```
#实例2：创建 MySQL 数据库时指定字符集和校对规则
mysql> CREATE DATABASE IF NOT EXISTS test_db_char
    -> DEFAULT CHARACTER SET utf8
    -> DEFAULT COLLATE utf8_chinese_ci;
```

二、查看数据库

```
mysql> SHOW DATABASES;
```

```
mysql> SHOW DATABASES LIKE '%test%';
```

三、修改数据库

```
ALTER DATABASE [数据库名] { [ DEFAULT ] CHARACTER SET <字符集名> |
[ DEFAULT ] COLLATE <校对规则名>}
```

```
ALTER DATABASE test_db {  DEFAULT  CHARACTER SET utf8
DEFAULT  COLLATE utf8_chinese_ci}
```

四、删除数据库

```
mysql> DROP DATABASE IF EXISTS test_db_del;
```

五、选择数据库实例

```
mysql> USE test_db;
```

六、mysql存储引擎

功能	MyISAM	MEMORY	InnoDB	Archive
存储限制	256TB	RAM	64TB	None
支持事务	No	No	Yes	No
支持全文索引	Yes	No	No	No
支持树索引	Yes	Yes	Yes	No
支持哈希索引	No	Yes	No	No
支持数据缓存	No	N/A	Yes	No
支持外键	No	No	Yes	No

```
#修改数据库临时的默认存储引擎
SET default_storage_engine=< 存储引擎名 >
```

七、mysql常见数据类型

1) 整数类型

包括 TINYINT、SMALLINT、MEDIUMINT、INT、BIGINT，浮点数类型 FLOAT 和 DOUBLE，定点数类型 DECIMAL。

类型名称	说明	存储需求
TINYINT	很小的整数	1个字节
SMALLINT	小的整数	2个字节
MEDIUMINT	中等大小的整数	3个字节
INT (INTEGHR)	普通大小的整数	4个字节
BIGINT	大整数	8个字节

类型名称	说明	存储需求
TINYINT	-128～127	0 ～255
SMALLINT	-32768～32767	0～65535
MEDIUMINT	-8388608～8388607	0～16777215
INT (INTEGER)	-2147483648～2147483647	0～4294967295
BIGINT	-9223372036854775808～9223372036854775807	0～18446744073709551615

2) 日期/时间类型

包括 YEAR、TIME、DATE、DATETIME 和 TIMESTAMP。

类型名称	日期格式	日期范围	存储需求
YEAR	YYYY	1901 ~ 2155	1 个字节
TIME	HH:MM:SS	-838:59:59 ~ 838:59:59	3 个字节
DATE	YYYY-MM-DD	1000-01-01 ~ 9999-12-3	3 个字节
DATETIME	YYYY-MM-DD HH:MM:SS	1000-01-01 00:00:00 ~ 9999-12-31 23:59:59	8 个字节
TIMESTAMP	YYYY-MM-DD HH:MM:SS	1980-01-01 00:00:01 UTC ~ 2040-01-19 03:14:07 UTC	4 个字节

3) 字符串类型

包括 CHAR、VARCHAR、BINARY、VARBINARY、BLOB、TEXT、ENUM 和 SET 等。

下表中列出了 MySQL 中的字符串数据类型，括号中的`M`表示可以为其指定长度。

类型名称	说明	存储需求
CHAR(M)	固定长度非二进制字符串	M 字节，1<=M<=255
VARCHAR(M)	变长非二进制字符串	L+1字节，在此，L< = M和 1<=M<=255
TINYTEXT	非常小的非二进制字符串	L+1字节，在此，L<2^8
TEXT	小的非二进制字符串	L+2字节，在此，L<2^16
MEDIUMTEXT	中等大小的非二进制字符串	L+3字节，在此，L<2^24
LONGTEXT	大的非二进制字符串	L+4字节，在此，L<2^32
ENUM	枚举类型，只能有一个枚举字符串值	1或2个字节，取决于枚举值的数目 (最大值为65535)
SET	一个设置，字符串对象可以有零个或 多个SET成员	1、2、3、4或8个字节，取决于集合 成员的数量（最多64个成员）

4) 二进制类型

包括 BIT、BINARY、VARBINARY、TINYBLOB、BLOB、MEDIUMBLOB 和 LONGBLOB。

下表中列出了 MySQL 中的二进制数据类型，括号中的`M`表示可以为其指定长度。

类型名称	说明	存储需求
BIT(M)	位字段类型	大约 (M+7)/8 字节
BINARY(M)	固定长度二进制字符串	M 字节
VARBINARY (M)	可变长度二进制字符串	M+1 字节
TINYBLOB (M)	非常小的BLOB	L+1 字节，在此，L<2^8
BLOB (M)	小 BLOB	L+2 字节，在此，L<2^16
MEDIUMBLOB (M)	中等大小的BLOB	L+3 字节，在此，L<2^24
LONGBLOB (M)	非常大的BLOB	L+4 字节，在此，L<2^32

八、创建表

```
CREATE TABLE `tb_emp1` (
  `id` int(11) PRIMARY KEY,
  `name` varchar(25) DEFAULT NULL,
  `deptId` int(11) DEFAULT NULL,
  `salary` float DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8
```

九、修改表

```
--添加字段
mysql> ALTER TABLE tb_emp1
    -> ADD COLUMN col1 INT FIRST;
```

```
--添加字段
mysql> ALTER TABLE tb_emp1
    -> ADD COLUMN col2 INT AFTER name;
```

```
--修改字段数据类型
mysql> ALTER TABLE tb_emp1
    -> MODIFY name VARCHAR(30);
```

```
--删除字段
mysql> ALTER TABLE tb_emp1
    -> DROP col2;
```

```
--修改字段名称
mysql> ALTER TABLE tb_emp1
    -> CHANGE col1 col3 CHAR(30);
```

```
--修改表名
mysql> ALTER TABLE tb_emp1
    -> RENAME TO tb_emp2;
```

十、删除表

```
SHOW TABLES;
DROP TABLE IF EXISTS tb_emp3;
```

十一、主键

- 每个表只能定义一个主键。
- 主键值必须唯一标识表中的每一行，且不能为 NULL，即表中不可能存在两行数据有相同的主键值。这是唯一性原则。
- 一个列名只能在复合主键列表中出现一次。
- 复合主键不能包含不必要的多余列。当把复合主键的某一列删除后，如果剩下的列构成的主键仍然满足唯一性原则，那么这个复合主键是不正确的。这是最小化原则。

```
CREATE TABLE tb_emp3
    -> (
    -> id INT(11) PRIMARY KEY,
    -> name VARCHAR(25),
    -> deptId INT(11),
    -> salary FLOAT
    -> );
```

```
mysql> CREATE TABLE tb_emp4
    -> (
    -> id INT(11),
    -> name VARCHAR(25),
    -> deptId INT(11),
    -> salary FLOAT,
    -> PRIMARY KEY(id)
    -> );
```

```
--复合主键,主键由多个字段联合组成
mysql> CREATE TABLE tb_emp5
    -> (
    -> name VARCHAR(25),
    -> deptId INT(11),
    -> salary FLOAT,
    -> PRIMARY KEY(id,deptId)
    -> );
```

```
DESC tb_emp2;
--添加主键
mysql> ALTER TABLE tb_emp2
    -> ADD PRIMARY KEY(id);
```

十二、外键约束

外键的主要作用是保持数据的一致性、完整性。例如，部门表 tb_dept 的主键是 id，在员工表 tb_emp5 中有一个键 deptId 与这个 id 关联。

- 主表（父表）：对于两个具有关联关系的表而言，相关联字段中主键所在的表就是主表。
- 从表（子表）：对于两个具有关联关系的表而言，相关联字段中外键所在的表就是从表。

定义一个外键时，需要遵守下列规则：

- 父表必须已经存在于数据库中，或者是当前正在创建的表。如果是后一种情况，则父表与子表是同一个表，这样的表称为自参照表，这种结构称为自参照完整性。
- 必须为父表定义主键。
- 主键不能包含空值，但允许在外键中出现空值。也就是说，只要外键的每个非空值出现在指定的主键中，这个外键的内容就是正确的。
- 在父表的表名后面指定列名或列名的组合。这个列或列的组合必须是父表的主键或候选键。
- 外键中列的数目必须和父表的主键中列的数目相同。
- 外键中列的数据类型必须和父表主键中对应列的数据类型相同。

```
--在创建表时设置外键约束
mysql> CREATE TABLE tb_dept1
    -> (
    -> id INT(11) PRIMARY KEY,
    -> name VARCHAR(22) NOT NULL,
    -> location VARCHAR(50)
    -> );

    mysql> CREATE TABLE tb_emp6
    -> (
    -> id INT(11) PRIMARY KEY,
    -> name VARCHAR(25),
    -> deptId INT(11),
    -> salary FLOAT,
    -> CONSTRAINT fk_emp_dept1
    -> FOREIGN KEY(deptId) REFERENCES tb_dept1(id)
    -> );
```

```
--在修改表时添加外键约束
mysql> ALTER TABLE tb_emp2
    -> ADD CONSTRAINT fk_tb_dept1
    -> FOREIGN KEY(deptId)
    -> REFERENCES tb_dept1(id);
```

```
--删除外键约束
mysql> ALTER TABLE tb_emp2
    -> DROP FOREIGN KEY fk_tb_dept1;
```

十三、唯一约束

```
--提示：UNIQUE 和 PRIMARY KEY 的区别：一个表可以有多个字段声明为 UNIQUE，但只能有一个 PRIMARY KEY 声明；声明为 PRIMAY KEY 的列不允许有空值，但是声明为 UNIQUE 的字段允许空值的存在。
mysql> CREATE TABLE tb_dept2
    -> (
    -> id INT(11) PRIMARY KEY,
    -> name VARCHAR(22) UNIQUE,
    -> location VARCHAR(50)
    -> );
```

```
--在修改表时添加唯一约束
mysql> ALTER TABLE tb_dept1
    -> ADD CONSTRAINT unique_name UNIQUE(name);
```

```
--删除唯一约束
mysql> ALTER TABLE tb_dept1
    -> DROP INDEX unique_name;
```

十四、检查约束check

MySQL检查约束（CHECK）可以通过 CREATE TABLE 或 ALTER TABLE 语句实现，根据用户实际的完整性要求来定义。它可以分别对列或表实施 CHECK 约束。

```
--在创建表时设置检查约束
mysql> CREATE TABLE tb_emp7
    -> (
    -> id INT(11) PRIMARY KEY,
    -> name VARCHAR(25),
    -> deptId INT(11),
    -> salary FLOAT,
    -> CHECK(salary>0 AND salary<100),
    -> FOREIGN KEY(deptId) REFERENCES tb_dept1(id)
    -> );
```

```
--在修改表时添加检查约束
mysql> ALTER TABLE tb_emp7
    -> ADD CONSTRAINT check_id
    -> CHECK(id>0);
```

```
--删除检查约束
ALTER TABLE <数据表名> DROP CONSTRAINT <检查约束名>;
```

十五、默认值

```
--在创建表时设置默认值约束
mysql> CREATE TABLE tb_dept3
    -> (
    -> id INT(11) PRIMARY KEY,
    -> name VARCHAR(22),
    -> location VARCHAR(50) DEFAULT 'Beijing'
    -> );
```

```
--在修改表时添加默认值约束
mysql> ALTER TABLE tb_dept3
    -> CHANGE COLUMN location
    -> location VARCHAR(50) DEFAULT 'Shanghai';
```

```
--删除默认值约束
mysql> ALTER TABLE tb_dept3
    -> CHANGE COLUMN location
    -> location VARCHAR(50) DEFAULT NULL;
```

十六、非空约束

非空约束（Not Null Constraint）指字段的值不能为空。对于使用了非空约束的字段，如果用户在添加数据时没有指定值，数据库系统就会报错。

```
--在创建表时设置非空约束
mysql> CREATE TABLE tb_dept4
    -> (
    -> id INT(11) PRIMARY KEY,
    -> name VARCHAR(22) NOT NULL,
    -> location VARCHAR(50)
    -> );
```

```
--在修改表时添加非空约束
mysql> ALTER TABLE tb_dept4
    -> CHANGE COLUMN location
    -> location VARCHAR(50) NOT NULL;
```

```
--删除非空约束
mysql> ALTER TABLE tb_dept4
    -> CHANGE COLUMN location
    -> location VARCHAR(50) NULL;
```

十六、查看表中的约束

在 MySQL中可以使用 SHOW CREATE TABLE 语句来查看表中的约束。

```
SHOW CREATE TABLE tb_emp8
```

十七、数据表查询

```
SELECT
{* | <字段列名>}
[
FROM <表 1>, <表 2>…
[WHERE <表达式>
[GROUP BY <group by definition>
[HAVING <expression> [{<operator> <expression>}…]]
[ORDER BY <order by definition>]
[LIMIT[<offset>,] <row count>]
]
```

其中，各条子句的含义如下：

- `{*|<字段列名>}` 包含星号通配符的字段列表，表示查询的字段，其中字段列至少包含一个字段名称，如果要查询多个字段，多个字段之间要用逗号隔开，最后一个字段后不要加逗号。
- `FROM <表 1>，<表 2>…`，表 1 和表 2 表示查询数据的来源，可以是单个或多个。
- WHERE 子句是可选项，如果选择该项，将限定查询行必须满足的查询条件。
- `GROUP BY< 字段 >`，该子句告诉 MySQL 如何显示查询出来的数据，并按照指定的字段分组。
- `[ORDER BY< 字段 >]`，该子句告诉 MySQL 按什么样的顺序显示查询出来的数据，可以进行的排序有升序（ASC）和降序（DESC）。
- `[LIMIT[<offset>，]<row count>]`，该子句告诉 MySQL 每次显示查询出来的数据条数。

```
--使用“*”查询表中的全部内容
mysql> SELECT * FROM tb_students_info;
```

```
mysql> SELECT id,name,dept_id,age,sex,height,login_date
    -> FROM tb_students_info;
```

十八、过滤重复数据

```
--SELECT DISTINCT <字段名> FROM <表名>;
mysql> SELECT  DISTINCT age
    -> FROM tb_students_info;
```

十九、别名

```
--表别名
mysql> SELECT stu.name,stu.height
    -> FROM tb_students_info AS stu;
```

```
--列别名
mysql> SELECT name AS student_name,
    -> age AS student_age
    -> FROM tb_students_info;
```

二十、LIMIT限制

在使用 MySQL SELECT 语句时往往返回的是所有匹配的行，有些时候我们仅需要返回第一行或者前几行，这时候就需要用到 MySQL LIMT 子句。

LIMIT 接受一个或两个数字参数。参数必须是一个整数常量。如果给定两个参数，第一个参数指定第一个返回记录行的偏移量，第二个参数指定返回记录行的最大数目。

```
mysql> SELECT * FROM tb_students_info LIMIT 4;
+----+-------+---------+------+------+--------+------------+
| id | name  | dept_id | age  | sex  | height | login_date |
+----+-------+---------+------+------+--------+------------+
|  1 | Dany  |       1 |   25 | F    |    160 | 2015-09-10 |
|  2 | Green |       3 |   23 | F    |    158 | 2016-10-22 |
|  3 | Henry |       2 |   23 | M    |    185 | 2015-05-31 |
|  4 | Jane  |       1 |   22 | F    |    162 | 2016-12-20 |
+----+-------+---------+------+------+--------+------------+
```

```
mysql> SELECT * FROM tb_students_info LIMIT 3,5;
+----+-------+---------+------+------+--------+------------+
| id | name  | dept_id | age  | sex  | height | login_date |
+----+-------+---------+------+------+--------+------------+
|  4 | Jane  |       1 |   22 | F    |    162 | 2016-12-20 |
|  5 | Jim   |       1 |   24 | M    |    175 | 2016-01-15 |
|  6 | John  |       2 |   21 | M    |    172 | 2015-11-11 |
|  7 | Lily  |       6 |   22 | F    |    165 | 2016-02-26 |
|  8 | Susan |       4 |   23 | F    |    170 | 2015-10-01 |
+----+-------+---------+------+------+--------+------------+
```

二十一、ORDER BY：对查询结果进行排序

SELECT 语句中，ORDER BY 子句主要用来将结果集中的数据按照一定的顺序进行排序。

使用 ORDER BY 子句应该注意以下几个方面：

- ORDER BY 子句中可以包含子查询。
- 当排序的值中存在空值时，ORDER BY 子句会将该空值作为最小值来对待。
- 当在 ORDER BY 子句中指定多个列进行排序时，MySQL 会按照列的顺序从左到右依次进行排序。
- 查询的数据并没有以一种特定的顺序显示，如果没有对它们进行排序，则将根据插入到数据表中的顺序显示。使用 ORDER BY 子句对指定的列数据进行排序。

```
mysql> SELECT * FROM tb_students_info ORDER BY height;
```

```
--先按 height 排序，再按 name 排序
mysql> SELECT name,height
    -> FROM tb_students_info
    -> ORDER BY height,name;
```

```
--先按 height 降序排序，再按 name 升序排序
mysql> SELECT name,height FROM tb_student_info ORDER BY height DESC,name ASC;
```

二十二、条件查询

在使用 MySQL SELECT语句时，可以使用 WHERE 子句来指定查询条件，从 FROM 子句的中间结果中选取适当的数据行，达到数据过滤的效果。

```
--单一条件的查询语句
mysql> SELECT name,height
    -> FROM tb_students_info
    -> WHERE height=170;

mysql> SELECT name,age
    -> FROM tb_students_info
    -> WHERE age<22;
```

```
--多条件的查询语句
mysql> SELECT * FROM tb_students_info
    -> WHERE age>21 AND height>=175;
```

like查询

利用通配符可以在不完全确定比较值的情形下创建一个比较特定数据的搜索模式，并置于关键字 LIKE 之后。可以在搜索模式的任意位置使用通配符，并且可以使用多个通配符。MySQL 支持的通配符有以下两种：

1）百分号（%）

百分号是 MySQL 中常用的一种通配符，在过滤条件中，百分号可以表示任何字符串，并且该字符串可以出现任意次。

使用百分号通配符要注意以下几点：

- MySQL 默认是不区分大小写的，若要区分大小写，则需要更换字符集的校对规则。
- 百分号不匹配空值。
- 百分号可以代表搜索模式中给定位置的 0 个、1 个或多个字符。
- 尾空格可能会干扰通配符的匹配，一般可以在搜索模式的最后附加一个百分号。

2）下划线（_）

下划线通配符和百分号通配符的用途一样，下画线只匹配单个字符，而不是多个字符，也不是 0 个字符。

> 注意：不要过度使用通配符，对通配符检索的处理一般会比其他检索方式花费更长的时间。

```
--使用 LIKE 的模糊查询
mysql> SELECT name FROM tb_students_info
    -> WHERE name LIKE 'T%';

mysql> SELECT name FROM tb_students_info
    -> WHERE name LIKE '%e%';

mysql> SELECT name FROM tb_students_info
    -> WHERE name LIKE '____y';
```

日期字段作为条件的查询语句

```
mysql> SELECT * FROM tb_students_info
    -> WHERE login_date<'2016-01-01';
```

```
mysql> SELECT * FROM tb_students_info
    -> WHERE login_date
    -> BETWEEN '2015-10-01'
    -> AND '2016-05-01';
```

二十三、常用运算符

算术运算符

算术运算符	说明
+	加法运算
-	减法运算
*	乘法运算
/	除法运算，返回商
%	求余运算，返回余数

比较运算符

比较运算符	说明
=	等于
<	小于
<=	小于等于
>	大于
>=	大于等于
<=>	安全的等于，不会返回 UNKNOWN
<> 或!=	不等于
IS NULL 或 ISNULL	判断一个值是否为 NULL
IS NOT NULL	判断一个值是否不为 NULL
LEAST	当有两个或多个参数时，返回最小值
GREATEST	当有两个或多个参数时，返回最大值
BETWEEN AND	判断一个值是否落在两个值之间
IN	判断一个值是IN列表中的任意一个值
NOT IN	判断一个值不是IN列表中的任意一个值
LIKE	通配符匹配
REGEXP	正则表达式匹配

逻辑运算符

逻辑运算符	说明
NOT 或者 !	逻辑非
AND 或者 &&	逻辑与
OR 或者 ||	逻辑或
XOR	逻辑异或

下面分别介绍不同的逻辑运算符的使用方法。

1) NOT 或者 !

逻辑非运算符 NOT 或者 !，表示当操作数为 0 时，返回值为 1；当操作数为非零值时，返回值为 0；当操作数为 NULL 时，返回值为 NULL。

2) AND 或者 &&

逻辑与运算符 AND 或者 &&，表示当所有操作数均为非零值并且不为 NULL 时，返回值为 1；当一个或多个操作数为 0 时，返回值为 0；其余情况返回值为 NULL。

3) OR 或者 ||

逻辑或运算符 OR 或者 ||，表示当两个操作数均为非 NULL 值且任意一个操作数为非零值时，结果为 1，否则结果为 0；当有一个操作数为 NULL 且另一个操作数为非零值时，结果为 1，否则结果为 NULL；当两个操作数均为 NULL 时，所得结果为 NULL。

4) XOR

逻辑异或运算符 XOR。当任意一个操作数为 NULL 时，返回值为 NULL；对于非 NULL 的操作数，若两个操作数都不是 0 或者都是 0 值，则返回结果为 0；若一个为 0，另一个不为非 0，则返回结果为 1。

位运算符

位运算符	说明
|	按位或
&	按位与
^	按位异或
<<	按位左移
>>	按位右移
~	按位取反，反转所有比特

下面分别介绍不同的位运算符的使用方法。

1) 位或运算符“|”

位或运算的实质是将参与运算的两个数据按对应的二进制数逐位进行逻辑或运算。若对应的二进制位有一个或两个为 1，则该位的运算结果为 1，否则为 0。 **2) 位与运算符“&”**

位与运算的实质是将参与运算的两个数据按对应的二进制数逐位进行逻辑与运算。若对应的二进制位都为 1，则该位的运算结果为 1，否则为 0。

3) 位异或运算符“^”

位异或运算的实质是将参与运算的两个数据按对应的二进制数逐位进行逻辑异或运算。对应的二进制位不同时，对应位的结果才为 1。如果两个对应位都为 0 或者都为 1，则对应位的结果为 0。

4) 位左移运算符“<<”

位左移运算符“<<”使指定的二进制值的所有位都左移指定的位数。左移指定位数之后，左边高位的数值将被移出并丢弃，右边低位空出的位置用 0 补齐。

语法格式为

```
表达式<<n
```

，这里 n 指定值要移位的位数。

5) 位右移运算符“>>”

位右移运算符“>>”使指定的二进制值的所有位都右移指定的位数。右移指定位数之后，右边高位的数值将被移出并丢弃，左边低位空出的位置用 0 补齐。

语法格式为

```
表达式>>n
```

，这里 n 指定值要移位的位数。

6) 位取反运算符“~”

位取反运算符的实质是将参与运算的数据按对应的二进制数逐位反转，即 1 取反后变 0，0 取反后变为 1。

二十四、INNER JOIN：内连接查询

内连接是通过在查询中设置连接条件的方式，来移除查询结果集中某些数据行后的交叉连接。简单来说，就是利用条件表达式来消除交叉连接的某些数据行。

```
mysql> SELECT id,name,age,dept_name
    -> FROM tb_students_info,tb_departments
    -> WHERE tb_students_info.dept_id=tb_departments.dept_id;
```

```
mysql> SELECT id,name,age,dept_name
    -> FROM tb_students_info INNER JOIN tb_departments
    -> WHERE tb_students_info.dept_id=tb_departments.dept_id;
```

提示：使用 WHERE 子句定义连接条件比较简单明了，而 INNER JOIN 语法是 ANSI SQL 的标准规范，使用 INNER JOIN 连接语法能够确保不会忘记连接条件，而且 WHERE 子句在某些时候会影响查询的性能。

二十五、LEFT/RIGHT JOIN：外连接查询

MySQL 中内连接是在交叉连接的结果集上返回满足条件的记录；而外连接先将连接的表分为基表和参考表，再以基表为依据返回满足和不满足条件的记录。外连接更加注重两张表之间的关系。按照连接表的顺序，可以分为左外连接和右外连接。

```
mysql> SELECT name,dept_name
    -> FROM tb_students_info s
    -> LEFT OUTER JOIN tb_departments d
    -> ON s.dept_id = d.dept_id;
```

```
mysql> SELECT name,dept_name
    -> FROM tb_students_info s
    -> RIGHT OUTER JOIN tb_departments d
    -> ON s.dept_id = d.dept_id;
```

二十六、子查询

子查询指一个查询语句嵌套在另一个查询语句内部的查询，这个特性从 MySQL4.1 开始引入，在 SELECT 子句中先计算子查询，子查询结果作为外层另一个查询的过滤条件，查询可以基于一个表或者多个表。子查询中常用的操作符有 ANY（SOME）、ALL、IN 和 EXISTS。

子查询中常用的运算符

```
IN子查询:<表达式> [NOT] IN <子查询>
mysql> SELECT name FROM tb_students_info
    -> WHERE dept_id IN
    -> (SELECT dept_id
    -> FROM tb_departments
    -> WHERE dept_type= 'A' );

    mysql> SELECT name FROM tb_students_info
    -> WHERE dept_id NOT IN
    -> (SELECT dept_id
    -> FROM tb_departments
    -> WHERE dept_type='A');
```

```
比较运算符子查询:<表达式> {= | < | > | >= | <= | <=> | < > | != }
{ ALL | SOME | ANY} <子查询>
 mysql> SELECT name FROM tb_students_info
    -> WHERE dept_id =
    -> (SELECT dept_id
    -> FROM tb_departments
    -> WHERE dept_name='Computer');

 mysql> SELECT name FROM tb_students_info
    -> WHERE dept_id <>
    -> (SELECT dept_id
    -> FROM tb_departments
    -> WHERE dept_name='Computer');
```

```
EXISTS <子查询>
mysql> SELECT * FROM tb_students_info
    -> WHERE EXISTS
    -> (SELECT dept_name
    -> FROM tb_departments
    -> WHERE dept_id=1);
```

二十七、GROUP BY：分组查询

在 MySQL SELECT 语句中，允许使用 GROUP BY 子句，将结果集中的数据行根据选择列的值进行逻辑分组，以便能汇总表内容的子集，实现对每个组而不是对整个结果集进行整合。

语法格式如下：

```
GROUP BY { <列名> | <表达式> | <位置> } [ASC | DESC]
```

语法说明如下：

- `<列名>`：指定用于分组的列。可以指定多个列，彼此间用逗号分隔。
- `<表达式>`：指定用于分组的表达式。通常与聚合函数一块使用，例如可将表达式 COUNT(*)AS' 人数 ' 作为 SELECT 选择列表清单的一项。
- `<位置>`：指定用于分组的选择列在 SELECT 语句结果集中的位置，通常是一个正整数。例如，GROUP BY 2 表示根据 SELECT 语句列清单上的第 2 列的值进行逻辑分组。
- `ASC|DESC`：关键字 ASC 表示按升序分组，关键字 DESC 表示按降序分组，其中 ASC 为默认值，注意这两个关键字必须位于对应的列名、表达式、列的位置之后。

  注意：GROUP BY 子句中的各选择列必须也是 SELECT 语句的选择列清单中的一项。

```
mysql> SELECT dept_id,GROUP_CONCAT(name) AS names
    -> FROM tb_students_info
    -> GROUP BY dept_id;
```

二十八、HAVING：指定过滤条件

SELECT 语句中，除了能使用 GROUP BY 子句分组数据外，还可以使用 HAVING 子句过滤分组，在结果集中规定了包含哪些分组和排除哪些分组。

```
HAVING <条件>
```

HAVING 子句和 WHERE 子句非常相似，HAVING 子句支持 WHERE 子句中所有的操作符和语法，但是两者存在几点差异：

- WHERE 子句主要用于过滤数据行，而 HAVING 子句主要用于过滤分组，即 HAVING 子句基于分组的聚合值而不是特定行的值来过滤数据，主要用来过滤分组。
- WHERE 子句不可以包含聚合函数，HAVING 子句中的条件可以包含聚合函数。
- HAVING 子句是在数据分组后进行过滤，WHERE 子句会在数据分组前进行过滤。WHERE 子句排除的行不包含在分组中，可能会影响 HAVING 子句基于这些值过滤掉的分组。

```
mysql> SELECT dept_id,GROUP_CONCAT(name) AS names
    -> FROM tb_students_info
    -> GROUP BY dept_id
    -> HAVING COUNT(name)>1;
```

二十九、REGEXP：正则表达式查询

MySQL中正式表达式通常被用来检索或替换符合某个模式的文本内容，根据指定的匹配模式匹配文中符合要求的特殊字符串。

MySQL 中使用 REGEXP 关键字指定正则表达式的字符匹配模式，下表列出了 REGEXP 操作符中常用的匹配列表。

选项	说明	例子	匹配值示例
^	匹配文本的开始字符	'^b' 匹配以字母 b 开头 的字符串	book、big、banana、 bike
$	匹配文本的结束字符	'st$' 匹配以 st 结尾的字 符串	test、resist、persist
.	匹配任何单个字符	'b.t' 匹配任何 b 和 t 之间有一个字符	bit、bat、but、bite
*	匹配零个或多个在它前面的字符	'f*n' 匹配字符 n 前面有 任意个字符 f	fn、fan、faan、abcn
+	匹配前面的字符 1 次或多次	'ba+' 匹配以 b 开头，后 面至少紧跟一个 a	ba、bay、bare、battle
<字符串>	匹配包含指定字符的文本	'fa'	fan、afa、faad
[字符集合]	匹配字符集合中的任何一个字符	'[xz]'匹配 x 或者 z	dizzy、zebra、x-ray、 extra
[^]	匹配不在括号中的任何字符	'[^abc]' 匹配任何不包 含 a、b 或 c 的字符串	desk、fox、f8ke
字符串{n,}	匹配前面的字符串至少 n 次	b{2} 匹配 2 个或更多 的 b	bbb、 bbbb、bbbbbbb
字符串{n,m}	匹配前面的字符串至少 n 次，至多 m 次	b{2,4} 匹配最少 2 个， 最多 4 个 b	bbb、 bbbb

查询以特定字符或字符串开头的记录

```
--查询 dept_name 字段以字母“C”开头的记录
mysql> SELECT * FROM tb_departments
    -> WHERE dept_name REGEXP '^C';
```

查询以特定字符或字符串结尾的记录

```
--查询 dept_name 字段以字母“y”结尾的记录
mysql> SELECT * FROM tb_departments
    -> WHERE dept_name REGEXP 'y$';
```

用符号“.”代替字符串中的任意一个字符

```
--查询 dept_name 字段值包含字母“o”与字母“y”且两个字母之间只有一个字母的记录
mysql> SELECT * FROM tb_departments
    -> WHERE dept_name REGEXP 'o.y';
```

使用“*”和“+”来匹配多个字符

```
--星号“*”匹配前面的字符任意多次，包括 0 次。加号“+”匹配前面的字符至少一次。
--查询 dept_name 字段值包含字母“c”，且“c”后面出现字母“h”的记录
mysql> SELECT * FROM tb_departments
    -> WHERE dept_name REGEXP '^Ch*';
```

```
--查询 dept_name 字段值包含字母“c”，且“c”后面出现字母“h”至少一次的记录
mysql> SELECT * FROM tb_departments
    -> WHERE dept_name REGEXP '^Ch+';
```

匹配指定字符串

正则表达式可以匹配指定字符串，只要这个字符串在查询文本中即可，若要匹配多个字符串，则多个字符串之间使用分隔符“|”隔开。

```
--查询 dept_name 字段值包含字符串“in”的记录
mysql> SELECT * FROM tb_departments
    -> WHERE dept_name REGEXP 'in';
```

```
--查询 dept_name 字段值包含字符串“in”或者“on”的记录
mysql> SELECT * FROM tb_departments
    -> WHERE dept_name REGEXP 'in|on';
```

匹配指定字符串中的任意一个

方括号“[]”指定一个字符集合，只匹配其中任何一个字符，即为所查找的文本。

```
--查询 dept_name 字段值包含字母“o”或者“i”的记录
mysql> SELECT * FROM tb_departments
    -> WHERE dept_name REGEXP '[io]';
```

```
--查询 dept_call 字段值中包含 1、2 或者 3 的记录;查询结果中，dept_call 字段值中有 1、2、3 三个数字中的一个即为匹配记录字段。
mysql> SELECT * FROM tb_departments
    -> WHERE dept_call REGEXP '[123]';
--匹配集合“[123]”也可以写成“[1-3]”，即指定集合区间。例如，“[a-z]”表示集合区间为a~z的字母，“[0-9]”表示集合区间为所有数字。
```

匹配指定字符以外的字符

“[^字符集合]”匹配不在指定集合中的任何字符。

```
--查询 dept_name 字段值包含字母 a~t 以外的字符的记录
mysql> SELECT * FROM tb_departments
    -> WHERE dept_name REGEXP '[^a-t]';
--返回记录中的 dept_name 字段值中包含了指定字母和数字以外的值，如 u、y 等，这些字母均不在 a~t 中，满足匹配条件。
```

三十、INSERT：插入数据

基本语法

```
-- INSERT…VALUES语句
INSERT INTO <表名> [ <列名1> [ , … <列名n>] ]
VALUES （值1）[… , （值n）];
```

```
-- INSERT…SET语句
INSERT INTO <表名>
SET <列名1> = <值1>,
        <列名2> = <值2>,
        …
```

由 INSERT 语句的两种形式可以看出：

- 使用 INSERT...VALUES 语句可以向表中插入一行数据，也可以插入多行数据；
- 使用 INSERT...SET 语句可以指定插入行中每列的值，也可以指定部分列的值；
- INSERT...SELECT 语句向表中插入其他表的数据。
- 采用 INSERT...SET 语句可以向表中插入部分列的值，这种方式更为灵活；
- INSERT...VALUES 语句可以一次插入多条数据。

向表中的全部字段添加值

```
mysql> INSERT INTO tb_courses
    -> (course_id,course_name,course_grade,course_info)
    -> VALUES(1,'Network',3,'Computer Network');
```

```
mysql> INSERT INTO tb_courses
    -> VLAUES(3,'Java',4,'Java EE');
```

注意：虽然使用 INSERT 插入数据时可以忽略插入数据的列名称，若值不包含列名称，则 VALUES 关键字后面的值不仅要求完整，而且顺序必须和表定义时列的顺序相同。如果表的结构被修改，对列进行增加、删除或者位置改变操作，这些操作将使得用这种方式插入数据时的顺序也同时改变。如果指定列名称，就不会受到表结构改变的影响。

使用 INSERT INTO...FROM 语句复制表数据

```
    mysql> INSERT INTO tb_courses_new
    -> (course_id,course_name,course_grade,course_info)
    -> SELECT course_id,course_name,course_grade,course_info
    -> FROM tb_courses;
```

三十一、UPDATE：修改数据

使用 UPDATE 语句修改单个表，语法格式为：

```
UPDATE <表名> SET 字段 1=值 1 [,字段 2=值 2… ] [WHERE 子句 ]
[ORDER BY 子句] [LIMIT 子句]
```

修改表中的数据

```
mysql> UPDATE tb_courses_new
    -> SET course_grade=4;
```

根据条件修改表中的数据

```
mysql> UPDATE tb_courses_new
    -> SET course_name='DB',course_grade=3.5
    -> WHERE course_id=2;
```

三十二、DELETE：删除数据

删除单个表中的数据

```
DELETE FROM <表名> [WHERE 子句] [ORDER BY 子句] [LIMIT 子句]
注意：在不使用 WHERE 条件的时候，将删除所有数据。
```

删除表中的全部数据

```
mysql> DELETE FROM tb_courses_new;
```

根据条件删除表中的数据

```
mysql> DELETE FROM tb_courses
    -> WHERE course_id=4;
```

三十三、视图简介

视图是一个虚拟表，其内容由查询定义。同真实表一样，视图包含一系列带有名称的列和行数据，但视图并不是数据库真实存储的数据表。

视图并不同于数据表，它们的区别在于以下几点：

- 视图不是数据库中真实的表，而是一张虚拟表，其结构和数据是建立在对数据中真实表的查询基础上的。
- 存储在数据库中的查询操作 SQL 语句定义了视图的内容，列数据和行数据来自于视图查询所引用的实际表，引用视图时动态生成这些数据。
- 视图没有实际的物理记录，不是以数据集的形式存储在数据库中的，它所对应的数据实际上是存储在视图所引用的真实表中的。
- 视图是数据的窗口，而表是内容。表是实际数据的存放单位，而视图只是以不同的显示方式展示数据，其数据来源还是实际表。
- 视图是查看数据表的一种方法，可以查询数据表中某些字段构成的数据，只是一些 SQL 语句的集合。从安全的角度来看，视图的数据安全性更高，使用视图的用户不接触数据表，不知道表结构。
- 视图的建立和删除只影响视图本身，不影响对应的基本表

视图具有如下优点：

- 定制用户数据，聚焦特定的数据

- 简化数据操作

在使用查询时，很多时候要使用聚合函数，同时还要显示其他字段的信息，可能还需要关联到其他表，语句可能会很长，如果这个动作频繁发生的话，可以创建视图来简化操作。

- 提高基表数据的安全性

视图是虚拟的，物理上是不存在的。可以只授予用户视图的权限，而不具体指定使用表的权限，来保护基础数据的安全。

- 共享所需数据

通过使用视图，每个用户不必都定义和存储自己所需的数据，可以共享数据库中的数据，同样的数据只需要存储一次。

- 更改数据格式

通过使用视图，可以重新格式化检索出的数据，并组织输出到其他应用程序中。

- 重用 SQL 语句

视图提供的是对查询操作的封装，本身不包含数据，所呈现的数据是根据视图定义从基础表中检索出来的，如果基础表的数据新增或删除，视图呈现的也是更新后的数据。视图定义后，编写完所需的查询，可以方便地重用该视图。

三十四、创建视图

基本语法

```
CREATE VIEW <视图名> AS <SELECT语句>
```

创建基于单表的视图

```
mysql> CREATE VIEW view_students_info
    -> AS SELECT * FROM tb_students_info;
```

```
mysql> CREATE VIEW v_students_info
    -> (s_id,s_name,d_id,s_age,s_sex,s_height,s_date)
    -> AS SELECT id,name,dept_id,age,sex,height,login_date
    -> FROM tb_students_info;
```

创建基于多表的视图

```

```

查询视图

视图一经定义之后，就可以如同查询数据表一样，使用 SELECT 语句查询视图中的数据，语法和查询基础表的数据一样。

```
--查看视图
DESCRIBE 视图名；
```

三十五、修改视图

语法格式如下：

```
ALTER VIEW <视图名> AS <SELECT语句>
```

修改视图内容

视图是一个虚拟表，实际的数据来自于基本表，所以通过插入、修改和删除操作更新视图中的数据，实质上是在更新视图所引用的基本表的数据。注意：对视图的修改就是对基本表的修改，因此在修改时，要满足基本表的数据定义。

某些视图是可更新的。也就是说，可以使用 UPDATE、DELETE 或 INSERT 等语句更新基本表的内容。对于可更新的视图，视图中的行和基本表的行之间必须具有一对一的关系。

还有一些特定的其他结构，这些结构会使得视图不可更新。更具体地讲，如果视图包含以下结构中的任何一种，它就是不可更新的：

- 聚合函数 SUM()、MIN()、MAX()、COUNT() 等。
- DISTINCT 关键字。
- GROUP BY 子句。
- HAVING 子句。
- UNION 或 UNION ALL 运算符。
- 位于选择列表中的子查询。
- FROM 子句中的不可更新视图或包含多个表。
- WHERE 子句中的子查询，引用 FROM 子句中的表。
- ALGORITHM 选项为 TEMPTABLE（使用临时表总会使视图成为不可更新的）的时候。

```
mysql> ALTER VIEW view_students_info
    -> AS SELECT id,name,age
    -> FROM tb_students_info;
```

```
mysql> UPDATE view_students_info
    -> SET age=25 WHERE id=1;
```

三十六、删除视图

语法格式如下：

```
DROP VIEW <视图名1> [ , <视图名2> …]
DROP VIEW IF EXISTS v_students_info;
```

三十七、自定义函数

MySQL 自带的函数可能完成不了我们的业务需求，这时候就需要自定义函数。

自定义函数是一种与存储过程十分相似的过程式数据库对象。它与存储过程一样，都是由 SQL 语句和过程式语句组成的代码片段，并且可以被应用程序和其他 SQL 语句调用。

自定义函数与存储过程之间存在几点区别：

- 自定义函数不能拥有输出参数，这是因为自定义函数自身就是输出参数；而存储过程可以拥有输出参数。
- 自定义函数中必须包含一条 RETURN 语句，而这条特殊的 SQL 语句不允许包含于存储过程中。
- 可以直接对自定义函数进行调用而不需要使用 CALL 语句，而对存储过程的调用需要使用 CALL 语句。

创建并使用自定义函数

```
CREATE FUNCTION <函数名> ( [ <参数1> <类型1> [ , <参数2> <类型2>] ] … )
  RETURNS <类型>
  <函数主体>

  mysql> CREATE FUNCTION StuNameById()
    -> RETURNS VARCHAR(45)
    -> RETURN
    -> (SELECT name FROM tb_students_info
    -> WHERE id=1);
```

```
SELECT <自定义函数名> ([<参数> [,...]])
mysql> SELECT StuNameById();
```

删除自定义函数

```
mysql> DROP FUNCTION StuNameById;
```

三十八、存储过程简介

我们前面所学习的 MySQL语句都是针对一个表或几个表的单条 SQL 语句，但是在数据库的实际操作中，并非所有操作都那么简单，经常会有一个完整的操作需要多条 SQL 语句处理多个表才能完成。例如，为了确认学生能否毕业，需要同时查询学生档案表、成绩表和综合表，此时就需要使用多条 SQL 语句来针对几个数据表完成这个处理要求。存储过程可以有效地完成这个数据库操作。

存储过程通常有如下优点：

- 封装性

存储过程被创建后，可以在程序中被多次调用，而不必重新编写该存储过程的 SQL 语句，并且数据库专业人员可以随时对存储过程进行修改，而不会影响到调用它的应用程序源代码。

- 可增强 SQL 语句的功能和灵活性

存储过程可以用流程控制语句编写，有很强的灵活性，可以完成复杂的判断和较复杂的运算。

- 可减少网络流量

由于存储过程是在服务器端运行的，且执行速度快，因此当客户计算机上调用该存储过程时，网络中传送的只是该调用语句，从而可降低网络负载。

- 高性能

存储过程执行一次后，产生的二进制代码就驻留在缓冲区，在以后的调用中，只需要从缓冲区中执行二进制代码即可，从而提高了系统的效率和性能。

- 提高数据库的安全性和数据的完整性

使用存储过程可以完成所有数据库操作，并且可以通过编程的方式控制数据库信息访问的权限。

三十九、创建存储过程

语法格式如下：

```
CREATE PROCEDURE <过程名> ( [过程参数[,…] ] ) <过程体>
[过程参数[,…] ] 格式
[ IN | OUT | INOUT ] <参数名> <类型>
```

```
--创建不带参数的存储过程
mysql> CREATE PROCEDURE ShowStuScore()
    -> BEGIN
    -> SELECT * FROM tb_students_score;
    -> END

mysql> CALL ShowStuScore();
```

```
--创建带参数的存储过程
mysql> CREATE PROCEDURE GetScoreByStu
    -> (IN name VARCHAR(30))
    -> BEGIN
    -> SELECT student_score FROM tb_students_score
    -> WHERE student_name=name;
    -> END
```

```
mysql> DROP PROCEDURE GetScoreByStu;
```

四十、触发器简介

数据库中触发器是一个特殊的存储过程，不同的是执行存储过程要使用 CALL 语句来调用，而触发器的执行不需要使用 CALL 语句来调用，也不需要手工启动，只要一个预定义的事件发生就会被 MySQL自动调用。

引发触发器执行的事件一般如下：

- 增加一条学生记录时，会自动检查年龄是否符合范围要求。
- 每当删除一条学生信息时，自动删除其成绩表上的对应记录。
- 每当删除一条数据时，在数据库存档表中保留一个备份副本。

触发程序的优点如下：

- 触发程序的执行是自动的，当对触发程序相关表的数据做出相应的修改后立即执行。
- 触发程序可以通过数据库中相关的表层叠修改另外的表。
- 触发程序可以实施比 FOREIGN KEY 约束、CHECK 约束更为复杂的检查和操作。

  触发器与表关系密切，主要用于保护表中的数据。特别是当有多个表具有一定的相互联系的时候，触发器能够让不同的表保持数据的一致性。

在 MySQL 中，只有执行 INSERT、UPDATE 和 DELETE 操作时才能激活触发器。

四十一、创建触发器

语法格式如下：

```
CREATE <触发器名> < BEFORE | AFTER >
<INSERT | UPDATE | DELETE >
ON <表名> FOR EACH Row<触发器主体>
```

```
--创建 BEFORE 类型触发器
--插入数据之前，对新插入的 salary 字段值进行求和计算
mysql> CREATE TRIGGER SumOfSalary
    -> BEFORE INSERT ON tb_emp8
    -> FOR EACH ROW
    -> SET @sum=@sum+NEW.salary;
```

```
--创建 AFTER 类型触发器
--向数据表 tb_emp6 中插入数据之后，再向数据表 tb_emp7 中插入相同的数据，并且 salary 为 tb_emp6 中新插入的 salary 字段值的 2 倍
mysql> CREATE TRIGGER double_salary
    -> AFTER INSERT ON tb_emp6
    -> FOR EACH ROW
    -> INSERT INTO tb_emp7
    -> VALUES (NEW.id,NEW.name,deptId,2*NEW.salary);
```

四十二、修改和删除触发器

语法格式如下：

```
    DROP TRIGGER [ IF EXISTS ] [数据库名] <触发器名>
    mysql> DROP TRIGGER double_salary;
```

四十三、索引简介

索引就是根据表中的一列或若干列按照一定顺序建立的列值与记录行之间的对应关系表，实质上是一张描述索引列的列值与原表中记录行之间一一对应关系的有序表。

索引访问是通过遍历索引来直接访问表中记录行的方式。使用这种方式的前提是对表建立一个索引，在列上创建了索引之后，查找数据时可以直接根据该列上的索引找到对应记录行的位置，从而快捷地查找到数据。索引存储了指定列数据值的指针，根据指定的排序顺序对这些指针排序。

例如，在学生基本信息表 students 中，如果基于 student_id 建立了索引，系统就建立了一张索引列到实际记录的映射表，当用户需要查找 student_id 为 12022 的数据的时候，系统先在 student_id 索引上找到该记录，然后通过映射表直接找到数据行，并且返回该行数据。因为扫描索引的速度一般远远大于扫描实际数据行的速度，所以采用索引的方式可以大大提高数据库的工作效率。

根据索引的具体用途，MySQL 中的索引在逻辑上分为以下 5 类：

- 普通索引

普通索引是最基本的索引类型，唯一任务是加快对数据的访问速度，没有任何限制。创建普通索引时，通常使用的关键字是 INDEX 或 KEY。

- 唯一性索引

唯一性索引是不允许索引列具有相同索引值的索引。如果能确定某个数据列只包含彼此各不相同的值，在为这个数据列创建索引的时候就应该用关键字 UNIQUE 把它定义为一个唯一性索引。

创建唯一性索引的目的往往不是为了提高访问速度，而是为了避免数据出现重复。

- 主键索引

主键索引是一种唯一性索引，即不允许值重复或者值为空，并且每个表只能有一个主键。主键可以在创建表的时候指定，也可以通过修改表的方式添加，必须指定关键字 PRIMARY KEY。

> 注意：主键是数据库考察的重点。注意每个表只能有一个主键。

- 空间索引

空间索引主要用于地理空间数据类型 GEOMETRY。

- 全文索引

全文索引只能在 VARCHAR 或 TEXT 类型的列上创建，并且只能在 MyISAM 表中创建。

索引的使用原则和注意事项

虽然索引可以加快查询速度，提高 MySQL 的处理性能，但是过多地使用索引也会造成以下弊端：

- 创建索引和维护索引要耗费时间，这种时间随着数据量的增加而增加。
- 除了数据表占数据空间之外，每一个索引还要占一定的物理空间。如果要建立聚簇索引，那么需要的空间就会更大。
- 当对表中的数据进行增加、删除和修改的时候，索引也要动态地维护，这样就降低了数据的维护速度。

> 注意：索引可以在一些情况下加速查询，但是在某些情况下，会降低效率。

索引只是提高效率的一个因素，因此在建立索引的时候应该遵循以下原则：

- 在经常需要搜索的列上建立索引，可以加快搜索的速度。
- 在作为主键的列上创建索引，强制该列的唯一性，并组织表中数据的排列结构。
- 在经常使用表连接的列上创建索引，这些列主要是一些外键，可以加快表连接的速度。
- 在经常需要根据范围进行搜索的列上创建索引，因为索引已经排序，所以其指定的范围是连续的。
- 在经常需要排序的列上创建索引，因为索引已经排序，所以查询时可以利用索引的排序，加快排序查询。
- 在经常使用 WHERE 子句的列上创建索引，加快条件的判断速度。

与此对应，在某些应用场合下建立索引不能提高 MySQL 的工作效率，甚至在一定程度上还带来负面效应，降低了数据库的工作效率，一般来说不适合创建索引的环境如下：

- 对于那些在查询中很少使用或参考的列不应该创建索引。因为这些列很少使用到，所以有索引或者无索引并不能提高查询速度。相反，由于增加了索引，反而降低了系统的维护速度，并增大了空间要求。
- 对于那些只有很少数据值的列也不应该创建索引。因为这些列的取值很少，例如人事表的性别列。查询结果集的数据行占了表中数据行的很大比例，增加索引并不能明显加快检索速度。
- 对于那些定义为 TEXT、IMAGE 和 BIT 数据类型的列不应该创建索引。因为这些列的数据量要么相当大，要么取值很少。
- 当修改性能远远大于检索性能时，不应该创建索引。因为修改性能和检索性能是互相矛盾的。当创建索引时，会提高检索性能，降低修改性能。当减少索引时，会提高修改性能，降低检索性能。因此，当修改性能远远大于检索性能时，不应该创建索引。

四十四、创建索引

基本语法

```
--使用 CREATE INDEX 语句
CREATE <索引名> ON <表名> (<列名> [<长度>] [ ASC | DESC])
```

```
--使用 CREATE TABLE 语句
KEY | INDEX [<索引名>] [<索引类型>] (<列名>,…)
```

```
--使用 ALTER TABLE 语句
ADD INDEX [<索引名>] [<索引类型>] (<列名>,…)
```

创建一般索引

```
mysql> CREATE TABLE tb_stu_info
    -> (
    -> id INT NOT NULL,
    -> name CHAR(45) DEFAULT NULL,
    -> dept_id INT DEFAULT NULL,
    -> age INT DEFAULT NULL,
    -> height INT DEFAULT NULL,
    -> INDEX(height)
    -> );
```

创建唯一索引

```
mysql> CREATE TABLE tb_stu_info2
    -> (
    -> id INT NOT NULL,
    -> name CHAR(45) DEFAULT NULL,
    -> dept_id INT DEFAULT NULL,
    -> age INT DEFAULT NULL,
    -> height INT DEFAULT NULL,
    -> UNIQUE INDEX(id)
    -> );
```

查看索引

```
mysql> SHOW INDEX FROM tb_stu_info2
```

四十五、修改和删除索引

基本语法

```
--使用 DROP INDEX 语句
DROP INDEX <索引名> ON <表名>
mysql> DROP INDEX height
    -> ON tb_stu_info;
```

```
-- 使用 ALTER TABLE 语句
mysql> ALTER TABLE tb_stu_info2
    -> DROP INDEX height;
```

四十六、用户管理

创建用户

```
CREATE USER <用户名> [ IDENTIFIED ] BY [ PASSWORD ] <口令>mysql> ALTER TABLE
mysql> CREATE USER 'james'@'localhost'
    -> IDENTIFIED BY 'tiger';
```

修改用户

```
--将用户名 james 修改为 jack，主机是 localhost
mysql> RENAME USER james@'localhost'
    -> TO jack@'localhost';
```

```
--使用 SET 语句将用户名为 jack 的密码修改为 lion，主机是 localhost
mysql> SET PASSWORD FOR 'jack'@'localhost'=
    -> PASSWORD('lion');
```

删除用户

```
--删除用户'jack'@'localhost'
mysql> DROP USER 'jack'@'localhost';
```

用户授权

```
使用 GRANT 语句创建一个新的用户 testUser，密码为 testPwd。用户 testUser 对所有的数据有查询、插入权限，并授予 GRANT 权限。
mysql> GRANT SELECT,INSERT ON *.*
    -> TO 'testUser'@'localhost'
    -> IDENTIFIED BY 'testPwd'
    -> WITH GRANT OPTION;
```

删除用户权限

```
第一种
REVOKE <权限类型> [ ( <列名> ) ] [ , <权限类型> [ ( <列名> ) ] ]…
ON <对象类型> <权限名> FROM <用户1> [ , <用户2> ]…
```

```
第二种
REVOKE ALL PRIVILEGES, GRANT OPTION
FROM user <用户1> [ , <用户2> ]…
```

```
mysql> REVOKE INSERT ON *.*
    -> FROM 'testUser'@'localhost';
```

四十七、事务

事务具有 4 个特性：原子性（Atomicity）、一致性（Consistency）、隔离性（Isolation）和持续性（Durability）。这 4 个特性简称为 ACID 特性。

开始事务

```
BEGIN TRANSACTION <事务名称> |@<事务变量名称>
```

提交事务

```
COMMIT TRANSACTION <事务名称> |@<事务变量名称>
```

撤销事务

```
ROLLBACK [TRANSACTION]
[<事务名称>| @<事务变量名称> | <存储点名称>| @ <含有存储点名称的变量名>
```