认识 sql 并学习数据库的基础操作

任务介绍：
1、什么是关系型和非关系型数据库，两者都包含哪些种类的数据库（理解两者的区别）

2、选择一种关系型数据库进行学习（选择自己不熟悉的进行学习，因为不同的数据库，其特性也不同，所以可以选择不熟悉或者感兴趣进行研究学习）

3、学习数据库中的字段类型并创建库和用户表，需要包含所有字段类型（主要熟悉数据库的基本使用，可以自由创建、删除、修改数据库和表）

4、学习数据库的增删改查，记录学习过程（重点是 sql 语句的理解）并形成报告（最终结果）

一、什么是关系型和非关系型数据库

1.基础概念

- **数据库**

对于数据库的概念，没有一个完全固定的定义，随着数据库历史的发展，定义的内容也有很大的差异，其中一个比较普遍观点认为，数据库(DataBase，DB)是一个长期存储在计算机内的、有组织的、有共享的、统一管理的数据集合。它是一个按照数据结构来存储和管理数据的计算机软件系统。即数据库包含两层含义：保管数据的“仓库”，以及数据管理的方法和技术。

- **数据库管理系统**

数据库管理系统(DataBase Management System， DBMS)是用户创建、管理和维护数据库时所使用的软件，位于用户与操作系统之间，对数据库进行统一管理。DBMS能定义数据存储结构，提供数据的操作机制，维护数据库的安全性、完整性和可靠性。

- **数据库应用程序**

数据库应用程序(DataBase Application)虽然已经有了DBMS,但是在很多情况下，DBMS无法满足对数据管理的要求。数据库应用程序的使用可以满足对数据管理的更高要求，还可以使数据管理过程更加直观和友好。数据库应用程序负责与DBMS进行通信，访问和管理DBMS中存储的数据，允许用户插入、修改、删除DB中的数据。

2.关系型数据库

- **概念**

关系型数据库是依据关系模型来创建的数据库。所谓关系模型就是“一对一、一对多、多对多”等关系模型，关系模型就是指二维表格模型,因而一个关系型数据库就是由二维表及其之间的联系组成的一个数据组织。主要代表：SQL Server，Oracle，MySQL(开源)，PostgreSQL(开源)。

- 关系型数据库特征

(1) 关系型数据库，是指采用了关系模型来组织。

(2) 关系型数据库的最大特点就是事务的一致性。

(3) 简单来说，关系模型指的就是二维表格模型。

- **优点**

(1) 保持数据的一致性（事务处理）。

(2) 由于以标准化为前提，数据更新的开销很小（相同的字段基本上都只有一处）。

(3) 可以进行Join等复杂查询。

(4) 容易理解，二维表结构是非常贴近逻辑世界一个概念，关系模型相对网状、层次等其他模型来说更容易理解。

(5) 使用方便，通用的SQL语言使得操作关系型数据库非常方便。

(6) 易于维护，丰富的完整性(实体完整性、参照完整性和用户定义的完整性)大大减低了数据冗余和数据不一致的概率。

- **缺点**

(1) 为了维护一致性所付出的巨大代价就是其读写性能比较差。

(2) 固定的表结构。

(3) 高并发读写需求。

(4) 海量数据的高效率读写。

3.非关系型数据库

- **概念**

非关系型数据库主要是基于“非关系模型”的数据库（由于关系型太大，所以一般用“非关系型”来表示其他类型的数据库）。非关系型模型比如有：

(1) 列模型：存储的数据是一列列的。关系型数据库以一行作为一个记录，列模型数据库以一列为一个记录。（这种模型，数据即索引，IO很快，主要是一些分布式数据库）。

(2) 键值对模型：存储的数据是一个个“键值对”，比如name:liming,那么name这个键里面存的值就是liming。

(3) 文档类模型：以一个个文档来存储数据，有点类似“键值对”。

- **非关系型数据库特征**

(1) 使用键值对存储数据。
(2) 分布式。
(3) 一般不支持ACID特性。
(4) 非关系型数据库严格上不是一种数据库，应该是一种数据结构化存储方法的集合。

- **优点**

(1) 成本：nosql数据库简单易部署，基本都是开源软件，不需要像使用oracle那样花费大量成本购买使用，相比关系型数据库价格便宜。

(2) 查询速度：nosql数据库将数据存储于缓存之中，关系型数据库将数据存储在硬盘中，自然查询速度远不及nosql数据库。

(3) 灵活的数据模型：nosql的存储格式是key,value形式、文档形式、图片形式等等，所以可以存储基础类型以及对象或者是集合等各种格式，而数据库则只支持基础类型。

(4) 扩展性：关系型数据库有类似join这样的多表查询机制的限制导致扩展很艰难。

(5) 高可用：NoSQL在不太影响性能的情况下，就可以方便地实现高可用的架构。比如Cassandra、HBase模型，通过复制模型也能实现高可用。

- **缺点**

(1) 没有标准,没有对NoSQL数据库定义的标准，所以没有两个NoSQL数据库是平等的。

(2) 维护的工具和资料有限，因为nosql是属于新的技术，不能和关系型数据库10几年的技术同日而语。

(3) 不提供对sql的支持，如果不支持sql这样的工业标准，将产生一定用户的学习和使用成本。

(4) 不提供关系型数据库对事物的处理。

4.关系型数据库与非关系型数据库的比较

(1) 实质，非关系型数据库产品是传统关系型数据库的功能阉割版本，通过减少用不到或很少用的功能，来大幅度提高产品性能。

(2) 成本，nosql数据库简单易部署，基本都是开源软件，不需要像使用oracle那样花费大量成本购买使用，相比关系型数据库价格便宜。

(3) 查询速度，nosql数据库将数据存储于缓存之中，关系型数据库将数据存储在硬盘中，自然查询速度远不及nosql数据库。

(4) 存储数据的格式，nosql的存储格式是key,value形式、文档形式、图片形式等等，所以可以存储基础类型以及对象或者是集合等各种格式，而数据库则只支持基础类型。

(5) 扩展性，关系型数据库有类似join这样的多表查询机制的限制导致扩展很艰难。

二、MySQL学习

1.基础

- **MySQL存储引擎**

MySQL提供多个不同的存储引擎，包括处理事务安全表的引擎和处理非事务安全表的引擎。可以使用SHOW ENGINES语句查看系统所支持的引擎类型，结果如下。

```
mysql> mysql> SHOW ENGINES;
+--------------------+---------+----------------------------------------------------------------+--------------+------+------------+
| Engine             | Support | Comment                                                        | Transactions | XA   | Savepoints |
+--------------------+---------+----------------------------------------------------------------+--------------+------+------------+
| InnoDB             | DEFAULT | Supports transactions, row-level locking, and foreign keys     | YES          | YES  | YES        |
| MRG_MYISAM         | YES     | Collection of identical MyISAM tables                          | NO           | NO   | NO         |
| MEMORY             | YES     | Hash based, stored in memory, useful for temporary tables      | NO           | NO   | NO         |
| BLACKHOLE          | YES     | /dev/null storage engine (anything you write to it disappears) | NO           | NO   | NO         |
| MyISAM             | YES     | MyISAM storage engine                                          | NO           | NO   | NO         |
| CSV                | YES     | CSV storage engine                                             | NO           | NO   | NO         |
| ARCHIVE            | YES     | Archive storage engine                                         | NO           | NO   | NO         |
| PERFORMANCE_SCHEMA | YES     | Performance Schema                                             | NO           | NO   | NO         |
| FEDERATED          | NO      | Federated MySQL storage engine                                 | NULL         | NULL | NULL       |
+--------------------+---------+----------------------------------------------------------------+--------------+------+------------+
9 rows in set (0.00 sec)
```

#InnoDB存储引擎

InnoDB事务型数据库的首选引擎，支持事务安全表(ACID)，支持行锁定和外键。 MySQL5.5.5之后， InnoDB作为默认存储引擎，InnoDB主要特性有：

(1) InnoDB给 MySQL提供了具有提交、回滚和崩溃恢复能力的事务安全(ACID兼容)存储引擎。 InnoDB锁定在行级并且也在 SELECT语句中提供一个类似 Oracle的非锁定读。这些功能增加了多用户部署和性能。在SQL查询中，可以自由地InnoDB类型的表与其他 MySQL的表的类型混合起来，甚至在同一个查询中也可以混合。

(2) InnoDB是为处理巨大数据量的最大性能设计。它的CPU效率可能是任何其他基于磁盘的关系数据库引擎所不能匹敌的。

(3) InnoDB存储引擎完全与 MySQL服务器整合， InnoDB存储引擎为在主内存中缓存数据和索引而维持它自己的缓冲池。 InnoDB将它的表和索引在一个逻辑表空间中，表空间可以包含数个文件(或原始磁盘分区)。这与 MyISAM表不同，比如在MyISAM表中每个表被存在分离的文件中。 InnoDB表可以是任何尺寸，即使在文件尺寸被限制为2GB的操作系统上。

(4) InnoDB支持外键完整性约束(FOREIGN KEY)。存储表中的数据时，每张表的存储都按主键顺序存放，如果没有显示在表定义时指定主键，InnoDB会为每一行生成一个6字节的 ROWID，并以此作为主键。

(5) InnoDB被用在众多需要高性能的大型数据库站点上。

InoDB不创建目录，使用 InnoDB时， MySQL将在 MySQL数据目录下创建一个名为 ibdata1的10MB大小的自动扩展数据文件，以及两个名为 ib logfile0和 ib logfile1的5MB大小的日志文件。

#MyISAM存储引擎

MyISAM基于ISAM存储引擎，并对其进行扩展。它是在web、数据仓储和其他应用环境下最常使用的存储引擎之一。 MyISAM拥有较高的插入、查询速度，但不支持事务。在 MySQL5.5.5之前的版本中， MyISAM是默认存储引擎。 MyISAM主要特性有：

(1) 大文件(达63位文件长度)在支持大文件的文件系统和操作系统上被支持。

(2) 当把删除和更新及插入操作混合使用的时候，动态尺寸的行产生更少碎片。这要通过合并相邻被删除的块，以及若下一个块被删除，就扩展到下一块来自动完成。

(3) 每个 MyISAM表最大索引数是64，这可以通过重新编译来改变。每个索引最大的列数是16个。

(4) 最大的键长度是1000字节，这也可以通过编译来改变。对于键长度超过250字节的情况，一个超过1024字节的键将被用上。

(5) BLOB和TEXT列可以被索引。

(6) NULL值被允许在索引的列中。这个值占每个键的0-1个字节。

(7) 所有数字键值以高字节优先被存储以允许一个更高地索引压缩。

(8) 每表一个 AUTO_INCREMENT列的内部处理。 MyISAM为 INSERT和 UPDATE操作自动更新这一列。这使得 AUTO_INCREMENT列更快(至少10%)。在序列顶的值被删除之后就不能再利用。

(9)可以把数据文件和索引文件放在不同目录。

(10)每个字符列可以有不同的字符集。

(11)有 VARCHAR的表可以固定或动态记录长度。

(12) VARCHAR和CHAR列可以多达64KB。

使用 MyISAM引擎创建数据库，将生产3个文件。文件的名字以表的名字开始，扩展名指出文件类型：frm文件存储表定义，数据文件的扩展名为MYD(MYData)，索引文件的扩展名是.MYI (MYIndex)。

#MEMORY存储引擎

MEMORY存储引擎将表中的数据存储到内存中，为查询和引用其它表数据提供快速访问。MEMORY主要特性有：

(1) MEMORY表的每个表可以有多达32个索引，每个索引16列，以及500字节的最大键长度。

(2) MEMORY存储引擎执行HASH和 BTREE索引。

(3)可以在一个 MEMORY表中有非唯一键。

(4) MEMORY表使用一个固定的记录长度格式。

(5) MEMORY不支持BLOB或TEXT列。

(6) MEMORY支持 AUTO_INCREMENT列和对可包含NULL值的列的索引。

(7) MEMORY表在所有客户端之间共享(就像其他任何非 TEMPORARY表)。

(8) MEMORY表内容被存在内存中，内存是 MEMORY表和服务器在查询处理时的空闲中，创建的内部表共享。

(9)当不再需要 MEMORY表的内容时，要释放被 MEMORY表使用的内存，应该执行 DELETE FROM或 TRUNCATE TABLE，或者删除整个表(使用 DROP TABLE)。

#存储引擎的选择

不同存储引擎都有各自的特点，以适应不同的需求，为了做出选择，首先需要考虑每一个存储引擎提供了那些不同的功能，如下表。

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
支持数索引	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

如果要提供提交，回滚和崩溃恢复能力的事务安全(ACID兼容)能力，并要求实现并发控制，InnoDB是个很好的选择。如果数据表主要用来插入和查询记录，则 MyISAM引擎能提供较高的处理效率；如果只是临时存放数据，数据量不大，并且不需要较高的数据安全性，可以选择将数据保存在内存中的 Memory引擎， MySQL中使用该引擎作为临时表，存放查询的中间结果。如果只有INSERT和SELECT操作，可以选择 Archive引擎， Archive存储引擎支持高并发的插入操作，但是本身并不是事务安全的。 Archive存储引擎非常适合存储归档数据，如记录日志信息可以使用Archive引擎。

使用哪一种引擎要根据需要灵活选择，一个数据库中多个表可以使用不同引擎以满足各种性能和实际需求。使用合适的存储引擎，将会提高整个数据库的性能。

- **数据类型**

#整数类型

数值型数据类型主要用来存储数字， MySQL提供了多种数值数据类型，不同的数据类型提供不同的取值范围，可以存储的值范围越大，其所需要的存储空间也会越大。 MySQL主要提供的整数类型有： TINYINT、 SMALLINT、 MEDIUMINT、INT(INTEGER)、 BIGINT。整数类型的属性字段可以添加AUTO_INCREMENT自增约束条件。

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(INTEGER）	普通大小的整数	4 个字节
BIGINT	大整数	8 个字节

从表中可以看出不同类型整数存储所需的字节数不同，还有取值范围也不同。

数据类型	有符号	无符号
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

#浮点数类型和定点数类型

MySQL中使用浮点数和定点数来表示小数。浮点类型有两种：单精度浮点类型(FLOAT)和双精度浮点类型(DOUBLE)。定点类型只有一种： DECIMAL。浮点类型和定点类型都可以用(M，N)来表示，其中M称为精度，表示总共的位数；N称为标度，是表示小数的位数。

类型名称	说明	存储需求
FLOAT	单精度浮点数	4 个字节
DOUBLE	双精度浮点数	8 个字节
DECIMAL（M,D），DEC	压缩的“严格”定点数	M+2 个字节

#日期和时间类型

MySQL有多种表示日期的数据类型，主要有DATETIME、DATE、TIMESTAMP、TIME和YEAR。

类型名称	日期格式	日期范围	存储需求
YEAR	YYYY	1901～2155	1 字节
TIME	HH:MM:SS	-838:59:59 ～838:59:59	3 字节
DATE	YYYY-MM-DD	1000-01-01～9999-12-3	3 字节
DATETIME	YYYY-MM-DD HH:MM:SS	1000-01-01 00:00:00～9999-12-31 23:59:59	8 字节
TIMESTAMP	YYYY-MM-DD HH:MM:SS	1970-01-01 00:00:01 UTC ～ 2038-01-19 03:14:07 UTC	4 字节

#字符串类型

字符串类型用来存储字符串数据，还可以存储其它类型的数据，比如图片和声音的二进制文件，字符串可以进行区分或者不区分大小写的串比较，因外，还可以进行模式匹配查找。

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 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）

2.数据库的基本操作

#查看当前所有存在的数据库

```
SHOW DATABASES;
```

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

#**创建数据库**

`CREATE DATABASE  database_name;`

```
mysql> CREATE DATABASE test_db;
Query OK, 1 row affected (0.00 sec)

mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
| test_db            |
+--------------------+
5 rows in set (0.00 sec)
```

#**查看数据库的定义**(\G后面不用再加分号;)

`SHOW CREATE DATABASE database_name\G`

```
mysql> SHOW CREATE DATABASE test_db\G
*************************** 1. row ***************************
       Database: test_db
Create Database: CREATE DATABASE `test_db` /*!40100 DEFAULT CHARACTER SET utf8 */
1 row in set (0.00 sec)
```

#**删除数据库**(使用此命令是需要十分谨慎，MySQL不会给任何提醒确认信息，而且不能恢复)

`DROP DATABASE database_name;`

```
mysql> CREATE DATABASE test_db;
Query OK, 1 row affected (0.00 sec)

mysql> DROP DATABASE test_db;
Query OK, 0 rows affected (0.00 sec)

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

3.数据表的基本操作

3.1 创建数据表

#数据表属于数据库，在创建数据表之前，应该使用语法“USE <数据库名>”指定在那个数据库进行，语法如下。

CREATE TABLE <表名>

```
(
    字段名1，数据类型 [列级别约束条件] [默认值],
    字段名2，数据类型 [列级别约束条件] [默认值],
    ......
    [表级约束条件]
)
```

#创建tb_tmp1表

```
mysql> USE test_db;
Database changed
mysql> CREATE TABLE tb_emp1
    -> (
    -> id INT(11),
    -> name VARCHAR(25),
    -> deptId INT(11),
    -> salary FLOAT
    -> );
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW TABLES;
+------------------+
| Tables_in_test_db |
+------------------+
| tb_emp1          |
+------------------+
1 row in set (0.00 sec)
```

- **使用主键约束**

#在定义列的同时指定主键

字段名　数据类型 PRIMARY KEY [默认值]，定义数据库tb_emp2，其主键为id

```
mysql> CREATE TABLE tb_emp2 (
    -> id INT(11) PRIMARY KEY,
    -> name VARCHAR(25),
    -> deptId INT(11),
    -> salary FLOAT
    -> );
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW TABLES;
+------------------+
| Tables_in_test_db |
+------------------+
| tb_emp1          |
| tb_emp2          |
+------------------+
2 rows in set (0.00 sec)
```

#在定义完所有列之后指定主键

```
mysql> CREATE TABLE tb_emp3 (
    -> id INT(11),
    -> name VARCHAR(25),
    -> deptId INT(11),
    -> salary FLOAT,
    -> PRIMARY KEY(id)
    -> );
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW TABLES;
+------------------+
| Tables_in_test_db |
+------------------+
| tb_emp1          |
| tb_emp2          |
| tb_emp3          |
+------------------+
3 rows in set (0.00 sec)
```

- **使用外键约束**

#语法规则如下，创建tb-emp5表、tb_dept1表tb_emp5的键deptId作为外键关联到tb_dept1的主键id

`[CONSTRAINT <外键名>] FOREIGN KEY 字段名1[,字段名2,...]`

`REFERENCES <主键名> 主键列1[,主键列2,...]`

```
mysql> CREATE TABLE tb_dept1 (
    -> id INT(11) PRIMARY KEY,
    -> name VARCHAR(25) NOT NULL,
    -> location VARCHAR(50)
    -> );
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE TABLE tb_emp5 (
    -> id INT(11) PRIMARY KEY,
    -> name VARCHAR(25),
    -> deptId INT(11),
    -> salary FLOAT,
    -> CONSTRAINT fk_emp_dept1 FOREIGN KEY(deptId) REFERENCES tb_dept1(id)
    -> );
Query OK, 0 rows affected (0.00 sec)
```

- **使用非空约束**

`字段名 字段类型 NOT NULL`

```
mysql> CREATE TABLE tb_emp6 (
    -> id INT(11) PRIMARY KEY,
    -> name VARCHAR(25) NOT NULL,
    -> deptId INT(11),
    -> salary FLOAT,
    -> CONSTRAINT fk_emp_dept2 FOREIGN KEY(deptId) REFERENCES tb_dept1(id)
    -> );
Query OK, 0 rows affected (0.00 sec)
```

- **使用唯一约束**

#定义完列后直接指定唯一约束

`字段名 数据类型 UNIQUE`

```
mysql> CREATE TABLE tb_dept2 (
    -> id INT(11) PRIMARY KEY,
    -> name VARCHAR(25) UNIQUE,
    -> location VARCHAR(50)
    -> );
Query OK, 0 rows affected (0.00 sec)
```

#定义完所有列之后指定唯一约束

`[CONSTRAINT <约束名>] UNIQUE(<字段名>)`

```
mysql> CREATE TABLE tb_dept3 (
    -> id INT(11) PRIMARY KEY,
    -> name VARCHAR(25),
    -> location VARCHAR(50),
    -> CONSTRAINT STH UNIQUE(name)
    -> );
Query OK, 0 rows affected (0.00 sec)
```

- **使用默认约束**

`字段名 数据类型 DEFAULT 默认值`

```
mysql> CREATE TABLE tb_emp7 (
    -> id INT(11) PRIMARY KEY,
    -> name VARCHAR(25) NOT NULL,
    -> deptId INT(11) DEFAULT 111,
    -> salary FLOAT,
    -> CONSTRAINT fk_emp_dept3 FOREIGN KEY(deptId) REFERENCES tb_dept1(id)
    -> );
Query OK, 0 rows affected (0.00 sec)
```

- **设置表的属性自动增加**

`字段名 数据类型 AUTO_INCREMENT`

```
mysql> CREATE TABLE tb_emp8 (
    -> id INT(11) PRIMARY KEY AUTO_INCREMENT,
    -> name VARCHAR(25) NOT NULL,
    -> deptId INT(11),
    -> salary FLOAT,
    -> CONSTRAINT fk_emp_dept4 FOREIGN KEY(deptId) REFERENCES tb_dept1(id)
    -> );
Query OK, 0 rows affected (0.01 sec)
```

- **查看表基本结构**

`DESC/DESCRIBE <表名>;`

```
mysql> DESC tb_dept1;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| id       | int(11)     | NO   | PRI | NULL    |       |
| name     | varchar(25) | NO   |     | NULL    |       |
| location | varchar(50) | YES  |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)

mysql> DESC tb_emp7;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| id     | int(11)     | NO   | PRI | NULL    |       |
| name   | varchar(25) | NO   |     | NULL    |       |
| deptId | int(11)     | YES  | MUL | 111     |       |
| salary | float       | YES  |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

- **查看表的详细结构**

`SHOW CREATE TABLE <表名>\G`

```
mysql> SHOW CREATE TABLE tb_emp7\G
*************************** 1. row ***************************
       Table: tb_emp7
Create Table: CREATE TABLE `tb_emp7` (
  `id` int(11) NOT NULL,
  `name` varchar(25) NOT NULL,
  `deptId` int(11) DEFAULT '111',
  `salary` float DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `fk_emp_dept3` (`deptId`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8
1 row in set (0.01 sec)
```

3.2 修改数据表

- **修改表名**

`ALTER TABLE <旧表名> RENAME <新表名>;`

```
mysql> ALTER TABLE tb_dept3 RENAME tb_deptment3;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW TABLES;
+------------------+
| Tables_in_test_db |
+------------------+
| tb_dept1          |
| tb_dept2          |
| tb_deptment3      |
| tb_emp1           |
| tb_emp2           |
| tb_emp3           |
| tb_emp5           |
| tb_emp6           |
| tb_emp7           |
| tb_emp8           |
+------------------+
10 rows in set (0.00 sec)
```

- **修改字段的数据类型**

`ALTER TABLE <表名> MODIFY <字段名> <数据类型>;`

```
mysql> DESC tb_dept1;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| id       | int(11)     | NO   | PRI | NULL    |       |
| name     | varchar(25) | NO   |     | NULL    |       |
| location | varchar(50) | YES  |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)

mysql> ALTER TABLE tb_dept1 MODIFY name VARCHAR(30);
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESC tb_dept1;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| id       | int(11)     | NO   | PRI | NULL    |       |
| name     | varchar(30) | YES  |     | NULL    |       |
| location | varchar(50) | YES  |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

- **修改字段名**

`ALTER TABLE <表名> CHANGE <旧字段名> <新字段名> <新数据类型>;`

```
mysql> DESC tb_dept1;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| id       | int(11)     | NO   | PRI | NULL    |       |
| name     | varchar(30) | YES  |     | NULL    |       |
| location | varchar(50) | YES  |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)

mysql> ALTER TABLE tb_dept1 CHANGE location loc VARCHAR(66);
Query OK, 0 rows affected (0.00 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESC tb_dept1;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| id    | int(11)     | NO   | PRI | NULL    |       |
| name  | varchar(30) | YES  |     | NULL    |       |
| loc   | varchar(66) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

- **添加字段**

#添加无完整性约束条件的字段

`ALTER TABLE tb_dept1 ADD managerId INT(10);`

```
mysql> ALTER TABLE tb_dept1 ADD managerId INT(10);
Query OK, 0 rows affected (0.00 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESC tb_dept1;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| id        | int(11)     | NO   | PRI | NULL    |       |
| name      | varchar(30) | YES  |     | NULL    |       |
| loc       | varchar(66) | YES  |     | NULL    |       |
| managerId | int(10)     | YES  |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

#添加有完整性约束条件的字段

```
ALTER TABLE tb_dept1 ADD column1 VARCHAR(10) NOT NULL;
```

```
mysql> ALTER TABLE tb_dept1 ADD column1 VARCHAR(10) NOT NULL;
Query OK, 0 rows affected (0.00 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESC tb_dept1;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| id        | int(11)     | NO   | PRI | NULL    |       |
| name      | varchar(30) | YES  |     | NULL    |       |
| loc       | varchar(66) | YES  |     | NULL    |       |
| managerId | int(10)     | YES  |     | NULL    |       |
| column1   | varchar(10) | NO   |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

#在表的第一列添加一个字段

```
ALTER TABLE tb_dept1 ADD column2 VARCHAR(11) NOT NULL FIRST;
```

```
mysql> ALTER TABLE tb_dept1 ADD column2 VARCHAR(11) NOT NULL FIRST;
Query OK, 0 rows affected (0.00 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESC tb_dept1;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| column2   | varchar(11) | NO   |     | NULL    |       |
| id        | int(11)     | NO   | PRI | NULL    |       |
| name      | varchar(30) | YES  |     | NULL    |       |
| loc       | varchar(66) | YES  |     | NULL    |       |
| managerId | int(10)     | YES  |     | NULL    |       |
| column1   | varchar(10) | NO   |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

#在表的指定列之后添加一个字段

```
ALTER TABLE tb_dept1 ADD column3 VARCHAR(12) NOT NULL AFTER name;
```

```
mysql> ALTER TABLE tb_dept1 ADD column3 VARCHAR(12) NOT NULL AFTER name;
Query OK, 0 rows affected (0.00 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESC tb_dept1;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| column2   | varchar(11) | NO   |     | NULL    |       |
| id        | int(11)     | NO   | PRI | NULL    |       |
| name      | varchar(30) | YES  |     | NULL    |       |
| column3   | varchar(12) | NO   |     | NULL    |       |
| loc       | varchar(66) | YES  |     | NULL    |       |
| managerId | int(10)     | YES  |     | NULL    |       |
| column1   | varchar(10) | NO   |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
7 rows in set (0.00 sec)
```

- **删除字段**

`ALTER TABLE <表名> DROP <字段名>;`

```
mysql> ALTER TABLE tb_dept1 DROP column2;
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESC tb_dept1;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| id        | int(11)     | NO   | PRI | NULL    |       |
| name      | varchar(30) | YES  |     | NULL    |       |
| column3   | varchar(12) | NO   |     | NULL    |       |
| loc       | varchar(66) | YES  |     | NULL    |       |
| managerId | int(10)     | YES  |     | NULL    |       |
| column1   | varchar(10) | NO   |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

- **修改字段的排列位置**

`ALTER TABLE <表名> MODIFY <字段1> <数据类型> FIRST | AFTER <字段2>;`

#修改字段为表的第一个字段

```
mysql> ALTER TABLE tb_dept1 MODIFY column1 VARCHAR(10) FIRST;
Query OK, 0 rows affected (0.00 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESC tb_dept1;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| column1   | varchar(10) | YES  |     | NULL    |       |
| id        | int(11)     | NO   | PRI | NULL    |       |
| name      | varchar(30) | YES  |     | NULL    |       |
| column3   | varchar(12) | NO   |     | NULL    |       |
| loc       | varchar(66) | YES  |     | NULL    |       |
| managerId | int(10)     | YES  |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

#修改字段到表的指定列之后

```
mysql> ALTER TABLE tb_dept1 MODIFY column1 VARCHAR(10) AFTER loc;
Query OK, 0 rows affected (0.00 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESC tb_dept1;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| id        | int(11)     | NO   | PRI | NULL    |       |
| name      | varchar(30) | YES  |     | NULL    |       |
| column3   | varchar(12) | NO   |     | NULL    |       |
| loc       | varchar(66) | YES  |     | NULL    |       |
| column1   | varchar(10) | YES  |     | NULL    |       |
| managerId | int(10)     | YES  |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

- **更改表的存储引擎**

`ALTER TABLE <表名> ENGINE=<更改后的存储引擎名>;`

```
mysql> ALTER TABLE tb_dept1 ENGINE=MyISAM;
Query OK, 0 rows affected (0.00 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> SHOW CREATE TABLE tb_dept1\G
*************************** 1. row ***************************
       Table: tb_dept1
Create Table: CREATE TABLE `tb_dept1` (
  `id` int(11) NOT NULL,
  `name` varchar(30) DEFAULT NULL,
  `column3` varchar(12) NOT NULL,
  `loc` varchar(66) DEFAULT NULL,
  `column1` varchar(10) DEFAULT NULL,
  `managerId` int(10) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8
1 row in set (0.00 sec)
```

- **删除表的外键约束**

`ALTER TABLE <表名> DROP FOREIGN KEY <外键约束名>;`

```
mysql> SHOW CREATE TABLE tb_emp8\G;
*************************** 1. row ***************************
       Table: tb_emp8
Create Table: CREATE TABLE `tb_emp8` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(25) NOT NULL,
  `deptId` int(11) DEFAULT NULL,
  `salary` float DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `fk_emp_dept4` (`deptId`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8
1 row in set (0.00 sec)

ERROR:
No query specified

mysql> ALTER TABLE tb_emp8 DROP FOREIGN KEY fk_emp_dept4;
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> SHOW CREATE TABLE tb_emp8\G;
*************************** 1. row ***************************
       Table: tb_emp8
Create Table: CREATE TABLE `tb_emp8` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(25) NOT NULL,
  `deptId` int(11) DEFAULT NULL,
  `salary` float DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `fk_emp_dept4` (`deptId`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8
1 row in set (0.00 sec)
```

3.3 删除数据表

- 删除没有被关联的表

`DROP TABLE [IF EXISTS] 表1,表2,...,表n;`

```
mysql> SHOW TABLES;
+-------------------+
| Tables_in_test_db |
+-------------------+
| tb_dept1          |
| tb_dept2          |
| tb_deptment3      |
| tb_emp1           |
| tb_emp2           |
| tb_emp3           |
| tb_emp5           |
| tb_emp6           |
| tb_emp7           |
| tb_emp8           |
+-------------------+
10 rows in set (0.00 sec)

mysql> DROP TABLE IF EXISTS tb_emp8;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW TABLES;
+-------------------+
| Tables_in_test_db |
+-------------------+
| tb_dept1          |
| tb_dept2          |
| tb_deptment3      |
| tb_emp1           |
| tb_emp2           |
| tb_emp3           |
| tb_emp5           |
| tb_emp6           |
| tb_emp7           |
+-------------------+
9 rows in set (0.00 sec)
```

- **删除被其它表关联的主表**

#存在外键关联的情况下，直接删除父表会显示失败，原因是破坏表的参考完整性，需要先删除子表再删除父表。但是有点情况下需要保留子表，这时候需要先将外键约束条件取消，再删除父表即可。

<!-------但是我在Linux平台使用MySQL5.7居然可以直接删除有外键关联的父表？！等有时间了再测试一下看看--------!>