第一章 MySQL基础

`20:30上课！！！`

1.数据库基本概念

2.MySQL的初步了解

3.创建/删除数据库

4.创建表以及表的操作

1.数据库基本概念

`数据库管理系统`(DataBase-Management System,DBMS)由一个相互关联的数据的集合和一组用以访问这些数据的程序组成。这个数据集合通常称作`数据库`(database)。

设计数据库的目的是为了管理大量数据。或者换句话来说，`python`或者其他语言，也是可以管理数据的，我们可以把数据放在文本里面，然后去管理，也是可以做到的，但是缺点也是很明显的，这样会异常麻烦，当我们要查询某个数据的时候，要写很的代码，但是用数据库，一条`sql`语句就搞定啦，所以使用数据库能够极大的提高我们的工作效率，同时数据库还有提供`事务`的功能。

`关系数据库`基于`关系模型`，使用一系列表来表达数据以及这些数据之间的关系。`MySQL`就是`关系型数据库`。关系模型已经成为当今主要的数据模型，它比之前的的网络模型和层次模型简化了编程者的工作。现在开始流行的`NoSQL`，泛指非关系型的数据库。

2.MySQL的初步了解

`MySQL`是一个开放源代码的关系型数据库，有企业版和社区版，其体积小、速度快、总体拥有成本低，因此应用广泛。

`linux`上`mysql`配置文件及目录是`/etc/my.cnf`，`windows`上面是安装目录下的`my.ini`。如果需要改`MySQL`的一些设置就需要在配置文件里面去改。

想要进入数据库先要满足几个条件：

1.`mysql`服务已经开启。

2.你有用户名及密码。

`linux` 上查看服务：`service mysqld status`

`windows`上打开任务管理器查看服务

连接数据库

```
本地连接：
mysql -u用户名 -p
输入密码

远程连接：
mysql  -hIP地址  -P端口 -u用户 -p
输入密码
```

查看有哪些数据库：
`SHOW DATABASES;`

使用/进入某个数据库

`USE mysql;`

判断是否在哪个数据库里:
`SELECT DATABASE();`

查看当前用户：
`SELECT user();`

3.创建/删除数据库

`CREATE  {DATABASE | SCHEMA} [IF NOT EXISTS] db_name`

```
mysql> CREATE DATABASE `mydb`;
Query OK, 1 row affected (0.00 sec)
mysql> CREATE DATABASE `mydb`;
ERROR 1007 (HY000): Can't create database 'mydb'; database exists
mysql> CREATE DATABASE IF NOT EXISTS `mydb`;
Query OK, 1 row affected, 1 warning (0.00 sec)
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mydb               |
| mysql              |
| performance_schema |
| sys                |
| test               |
+--------------------+
6 rows in set (0.00 sec)
mysql>
```

删除数据库

`DROP {DATABASE | SCHEMA} [IF EXISTS] dbname;`

```
mysql> DROP DATABASE `mydb`;
Query OK, 0 rows affected (0.00 sec)
mysql> DROP DATABASE `mydb`;
ERROR 1008 (HY000): Can't drop database 'mydb'; database doesn't exist
mysql> DROP DATABASE IF EXISTS `mydb`;
Query OK, 0 rows affected, 1 warning (0.00 sec)
mysql>
```

注意：

```
MySQL 语句的规范
关键字与函数名称全部大写
数据库名称、表名称、字段名称全部小写，用反引号引起来
SQL语句必须以分号结尾
```

打开数据库

USE 数据库名称

进入mysql后，使用`SELECT DATABASE();`后会发现当前并没有进入到某个数据库中，需要使用`use`来进入某个数据库中。

4.创建表以及表的操作

查看数据库中的表

数据表（或称表）
是数据库最重要的组成部分之一，是其他对象的基础

查看数据表列表
`SHOW TABLES [FROM db_name]`

`SHOW TABLES`查看当前数据库中的数据表。

`SHOW TABLES FROM 'mysql'`查看`mysql`这个数据库中的数据表。

创建数据表

```
CREATE TABLE [IF NOT EXISTS] table_name(
   column_name data_type,
)
```

```
例：
mysql> CREATE TABLE `tb1`(
    -> `id` INT,
    -> `name` VARCHAR(20)
    -> );
Query OK, 0 rows affected (0.02 sec)
mysql>
```

数据类型：
INT 整数类型
VARCHAR 变长字符串

查看创建的表:

`SHOW CREATE TABLE tb_name;`

```
例：
mysql> SHOW CREATE TABLE `tb1`;
************** 1. row **********************
       Table: tb1
Create Table: CREATE TABLE `tb1` (
  `id` int(11) DEFAULT NULL,
  `name` varchar(20) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8
1 row in set (0.00 sec)
mysql>
```

查看数据表结构
`DESCRIBE tb_name;`
`SHOW COLUMNS FROM tb_name;`

```
例：
mysql> DESCRIBE `tb1`;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| id    | int(11)     | YES  |     | NULL    |       |
| name  | varchar(20) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

删除数据表

`DROP TABLE tablename;`

删除数据

DELETE FROM tablename WHERE condition

DELETE FROM tablename ORDER BY c1, c2, ...

LIMT row_count

修改数据表结构

修改字段属性

```
修改字段属性
alter table chengyu modify column reference VARCHAR(120);
```

添加单列：

```
ALTER TABLE tb1_name ADD [COLUNM] col_name
column_definition [FIRST|AFTER col-name]
```

例：

```
mysql> ALTER TABLE `tb1`
    -> ADD `age` INT
    -> ;
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0
mysql> ALTER TABLE `tb1`
    -> ADD `number` INT FIRST
    -> ;
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

添加多列：

```
ALTER TABLE tbl_name ADD [COLUMN]
(col_name column_definition,...)
```

例：

```
mysql> ALTER TABLE `tb1`
    -> ADD (`aa` INT,
    ->      `bb` INT,
    ->      `cc` INT
    -> );
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0
mysql>
```

删除数据表中的列

```
ALTER TABLE tbl_name DROP [COLUMN] col_name ;
```

例：

```
mysql> ALTER TABLE `tb1`
    -> DROP `aa`
    -> ;
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0
mysql> ALTER TABLE `tb1`
    -> DROP `bb`,
    -> DROP `cc`
    -> ;
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

向数据表中输入数据

`INSERT [INTO] tb1_name [(col_name,..)] VALUES(val,...)`

```
例：
mysql> INSERT INTO `tb1`(`id`,`name`)
    -> VALUES(1,'rose'),
    ->       (2,'daxia')
    -> ;
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM `tb1`;
+------+------+--------+------+
| id   | name | gender | age  |
+------+------+--------+------+
|    1 | rose | f      |   18 |
| NULL | taka | NULL   |   18 |
+------+------+--------+------+
2 rows in
```

查看表中的数据

SELECT * FROM tablename

SELECT col_name, col_name... FROM tablename

mysql 报错解决方法：

ERROR 1045 (28000): Access denied for user 'root'@'localhost' (using password: YES)

输入 mysql -uroot -proot

更新数据

UPDATE tablename SET col_name1='', col_name2='' WHERE id='';

MySQL 如何删除有外键约束的表数据

在MySQL中删除一张表或一条数据的时候，出现

[Err] 1451 -Cannot **delete**or**update**a parent row:

a**foreign****keyconstraint****fails (...)

这是因为MySQL中设置了foreign key关联，造成无法更新或删除数据。可以通过设置FOREIGN_KEY_CHECKS变量来避免这种情况。

禁用外键约束，我们可以使用:

SET FOREIGN_KEY_CHECKS=0;

然后再删除数据

启动外键约束，我们可以使用:

SET FOREIGN_KEY_CHECKS=1;

查看当前**FOREIGN_KEY_CHECKS**的值，可用如下命令：

```
SELECT @@FOREIGN_KEY_CHECKS;
```

第二章 MySQL表约束

```
20:30上课!!!
```

1.表结构操作

2.非空约束

3.唯一约束

4.主键约束

5.自增长

6.默认约束

7.外键约束

1.表结构操作

```
ALTER TABLE 'tbname'
```

增加：ADD
删除：DROP
修改：MODIFY #改列的数据类型

CHANGE #改列名和数据类型

RENAME #改表名

```
#ADD
mysql> ALTER TABLE `tb1`
    -> ADD (`age` INT,
    -> `number` INT);

#DROP
mysql> ALTER TABLE `tb1`
    -> DROP `number`;

#MODIFY  重点
mysql> ALTER TABLE `tb1`
    -> MODIFY `age` VARCHAR(4);

#CHANGE
mysql> ALTER TABLE `tb1`
    -> CHANGE `age` `ages` INT;

#RENAME
mysql> ALTER TABLE `tb1` RENAME `tb2`;

#表名和字段名，尽量避免修改，即便是在封装得很好的情况下也要修改代码，如果是上线的东西就更加不要修改
```

2.非空约束

数据库字段的某个值是否可以为空，`NULL` 字段值可以为空，`NOT NULL` 字段值不能为空。

```
mysql> DESC `tb2`;
mysql> SELECT * FROM `tb2`;

mysql> INSERT INTO `tb2`(`id`)
    -> VALUES(4);
#允许为空，所以字段不添加值也可以插入

mysql> ALTER TABLE `tb2`
    -> MODIFY `id` INT NOT NULL;
mysql> DESC `tb2`;
mysql> INSERT `tb2`(`name`) VALUES('haha');
ERROR 1364 (HY000): Field 'id' doesn't have a default value
#在限定字段不能为空之后，插入数据的时候就必须插入数据，否则就会报错
```

3.唯一约束

字段添加唯一约束之后，该字段的值不重复，也就是该字段的值在该表中唯一`unique key`

```
#添加唯一约束
ALTER TABLE tbl_name ADD [CONSTRAINT[symbol]]
UNIQUE [INDEX|KEY] [index_name] [index_type]
(index_col_name)

#删除唯一约束

ALTERT TABLE tbl_name DROP {INDEX|KEY} index_name
```

```
mysql> ALTER TABLE `tb2` ADD UNIQUE(`id`);
mysql> INSERT INTO `tb2`(`id`)
    -> VALUES(1);
ERROR 1062 (23000): Duplicate entry '1' for key 'id'
#已经添加的值不能再重复的插入

#删除唯一约束
mysql> ALTER TABLE  `tb2` ADD UNIQUE `name_a` (`name`);
mysql> ALTER TABLE `tb2` DROP KEY `name_a`;
```

4.主键约束

主键保证记录的唯一性，主键自动为`NOT NULL`
每张数据表只能存在一个主键
`NOT NULL + UNIQUE KEY`

一个`UNIQUE KEY`又是一个`NOT NULL`的时候，那么它被当做`PRIMARY KEY`主键
当一张表里没有一个主键的时候，第一个出现的非空且为唯一的列被视为有主键。

```
#添加主键约束
ALTER TABLE tbl_name ADD [CONSTRAINT[sysbol]]
PRIMARY KEY [index_type] (index_col_name)

#删除主键约束
ALTER TABLE tbl_name DROP PRIMARY KEY
```

```
mysql> DESCRIBE `tb2`;
#id没有申明它为主键，但是它有唯一约束和非空约束，就会默认为主键

#在创建表时声明为主键
mysql> CREATE TABLE `user`(
    -> `id` INT PRIMARY KEY,
    -> `name` VARCHAR(10),
    -> `number` INT
    -> );
mysql> DESC `user`;

#删除主键
mysql> ALTER TABLE `user` DROP PRIMARY KEY;   #保留非空特性
mysql> DESC `user`;

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

#插入值
mysql> INSERT INTO `user`(`id`,`name`,`number`)
    -> VALUES(1,'budong',1),
    -> (3,'tuple',3);

#再次插入就会报主键冲突
mysql> INSERT INTO `user`(`id`,`name`,`number`) VALUES(1,'budong',1), (3,'tuple',3);
ERROR 1062 (23000): Duplicate entry '1' for key 'PRIMARY'

#大家要学会看错误码 https://wenku.baidu.com/view/f5ed2897250c844769eae009581b6bd97f19bc9d
```

5.自增长`AUTO_INCREMENT`

`AUTO_INCREMENT` 自动编号，且必须与主键组合使用
默认情况下，起始值为1，每次的增量为1。
当插入记录时，如果为`AUTO_INCREMENT`数据列明确指定了一个数值，则会出现两种情况，

情况一，如果插入的值与已有的编号重复，则会出现出错信息，因为AUTO_INCREMENT数据列的值必须是唯一的；

情况二，如果插入的值大于已编号的值，则会把该插入到数据列中，并使在下一个编号将从这个新值开始递增。也就是说，可以跳过一些编号。如果自增序列的最大值被删除了，则在插入新记录时，该值被重用。

```
#添加自增长
mysql> ALTER TABLE `user` CHANGE `id` `id` INT NOT NULL AUTO_INCREMENT;
mysql> DESCRIBE `user`;
mysql> SELECT * FROM `user`;

#插入值
mysql> INSERT INTO `user`(`name`,`number`)
    -> VALUES('take',2),
    -> ('which',4)
    -> ;
mysql> SELECT * FROM `user`;

#删除自增长
mysql> ALTER TABLE `user` CHANGE `id` `id` INT NOT NULL ;
mysql> DESCRIBE `user`;

#一个表里只能有一个自增，所以一般和主键联用
```

6.默认约束

`DEFAULT`（默认约束）
初始值设置，插入记录时，如果没有明确为字段赋值，则自动赋予默认值。

```
#添加/删除默认约束
ALTER TABLE tbl_name ALTER [COLUMN] col_name
{SET DEFAULT literal | DROP DEFAULT}
```

```
#设置默认值
mysql> ALTER TABLE `user` ALTER `number` SET DEFAULT 0;
mysql> DESCRIBE `user`;

#插入值
mysql> ALTER TABLE `user` CHANGE `id` `id` INT NOT NULL AUTO_INCREMENT;
mysql> INSERT INTO `user`(`name`) VALUES('rock');
mysql> SELECT * FROM `user`;
```

7.外键约束`FOREING KEY`

```
#外键约束要求数据表的存储引擎只能为InnoDB

#查看当前mysql服务器支持的存储引擎
SHOW ENGINES;

#编辑数据表的默认存储引擎

MySQL配置文件  ->/etc/my.cnf

[mysqld]
default-storage-engine=INNODB

#改完配置文件要重启服务
sudo service mysqld restart
```

```
mysql> CREATE TABLE `department`(
    -> `d_id` INT PRIMARY KEY AUTO_INCREMENT,
    -> `name` VARCHAR(20) NOT NULL
    -> );
Query OK, 0 rows affected (0.02 sec)

mysql> CREATE TABLE `students`(
    -> `s_id` INT PRIMARY KEY AUTO_INCREMENT,
    -> `name` VARCHAR(20) NOT NULL,
    -> `dept_id` INT,
    -> FOREIGN KEY(`dept_id`) REFERENCES `department`(`d_id`)
    -> );
Query OK, 0 rows affected (0.02 sec)
```

第三章 MySQL表关系

`20:30上课！！！`

1.数据的增删改

2.外键约束要求

3.表关系

4.外键约束的参照操作

5.数据库的三范式

1.数据的增删改

插入数据

方法一：

```
INSERT [INTO] table_name [(column_name,...)]

{VALUES|VALUE} ({expr|DEFAULT},...),(...),...;   #使用[]表示可选项，{}表示二选一
```

方法二：

```
INSERT [INTO] tbl_name SET col_name={expr|DEFAULT},...;
```

```
mysql> CREATE TABLE `information`(
    -> `id` INT PRIMARY KEY AUTO_INCREMENT,
    -> `address` VARCHAR(20) NOT NULL,
    -> `number` INT DEFAULT 1
    -> );

mysql> INSERT INTO `information`(`address`,`number`)
    -> VALUES ('changsha',0731),
    -> ('guangzhou',DEFAULT)  #可以使用默认值
    -> ;

mysql> INSERT INTO `information` SET `address`='beijing' ,`number`=010;

#这两种插入方法，推荐大家使用第一种，使用更多
```

更新数据

```
UPDATE  tb_name
SET col_name1={expr1|DEFAULT}[,col_name2={expr2|DEFAULT}]...
[WHERE where_condition];
```

```
mysql> SELECT * FROM `information`;

mysql> UPDATE `information` SET `address` ='shanghai',`number`=021
    -> WHERE `id`=10;
#虽然where是可选的，但是在实际当中，where往往是必须带上的，很少有情况更新整张表的某个字段，切记切记！！

mysql> CREATE TABLE `province`(
    -> `id` INT PRIMARY KEY AUTO_INCREMENT,
    -> `province` VARCHAR(20) NOT NULL,
    -> `capital` VARCHAR(20) NOT NULL,
    -> `code` INT DEFAULT 01
    -> );

mysql> INSERT INTO `province`(`province`,`capital`,`code`)
    -> VALUES('Beijing','beijing',01),
    -> ('Hunan','changsha',0731);

mysql> UPDATE `information` i ,`province` p
    -> SET i.`address` = p.`province`
    -> WHERE i.`number` = p.`code`;

mysql> SELECT * FROM `information`;
#以上是用一张表数据更新另一张表的基本用法，在Oracle和DB2中有个merge into，当更新的数据量相当大的时候，上面的方式会很慢，但是使用merge into却要快很多，大家今后如果使用到这两个数据库的时候，可以去了解一下。
```

删除数据

```
DELETE FROM tbl_name [WHERE where_conditon];
```

不添加WHERE则会删除全部记录

```
mysql>SELECT * FROM `tb2`;

mysql> DELETE FROM `tb2` WHERE `id`=4;
mysql> DELETE FROM `tb2` WHERE `id` IN (4,5);
#同样的，删除的数据的时候一定要带上where，否则就是整张表都清空！

mysql> SELECT * FROM `tb2`;
mysql> TRUNCATE TABLE `tb2`;
mysql> SELECT * FROM `tb2`;
#这里的TRUNCATE是清空的意思，这里是清空表，不返回清楚的数据，执行非常快，常用在临时表里面。
```

2.外键约束要求

外键约束`FOREIGN KEY`，保持数据一致性，完整性实现一对一或一对多关系。

外键约束的要求：

数据表的存储引擎只能为InnoDB
外键列和参照列数据类型一致
外键必须关联到键上面去

```
#添加外键的格式:
ALTER TABLE yourtablename  #需要添加外键的表
    ADD [CONSTRAINT 外键名] FOREIGN KEY [id] (index_col_name, ...)
    REFERENCES tbl_name (index_col_name, ...)
    [ON DELETE {CASCADE | SET NULL | NO ACTION | RESTRICT}]
    [ON UPDATE {CASCADE | SET NULL | NO ACTION | RESTRICT}]

 CASCADE     删除包含与已删除键值有参照关系的所有记录
 SET NULL    修改包含与已删除键值有参照关系的所有记录，使用NULL值替换（只能用于已标记为NOT NULL的字段）
 RESTRICT    拒绝删除要求，直到使用删除键值的辅助表被手工删除，并且没有参照时(这是默认设置，也是最安全的设置)
 NO ACTION   啥也不做
```

一对多关系

`一对多`与`多对一`是一个概念，指的是一个实体的某个数据与另外一个实体的多个数据有关联关系。

举例，学校中一个学院可以有很多的学生，而一个学生只属于某一个学院（通常情况下），学院与学生之间的关系就是一对多的关系，通过外键关联来实现这种关系。

```
#创建学院表：
mysql> CREATE TABLE `department`(
    -> `id` INT PRIMARY KEY AUTO_INCREMENT,
    -> `name` VARCHAR(10) NOT NULL,
    -> `code` INT NOT NULL
    -> );
#创建学生表
mysql> CREATE TABLE `student`(

    -> `id` INT PRIMARY KEY AUTO_INCREMENT,
    -> `name` VARCHAR(10) NOT NULL,
    -> `dep_id` INT,
    -> CONSTRAINT `stu_dep_for_key` FOREIGN KEY (`dep_id`) REFERENCES `department`(`id`) ON
DELETE RESTRICT
    -> );

#插入数据
mysql> INSERT INTO `department`(`name`,`code`)
    -> VALUES('理学院',01),
    -> ('计算机学院',02)
    -> ;
mysql> SELECT * FROM `department`;

mysql> INSERT INTO `student`(`name`,`dep_id`)
    -> VALUES('budong',1),
    -> ('Tuple',1),
    -> ('Which',2);
mysql> SELECT * FROM `student`;
```

一对一关系

举例，学生表中有学号、姓名、学院，但学生还有些比如电话，家庭住址等比较私密的信息，这些信息不会放在学生表当中，会新建一个学生的详细信息表来存放。这时的学生表和学生的详细信息表两者的关系就是一对一的关系，因为一个学生只有一条详细信息。用外键加主键的方式来实现这种关系。

```
mysql> DESCRIBE `student`;

#建立学生的详细信息表
mysql> CREATE TABLE `student_details`(
    -> `id` INT PRIMARY KEY,
    -> `id_card` INT NOT NULL UNIQUE KEY,
    -> `telephone` INT,
    -> CONSTRAINT `stu_deta_for_key` FOREIGN KEY (`id`) REFERENCES `student`(`id`) ON DELETE
CASCADE
    -> );
#插入数据
mysql> INSERT INTO `student_details`(`id`,`id_card`,`telephone`)
    -> VALUES(1,4301,133),
    -> (2,4302,133);
#这里信息一一对应，所以一般会同步插入
```

多对多关系

一个实体的数据对应另外一个实体的多个数据，另外实体的数据也同样对应当前实体的多个数据。

举例，学生要报名选修课，一个学生可以报名多门课程，一个课程有很多的学生报名，那么学生表和课程表两者就形成了多对多关系。对于多对多关系，需要创建第三张关系表，关系表中通过外键加主键的形式实现这种关系。

```
#创建课程表
mysql> CREATE TABLE `course`(
    -> `id` INT PRIMARY KEY AUTO_INCREMENT,
    -> `name` VARCHAR(20) NOT NULL
    -> );

mysql> CREATE TABLE `select`(
    -> `stu_id` INT,
    -> `coures_id` INT,
    -> PRIMARY KEY(`stu_id`,`coures_id`),
    -> CONSTRAINT `select_stu_id_for_key` FOREIGN KEY (`stu_id`) REFERENCES `student`(`id`),
    -> CONSTRAINT `select_coures_id_for_key` FOREIGN KEY (`coures_id`) REFERENCES `course`(`id`)
    -> );
```

4.外键约束的参照操作：

```
1.CASCADE   从父表删除或更新时自动删除或更新子表中的匹配行
2.SET NULL  从父表删除或更新行时，设置子表中的外键列为NULL。如果使用该选项，必须保证子表列没有指定NOT
NULL
3.RESTRICT  拒绝对父表的删除或更新操作
4.NO ACTION 标准的SQL关键字，在mysql中与RESTRICT作用相同
```

```
on delete RESTRICT   #拒绝父表删除，当子表中有记录关联，则父表中不能删除该记录，除非子表更改关联记录
on update CASCADE    #父表删除，子表对应记录也删除，父表更新，子表也更新

#这里的删除更新值的是子表关联父表中的父表字段，不是父表所有的字段都不能更新
```

```
mysql> SELECT * FROM `department`;
+----+-----------------+------+
| id | name            | code |
+----+-----------------+------+
|  1 | 理学院          |    1 |
|  2 | 计算机学院      |    2 |
+----+-----------------+------+

mysql> SELECT * FROM `student`;
+----+--------+--------+
| id | name   | dep_id |
+----+--------+--------+
|  1 | budong |      1 |
|  2 | Tuple  |      1 |
|  3 | Which  |      2 |
+----+--------+--------+

mysql> DELETE FROM `department` WHERE `id`=1;
ERROR 1451 (23000): Cannot delete or update a parent row: a foreign key constraint fails
(`mydb`.`student`, CONSTRAINT `stu_dep_for_key` FOREIGN KEY (`dep_id`) REFERENCES `department`
(`id`))
#因为外键关联的原因，不能删除父表中的记录

mysql> UPDATE `student` SET `dep_id`=1 WHERE `id`=3;
mysql> DELETE FROM `department` WHERE `id`=2;
#因为student表中的dep_id没有值关联department中的id=2的数据，所以可以删除department表中id=2的数据
```

5.数据库的三范式

第一范式（1NF）：符合1NF的关系中的每个属性都不可再分。1NF是所有关系型数据库的最基本要求。

第二范式（2NF）：2NF在1NF的基础之上，消除了非主属性对于码的部分函数依赖。

第三范式（3NF）：3NF在2NF的基础之上，消除了非主属性对于码的传递函数依赖。

> 在上面的department和student表中，虽然可以把两个表合在一起，但是那样就会造成数据的冗余，不满足第一范式。
> 第二范式简单的说就是表里的每列与主属性是完全依赖，如果有部分依赖应该分离出来，形成一对多的关系。
> 第三范式简单的说就是所有的非主属性只在整个数据库里面出现一次。
> 这个解释是不严格的解释，讲三范式是告诉大家一个方向，如果今后大家要设计表结构的时候，应该从哪些方向去考虑。

总结

数据的增删改，相对用得最多的是增改，大家要熟练地写出来。

表关系我们常用外键来表示，但是也不是说非得用外键才能表示变关系，这个根据实际情况来灵活处理的，在是用外键的时候要记住外键的使用条件：`数据表的存储引擎只能为InnoDB`、`外键列和参照列数据类型一致`和`外键必须关联到键上面去`

外键的好处与不方便的地方都是在于外键约束，外键约束合理的使用能够保证数据的一致性，但是滥用同样会带来很多的麻烦。

数据库的三范式是设计数据库的基本思路，它的作用就是减少数据的冗余，减少脏数据的的出现，保证数据的稳健。除此之外，还有4NF、5NF和BCNF，这些都是数据库设计表结构的时候用的，如果有需要，大家可以尝试着去学习一下。

第四章 MySQL查询

`20:30上课！！！`

1.单表查询

2.多表查询

3.MySQL函数

4.查询SQL的优化

1.单表查询

查询需要数据，我们先插入一些数据。

```
mysql> SELECT * FROM `department`;
mysql> SELECT * FROM `student`;
mysql> SELECT * FROM `student_details`;
mysql> SELECT * FROM `course`;
mysql> SELECT * FROM `select`;

#学院表添加数据
mysql> INSERT INTO `department`(`name`,`code`)
    -> VALUES('计算机学院',2),
    -> ('通信学院',3),
    -> ('外国语学院',4),
    -> ('化工学院',5);

#学生表添加数据
mysql> INSERT INTO `student`(`name`,`dep_id`)
    -> VALUES('小明',3),
    -> ('小明爸',4),
    -> ('小明妈',5),
    -> ('小新',6);

#学生详细信息表添加数据
mysql> INSERT INTO `student_details`(`id`,`id_card`,`telephone`)
    -> VALUES(3,4303,137),
    -> (8,4304,155),
    -> (9,4305,136),
    -> (10,3701,158),
    -> (11,2301,188);

#课程表添加数据
mysql> INSERT INTO `course`(`name`)
    -> VALUES('心理学'),
    -> ('音乐鉴赏'),
    -> ('近代史'),
    -> ('影视鉴赏');

#选课表里里面插入数据
mysql> INSERT INTO `select`(`stu_id`,`coures_id`)
    -> VALUES(8,1),
    -> (8,2),
    -> (8,3),
    -> (9,2),
    -> (9,4),
    -> (10,1),
    -> (10,3),
    -> (10,2),
    -> (11,1),
    -> (11,3);
```

查询所有记录

```
SELECT * FROM  tb_name;
```

查询选中列记录

`SELECT col_name1,col_name2 FROM tb_name;`

查询指定条件下的记录

`SELECT col_name FROM  tb_name  WHERE 条件`

查询后为列取别名

`SELECT  col_name  AS  new_name  FROM  tab_name`

```
#查询所以记录
mysql> SELECT * FROM `student`;

#查询选中的记录
mysql> SELECT `name` FROM `student`;

#查询指定条件下的记录
mysql> SELECT * FROM `student` WHERE `id`>3;

#查询中的别名
mysql> SELECT s.`name` AS 姓名,s.`dep_id` 学院id FROM `student` s;
#可以给查询的表命名别名，也可以给查的字段重命名，最好养成给表命别名的习惯，尤其在多表查询的时候
```

2.多表查询

联表查询join

内连接 `[INNER| CROSS] JOIN`

在`MySQL`里面`inner join`和`cross join`是一样的，只是写法有点不一样，当使用`JOIN`时，默认是`inner join`

```
mysql> SELECT * FROM `department` INNER JOIN `student`;
mysql> SELECT * FROM `department` , `student`;
mysql> SELECT * FROM `department` CROSS JOIN `student`;
#这三个是都是等价的，常用上两种
```

上面的是`无条件连接`，又名`交叉连接`/`笛卡尔连接`，第一张表种的每一向会和另一张表的每一项依次组合。

除了无条件连接外，还有 有条件连接，只需要在无条件连接后面加上ON子句，就是有条件，这是时候数据会根据连接条件来进行连接。

```
mysql> SELECT * FROM `department` INNER JOIN `student` ON `department`.`id` = `student`.`dep_id`;

mysql> SELECT * FROM `department` p , `student` s WHERE  p.`id`=s.`dep_id`;

mysql> SELECT * FROM `department` p CROSS JOIN `student` s WHERE p.`id` = s.`dep_id`;
```

左连接 `LEFT JOIN`

`A LEFT JOIN B`会以左边的表为主，展式左边表的所有数据，展式右边表中符合`ON`子句中条件的数据，没有为空。

```
mysql> SELECT * FROM `student` s LEFT JOIN `student_details` sd ON s.`id`=sd.`id`;
mysql> DELETE FROM `student_details` WHERE `id`=11;
mysql> SELECT * FROM `student` s LEFT JOIN `student_details` sd ON s.`id`=sd.`id`;

#注意以下两条SQL的区别
mysql> SELECT * FROM `student` s LEFT JOIN `student_details` sd ON s.`id`=sd.`id` AND sd.`id` IS
NULL;
mysql> SELECT * FROM `student` s LEFT JOIN `student_details` sd ON s.`id`=sd.`id` WHERE sd.`id`
IS NULL;
#在 ON 后的时候，是在表关联的时候执行的，在 WHERE 后面是在关联之后判断的，执行顺序会在后面讲解
```

右连接 `RIGHT JOIN`

右连接和左连接类似，只是作用相反

```
mysql> SELECT * FROM `student` s RIGHT JOIN `student_details` sd ON s.`id`=sd.`id`;
#这里展式出所有 `student_details` 中的数据，`student` 中多出的数据不会展式
```

全连接 `UNION`

`UNION` 用于合并两个或多个 `SELECT` 语句的**结果集**，并**消去**表中任何**重复行**。

```
mysql> SELECT s.`name`,c.`name` FROM `student` s LEFT JOIN `select` se ON s.`id`=se.`stu_id` LEFT
JOIN `course` c ON c.`id`=se.`coures_id`
    -> UNION
    -> SELECT s.`name`,c.`name` FROM `student` s LEFT JOIN `select` se ON s.`id`=se.`stu_id` LEFT
JOIN `course` c ON c.`id`=se.`coures_id`
    -> ;
#两个同样的SQL，但是数据没有重复的，如果如要重复的，使用 UNION ALL 即可。
```

子表查询

在一个SQL语句中出现两个SQL语句，就是子表查询

```
SELECT
    s.`name`,
    e.`name`
FROM
    `student` s
LEFT JOIN(
        SELECT
            se.`stu_id`,
            c.`name`
        FROM
            `select` se
        JOIN `course` c ON se.`coures_id` = c.`id`
    ) e ON s.`id`=e.`stu_id`
#LEFT JOIN 里面也有一个查询语句，这种格式，希望大家能够记住，今后写子查询的时候也用这种格式

SELECT
    *
FROM
    `student` s
WHERE
    s.`dep_id` = (
        SELECT
            `id`
        FROM
            `department` d
        WHERE
            d.`name` = '外国语学院'
    )
#子查询不但可以放在JOIN 哪里，也可以放在 WHERE 后面
```

查询的其他操作

排序 `ORDER BY`

我们可以对查询出来的结果进行排序， `ASC` 升序(默认) `DESC` 降序

```
mysql> SELECT * FROM `student` s JOIN `student_details` sd ON s.`id`=sd.`id` ORDER BY s.`id`;

mysql> SELECT * FROM `student` s JOIN `student_details` sd ON s.`id`=sd.`id` ORDER BY  s.`id`
DESC;
```

限制显示的行数 `LIMIT`

```
mysql> SELECT * FROM `student` s JOIN `student_details` sd ON s.`id`=sd.`id` ORDER BY  s.`id`
DESC LIMIT 3;
#限制显示出的记录数
```

分组查询 `GROUP BY`

```
mysql> SELECT d.`id`,d.`name`,COUNT(*) FROM `department` d LEFT JOIN `student` s ON
d.`id`=s.`dep_id` GROUP BY d.`id`,d.`name`;

#分组是个常见的操作，常用于分组统计，使用GROUP BY后，会按照GROUP BY后面的字段进行分组，且必须是明确的字段，不能是*，因此SELECT后面也不能是*

mysql> SELECT d.`id`,d.`name`,COUNT(*) FROM `department` d LEFT JOIN `student` s ON
d.`id`=s.`dep_id` GROUP BY d.`id`,d.`name` HAVING COUNT(*)>1;

#使用 HAVING 可以对分组之后的结果进行筛选，注意：HAVING 后的字段必须是SELECT后出现过的
```

3.MySQL函数

MySQL同样也有很多的函数，这些函数可以方便的我们处理一些我们查询出来的数据，合理使用这些函数，可以减少我们的代码，对于我们拿的数据进行处理提供更加便捷的方式，但是MySQL的函数也是十分多的，我这里举几个常见的。

```
#处理NULL，把NULL处理成自己指定的数据
mysql> SELECT s.`name`,IFNULL(sd.`id_card`,430) FROM `student` s LEFT JOIN `student_details` sd
ON s.`id`=sd.`id`;
#在Oracle和DB2中使用的NVL，用法一样

#字符串长度截取，left是从左边开始截取，right是用右边开始截取
mysql> SELECT LEFT(s.`name`,2),IFNULL(sd.`id_card`,430) FROM `student` s LEFT JOIN
`student_details` sd ON s.`id`=sd.`id`;

mysql> SELECT RIGHT(s.`name`,2),IFNULL(sd.`id_card`,430) FROM `student` s LEFT JOIN
`student_details` sd ON s.`id`=sd.`id`;
#SUBSTRING也是字符截取，但是可以指定其实和结束的位置
mysql> SELECT SUBSTRING(s.`name`,2,5),IFNULL(sd.`id_card`,430) FROM `student` s LEFT JOIN
`student_details` sd ON s.`id`=sd.`id``;

#字符串拼接
mysql> SELECT CONCAT(s.`name`,sd.`id_card`) FROM `student` s LEFT JOIN `student_details` sd ON
s.`id`=sd.`id`;

#类型转换CAST
mysql> SELECT CAST(sd.`id_card` AS CHAR) FROM `student` s LEFT JOIN `student_details` sd ON
s.`id`=sd.`id`;

#CONVERT也是类型转换
mysql> SELECT CONVERT(s.`dep_id`,SIGNED) FROM `student` s ;

#DAY时间函数
mysql> SELECT DAY('2017-08-18')-DAY('2017-08-01');
#查看当前时间
mysql> SELECT NOW();
#时间函数也是用的比较多的函数，大家需要处理时间格式的时候，在晚上找下资料一般都能解决
```

`MySQL`的函数还有很多，许多也和我们平常用的一样，比如`ABS`,`MAX`,`MIN`,`ROUND`,`AVG`,`SUM`等等，用法也是一样的，大家如果有需求，直接查下资料就可以，如果需要写原生的`SQL`，能够用`MySQL`的函数处理的就尽量用`MySQL`自带的函数。

4.查询SQL的优化

MySQL的执行顺序

```
1.FORM: 对FROM的左边的表和右边的表计算笛卡尔积。产生虚表VT1

2.ON: 对虚表VT1进行ON筛选，只有那些符合<join-condition>的行才会被记录在虚表VT2中。

3.JOIN: 如果指定了OUTER JOIN（比如left join、 right join），那么保留表中未匹配的行就会作为外部行添加到虚拟表VT2中，产生虚拟表VT3, rug from子句中包含两个以上的表的话，那么就会对上一个join连接产生的结果VT3和下一个表重复执行步骤1~3这三个步骤，一直到处理完所有的表为止。

4.WHERE: 对虚拟表VT3进行WHERE条件过滤。只有符合<where-condition>的记录才会被插入到虚拟表VT4中。

5.GROUP BY: 根据group by子句中的列，对VT4中的记录进行分组操作，产生VT5.

6.CUBE | ROLLUP: 对表VT5进行cube或者rollup操作，产生表VT6.

7.HAVING: 对虚拟表VT6应用having过滤，只有符合<having-condition>的记录才会被 插入到虚拟表VT7中。

8.SELECT: 执行select操作，选择指定的列，插入到虚拟表VT8中。

9.DISTINCT: 对VT8中的记录进行去重。产生虚拟表VT9.

10.ORDER BY: 将虚拟表VT9中的记录按照<order_by_list>进行排序操作，产生虚拟表VT10.

11.LIMIT：取出指定行的记录，产生虚拟表VT11, 并将结果返回。
```

通过上面的执行顺序不难想到，要想SQL执行更快，就必须把筛选条件尽量的往前面放。如下：

```
SELECT
    s.`name`,
    e.`name`
FROM
    `student` s
LEFT JOIN(
        SELECT
            se.`stu_id`,
            c.`name`
        FROM
            `select` se
        JOIN `course` c ON se.`coures_id` = c.`id`
    ) e ON s.`id`=e.`stu_id`


SELECT
    *
FROM
    `student` s
WHERE
    s.`dep_id` = (
        SELECT
            `id`
        FROM
            `department` d
        WHERE
            d.`name` = '外国语学院'
    )
```

在这两个例子中，第一个SQL中的子表只会被查询一次，但是在第二个SQL中，子表会被执行n次，这个n取决student表中的数据条数，如果子表的数据量很大的话，那么SQL的执行速度会十分慢。

这是典型的通过执行顺序来优化SQL，除此之外，要想SQL执行快一点，应该尽量避免模糊匹配，如：like,in,not in等这些匹配条件。

还有几点建议给大家：

1.尽量避免整表扫描，如SELECT *

2.建立合适的索引

3.使用合适的存储引擎

4.在JOIN中，尽量用小表LEFT JOIN 大表

5.除非十分必要，尽量不要使用ORDER BY,GROUP BY 和 DISTINCT(去重)，尽量用索引来代替

总结

数据的查询是使用数据库最基本，最常用的操作，大家一定要十分熟练，尤其是多表查询的时候，一点要先弄清楚业务需求，再去写相应的`SQL`语句。

在查询过程中，大家可以采用先分再总的顺序来查询，先分多个`SQL`把各个表中需要的数据查询出来，在把这些数据组合起来，在组合的过程用，合理的使用各种`JOIN`方法，最常用的是`LEFT JOIN`。

SQL的优化是数据库里面永恒的话题，几乎是无止境的，上面介绍的是几点基本的原则，想要写出好的SQL，需要大家不断的积累和总结。

数据库的后续学习建议

我们这几天的学习，基本上把数据库里面基本的操作都学习了一遍，其实也不多，多操作就好，但是想做DBA还远远不够，加上每个数据库都不一样，一般的DBA只能是某个数据库或者一两个数据库的DBA，不过只是单纯的想更加深入的学习数据库可以先去理解数据库的索引，视图等功能，之后再考虑分区分页等存储问题，再后面可以考虑主从，分布式的实现方法和原理，这些是对有兴趣的同学给个参考的学习路线，希望大家都能够熟练的使用数据库，写出漂亮的SQL。