

MySQL学习笔记

数据库

✧ 数据库相关概念

数据库

- 存储和管理数据的仓库，数据是有组织的进行存储。
- 数据库英文名是 DataBase，简称DB。

数据库就是将数据存储硬盘上，可以达到持久化存储的效果。那又是如何解决上述问题的？使用数据库管理系统。

数据库管理系统

- 管理数据库的大型软件
- 英文：DataBase Management System，简称 DBMS

在电脑上安装了数据库管理系统后，就可以通过数据库管理系统创建数据库来存储数据，也可以通过该系统对数据库中的数据进行数据的增删改查相关的操作。我们平时说的MySQL数据库其实是MySQL数据库管理系统。

常见的数据库管理系统

- Oracle：收费的大型数据库，Oracle 公司的产品

- MySQL：开源免费的中小型数据库。后来 Sun公司收购了 MySQL，而 Sun 公司又被 Oracle 收购
- SQL Server：MicroSoft 公司收费的中型的数据库。C#、.net 等语言常使用
- PostgreSQL：开源免费中小型的数据库
- DB2：IBM 公司的大型收费数据库产品
- SQLite：嵌入式的微型数据库。如：作为 Android 内置数据库
- MariaDB：开源免费中小型的数据库

SQL

- 英文：Structured Query Language，简称 SQL，结构化查询语言
- 操作关系型数据库的编程语言
- 定义操作所有关系型数据库的统一标准，可以使用SQL操作所有的关系型数据库管理系统，以后工作中如果使用到了其他的数据库管理系统，也同样的使用SQL来操作。

✧ MySQL数据库

MySQL数据模型

关系型数据库：关系型数据库是建立在关系模型基础上的数据库，简单说，关系型数据库是由多张能互相连接的 **二维表** 组成的数据库

关系型数据库的优点：

- 都是使用表结构，格式一致，易于维护。
- 使用通用的 SQL 语言操作，使用方便，可用于复杂查询。
 - 关系型数据库都可以通过SQL进行操作，所以使用方便。
 - 复杂查询。现在需要查询001号订单数据，我们可以看到该订单是1号客户的订单，而1号订单是李聪这个客户。以后也可以在一张表中进行统计分析等操作。

- 数据存储于磁盘中，安全。

小结：

- MySQL中可以创建多个数据库，每个数据库对应到磁盘上的一个文件夹
- 在每个数据库中创建多个表，每张都对应到磁盘上一个 `frm` 文件
- 每张表可以存储多条数据，数据会被存储到磁盘中 `MYD` 文件中


SQL

SQL简介

- 英文：Structured Query Language，简称 SQL
- 结构化查询语言，一门操作关系型数据库的编程语言
- 定义操作所有关系型数据库的统一标准
- 对于同一个需求，每一种数据库操作的方式可能会存在一些不一样的地方，我们称为“方言”

通用语法

- SQL 语句可以单行或多行书写，以分号结尾。
- MySQL 数据库的 SQL 语句不区分大小写，关键字建议使用大写。
- 注释
 - 单行注释: `-- 注释内容` 或 `#注释内容`(MySQL 特有)
 - 多行注释: `/* 注释 */`

 注意：使用 `--` 添加单行注释时，`--`后面一定要加空格，而 `#`没有要求。

SQL分类

- DDL(Data Definition Language)：数据定义语言，用来定义数据库对象：数据库，表，列等

- DDL简单理解就是用来操作数据库，表等
- DML(Data Manipulation Language) 数据操作语言，用来对数据库中表的数据进行增删改
 - DML简单理解就对表中数据进行增删改
- DQL(Data Query Language) 数据查询语言，用来查询数据库中表的记录(数据)
 - DQL简单理解就是对数据进行查询操作。从数据库表中查询到我们想要的
- DCL(Data Control Language) 数据控制语言，用来定义数据库的访问权限和安全级别，及创建用户
 - DML简单理解就是对数据库进行权限控制。比如我让某一个数据库表只能让某一个用户进行操作等



注意：以后我们最常操作的是 DML 和 DQL，因为我们开发中最常操作的就是数据。

✧ DDL:操作数据库

查询

查询所有的数据库

```
1 SHOW DATABASES;
```

创建数据库

- 创建数据库：

```
1 CREATE DATABASE 数据库名称;
```

- 创建数据库(判断，如果不存在则创建)

```
1 CREATE DATABASE IF NOT EXISTS 数据库名称;
```

删除数据库

- 删除数据库

```
1 DROP DATABASE 数据库名称;
```

- 删除数据库(判断, 如果存在则删除)

```
1 DROP DATABASE IF EXISTS 数据库名称;
```

使用数据库

- 使用数据库

```
1 USE 数据库名称;
```

- 查看当前使用的数据库

```
1 SELECT DATABASE();
```

✧ DDL:操作表

查询表

- 查询当前数据库下所有表名称

```
1 SHOW TABLES;
```

- 查询表结构

```
1 DESC 表名称;
```

创建表

- 创建表

```
1 CREATE TABLE 表名 (  
2     字段名1 数据类型1,  
3     字段名2 数据类型2,  
4     ...  
5     字段名n 数据类型n  
6 );
```

i 注意：最后一行末尾，不能加逗号

数据类型

○ 数值

```
1 tinyint : 小整数型, 占一个字节
2 int : 大整数类型, 占四个字节
3     eg : age int
4 double : 浮点类型
5     使用格式: 字段名 double(总长度, 小数点后保留的位数)
6     eg : score double(5,2)
```

○ 日期

```
1 date : 日期值。只包含年月日
2     eg : birthday date :
3 datetime : 混合日期和时间值。包含年月日时分秒
```

○ 字符串

```
1 char : 定长字符串。
2     优点: 存储性能高
3     缺点: 浪费空间
4     eg : name char(10) 如果存储的数据字符个数不足10个, 也会占10个的空间
5 varchar : 变长字符串。
6     优点: 节约空间
7     缺点: 存储性能底
8     eg : name varchar(10) 如果存储的数据字符个数不足10个, 那就数据字符个数是几就占几个的空间
```

i 注意：其他类型参考资料中的《MySQL数据类型.xlsx》

案例

需求：设计一张学生表，请注重数据类型、长度的合理性

- 1 编号
- 2 姓名，姓名最长不超过10个汉字
- 3 性别，因为取值只有两种可能，因此最多一个汉字

- ④ 生日，取值为年月日
- ⑤ 入学成绩，小数点后保留两位
- ⑥ 邮件地址，最大长度不超过 64
- ⑦ 家庭联系电话，不一定是手机号码，可能会出现 - 等字符
- ⑧ 学生状态（用数字表示，正常、休学、毕业...）

```
1  create table student (  
2      id int,  
3      name varchar(10),  
4      gender char(1),  
5      birthday date,  
6      score double(5,2),  
7      email varchar(15),  
8      tel varchar(15),  
9      status tinyint  
10 );
```

删除表

- 删除表

```
1  DROP TABLE 表名;
```

- 删除表时判断表是否存在

```
1  DROP TABLE IF EXISTS 表名;
```

修改表

- 修改表名

```
1  ALTER TABLE 表名 RENAME TO 新的表名;  
2  
3  -- 将表名student修改为stu  
4  alter table student rename to stu;
```

- 添加一列

```
1  ALTER TABLE 表名 ADD 列名 数据类型;  
2  -- 给stu表添加一列address, 该字段类型是varchar(50)  
3  alter table stu add address varchar(50);
```

- 修改数据类型

```
1 ALTER TABLE 表名 MODIFY 列名 新数据类型;  
2  
3 -- 将stu表中的address字段的类型改为 char(50)  
4 alter table stu modify address char(50);
```

- 修改列名和数据类型

```
1 ALTER TABLE 表名 CHANGE 列名 新列名 新数据类型;  
2  
3 -- 将stu表中的address字段名改为 addr, 类型改为varchar(50)  
4 alter table stu change address addr varchar(50);
```

- 删除列

```
1 ALTER TABLE 表名 DROP 列名;  
2  
3 -- 将stu表中的addr字段 删除  
4 alter table stu drop addr;
```

✧ DML

DML主要是对数据进行增（insert）删（delete）改（update）操作。

添加数据

- 给指定列添加数据

```
1 INSERT INTO 表名(列名1,列名2,...) VALUES(值1,值2,...);
```

- 给全部列添加数据

```
1 INSERT INTO 表名 VALUES(值1,值2,...);
```

- 批量添加数据

```
1 INSERT INTO 表名(列名1,列名2,...) VALUES(值1,值2,...),(值1,值2,...),(值1,值  
2,...) ...;  
2 INSERT INTO 表名 VALUES(值1,值2,...),(值1,值2,...),(值1,值2,...) ...;
```


修改数据

- 修改表数据

```
1 UPDATE 表名 SET 列名1=值1,列名2=值2,... [WHERE 条件];
```

注意:

- i** ① 修改语句中如果不加条件，则将所有数据都修改！
- ② 像上面的语句中的中括号，表示在写sql语句中可以省略这部分

练习

- 将张三的性别改为女

```
1 update stu set sex = '女' where name = '张三';
```

- 将张三的生日改为 1999-12-12 分数改为99.99

```
1 update stu set birthday = '1999-12-12', score = 99.99 where name = '张三';
```

- 注意: 如果update语句没有加where条件，则会将表中所有数据全部修改！

```
1 update stu set sex = '女';
```

删除数据

- 删除数据

```
1 DELETE FROM 表名 [WHERE 条件];
```

练习

```
1 -- 删除张三记录
2 delete from stu where name = '张三';
3
4 -- 删除stu表中的所有数据
5 delete from stu;
```

基础查询

- 查询多个字段

```
1 SELECT 字段列表 FROM 表名;  
2 SELECT * FROM 表名; -- 查询所有数据
```

- 去除重复记录

```
1 SELECT DISTINCT 字段列表 FROM 表名;
```

- 起别名

```
1 AS: AS 也可以省略
```

练习

- 查询name、age两列

```
1 select name,age from stu;
```

- 查询所有列的数据，列名的列表可以使用*替代

```
1 select * from stu;
```

- 查询地址信息

```
1 select address from stu;
```

- 去除重复记录

```
1 select distinct address from stu;
```

- 查询姓名、数学成绩、英语成绩。并通过as给math和english起别名（as关键字可以省略）

```
1 select name,math as 数学成绩,english as 英文成绩 from stu;  
2 select name,math 数学成绩,english 英文成绩 from stu;
```

条件查询

i 语法

1 **SELECT** 字段列表 **FROM** 表名 **WHERE** 条件列表;

i 条件

符号	功能
>	大于
<	小于
>=	大于等于
<=	小于等于
=	等于
<>或!=	不等于
BETWEEN...AND...	在某个范围之内（都包含）
IN(...)	多选一
LIKE 占位符	摩奴查询, <code>_</code> 单个任意字符, <code>%</code> 多个任意字符
IS NULL	是 NULL
IS NOT NULL	不是 NULL
AND 或 &&	并且
OR 或	或者
NOT 或 !	非, 不是

条件查询练习

- 查询年龄大于20岁的学员信息

1 **select** * **from** stu **where** age > 20;

- 查询年龄大于等于20岁的学员信息

```
1 select * from stu where age ≥ 20;
```

- 查询年龄大于等于20岁 并且 年龄 小于等于 30岁 的学员信息

```
1 select * from stu where age ≥ 20 && age ≤ 30;  
2 select * from stu where age ≥ 20 and age ≤ 30;
```



上面语句中 && 和 and 都表示并且的意思。建议使用 and 。
也可以使用 between ... and 来实现上面需求

```
1 select * from stu where age BETWEEN 20 and 30;
```

- 查询入学日期在'1998-09-01' 到 '1999-09-01' 之间的学员信息

```
1 select * from stu where hire_date BETWEEN '1998-09-01' and '1999-09-01';
```

- 查询年龄等于18岁的学员信息

```
1 select * from stu where age = 18;
```

- 查询年龄不等于18岁的学员信息

```
1 select * from stu where age ≠ 18;  
2 select * from stu where age <> 18;
```

- 查询年龄等于18岁 或者 年龄等于20岁 或者 年龄等于22岁的学员信息

```
1 select * from stu where age = 18 or age = 20 or age = 22;  
2 select * from stu where age in (18,20 ,22);
```

- 查询英语成绩为 null的学员信息
 - null值的比较不能使用 = 或者 != 。需要使用 is 或者 is not

```
1 select * from stu where english = null; -- 这个语句是不行的  
2 select * from stu where english is null;  
3 select * from stu where english is not null;
```

模糊查询练习

模糊查询使用like关键字，可以使用通配符进行占位:



- (1) _ : 代表单个任意字符
- (2) % : 代表任意个数字符

- 查询姓'马'的学员信息

```
1 select * from stu where name like '马%';
```

- 查询第二个字是'花'的学员信息

```
1 select * from stu where name like '_花%';
```

- 查询名字中包含'德'的学员信息

```
1 select * from stu where name like '%德%';
```

排序查询

```
1 SELECT 字段列表 FROM 表名 ORDER BY 排序字段名1 [排序方式1], 排序字段名2 [排序方式2] ...;
```

- ASC：升序排列（默认值）
- DESC：降序排列



注意：如果有多个排序条件，当前边的条件值一样时，才会根据第二条件进行排序

练习

- 查询学生信息，按照年龄升序排列

```
1 select * from stu order by age;
```

- 查询学生信息，按照数学成绩降序排列

```
1 select * from stu order by math desc;
```

- 查询学生信息，按照数学成绩降序排列，如果数学成绩一样，再按照英语成绩升序排列

```
1 select * from stu order by math desc , english asc;
```

聚合函数



概念

将一列数据作为一个整体，进行纵向计算。

i 聚合函数分类

函数名	功能
count(列名)	统计数量（一般选用不为null的列）
max(列名)	最大值
min(列名)	最小值
sum(列名)	求和
avg(列名)	平均值

i 聚合函数语法

```
1 SELECT 聚合函数名(列名) FROM 表;
```

i 注意：null 值不参与所有聚合函数运算

练习

- 统计班级一共有多少个学生

```
1 select count(id) from stu;  
2 select count(english) from stu;
```

- 上面语句根据某个字段进行统计，如果该字段某一行的值为null的话，将不会被统计。所以可以在count() 来实现。 表示所有字段数据，一行中也不可能所有的数据都为null，所以建议使用 count(*)

```
1 select count(*) from stu;
```

- 查询数学成绩的最高分

```
1 select max(math) from stu;
```

- 查询数学成绩的最低分

```
1 select min(math) from stu;
```

- 查询数学成绩的总分

```
1 select sum(math) from stu;
```

- 查询数学成绩的平均分

```
1 select avg(math) from stu;
```

- 查询英语成绩的最低分

```
1 select min(english) from stu;
```

分组查询

```
1 SELECT 字段列表 FROM 表名 [WHERE 分组前条件限定] GROUP BY 分组字段名  
[HAVING 分组后条件过滤];
```



注意：分组之后，查询的字段为聚合函数和分组字段，查询其他字段无任何意义

练习

- 查询男同学和女同学各自的数学平均分

```
1 select sex, avg(math) from stu group by sex;
```



注意：分组之后，查询的字段为聚合函数和分组字段，查询其他字段无任何意义

```
1 select name, sex, avg(math) from stu group by sex; -- 这里查询name字  
段就没有任何意义
```

- 查询男同学和女同学各自的数学平均分，以及各自人数

```
1 select sex, avg(math), count(*) from stu group by sex;
```

- 查询男同学和女同学各自的数学平均分，以及各自人数，要求：分数低于70分的不参与分组

```
1 select sex, avg(math), count(*) from stu where math > 70 group by  
sex;
```

- 查询男同学和女同学各自的数学平均分，以及各自人数，要求：分数低于70分的不参与分组，分组之后人数大于2个的

```
1 select sex, avg(math), count(*) from stu where math > 70 group by sex having count(*) > 2;
```

where 和 having 区别：

- 执行时机不一样：where 是分组之前进行限定，不满足where条件，则不参与分组，而having是分组之后对结果进行过滤。
- 可判断的条件不一样：where 不能对聚合函数进行判断，having 可以

分页查询

```
1 SELECT 字段列表 FROM 表名 LIMIT 起始索引 , 查询条目数;
```

i 注意：上述语句中的起始索引是从0开始

练习

- 从0开始查询，查询3条数据

```
1 select * from stu limit 0 , 3;
```

- 每页显示3条数据，查询第1页数据

```
1 select * from stu limit 0 , 3;
```

- 每页显示3条数据，查询第2页数据

```
1 select * from stu limit 3 , 3;
```

- 每页显示3条数据，查询第3页数据

```
1 select * from stu limit 6 , 3;
```

i 从上面的练习推导出起始索引计算公式：

```
1 起始索引 = (当前页码 - 1) * 每页显示的条数
```


Tips:

- 分页查询 limit 是MySQL数据库的方言
- Oracle 分页查询使用 rownumber
- SQL Server 分页查询使用 top

mysql高级

约束

概念

- 约束是作用于表中列上的规则，用于限制加入表的数据
 - 例如：我们可以给id列加约束，让其值不能重复，不能为null值。
- 约束的存在保证了数据库中数据的正确性、有效性和完整性
 - 添加约束可以在添加数据的时候就限制不正确的数据，年龄是3000，数学成绩是-5分这样无效的数据，继而保障数据的完整性。

分类

- 非空约束：关键字是 **NOT NULL**
 - 保证列中所有的数据不能有null值。
 - 例如：id列在添加 马花疼 这条数据时就不能添加成功。
- 唯一约束：关键字是 **UNIQUE**
 - 保证列中所有数据各不相同。
 - 例如：id列中三条数据的值都是1，这样的数据在添加时是绝对不允许的。
- 主键约束：关键字是 **PRIMARY KEY**

- 主键是一行数据的唯一标识，要求非空且唯一。一般我们都会给没张表添加一个主键列用来唯一标识数据。
- 例如：上图表中id就可以作为主键，来标识每条数据。那么这样就要求数据中id的值不能重复，不能为null值。
- 检查约束： 关键字是 **CHECK**
 - 保证列中的值满足某一条件。
 - 例如：我们可以给age列添加一个范围，最低年龄可以设置为1，最大年龄就可以设置为300，这样的数据才更合理些。

注意：MySQL不支持检查约束。

i 这样是不是就没办法保证年龄在指定的范围内了？从数据库层面不能保证，以后可以在java代码中进行限制，一样也可以实现要求。

- 默认约束： 关键字是 **DEFAULT**
 - 保存数据时，未指定值则采用默认值。
 - 例如：我们在给english列添加该约束，指定默认值是0，这样在添加数据时没有指定具体值时就会采用默认给定的0。
- 外键约束： 关键字是 **FOREIGN KEY**
 - 外键用来让两个表的数据之间建立链接，保证数据的一致性和完整性。
 - 外键约束现在可能还不太好理解，后面我们会重点进行讲解。

非空约束

- 概念
 - 非空约束用于保证列中所有数据不能有NULL值
- 语法
 - 添加约束

```
1  -- 创建表时添加非空约束
2  CREATE TABLE 表名(
3      列名 数据类型 NOT NULL,
4      ...
5  );
```

```
1  -- 建完表后添加非空约束
2  ALTER TABLE 表名 MODIFY 字段名 数据类型 NOT NULL;
```

- 删除约束

```
1 ALTER TABLE 表名 MODIFY 字段名 数据类型;
```

唯一约束

- 概念
 - 唯一约束用于保证列中所有数据各不相同
- 语法
 - 添加约束

```
1 -- 创建表时添加唯一约束
2 CREATE TABLE 表名(
3     列名 数据类型 UNIQUE [AUTO_INCREMENT],
4     -- AUTO_INCREMENT: 当不指定值时自动增长
5     ...
6 );
7 CREATE TABLE 表名(
8     列名 数据类型,
9     ...
10    [CONSTRAINT] [约束名称] UNIQUE(列名)
11 );
```

```
1 -- 建完表后添加唯一约束
2 ALTER TABLE 表名 MODIFY 字段名 数据类型 UNIQUE;
```

- 删除约束

```
1 ALTER TABLE 表名 DROP INDEX 字段名;
```

主键约束

- 概念
 - 主键是一行数据的唯一标识，要求非空且唯一
 - 一张表只能有一个主键
- 语法
 - 添加约束

```

1  -- 创建表时添加主键约束
2  CREATE TABLE 表名(
3      列名 数据类型 PRIMARY KEY [AUTO_INCREMENT],
4      ...
5  );
6  CREATE TABLE 表名(
7      列名 数据类型,
8      [CONSTRAINT] [约束名称] PRIMARY KEY(列名)
9  );

```

```

1  -- 建完表后添加主键约束
2  ALTER TABLE 表名 ADD PRIMARY KEY(字段名);

```

- 删除约束

```

1  ALTER TABLE 表名 DROP PRIMARY KEY;

```

默认约束

- 概念
 - 保存数据时，未指定值则采用默认值
- 语法
 - 添加约束

```

1  -- 创建表时添加默认约束
2  CREATE TABLE 表名(
3      列名 数据类型 DEFAULT 默认值,
4      ...
5  );
6  -- 建完表后添加默认约束
7  ALTER TABLE 表名 ALTER 列名 SET DEFAULT 默认值;

```

- 删除约束

```

1  ALTER TABLE 表名 ALTER 列名 DROP DEFAULT;

```

约束练习

根据需求，为表添加合适的约束

```

1  -- 员工表
2  CREATE TABLE emp (
3      id INT, -- 员工id, 主键且自增长
4      ename VARCHAR(50), -- 员工姓名, 非空且唯一
5      joindate DATE, -- 入职日期, 非空
6      salary DOUBLE(7,2), -- 工资, 非空
7      bonus DOUBLE(7,2) -- 奖金, 如果没有将近默认为0
8  );

```

上面一定给出了具体的要求，我们可以根据要求创建这张表，并为每一列添加对应的约束。建表语句如下：

```

1  DROP TABLE IF EXISTS emp;
2
3  -- 员工表
4  CREATE TABLE emp (
5      id INT PRIMARY KEY, -- 员工id, 主键且自增长
6      ename VARCHAR(50) NOT NULL UNIQUE, -- 员工姓名, 非空并且唯一
7      joindate DATE NOT NULL, -- 入职日期, 非空
8      salary DOUBLE(7,2) NOT NULL, -- 工资, 非空
9      bonus DOUBLE(7,2) DEFAULT 0 -- 奖金, 如果没有奖金默认为0
10 );

```

通过上面语句可以创建带有约束的 `emp` 表，约束能不能发挥作用呢。接下来我们一一进行验证，先添加一条没有问题的数据

```

1  INSERT INTO emp(id,ename,joindate,salary,bonus) values(1,'张三','1999-11-11',8800,5000);

```

- 验证主键约束，非空且唯一

```

1  INSERT INTO emp(id,ename,joindate,salary,bonus) values(null,'张三','1999-11-11',8800,5000);

```

- 验证非空约束

```

1  INSERT INTO emp(id,ename,joindate,salary,bonus)
    values(3,null,'1999-11-11',8800,5000);

```

- 验证唯一约束

```

1  INSERT INTO emp(id,ename,joindate,salary,bonus) values(3,'李四','1999-11-11',8800,5000);

```

- 验证默认约束

```
1 INSERT INTO emp(id,ename,joindate,salary) values(3,'王五','1999-11-11',8800);
```



注意：默认约束只有在不给值时才会采用默认值。如果给了**null**，那值就是**null**值。

- 验证自动增长： `auto_increment` 当列是数字类型 并且唯一约束
 - 重新创建 `emp` 表，并给 `id` 列添加自动增长

```
1 -- 员工表
2 CREATE TABLE emp (
3     id INT PRIMARY KEY auto_increment, -- 员工id, 主键且自增长
4     ename VARCHAR(50) NOT NULL UNIQUE, -- 员工姓名, 非空并且唯一
5     joindate DATE NOT NULL , -- 入职日期, 非空
6     salary DOUBLE(7,2) NOT NULL , -- 工资, 非空
7     bonus DOUBLE(7,2) DEFAULT 0 -- 奖金, 如果没有奖金默认为0
8 );
```

- 接下来给 `emp` 添加数据，分别验证不给 `id` 列添加值以及给 `id` 列添加 `null` 值，`id` 列的值会不会自动增长：

```
1 INSERT INTO emp(ename,joindate,salary,bonus) values('赵六','1999-11-11',8800,null);
2 INSERT INTO emp(id,ename,joindate,salary,bonus) values(null,'赵六2','1999-11-11', 8800,null);
3 INSERT INTO emp(id,ename,joindate,salary,bonus) values(null,'赵六3','1999-11-11', 8800,null);
```

外键约束

外键用来让两个表的数据之间建立链接，保证数据的一致性和完整性。



语法

- 添加外键约束

```
1 -- 创建表时添加外键约束
2 CREATE TABLE 表名(
3     列名 数据类型,
4     ...
5     [CONSTRAINT] [外键名称] FOREIGN KEY(外键列名) REFERENCES 主表(主表列名)
6 );
```

```
1  -- 建完表后添加外键约束
2  ALTER TABLE 表名 ADD CONSTRAINT 外键名称 FOREIGN KEY (外键字段名称)
   REFERENCES 主表名称(主表列名称);
```

- 删除外键约束

```
1  ALTER TABLE 表名 DROP FOREIGN KEY 外键名称;
```

练习

根据上述语法创建员工表和部门表，并添加上外键约束：

```
1  -- 删除表
2  DROP TABLE IF EXISTS emp;
3  DROP TABLE IF EXISTS dept;
4
5  -- 部门表
6  CREATE TABLE dept(
7      id int primary key auto_increment,
8      dep_name varchar(20),
9      addr varchar(20)
10 );
11
12 -- 员工表
13 CREATE TABLE emp(
14     id int primary key auto_increment,
15     name varchar(20),
16     age int,
17     dep_id int,
18
19     -- 添加外键 dep_id,关联 dept 表的id主键
20     CONSTRAINT fk_emp_dept FOREIGN KEY(dep_id) REFERENCES
    dept(id)
21 );
```

- 添加数据

```

1  -- 添加 2 个部门
2  insert into dept(dep_name,addr) values
3  ('研发部','广州'),('销售部','深圳');
4
5  -- 添加员工,dep_id 表示员工所在的部门
6  INSERT INTO emp (NAME, age, dep_id) VALUES
7  ('张三', 20, 1),
8  ('李四', 20, 1),
9  ('王五', 20, 1),
10 ('赵六', 20, 2),
11 ('孙七', 22, 2),
12 ('周八', 18, 2);

```

- 删除外键

```

1  alter table emp drop FOREIGN key fk_emp_dept;

```

- 重新添加外键

```

1  alter table emp add CONSTRAINT fk_emp_dept FOREIGN key(dep_id)
   REFERENCES dept(id);

```

✳ 数据库设计

- 数据库设计概念
 - 数据库设计就是根据业务系统的具体需求，结合我们所选用的DBMS，为这个业务系统构造出最优的数据存储模型。
 - 建立数据库中的表结构以及表与表之间的关联关系的过程。
 - 有哪些表？表里有哪些字段？表和表之间有什么关系？
- 数据库设计的步骤
 - 需求分析（数据是什么？数据具有哪些属性？数据与属性的特点是什么）
 - 逻辑分析（通过ER图对数据库进行逻辑建模，不需要考虑我们所选用的数据库管理系统）
 - 物理设计（根据数据库自身的特点把逻辑设计转换为物理设计）
 - 维护设计（1.对新的需求进行建表；2.表优化）
- 表关系

- 一对一
 - 如：用户 和 用户详情
 - 一对一关系多用于表拆分，将一个实体中经常使用的字段放一张表，不经常使用的字段放另一张表，用于提升查询性能
- 一对多
 - 如：部门 和 员工
 - 一个部门对应多个员工，一个员工对应一个部门。
- 多对多
 - 如：商品 和 订单
 - 一个商品对应多个订单，一个订单包含多个商品。

表关系(一对多)

- 一对多
 - 如：部门 和 员工
 - 一个部门对应多个员工，一个员工对应一个部门。
- 实现方式
 - 在多的一方建立外键，指向一的一方的主键
- 建表语句如下：

```
1  -- 删除表
2  DROP TABLE IF EXISTS tb_emp;
3  DROP TABLE IF EXISTS tb_dept;
4
5  -- 部门表
6  CREATE TABLE tb_dept(
7      id int primary key auto_increment,
8      dep_name varchar(20),
9      addr varchar(20)
10 );
11
12 -- 员工表
13 CREATE TABLE tb_emp(
14     id int primary key auto_increment,
15     name varchar(20),
16     age int,
```

```

17     dep_id int,
18
19     -- 添加外键 dep_id,关联 dept 表的id主键
20     CONSTRAINT fk_emp_dept FOREIGN KEY(dep_id) REFERENCES
    tb_dept(id)
21 );

```

表关系(多对多)

- 多对多
 - 如：商品 和 订单
 - 一个商品对应多个订单，一个订单包含多个商品
- 实现方式
 - 建立第三张中间表，中间表至少包含两个外键，分别关联两方主键
- 建表语句如下：

```

1  -- 删除表
2  DROP TABLE IF EXISTS tb_order_goods;
3  DROP TABLE IF EXISTS tb_order;
4  DROP TABLE IF EXISTS tb_goods;
5
6  -- 订单表
7  CREATE TABLE tb_order(
8      id int primary key auto_increment,
9      payment double(10,2),
10     payment_type TINYINT,
11     status TINYINT
12 );
13
14 -- 商品表
15 CREATE TABLE tb_goods(
16     id int primary key auto_increment,
17     title varchar(100),
18     price double(10,2)
19 );
20
21 -- 订单商品中间表
22 CREATE TABLE tb_order_goods(
23     id int primary key auto_increment,
24     order_id int,
25     goods_id int,

```

```

26     count int
27 );
28
29 -- 建完表后, 添加外键
30 alter table tb_order_goods add CONSTRAINT fk_order_id FOREIGN
    key(order_id) REFERENCES
31 tb_order(id);
32 alter table tb_order_goods add CONSTRAINT fk_goods_id FOREIGN
    key(goods_id) REFERENCES
33 tb_goods(id);

```

表关系(一对一)

- 一对一
 - 如: 用户 和 用户详情
 - 一对一关系多用于表拆分, 将一个实体中经常使用的字段放一张表, 不经常使用的字段放另一张表, 用于提升查询性能
- 实现方式
 - 在任意一方加入外键, 关联另一方主键, 并且设置外键为唯一(UNIQUE)
- 建表语句如下:

```

1  create table tb_user_desc (
2      id int primary key auto_increment,
3      city varchar(20),
4      edu varchar(10),
5      income int,
6      status char(2),
7      des varchar(100)
8  );
9
10 create table tb_user (
11     id int primary key auto_increment,
12     photo varchar(100),
13     nickname varchar(50),
14     age int,
15     gender char(1),
16     desc_id int unique,
17     -- 添加外键
18     CONSTRAINT fk_user_desc FOREIGN KEY(desc_id) REFERENCES
    tb_user_desc(id)
19 );

```

多表查询

多表查询顾名思义就是从多张表中一次性的查询出我们想要的数据库。我们通过具体的sql给他们演示，先准备环境

```
1 DROP TABLE IF EXISTS emp;
2 DROP TABLE IF EXISTS dept;
3
4 # 创建部门表
5 CREATE TABLE dept(
6     did INT PRIMARY KEY AUTO_INCREMENT,
7     dname VARCHAR(20)
8 );
9
10 # 创建员工表
11 CREATE TABLE emp (
12     id INT PRIMARY KEY AUTO_INCREMENT,
13     NAME VARCHAR(10),
14     gender CHAR(1), -- 性别
15     salary DOUBLE, -- 工资
16     join_date DATE, -- 入职日期
17     dep_id INT,
18     FOREIGN KEY (dep_id) REFERENCES dept(did) -- 外键，关联部门表(部门表的主键)
19 );
20
21 -- 添加部门数据
22 INSERT INTO dept (dNAME) VALUES ('研发部'),('市场部'),('财务部'),('销售部');
23
24 -- 添加员工数据
25 INSERT INTO emp(NAME,gender,salary,join_date,dep_id) VALUES
26 ('孙悟空','男',7200,'2013-02-24',1),
27 ('猪八戒','男',3600,'2010-12-02',2),
28 ('唐僧','男',9000,'2008-08-08',2),
29 ('白骨精','女',5000,'2015-10-07',3),
30 ('蜘蛛精','女',4500,'2011-03-14',1),
31 ('小白龙','男',2500,'2011-02-14',null);
```

- 执行下面的多表查询语句

```
1 select * from emp , dept; -- 从emp和dept表中查询所有的字段数据
```

- 孙悟空 这个员工属于1号部门，但也同时关联的2、3、4号部门。所以我们要通过限制员工表中的 `dep_id` 字段的值和部门表 `did` 字段的值相等来消除这些无效的数据，

```
1 select * from emp , dept where emp.dep_id = dept.did;
```

多表查询分类

- 连接查询
 - 内连接查询：相当于查询AB交集数据
 - 外连接查询
 - 左外连接查询：相当于查询A表所有数据和交集部门数据
 - 右外连接查询：相当于查询B表所有数据和交集部分数据
- 子查询

内连接查询

- 语法

```
1 -- 隐式内连接
2 SELECT 字段列表 FROM 表1,表2... WHERE 条件;
3
4 -- 显示内连接
5 SELECT 字段列表 FROM 表1 [INNER] JOIN 表2 ON 条件;
```

内连接相当于查询 A B 交集数据

- 案例
 - 隐式内连接

```
1 SELECT
2     *
3 FROM
4     emp,
5     dept
6 WHERE
7     emp.dep_id = dept.did;
```

- 查询 `emp` 的 `name`，`gender`，`dept` 表的 `dname`

```

1  SELECT
2      emp. NAME,
3      emp.gender,
4      dept.dname
5  FROM
6      emp,
7      dept
8  WHERE
9      emp.dep_id = dept.did;

```

- 上面语句中使用表名指定字段所属有点麻烦，sql也支持给表指别名，上述语句可以改进为

```

1  SELECT
2      t1. NAME,
3      t1.gender,
4      t2.dname
5  FROM
6      emp t1,
7      dept t2
8  WHERE
9      t1.dep_id = t2.did;

```

- 显式内连接

```

1  select * from emp inner join dept on emp.dep_id = dept.did;
2  -- 上面语句中的inner可以省略，可以书写为如下语句
3  select * from emp join dept on emp.dep_id = dept.did;

```

外连接查询

- 语法

```

1  -- 左外连接
2  SELECT 字段列表 FROM 表1 LEFT [OUTER] JOIN 表2 ON 条件;
3
4  -- 右外连接
5  SELECT 字段列表 FROM 表1 RIGHT [OUTER] JOIN 表2 ON 条件;

```



左外连接：相当于查询A表所有数据和交集部分数据
右外连接：相当于查询B表所有数据和交集部分数据

- 案例

- 查询emp表所有数据和对应的部门信息（左外连接）

```
1 select * from emp left join dept on emp.dep_id = dept.did;
```

- 查询dept表所有数据和对应的员工信息（右外连接）

```
1 select * from emp right join dept on emp.dep_id = dept.did;
```

- 要查询出部门表中所有的数据，也可以通过左外连接实现，只需要将两个表的位置进行互换：

```
1 select * from dept left join emp on emp.dep_id = dept.did;
```

子查询

查询中嵌套查询，称嵌套查询为子查询。

i 需求：查询工资高于猪八戒的员工信息。

- 来实现这个需求，我们就可以通过二步实现，第一步：先查询出来 猪八戒的工资

```
1 select salary from emp where name = '猪八戒';
```

- 第二步：查询工资高于猪八戒的员工信息

```
1 select * from emp where salary > 3600;
```

- 第二步中的3600可以通过第一步的sql查询出来，所以将3600用第一步的sql语句进行替换

```
1 select * from emp where salary > (select salary from emp where name = '猪八戒');
```

- 子查询根据查询结果不同，作用不同
 - 子查询语句结果是单行单列，子查询语句作为条件值，使用 = != > < 等进行条件判断

```
1 SELECT 字段列表 FROM 表 WHERE 字段名 = (子查询);
```

- 子查询语句结果是多行单列，子查询语句作为条件值，使用 in 等关键字进行条件判断

```
1 SELECT 字段列表 FROM 表 WHERE 字段名 IN (子查询);
```

- 子查询语句结果是多行多列，子查询语句作为虚拟表

```
1 SELECT 字段列表 FROM (子查询) WHERE 条件;
```

- 案例

- 查询 '财务部' 和 '市场部' 所有的员工信息

```
1  -- 查询 '财务部' 或者 '市场部' 所有的员工的部门did
2  select did from dept where dname = '财务部' or dname = '市场部';
3
4  select * from emp where dep_id in (select did from dept where
    dname = '财务部' or dname = '市场部');
```

- 查询入职日期是 '2011-11-11' 之后的员工信息和部门信息

```
1  -- 查询入职日期是 '2011-11-11' 之后的员工信息
2  select * from emp where join_date > '2011-11-11' ;
3
4  -- 将上面语句的结果作为虚拟表和dept表进行内连接查询
5  select * from (select * from emp where join_date > '2011-11-
    11' ) t1, dept where t1.dep_id = dept.did;
```

案例

- 环境准备:

```
1  DROP TABLE IF EXISTS emp;
2  DROP TABLE IF EXISTS dept;
3  DROP TABLE IF EXISTS job;
4  DROP TABLE IF EXISTS salarygrade;
5
6  -- 部门表
7  CREATE TABLE dept (
8      did INT PRIMARY KEY PRIMARY KEY, -- 部门id
9      dname VARCHAR(50), -- 部门名称
10     loc VARCHAR(50) -- 部门所在地
11 );
12
13 -- 职务表, 职务名称, 职务描述
14 CREATE TABLE job (
15     id INT PRIMARY KEY,
16     jname VARCHAR(20),
17     description VARCHAR(50)
18 );
19
20 -- 员工表
21 CREATE TABLE emp (
22     id INT PRIMARY KEY, -- 员工id
23     ename VARCHAR(50), -- 员工姓名
```



```

24     job_id INT, -- 职务id
25     mgr INT , -- 上级领导
26     joindate DATE, -- 入职日期
27     salary DECIMAL(7,2), -- 工资
28     bonus DECIMAL(7,2), -- 奖金
29     dept_id INT, -- 所在部门编号
30     CONSTRAINT emp_jobid_ref_job_id_fk FOREIGN KEY (job_id)
REFERENCES job (id),
31     CONSTRAINT emp_deptid_ref_dept_id_fk FOREIGN KEY (dept_id)
REFERENCES dept (id)
32 );
33
34 -- 工资等级表
35 CREATE TABLE salarygrade (
36     grade INT PRIMARY KEY, -- 级别
37     losalary INT, -- 最低工资
38     hisalary INT -- 最高工资
39 );
40
41 -- 添加4个部门
42 INSERT INTO dept(did,dname,loc) VALUES
43 (10,'教研部','北京'),
44 (20,'学工部','上海'),
45 (30,'销售部','广州'),
46 (40,'财务部','深圳');
47
48 -- 添加4个职务
49 INSERT INTO job (id, jname, description) VALUES
50 (1, '董事长', '管理整个公司, 接单'),
51 (2, '经理', '管理部门员工'),
52 (3, '销售员', '向客人推销产品'),
53 (4, '文员', '使用办公软件');
54
55 -- 添加员工
56 INSERT INTO
57 emp(id,ename,job_id,mgr,joindate,salary,bonus,dept_id) VALUES
58 (1001,'孙悟空',4,1004,'2000-12-17','8000.00',NULL,20),
59 (1002,'卢俊义',3,1006,'2001-02-20','16000.00','3000.00',30),
60 (1003,'林冲',3,1006,'2001-02-22','12500.00','5000.00',30),
61 (1004,'唐僧',2,1009,'2001-04-02','29750.00',NULL,20),
62 (1005,'李逵',4,1006,'2001-09-28','12500.00','14000.00',30),
63 (1006,'宋江',2,1009,'2001-05-01','28500.00',NULL,30),
64 (1007,'刘备',2,1009,'2001-09-01','24500.00',NULL,10),
65 (1008,'猪八戒',4,1004,'2007-04-19','30000.00',NULL,20),
66 (1009,'罗贯中',1,NULL,'2001-11-17','50000.00',NULL,10),

```

```

66 (1010, '吴用', 3, 1006, '2001-09-08', '15000.00', '0.00', 30),
67 (1011, '沙僧', 4, 1004, '2007-05-23', '11000.00', NULL, 20),
68 (1012, '李逵', 4, 1006, '2001-12-03', '9500.00', NULL, 30),
69 (1013, '小白龙', 4, 1004, '2001-12-03', '30000.00', NULL, 20),
70 (1014, '关羽', 4, 1007, '2002-01-23', '13000.00', NULL, 10);
71
72 -- 添加5个工资等级
73 INSERT INTO salarygrade(grade, losalary, hisalary) VALUES
74 (1, 7000, 12000),
75 (2, 12010, 14000),
76 (3, 14010, 20000),
77 (4, 20010, 30000),
78 (5, 30010, 99990);

```

○ 需求

- 查询所有员工信息。查询员工编号，员工姓名，工资，职务名称，职务描述

```

1  /*
2     分析：
3         1. 员工编号，员工姓名，工资 信息在emp 员工表中
4         2. 职务名称，职务描述 信息在 job 职务表中
5         3. job 职务表 和 emp 员工表 是 一对多的关系 emp.job_id =
        job.id
6  */
7
8  -- 方式一：隐式内连接
9  SELECT
10     emp.id,
11     emp.ename,
12     emp.salary,
13     job.jname,
14     job.description
15  FROM
16     emp,
17     job
18  WHERE
19     emp.job_id = job.id;
20
21  -- 方式二：显式内连接
22  SELECT
23     emp.id,
24     emp.ename,
25     emp.salary,
26     job.jname,

```

```

27     job.description
28 FROM
29     emp
30 INNER JOIN job ON emp.job_id = job.id;

```

- 查询员工编号，员工姓名，工资，职务名称，职务描述，部门名称，部门位置

```

1  /*
2      分析：
3          1. 员工编号，员工姓名，工资 信息在emp 员工表中
4          2. 职务名称，职务描述 信息在 job 职务表中
5          3. job 职务表 和 emp 员工表 是一对多的关系 emp.job_id =
            job.id
6          4. 部门名称，部门位置 来自于 部门表 dept
7          5. dept 和 emp 一对多关系 dept.id = emp.dept_id
8  */
9
10 -- 方式一 ： 隐式内连接
11 SELECT
12     emp.id,
13     emp.ename,
14     emp.salary,
15     job.jname,
16     job.description,
17     dept.dname,
18     dept.loc
19 FROM
20     emp,
21     job,
22     dept
23 WHERE
24     emp.job_id = job.id
25     and dept.id = emp.dept_id;
26
27 -- 方式二 ： 显式内连接
28 SELECT
29     emp.id,
30     emp.ename,
31     emp.salary,
32     job.jname,
33     job.description,
34     dept.dname,
35     dept.loc
36 FROM
37     emp

```

```

38 INNER JOIN job ON emp.job_id = job.id
39 INNER JOIN dept ON dept.id = emp.dept_id

```

- 查询员工姓名，工资，工资等级

```

1  /*
2      分析：
3          1. 员工姓名, 工资 信息在emp 员工表中
4          2. 工资等级 信息在 salarygrade 工资等级表中
5          3. emp.salary ≥ salarygrade.losalary and emp.salary ≤
salarygrade.hisalary
6  */
7
8  SELECT
9      emp.ename,
10     emp.salary,
11     t2.*
12 FROM
13     emp,
14     salarygrade t2
15 WHERE
16     emp.salary ≥ t2.losalary
17     AND emp.salary ≤ t2.hisalary

```

- 查询员工姓名，工资，职务名称，职务描述，部门名称，部门位置，工资等级

```

1  /*
2      分析：
3          1. 员工编号, 员工姓名, 工资 信息在emp 员工表中
4          2. 职务名称, 职务描述 信息在 job 职务表中
5          3. job 职务表 和 emp 员工表 是 一对多的关系 emp.job_id =
job.id
6          4. 部门名称, 部门位置 来自于 部门表 dept
7          5. dept 和 emp 一对多关系 dept.id = emp.dept_id
8          6. 工资等级 信息在 salarygrade 工资等级表中
9          7. emp.salary ≥ salarygrade.losalary and emp.salary ≤
salarygrade.hisalary
10 */
11
12 SELECT
13     emp.id,
14     emp.ename,
15     emp.salary,
16     job.jname,
17     job.description,
18     dept.dname,

```

```

19     dept.loc,
20     t2.grade
21 FROM
22     emp
23 INNER JOIN job ON emp.job_id = job.id
24 INNER JOIN dept ON dept.id = emp.dept_id
25 INNER JOIN salarygrade t2 ON emp.salary BETWEEN t2.losalary and
    t2.hisalary;

```

- 查询出部门编号、部门名称、部门位置、部门人数

```

1  /*
2      分析:
3          1. 部门编号、部门名称、部门位置 来自于部门 dept 表
4          2. 部门人数: 在emp表中 按照dept_id 进行分组, 然后count(*)统计数
           量
5          3. 使用子查询, 让部门表和分组后的表进行内连接
6  */
7
8  -- 根据部门id分组查询每一个部门id和员工数
9  select dept_id, count(*) from emp group by dept_id;
10
11 SELECT
12     dept.id,
13     dept.dname,
14     dept.loc,
15     t1.count
16 FROM
17     dept,
18     (
19         SELECT
20             dept_id,
21             count(*) count
22         FROM
23             emp
24         GROUP BY
25             dept_id
26     ) t1
27 WHERE
28     dept.id = t1.dept_id

```

概述

数据库的事务（Transaction）是一种机制、一个操作序列，包含了一组数据库操作命令。

- i** 事务把所有的命令作为一个整体一起向系统提交或撤销操作请求，即这一组数据库命令要么同时成功，要么同时失败。

事务是一个不可分割的工作逻辑单元。

语法

- 开启事务

```
1 START TRANSACTION;  
2 或者  
3 BEGIN;
```

- 提交事务

```
1 commit;
```

- 回滚事务

```
1 rollback;
```

代码验证

- 环境准备

```
1 DROP TABLE IF EXISTS account;  
2  
3 -- 创建账户表  
4 CREATE TABLE account(  
5     id int PRIMARY KEY auto_increment,  
6     name varchar(10),  
7     money double(10,2)  
8 );  
9  
10 -- 添加数据  
11 INSERT INTO account(name,money) values('张三',1000),('李四',1000);
```

- 不加事务演示问题

```

1  -- 转账操作
2  -- 1. 查询李四账户金额是否大于500
3
4  -- 2. 李四账户 -500
5  UPDATE account set money = money - 500 where name = '李四';
6  出现异常了... -- 此处不是注释，在整体执行时会出问题，后面的sql则不执行
7
8  -- 3. 张三账户 +500
9  UPDATE account set money = money + 500 where name = '张三';

```

- 添加事务sql如下：

```

1  -- 开启事务
2  BEGIN;
3  -- 转账操作
4  -- 1. 查询李四账户金额是否大于500
5
6  -- 2. 李四账户 -500
7  UPDATE account set money = money - 500 where name = '李四';
8  出现异常了... -- 此处不是注释，在整体执行时会出问题，后面的sql则不执行
9
10 -- 3. 张三账户 +500
11 UPDATE account set money = money + 500 where name = '张三';
12
13 -- 提交事务
14 COMMIT;
15
16 -- 回滚事务
17 ROLLBACK;

```

- 上面sql中的执行成功进选择执行提交事务，而出现问题则执行回滚事务的语句。以后我们肯定不可能这样操作，而是在java中进行操作，在java中可以抓取异常，没出现异常提交事务，出现异常回滚事务。

事务的四大特征

- 原子性（Atomicity）：事务是不可分割的最小操作单位，要么同时成功，要么同时失败
- 一致性（Consistency）：事务完成时，必须使所有的数据都保持一致状态
- 隔离性（Isolation）：多个事务之间，操作的可见性

- 持久性（Durability）:事务一旦提交或回滚，它对数据库中的数据的改变就是永久的

说明:

mysql中事务是自动提交的。

也就是说我们不添加事务执行sql语句，语句执行完毕会自动的提交事务。

可以通过下面语句查询默认提交方式:



```
1 SELECT @@autocommit;
```

查询到的结果是1 则表示自动提交，结果是0表示手动提交。当然也可以通过下面语句修改提交方式:

```
1 set @@autocommit = 0;
```