

1- 数据库的基本概念

1.1 什么是数据库

- 数据库(DataBase) 就是存储和管理数据的仓库。
- 数据库本质是一个文件系统, 还是以文件的方式,将数据保存在电脑上。

1.2 为什么使用数据库

存储方式	优点	缺点
内存	速度快	不能够永久保存,数据是临时状态的
文件	数据是可以永久保存的	使用IO流操作文件, 不方便
数据库	1.数据可以永久保存 2.方便存储和管理数据 3.使用统一的方式操作数据库 (SQL)	占用资源,有些数据库需要付费(比如Oracle数据库)

1.3 常见的数据库

数据库名	介绍
MySQL 数据库	开源免费的数据库因为免费开源、运作简单的特点，常作为中小型的项目的数据库首选。
Oracle 数据库	收费的大型数据库，Oracle公司的核心产品。安全性高
DB2	IBM公司的数据库产品,收费的超大型数据库。常在银行系统中使用
SQL Server	MicroSoft 微软公司收费的中型的数据库。C#、.net等语言常使用。但该数据库只能运行在windows机器上，扩展性、稳定性、安全性、性能都表现平平。

SQL: Structured Query Language 结构化的查询语言，所有关系型数据库通用的语言。用来对数据库和数据进行管理的命令。

1.4 命令方式启动MySQL

启动MySQL

```
net start mysql
```

关闭MySQL

```
net stop mysql
```

1.5 命令行登录数据库

命令	说明
mysql -u 用户名 -p 密码	使用指定用户名和密码登录当前计算机中的MySQL数据库
mysql -h 主机IP -u 用户名 -p 密码	-h 指定IP 方式,进行 登录

登录数据库

```
mysql -uroot -p123
```

退出数据库

```
exit 或者 quit
```

2- MySql的目录结构

2.1 MySQL安装目录

MySQL的默认安装目录在 C:\Program Files\MySQL\MySQL Server 5.7

目录	目录内容
bin	放置一些可执行文件
docs	文档
include	包含头文件
lib	依赖库
share	用于存放字符集、语言等信息。

MySQL配置文件与数据库及数据表所在目录

(C:) > ProgramData > MySQL > MySQL Server 5.7			
名称		修改日期	类型
Data	保存的是数据库和数据表的信息	2020/5/9 16:17	文件夹
Uploads		2020/5/9 14:27	文件夹
installer_config.xml		2020/5/9 14:27	XML 文档
my.ini	Mysql的配置文件	2020/5/9 14:27	配置设置

2.2 数据库管理系统

2.2.1 什么是数据库管理系统

- 数据库管理系统 (DataBase Management System, DBMS) : 指一种操作和管理维护数据库的大型软件。
- MySQL就是一个 数据库管理系统软件, 安装了Mysql的电脑,我们叫它数据库服务器。

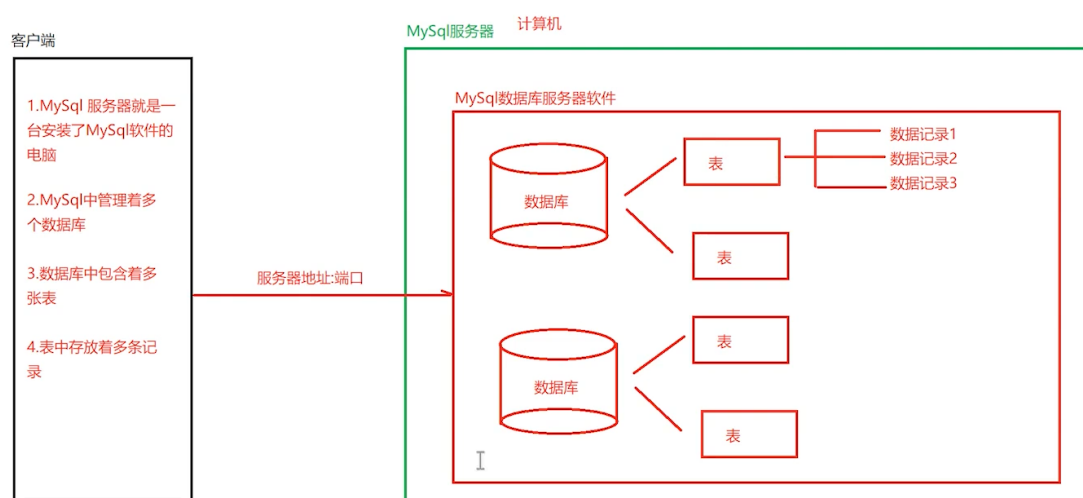
2.2.2 数据库管理系统的作用

用于建立、使用和维护数据库, 对数据库进行统一的管理。

2.2.3 数据库 和表之间的关系

- MySQL中管理着很多数据库, 在实际开发环境中 一个数据库一般对应了一个的应用。
- 数据库当中保存着多张表, 每一张表对应着不同的业务, 表中保存着对应业务的数据。

图解示例



3-SQL语句

3.1 SQL的概念

3.1.1 什么是SQL

- 结构化查询语言(Structured Query Language)简称SQL, 是一种特殊目的的编程语言, 是一种数据库查询和程序设计语言。
- 用于存取数据以及查询、更新和管理关系数据库系统。

3.1.2 SQL 的作用

- 是所有关系型数据库的统一查询规范, 不同的关系型数据库都支持SQL。
- 所有的关系型数据库都可以使用SQL, 不同数据库之间的SQL 有一些区别方言。

3.2 SQL通用语法

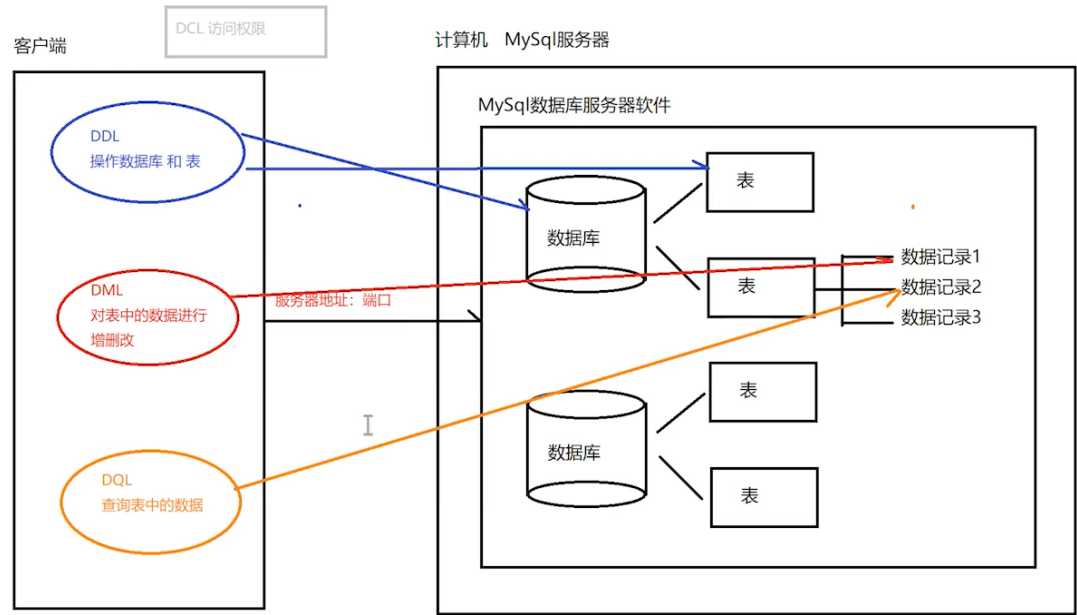
- SQL语句可以单行 或者 多行书写, 以分号 结尾;
- 可以使用空格和缩进来增加语句的可读性。
- MySQL中使用SQL不区分大小写, 一般关键字大写, 数据库名 表名列名 小写。
- 注释方式

注释语法	注解说明
-- 空格	单行注释
/* */	多行注释
#	MySql特有的单行注释

3.3 SQL的分类

分类	说明
数据定义语言	简称DDL(Data Definition Language)，用来定义数据库对象：数据库，表，列等。
数据操作语言	简称DML(Data Manipulation Language)，用来对数据库中表的记录进行更新。
数据查询语言	简称DQL(Data Query Language)，用来查询数据库中表的记录。
数据控制语言	简称DCL(Data Control Language)，用来定义数据库的访问权限和安全级别，及创建用户。

图示



4- DDL:数据库定义语言

4.1 数据库操作

4.1.1 创建数据库

```
-- 创建数据库
CREATE DATABASE db1;

-- 判断是否存在，如果不存在则创建数据库db2
create database if not exists db2;

-- 创建数据库并指定字符集为gbk
create database if not exists db3 character set gbk;
```

4.1.2 查看数据库

```
-- 查看数据库
SHOW DATABASES;

-- 查看某个数据库的信息
SHOW CREATE DATABASE db3;
```

4.1.3 修改数据库

```
-- 修改数据库
ALTER DATABASE db3 CHARACTER SET utf8;

-- 查看某个数据库的信息
SHOW CREATE DATABASE db3;
```

4.1.4 删除数据库

```
-- 创建db2数据库
CREATE DATABASE if not exists db2;

-- 删除db2数据库
DROP DATABASE db2;

-- 查看数据库
SHOW DATABASES;
```

4.1.5 使用当前数据库

```
-- 查看正在使用的数据库
SELECT DATABASE();

-- 使用db3数据库
USE db3;
```

4.2 数据表操作

基本语法

```
-- 字段名就是列名，字段类型：数据类型，约束。多个字段之间使用逗号来分隔
create table 表名 (
    字段名1 字段类型1 约束1,
    字段名2 字段类型2 约束2
)
```

MySQL数据类型

数据类型	关键字
整型	int或integer
浮点型	double或float
字符串型	varchar或char
日期类型	date或datetime

图示:

分类	类型名称	类型说明
整数	tinyInt	微整型：很小的整数(占 8 位二进制)
	smallint	小整型：小的整数(占 16 位二进制)
	mediumint	中整型：中等长度的整数(占 24 位二进制)
	int(integer)	整型：整数类型(占 32 位二进制)
	bigint	大整型：占 64 位二进制
小数	float	单精度浮点数，占 4 个字节
	double	双精度浮点数，占 8 个字节
	decimal(m,n)	数值类型，m 表示数值的长度，n 表示小数的位数
日期	time	只表示时间类型
	date	只表示日期类型
	datetime	表示日期和时间类型 时间范围为：'1000-01-01 00:00:00' 到 '9999-12-31 23:59:59'
	timestamp	表示日期和时间类型 时间范围为：'1970-01-01 00:00:01' 到 '2038-01-19 03:14:07'
字符串	char(m)	固定长度的字符串，无论使用几个字符都占满全部，M 为 0~255 之间的整数
	varchar(m)	可变长度的字符串，使用几个字符就占用几个，M 为 0~65535 之间的整数

4.2.1 创建表结构

```
-- 创建数据库test1
create database if not exists test1;

-- 使用数据库test1
use test1;

-- 创建student表包含id整数,name变长字符串长20,sex性别定长型1,birthday字段日期类型
create table student(
    -- 字段名,字段类型
    id int,
    name VARCHAR(20),
    sex char(1),
    birthday date
);
```

4.2.2 查看表结构

```
-- 查看数据库的所有表
SHOW TABLES;

-- 查看某个表结构
```

```
DESC student;

-- 查看创建表的SQL语句
SHOW CREATE TABLE student;

-- 创建相同的表结构
CREATE TABLE stu1 LIKE student;

-- 查看stu1表结构
DESC stu1;
```

4.2.3 删除表结构

```
-- 使用test1数据库
use test1
-- 显示数据库中所有表
show tables;

-- 创建数据表
create table stu2(
    -- 字段名,字段类型
    id int,
    name VARCHAR(20),
    sex char(1)
);

-- 直接删除表
DROP TABLE stu1;

-- 判断表是否存在并且删除stu2
DROP TABLE if EXISTS stu2;
```

4.2.4 修改表结构

添加表列ADD

```
alter table 表名 add 字段名 数据类型
```

修改列类型MODIFY

```
alter table 表名 modify 字段名 新的数据类型
```

修改列名和类型 CHANGE

```
alter table 表名 change 旧列名 新列名 数据类型;
```

删除列 DROP

```
alter table 表名 drop 列名;
```

修改表名

注：MySQL中没有直接修改库名的语句

```
rename table 旧表名 to 新的表名;
```

案例语句

```
-- 查看student的结构
DESC student;

-- 为学生表添加一个新的字段remark,类型为char(20)
ALTER TABLE student ADD remark CHAR(20);

-- 将student表中的remark字段的改成varchar(100)
ALTER TABLE student MODIFY remark VARCHAR(100);

-- 将student表中的remark字段名改成intro, 类型varchar(30)
ALTER TABLE student CHANGE remark intro VARCHAR(30);

-- 删除student表中的字段intro
alter TABLE student DROP intro;

-- 将学生表student改成student2
RENAME TABLE student to student2;

-- 查看所有test01的数据表
SHOW TABLES;
```

5-DML:数据库操纵语言

5.1 插入记录

5.1.1 插入全部字段

```
insert into 表名 values(值1, 值2.....)
```

5.1.2-插入部分字段

```
insert into 表名(列名1, 列名2) values(值1, 值2);
```

注: 没有添加数据的字段值为NULL

5.1.3-插入多条记录

```
insert into 表名 values(值1, 值2), (值1, 值2), (值1, 值2)
```

案例语句

```
-- 查询表中的记录
SELECT * from student;

-- 注意:在插入 varchar char date 类型的时候,必须要使用单引号或者双引号进行包裹

-- 向表中插入所有字段
INSERT INTO student VALUES(1, 'curry', '男', '1988-3-14');

-- now() 函数表示现在的时间: 同时包含日期和时间
```



```

INSERT INTO student VALUES(2, 'James', '男', now());

-- 插入部分数据，往学生表中添加id, name数据
INSERT INTO student(id, name) VALUES(3, 'Rondo');
delete from student where id=4;

-- 一条insert语句插入多条记录
INSERT INTO student VALUES (4, "guardwhy", "男", "1993-6-19"), (5, "邓肯", "男", "1976-4-25"), (6, "kobe", "男", "1978-8-23");

```

5.2-更新表记录

5.2.1 不带条件修改数据

修改所有行的记录

```
update 表名 set 字段名 = 值
```

5.2.2 带条件修改数据

修改符合条件的数据

```
update 表名 set 字段名 = 值 where 条件
```

案例语句

```

-- 查询表中的记录
SELECT * from stu2;

-- 为学生表添加一个新的字段birthday,类型为date
ALTER TABLE stu2 ADD birthday date;

-- 插入数据
INSERT INTO stu2 VALUES (1, "小刘", "男", "1993-6-30"), (2, "李红", "女", "1976-4-11"), (3, "小李", "男", "1998-8-13");

-- 不带条件修改数据，将所有的性别改成女
UPDATE stu2 set sex = '女';

-- 带条件修改数据，将id号为2的学生性别改成男
UPDATE stu2 set sex= "男" where id = 2;

-- 一次修改多个列，把id为3的学生，修改性别为男，生日：1996-03-18
UPDATE stu2 set sex = "男", birthday = "1996-03-18" WHERE id = 3;

```

5.3-删除表记录

5.3.1 不带条件删除数据

```

-- 删除表中所有的记录。
delete from 表名;

```

5.3.2 带条件删除数据

```
-- 删除符合条件的记录
delete from 表名 where 条件
```

5.3.3 删除表中所有记录

```
-- 与前两条语法的区别:相当于先删除表结构:drop table 表名; 然后在创建一张相同结构的表:create
table 表名;
truncate 表名;
```

案例语句

```
use test1;

-- 查看表结构
SHOW TABLES;

-- 查看stu2表结构
select * from stu2;

-- 带条件删除数据,删除id为3的记录
DELETE from stu2 WHERE id = 3;

-- 不要带条件删除数据,删除表中的所有数据
DELETE from stu2;

-- 查看stu2表结构
SELECT * FROM stu2;

-- 插入多条记录
INSERT INTO stu2 VALUES (1, "guardwhy", "男", "1993-6-19"), (2, "邓肯", "男", "1976-
4-25"), (3, "kobe", "男", "1978-8-23");

-- 删除表结构再创建表
truncate stu2;
```

6-DCL:数据库控制语言

6.1创建用户

语法:

```
create user '用户名'@'主机名' identified by '密码';
主机名:a.如果是本地用户用localhost,如果想让该用户可以从任意远程主机登陆,可以使用通配符%。
```

案例语句

```
用户1: create user 'guardwhy1' @'localhost' identified by '123';
用户2: create user 'guardwhy2' @'%' identified by '123';
```

6.2 给用户授权

语法：

```
grant 权限 on 数据库名.表名 to '用户名'@'主机名'
```

权限：

授予用户的权限，如 CREATE、ALTER、SELECT、INSERT、UPDATE、DELETE等，如果要授予所有的权限则使用ALL。

案例语句

1、给 guardwhy1用户分配对 stu1这个数据库操作的权限：创建表，修改表，插入记录，更新记录。

```
grant create, alter, insert, update, select on stu1.* to 'guardwhy1'@'localhost';
```

2、给guardwhy2用户分配所有权限，对所有数据库的所有表。

```
grant all on *.* to 'guardwhy2'@'%';
```

6.3 撤销授权

语法

```
revoke 权限 on 数据库名.表名 from '用户名'@'主机名'
```

权限：

授予用户的权限，如 CREATE、ALTER、SELECT、INSERT、UPDATE、DELETE等，如果要授予所有的权限则使用ALL。

给哪个用户撤销，要加上单引号。与创建用户时的用户名和主机名要相同。

案例语句

撤销 guardwhy1 用户对 stu1数据库所有表的操作的权限。

```
revoke all on stu1.* from 'guardwhy1'@'localhost'
```

6.4 删改操作

6.4.1-删除用户

语法：

```
drop user '用户名'@'主机名'
```

案例语句：

```
drop user 'guardwhy2'@'%';
```

6.4.2 修改管理员密码

语法:

```
mysqladmin -u 用户名 -p password 新密码
```

案例语句:

```
mysqladmin -u root -p password 123
```

6.4.3 修改用户自己的密码

语法

```
set password = password('密码');
```

案例语句

- 1、以guardwhy1用户登陆。
- 2、修改guardwhy1的密码为abcd
- 3、重新以新的密码登陆。

```
set password = password('abcd');  
mysql -u guardwhy1 -p abcd
```

7-DQL 查询表中数据

7.1 基本查询

7.1.1 查询所有的数据

```
select * from 表名;
```

7.1.2 查询指定的多个列

```
select 列名1, 列名2 from 表名;
```

案例语句

```
-- 使用数据库  
USE test1;  
  
-- 查询数据表  
SELECT * FROM student;  
  
-- 查询student表中的id和name列  
SELECT id, name FROM student;  
  
-- 没有指定条件, 查询的是所有的行。条件是对行进行过滤  
SELECT name, birthday FROM student;
```

7.1.3 指定列的别名

AS关键字

```
-- 使用别名,定义别名关键字是:AS
SELECT id AS 编号, name AS 姓名 FROM student;
-- AS关键字可以省略
SELECT name 姓名, birthday 生日 FROM student;
```

7.1.4 清除重复值

```
select distinct 1个或多个字段名 from 表名
```

案例语句

```
-- 添加一个字段address VARCHAR(50)
ALTER TABLE student ADD address VARCHAR(50);

UPDATE student SET address = "广州";

-- 查询学生来自于哪些地方,并且去掉重复行
SELECT address FROM student;
SELECT DISTINCT address FROM student;

-- 查询学生的姓名和地址,去掉重复行.必须几个列都相同,才会去除
SELECT DISTINCT name, address FROM student;
```

7.1.5 查询结果参与运算

注意: 参与运算的必须是数值类型

1、固定值运算

```
select 列+数值 from 表名
```

2、其他列数据参与运算

```
select 列表1+列表2 from 表名
```

案例语句

```
-- 使用数据库
USE test1;

-- 查询数据表
SELECT * FROM student;

-- 修改student表结构,添加数学和英语成绩列
ALTER TABLE student ADD math int, ADD English int;

-- 将student中的math字段名改成Math, 类型为INT
ALTER TABLE student CHANGE math Math INT;

-- 查询name Math字段
```

```

select name, Math FROM student;

-- 更新数据
UPDATE student set math = "90", English = "96" WHERE id = 1;
UPDATE student set math = "99", English = "95" WHERE id = 2;
UPDATE student set math = "91", English = "88" WHERE id = 3;
UPDATE student set math = "90", English = "96" WHERE id = 4;
UPDATE student set math = "95", English = "100" WHERE id = 5;
UPDATE student set math = "98", English = "98" WHERE id = 6;

-- 只影响查询结果
SELECT name, math-10 from student;

-- 查询所有列与math + english的和并使用别名"总成绩"
select *,(math+english) 总成绩 from student;

```

7.2 条件查询

7.2.1 运算符查询

```
select 列名 from 表名 where 条件表达式
```

7.2.2 比较运算符

运算符	作用
> < <= >= = <> !=	大于、小于、大于(小于)等于、不等于
BETWEEN ...AND...	显示在某一区间的值 例如: 2000-10000之间: Between 2000 and 10000
IN(集合)	集合表示多个值,使用逗号分隔,例如: name in (悟空, 八戒) in中的每个数据都会作为一次条件,只要满足条件就会显示
LIKE '%刘%'	模糊查询
IS NULL	查询某一列为NULL的值, 注: 不能写 = NULL

7.2.2-逻辑运算符

and 或 &&	全真为真
or 或	见真为真
not 或 !	取反

案例语句

```

-- 使用数据库
USE test1;

-- 创建数据表
CREATE TABLE stu3(
    id INT, -- 编号
    NAME VARCHAR(20), -- 姓名

```

```

age INT, -- 年龄
sex VARCHAR(5), -- 性别
address VARCHAR(100), -- 地址
Math INT, -- 数学
English INT -- 英语
);

-- 插入数据
INSERT INTO stu3(id,NAME,age,sex,address,Math,English) VALUES (1,'刘
云',18,'男','广州',66,78),(2,'刘涛',30,'女','深圳',98,87),

(3,'马小天',35,'男','北京',56,77),(4,'柳云',20,'女','湖南',76,65),

(5,'张晓天',20,'女','北京',86,NULL),(6,'侯大利',27,'男','重庆',99,99),

(7,'田甜',42,'女','重庆',99,99),(8,'朱琳',50,'男','南京',56,65);

-- 查询数据表
SELECT * FROM stu3;

-- 查询math分数大于80分的学生
select * from stu3 where Math > 80;

-- 查询english分数小于或等于78分的学生
SELECT * FROM stu3 WHERE Math <= 77;

-- 查询age等于20岁的学生
SELECT * FROM stu3 WHERE age = 20;

-- 查询age不等于20岁的学生
SELECT *FROM stu3 WHERE age <>20;
SELECT * FROM stu3 where age!= 20;

-- 查询age大于35且性别为男的学生(两个条件同时满足)
select * FROM stu3 where age>35 && sex="男";
select * FROM stu3 where age>35 AND sex="男";

-- 查询age大于35或性别为男的学生(两个条件其中一个满足)
SELECT * FROM stu3 where age > 35 or sex = "男";

-- 查询id是1或者3或者5的学生
SELECT * FROM stu3 WHERE id = 1 OR id = 3 OR id = 5;

```

7.4 关键字查询

7.4.1-in关键字

语法: in里面的每个数据都会作为一次匹配条件,只要满足条件的就会显示。

```
select 列名 from 表名 where 列名 in (值1, 值2, 值3);
```

7.4.2-范围查询

语法：表示从值1到值2范围，包头又包尾

```
select 列名 from 表名 where 列名 between 小值 and 大值
```

7.4.3-like关键字

语法：用于字符串的模糊查询

```
select 列名 from 表名 where 列名 like '字符串'
```

7.4.4-MySQL通配符

语法：如果没有通配符，相当于**等于号**。char, varchar

通配符	说明
%	匹配多个字符
-	匹配一个字符

7.4.5-查询为空的列

语法：判断列是否为空

```
IS NULL 不写成=NULL
```

案例语句

```
# 模糊查询

-- 查询姓马的学生
SELECT * from stu3 where `NAME` LIKE '刘%';
-- 查询姓名中包含'天'的姓名
SELECT * FROM stu3 WHERE name LIKE "%天%";
-- 查询姓刘,且姓名中有2个字的学生
SELECT * FROM stu3 where `NAME` LIKE "刘_";

# 查询为空的列

-- 查询英文成绩为NULL的学生
SELECT * FROM stu3 WHERE English IS NULL;
-- 查询英语成绩不为NULL的学生
SELECT * FROM stu3 WHERE English IS NOT NULL;
-- 查询姓名和英语成绩，如果英语为null，则显示为0分
SELECT `NAME`,English FROM stu3;
-- IFNULL(列名,默认值)：如果这一列有值,则显示它的值,如果为NULL,则显示后面默认值
SELECT name, English(English,0) 英语 FROM stu3;
```

8-DQL操作单表

8.1 查询排序

语法：对指定的列进行排序，默认是asc升序。

```
select 列名 from 表名 order by 列名 asc/desc      升序:asc   降序:desc
```

案例语句：

```
# 排序
-- 查询所有数据,使用年龄降序排序
SELECT * FROM stu3 ORDER BY age DESC;
-- 查询所有数据大于20岁的学生,在年龄降序排序的基础上,如果年龄相同再以数学成绩升序排序
SELECT * FROM stu3 ORDER BY age , Math ASC
```

8.2 聚合函数

```
SELECT 聚合函数(字段名) FROM 表名;
```

常用的聚合函数

SQL中的聚合函数	作用
count(列名)	统计个数(行)
sum(列名)	求和，对数值类型的列求和
avg(列名)	求平均，对数值类型的列求平均
max(列名)	求这一列中最大值
min(列名)	求这一列中最小值

案例语句：

```
# 聚合函数
-- 查询学生总数
SELECT COUNT(id) FROM stu3; -- 8
SELECT COUNT(English) FROM stu3; -- 7
SELECT count(*) FROM stu3; -- 8

-- 查询年龄大于40的总数
SELECT COUNT(id) FROM stu3 WHERE age > 40; -- 2

-- 查询数学成绩总分
SELECT SUM(Math) 数学总分 FROM stu3; -- 636

-- 查询数学成绩的平均分
SELECT AVG(Math) 数学平均分 FROM stu3; -- 79.5000

-- 查询数学成绩最高分
SELECT MAX(Math) 数学最高分 FROM stu3; -- 99

-- 查询数学成绩最低分
SELECT MIN(Math) 数学最低分 FROM stu3; -- 56
```

8.3 分组查询

语法：对表中所有的行进行分组查询，默认是返回每一组的第一行记录，可以使用having对分组后的结果再进行过滤。

```
select 列名 from 表名 group by 列名 having 条件
```

注意：MySQL5.7以上版本使用group by关键字出现报错。

解决方案：

找到数据库配置文件 `my.cnf`，在里面的 `[mysqld]` 中添加以下语句：

```
sql_mode =  
"STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_  
ENGINE_SUBSTITUTION"
```

having与where的区别

关键字	功能
where 子句	1. 先过滤，再进行分组 2. where后面不能使用聚合函数
having子句	1. 先分组，再进行过滤 2. having后面可以使用聚合函数，having用在group by后面

8.3.1 直接分组

将分组字段结果中相同内容作为一组，如按性别将学生分成2组。

```
-- 按性别分组,查询每组的第一条记录  
select * from stu3 group by sex ;
```

8.3.2 和聚合函数使用

```
-- 按性别分组,求男女数学平均分  
SELECT sex, AVG(Math) FROM stu3 GROUP BY sex;  
-- 按性别分组,求男女数学总和  
SELECT sex, SUM(Math) FROM stu3 GROUP BY sex;  
  
-- 查询年龄大于25岁的人,按性别分组,统计每组的人数  
SELECT sex, COUNT(*) FROM stu3 WHERE age > 25 GROUP BY sex;
```

8.3.3 对分组后的结果过滤

```
-- 查询年龄大于25岁的人,按性别分组,统计每组的人数,并只显示性别人数大于2的数据  
SELECT sex, COUNT(*) FROM stu3 WHERE age > 25 GROUP BY sex HAVING COUNT(*) > 2;
```

8.4 分页操作

8.4.1 limit的作用

- limit是限制的意思,用于 限制返回的查询结果的行数 (可以通过limit指定查询多少行数据)。
- limit 语法是 MySql的方言,用来完成分页。

语法格式

```
SELECT 字段1,字段2... FROM 表名 LIMIT offset , length;
```

参数说明

limit offset , length; 关键字可以接受一个 或者两个 为0 或者正整数的参数

offset 起始行数, 从0开始记数, 如果省略 则默认为 0。

length 返回的行数。

案例语句：

```
SELECT * from stu3;

-- 插入数据
INSERT INTO stu3(id,NAME,age,sex,address,Math,English) VALUES
(9,'侯国龙',50,'男','江州',87,78),
(10,'黄晓军',18,'男','广州',100,66),
(11,'杜强',22,'男','江州',58,78),
(12,'石秋菊',50,'男','扬州',77,88),
(13,'朱倩',22,'女','北京',66,66),
(14,'张晓欣',23,'女','江州',88,88);

# limit的作用
-- 返回0到3这三条记录 (起始行,返回行数)
SELECT * from stu3 LIMIT 0,3;

-- 查询学生表中数据, 从第3条开始显示, 显示6条。
SELECT * FROM stu3 LIMIT 2,6;

-- 每页显示5条
SELECT * FROM stu3 LIMIT 0,5;
SELECT * FROM stu3 LIMIT 5,5;
SELECT * FROM stu3 LIMIT 10,5;
```

8.4.2 增删改查关键字

操作类型	增	删	改	查
DDL对表和库的操作	create	drop	alter	show
DML和DQL对记录的操作	insert	delete	update	select

9- SQL约束

9.1 约束种类

约束名	约束关键字
主键	primary key
非空	not null
唯一	unique
外键	foreign key ... references
默认	default

9.2 约束的作用

约束通常是在创建表结构的时候创建的，表中约束防止不符合要求的数据添加到表中，保证表中数据的正确性。如果表中已经有违反约束的数据，会导致约束创建失败。

9.3 主键约束

9.3.1 主键特点

- 唯一标识表中每一条记录，通过主键来查询到每一条记录。
- 非空:不能为NULL。
- 唯一:不能重复。

9.3.2 创建主键

1. 在创建表的时候给字段添加主键
`create table` 表名(
 字段名字 数据类型 `primary key`
)
2. 在已有表中添加主键, 修改表结构
`alter table` 表名 `add primary key`(列名)

案例语句

```
-- 使用数据库
USE test1;

-- 创建学生表stu4,包括字段(id, name, age)将id作为主键
CREATE TABLE stu4(
    id INT PRIMARY KEY,
    NAME VARCHAR(20),
    age INT
);

-- 插入主键值

INSERT INTO stu4 VALUES(1, "张三", 18);

INSERT INTO stu4 VALUES(2, "小刘", 21);
INSERT INTO stu4 VALUES(3, "李刚", 51);
INSERT INTO stu4 VALUES(4, "科比", 41);
```

```
-- 删除stu4表的主键
ALTER TABLE stu4 DROP PRIMARY KEY;

-- 表存在的情况下,添加主键
ALTER TABLE stu4 ADD PRIMARY KEY(id);

SELECT * FROM stu4;

-- stu4的结构
DESC stu4;
```

9.3.3 主键自增长

由系统自动增长主键,不需要人为添加主键的值避免重复,auto_increment只能用在主键上,不能用在其它的字段上。

语法:

```
create table 表名(
    字段名字 数字类型 primary key auto_increment
)
```

案例语句

```
-- 创建学生表st5, 包含字段(id, name, age)将id做为主键并自动增长
CREATE TABLE stu5(
    id INT PRIMARY KEY auto_increment,
    NAME VARCHAR(20),
    age INT
);

-- 插入数据
INSERT INTO stu5(`NAME`, age) VALUES("张三", 10);
INSERT INTO stu5(`NAME`, age) VALUES("李四", 23);

-- 将主键的起始值设置为100
ALTER TABLE stu5 auto_increment = 100;

-- 查询stu5
SELECT * FROM stu5;

-- stu5结构
DESC stu5;

-- 删除自增长的值(数据表)
DELETE FROM stu5;

-- 删除表中所有记录
TRUNCATE stu5;
```

9.4 唯一约束

指定某一列不能出现相同的值.

语法:

```
create table 表名(  
    字段名 字段类型 unique  
)
```

案例语句

```
-- 创建学生表stu6, 包含字段(id,name), NAME这一列设置唯一约束, 不能出现同名的学生  
CREATE TABLE stu6(  
    id int,  
    NAME VARCHAR(20) UNIQUE  
);  
-- 表结构  
DESC stu6;  
  
-- 添加数据  
INSERT INTO stu6 VALUES(1, "张三");  
-- 再次添加  
# INSERT INTO stu6 VALUES(2, "张三"); -- 插入失败, Duplicate entry '张三' for key  
'name'  
  
-- 删除数据  
DELETE from stu6 WHERE id = 1;  
  
-- 查询表数据  
SELECT * FROM stu6;
```

9.5 非空约束

设置某列数据不能为空, 必须要输入。

语法格式:

```
create table 表名(  
    字段名, 字段类型 not null  
)
```

案例语句

```
# 非空约束  
-- 创建表学生表stu7, 包含字段(id,name,gender)其中name不能为NULL  
CREATE TABLE stu7(  
    id INT,  
    NAME VARCHAR(20) NOT NULL,  
    age INT  
);  
  
-- 查看表结构  
DESC stu7;  
  
-- 添加一条记录其中姓名不赋值  
# INSERT INTO stu7 VALUES(1, NULL, 20); -- 1048 - Column 'NAME' cannot be null  
# INSERT INTO stu7(id, age) VALUES(1, 30); -- 1364 - Field 'NAME' doesn't have  
a default value
```

```
-- 添加数据
INSERT INTO stu7 VALUES(1, "李四", 20);
-- 查询表中数据
SELECT * FROM stu7;
```

9.6 默认值约束

如果一个字段没有设置它的值，将使用默认值。

语法格式:

```
create table 表名(
    字段名 字段类型 default 默认值
)
```

案例语句

```
# 默认值约束
-- 创建一个学生表,包含字段(id, name, address),地址默认值是广州
CREATE TABLE stu5(
    id INT,
    NAME VARCHAR(20),
    address VARCHAR(30) DEFAULT "广州"
);

-- 查看表结构
DESC stu5;

-- 添加一条记录,使用默认地址
INSERT INTO stu5(id, `NAME`) VALUES(1, "刘菲");
INSERT INTO stu5 VALUES (2, "侯大利", DEFAULT);

-- 添加一条记录,不使用默认地址
INSERT INTO stu5 VALUES(3, "杨帆", "深圳");
-- stu5表信息
SELECT * FROM stu5;
-- 删除id=3的记录
DELETE FROM stu5 WHERE id = 3;
```

9.7 外键约束

9.7.1 创建外键

新表创建时候添加外键

```
create table 表名 (
    外键字段名 字段类型,
    constraint 约束名 foreign key(外键字段名)references 主表(主键)
)
```

已有表添加外键

```
alter table 表名 add constraint 约束名 foreign key(外键字段名) references 主表(主键)
```

9.7.2 删除外键

```
alter table 从表 drop foreign key 约束名;
```

sql语句

```
use db_mysql;

-- 数据库所有表
SHOW TABLES;

-- 部门表
CREATE TABLE department(
    id INT PRIMARY KEY auto_increment, -- 部门表主键
    dep_name VARCHAR(10), -- 部门名
    dep_location VARCHAR(20) -- 部门所在城市
);

-- 添加部门表的记录
insert into department values(null, '刑警部', '江州'), (null, '法医部', '阳州');

-- 员工表
CREATE TABLE employee(
    id INT PRIMARY KEY auto_increment, -- 主键自增长
    NAME VARCHAR(20), -- 姓名
    age INT,
    dept_id INT -- 外键, 引用部门表中的主键
    -- CONSTRAINT fk_ep_id FOREIGN KEY(dept_id) REFERENCES department(id)
);

-- 删除外键
alter table employee drop foreign key fk_ep_id;

-- 添加从表存在的情况下添加外键
alter TABLE employee ADD constraint fk_ep_id FOREIGN KEY(dept_id) REFERENCES
department(id);

# 向员工表中添加元素
INSERT INTO employee (NAME, age, dept_id) VALUES ('侯大利', 27, 1);
INSERT INTO employee (NAME, age, dept_id) VALUES ('朱琳', 50, 1);
INSERT INTO employee (NAME, age, dept_id) VALUES ('张小天', 30, 1);
INSERT INTO employee (NAME, age, dept_id) VALUES ('田甜', 26, 2);
INSERT INTO employee (NAME, age, dept_id) VALUES ('老李', 42, 2);
INSERT INTO employee (NAME, age, dept_id) VALUES ('樊勇', 35, 2);

-- 查询部门表信息
SELECT * FROM department;

-- 查询员工表信息
SELECT * FROM employee;

-- 删除表结构
DROP table department;
DROP table employee;
```


9.7.3 外键约束注意事项

1. 从表外键类型必须与主表主键类型一致 否则创建失败。
- 2) 添加数据时, 应该先添加主表中的数据。

```
-- 添加一个新的部门
INSERT INTO department(dep_name,dep_location) VALUES('缉毒部','秦阳');
-- 添加一个属于市场部的员工
INSERT INTO employee(ename,age,dept_id) VALUES('张小欣',24,3);
```

3. 删除数据时,应该先删除从表中的数据。

```
-- 先删除从表的关联数据
DELETE FROM employee WHERE dept_id = 3;
-- 再删除主表的数据
DELETE FROM department WHERE id = 3;
```

9.7.4 级联操作

语法: 写在外键约束的后面, 在创建外键约束的时候创建级联操作。

级联操作	语法
级联更新	on update cascade
级联删除	on delete cascade

案例语句

```
-- 删除外键约束
ALTER TABLE employee DROP FOREIGN KEY fk_ep_id;
-- 添加外键约束,级联更新和级联删除
ALTER TABLE employee ADD CONSTRAINT fk_ep_id FOREIGN KEY(dep_id) REFERENCES
department(id) ON UPDATE ON DELETE CASCADE;

SELECT * from employee;

-- 删除员工表id=6的值
DELETE FROM employee WHERE id = 6;
-- 把部门表中id等于2的部门改成id等于3
UPDATE department set id=3 WHERE id=2

-- 删除部门号是3的部门
DELETE FROM department WHERE id = 3;

-- 查询部门表信息
SELECT * FROM department;
```

10- 数据库事务

10.1 什么是事务

- 事务是一个整体,由一条或者多条SQL 语句组成,这些SQL语句要么都执行成功,要么都执行失败。
- 只要有一条SQL出现异常,整个操作就会回滚,整个业务执行失败。

比如：银行的转账业务,kobe给curry转账500元 , 至少要操作两次数据库, kobe -500, curry + 500。

这中间任何一步出现问题,整个操作就必须全部回滚,这样才能保证用户和银行都没有损失。

10.2 模拟转账操作

(1) 创建账户表

```
-- 创建账户表
CREATE TABLE account(
    -- 主键
    id INT PRIMARY KEY AUTO_INCREMENT,
    -- 姓名
    NAME VARCHAR(10),
    -- 余额
    money DOUBLE
);

select * from account;
-- 添加用户
insert into account(NAME, money) VALUES('kobe', 1000),('curry',1000);
```

(2) 模拟kobe 给 curry 转 500元钱, 一个转账的业务操作最少要执行下面语句

```
-- 给kobe-500
update account set money = money - 500 where name = 'kobe';
-- 给kobe+500
update account set money = money + 500 where name = 'curry';
```

10.3 事务回滚

即在事务运行的过程中发生了某种故障,事务不能继续执行,系统将事务中对数据库的所有已完成的操作全部撤销,滚回到事务开始时的状态。(在提交之前执行)

10.4 MySQL事务操作

MYSQL 中可以有两种方式进行事务的操作:手动提交事务和自动提交事务。

10.4.1 手动提交事务

功能	语句
开启事务	start transaction;
提交事务	commit;
回滚事务	rollback;

START TRANSACTION

- 显式地标记一个事务的起始点。

COMMIT

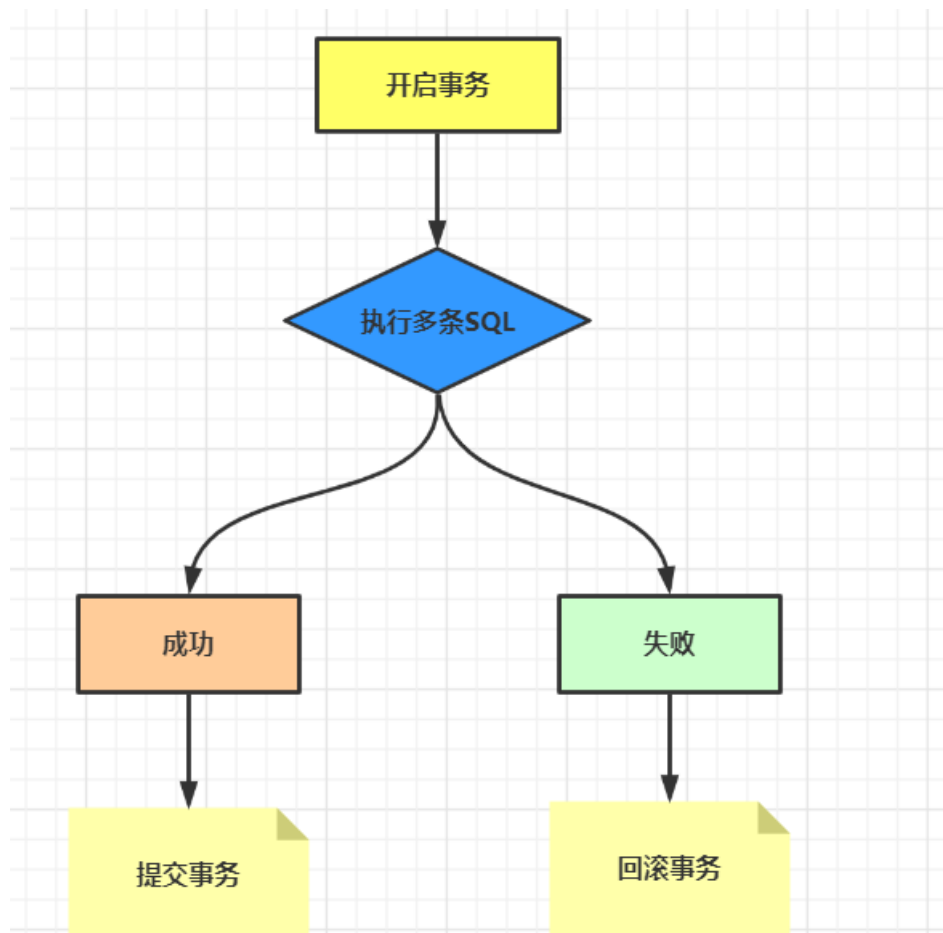
- 表示提交事务，即提交事务的所有操作，具体地说。
- 就是将事务中所有对数据库的更新都写到磁盘上的物理数据库中，事务正常结束。

ROLLBACK

- 表示撤销事务，即在事务运行的过程中发生了某种故障，事务不能继续执行。
- 系统将事务中对数据库的所有已完成的操作全部撤销，回滚到事务开始时的状态。

10.4.2 手动提交事务流程

- 执行成功的情况：开启事务 -> 执行多条 SQL 语句 -> 成功提交事务
- 执行失败的情况：开启事务 -> 执行多条 SQL 语句 -> 事务的回滚



10.4.3 成功案例

(1) 命令行登录数据库

```

PS C:\Users\linux> mysql -u root -p
Enter password: ****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 28
Server version: 5.7.30 MySQL Community Server (GPL)

Copyright (c) 2000, 2020, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

```

(2) 使用数据库

```
use db_mysql;
```

(3) 开启事务

```
start transaction;
```

(4) 执行转账操作

```

-- 给kobe-500
update account set money = money - 500 where name = 'kobe';
-- 给kobe+500
update account set money = money + 500 where name = 'curry';

```

(5). 执行commit操作

```
commit;
```

(6) 查看结果

信息	结果 1	剖析	状态
	id	NAME	money
▶	1	kobe	500
	2	curry	1500

10.4.4 自动提交事务

- MySQL 默认每一条 DML(增删改)语句都是一个单独的事务，每条语句都会自动开启一个事务。
- 语句执行完毕 自动提交事务，MySQL 默认开始自动提交事务。

10.4.5 取消自动提交

MySQL默认是自动提交事务,设置为手动提交。

(1) 登录mysql, 查看autocommit状态。

```
SHOW VARIABLES LIKE 'autocommit';
```

```
mysql> ^C
mysql> SHOW VARIABLES LIKE 'autocommit';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| autocommit    | ON    |
+-----+-----+
1 row in set, 1 warning (0.00 sec)

mysql>
```

on : 自动提交

off : 手动提交

(2) 把 autocommit 改成 off;

```
SET @@autocommit=off;
```

```
mysql> SET @@autocommit=off;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW VARIABLES LIKE 'autocommit';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| autocommit    | OFF   |
+-----+-----+
1 row in set, 1 warning (0.00 sec)

mysql>
```

10.5 事务的四大特性 ACID

特性	含义
原子性	每个事务都是一个整体，不可再拆分，事务中所有的 SQL 语句要么都执行成功，要么都失败。
一致性	事务在执行前数据库的状态与执行后数据库的状态保持一致。
隔离性	事务与事务之间不应该相互影响，执行时保持隔离的状态。
持久性	一旦事务执行成功，对数据库的修改是持久的。就算关机，数据也是要保存下来的。

10.6 事务隔离级别

10.6.1 数据并发访问

- 一个数据库可能拥有多个访问客户端,这些客户端都可以并发方式访问数据库.
- 数据库的相同数据可能被多个事务同时访问,如果不采取隔离措施,就会导致各种问题,破坏数据的完整性。

10.6.2 并发访问产生的问题

- 事务在操作时的理想状态：所有的事务之间保持隔离，互不影响。
- 因为并发操作，多个用户同时访问。同一个数据。可能引发并发访问的问题

并发访问的问题	含义
脏读	一个事务读取到了另一个事务中尚未提交的数据。
不可重复读	一个事务中两次读取的数据内容不一致, 要求的是在一个事务中多次读取时数据是一致的. 这是进行 update 操作时引发的问题
幻读	一个事务中,某一次的 select 操作得到的结果所表征的数据状态, 无法支撑后续的业务操作. 查询得到的数据状态不准确,导致幻读.

10.6.3 四种隔离级别

通过设置隔离级别,可以防止上面的三种并发问题。

- MySQL数据库有四种隔离级别 上面的级别最低，下面的级别最高。
- ✓ 会出现问题，✗ 不会出现问题。

级别	常见操作	隔离级别	脏读	不可重复读	幻读	数据库的默认隔离级别
1	读未提交	read uncommitted	✓	✓	✓	
2	读已提交	read committed	X	✓	✓	Oracle和SQLServer
3	可重复读	repeatable read	X	X	✓	MySql
4	串行化	serializable	X	X	X	

10.6.4 隔离级别相关命令

(1).查看隔离级别

```
select @@tx_isolation;
```

```
mysql> select @@tx_isolation;
+-----+
| @@tx_isolation |
+-----+
| REPEATABLE-READ |
+-----+
1 row in set, 1 warning (0.00 sec)
```

MySQL的默认级别，可重复读可以防止脏读与不可重复读，不能防止幻读。

(2).设置事务隔离级别，需要退出 MySQL 再重新登录才能看到隔离级别的变化。

```
set global transaction isolation level 级别名称;
read uncommitted 读未提交
read committed 读已提交
repeatable read 可重复读
serializable 串行化
```

10.7 隔离性问题

10.7.1 脏读演示

脏读: 一个事务读取到了另一个事务中尚未提交的数据。

打开窗口登录 MySQL，设置全局的隔离级别为最低

(1).登录MySQL

```
PS C:\Users\linux> mysql -u root -p
Enter password: ****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 3
Server version: 5.7.30 MySQL Community Server (GPL)

Copyright (c) 2000, 2020, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

(2) 使用db_mysql数据库

```
use db_mysql
```

(3) 设置隔离级别为最低 读未提交

```
set global transaction isolation level read uncommitted;
```

关闭窗口,开一个新的窗口A,再次查询隔离级别

(1). 开启新的窗口A

```
PS C:\Users\linux> mysql -u root -p
Enter password: ****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 3
Server version: 5.7.30 MySQL Community Server (GPL)

Copyright (c) 2000, 2020, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

(2). 查询隔离级别

```
select @@tx_isolation;
```



```
Type 'help;' or '\h' for help. Type '\c' to clear the c
mysql> select @@tx_isolation;
+-----+
| @@tx_isolation |
+-----+
| READ-UNCOMMITTED |
+-----+
1 row in set, 1 warning (0.00 sec)
```

已修改为读未提交

再开启一个新的窗口 B

(1) 登录数据库

```
Copyright (c) 2000, 2020, Oracle and/or its affiliates. All
Oracle is a registered trademark of Oracle Corporation and/or
affiliates. Other names may be trademarks of their respectiv
owners.
Type 'help;' or '\h' for help. Type '\c' to clear the curren
mysql>
```

窗口 B

(2) 使用db_mysql数据库

```
use db_mysql
```

(3) 开启事务

```
start transaction;
```

```
mysql> select * from account;
+----+-----+-----+
| id | NAME  | money |
+----+-----+-----+
| 1  | kobe  | 1000  |
| 2  | curry | 1000  |
+----+-----+-----+
2 rows in set (0.00 sec)
```

A窗口执行

(1) 使用db_mysql数据库

```
use db_mysql
```

(2) 开启事务

```
start transaction;
```

```
mysql> use db_mysql
Database changed
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)
```

A窗口

(3) 执行修改操作

```
-- 给kobe-500
update account set money = money - 500 where name = 'kobe';
-- 给curry+500
update account set money = money + 500 where name = 'curry';
```

B 窗口查询数据

(1) 查询账户信息

```
select * from account;
```

```
mysql> update account set money = money - 500 where name = 'kobe';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> update account set money = money + 500 where name = 'curry';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql>
```

A窗口中开启事务，对数据进行修改，但是没有提交事务。

```
mysql> select * from account;
+----+-----+-----+
| id | NAME | money |
+----+-----+-----+
| 1  | kobe | 500   |
| 2  | curry | 1500  |
+----+-----+-----+
2 rows in set (0.00 sec)

mysql>
```

B窗口中，查询到了A窗口未提交数据，出现了脏读

A窗口转账异常,进行回滚

```
rollback;
```

```
mysql> rollback;
Query OK, 0 rows affected (0.01 sec)

mysql>
```

B 窗口再次查询账户

```
select * from account
```

```
mysql> select * from account;
+----+-----+-----+
| id | NAME  | money |
+----+-----+-----+
|  1 | kobe  |  1000 |
|  2 | curry |  1000 |
+----+-----+-----+
2 rows in set (0.00 sec)
```

10.7.2 解决脏读问题

- 在 A 窗口设置全局的隔离级别为 read committed

```
set global transaction isolation level read committed;
```

- 重新开启A窗口, 查看设置是否成功

```
select @@tx_isolation;
```

```
mysql> select @@tx_isolation;
+-----+
| @@tx_isolation |
+-----+
| READ-COMMITTED |
+-----+
1 row in set, 1 warning (0.00 sec)
```

- 开启B 窗口, A 和 B 窗口选择数据库后, 都开启事务。

```
start transaction;
```

```
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> A窗口, 选择数据库, 开启事务

mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> use db_mysql;
Database changed
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)
```

- A窗口 只是更新两个人的账户, 不提交事务

```
-- 给kobe-500
update account set money = money - 500 where name = 'kobe';
-- 给curry+500
update account set money = money + 500 where name = 'curry';
```

- B窗口进行查询,没有查询到未提交的数据

```
select * from account;
```

```
mysql> use db_mysql;
Database changed
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> update account set money = money - 500 where name = 'kobe';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql> update account set money = money + 500 where name = 'curry';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql>

mysql> select * from account;
+----+-----+-----+
| id | NAME  | money |
+----+-----+-----+
| 1  | kobe  | 1000  |
| 2  | curry | 1000  |
+----+-----+-----+
2 rows in set (0.01 sec)
```

A窗口开启事务，对数据进行修改，但是不提交。

B窗口进行查询，数据没有改变

- A窗口commit提交数据

```
commit;
```

- B窗口查看数据

```
select * from account;
```

```
mysql> commit;
Query OK, 0 rows affected (0.17 sec)

mysql>
```

A窗口提交事务

```
mysql> select * from account;
+----+-----+-----+
| id | NAME  | money |
+----+-----+-----+
| 1  | kobe  | 500   |
| 2  | curry | 1500  |
+----+-----+-----+
2 rows in set (0.00 sec)
```

B窗口查询

10.7.3 不可重复读演示

不可重复读: 同一个事务中,进行查询操作,但是每次读取的数据内容是不一样的

- 打开两个窗口A和窗口B,选择数据库后开启事务

```
use db_mysql;
start transaction;
```

```
mysql> use db_mysql;
Database changed
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)
mysql>
```

A窗口, 选择数据库, 开启事务

```
mysql> use db_mysql;
Database changed
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)
mysql>
```

B窗口, 选择数据库, 开启事务

- B窗口开启事务后, 先进行一次数据查询

```
select * from account;
```

```
mysql> select * from account;
+----+-----+-----+
| id | NAME  | money |
+----+-----+-----+
|  1 | kobe  |  1000 |
|  2 | curry |  1000 |
+----+-----+-----+
2 rows in set (0.00 sec)
```

- 在 A 窗口开启事务后, 将用户kobe的账户 + 500 ,然后提交事务

```
-- 给kobe+500
update account set money = money + 500 where name = 'kobe';
-- 提交事务
commit;
```

```
mysql> use db_mysql;
Database changed
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> update account set money = money + 500 where name = 'kobe';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> commit;
Query OK, 0 rows affected (0.00 sec)
```

A窗口修改完数据之后, 提交事务。

- 同一个事务, B窗口再次查询数据。

```
mysql> use db_mysql;
Database changed
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from account;
+----+-----+-----+
| id | NAME  | money |
+----+-----+-----+
| 1  | kobe  | 1000  |
| 2  | curry | 1000  |
+----+-----+-----+
2 rows in set (0.00 sec)

mysql> select * from account;
+----+-----+-----+
| id | NAME  | money |
+----+-----+-----+
| 1  | kobe  | 1500  |
| 2  | curry | 1000  |
+----+-----+-----+
2 rows in set (0.00 sec)
```

第一次查询

第二次查询

同一个事务，两次查询结果不一样
出现了不可重复读的问题

10.7.4 解决不可重复读问题

将全局的隔离级别进行提升为：repeatable read

- 恢复数据

```
UPDATE account SET money = 1000
```

- 打开A 窗口, 设置隔离级别为：repeatable read

```
-- 查看事务隔离级别
select @@tx_isolation;
-- 设置事务隔离级别为 repeatable read
set global transaction isolation level repeatable read;
```

```
mysql> select @@tx_isolation;
+-----+
| @@tx_isolation |
+-----+
| READ-COMMITTED |
+-----+
1 row in set, 1 warning (0.02 sec)

mysql> set global transaction isolation level repeatable read;
Query OK, 0 rows affected (0.00 sec)
```

- 重新开启 A,B 窗口 选择数据库 ,同时开启事务

```
use db_mysql;
start transaction;
```

```
mysql> use db_mysql;
Database changed
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)
mysql>
```

A窗口, 选择数据库, 开启事务

```
mysql> use db_mysql;
Database changed
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)
mysql>
```

B窗口, 选择数据库, 开启事务

- B窗口事务 先进行第一次查询

```
select * from account;
```

- A窗口更新数据, 然后提交事务

```
update account set money = money + 500 where name = 'kobe';
-- 提交
commit;
```

```
mysql> use db_mysql;
Database changed
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> update account set money = money + 500 where name = 'kobe';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> commit;
Query OK, 0 rows affected (0.17 sec)
```

- B窗口再次查询

```
select * from account;
```

- 同一个事务中为了保证多次查询数据一致, 必须使用 repeatable read 隔离级别

```
mysql> use db_mysql;
Database changed
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from account;
+----+-----+-----+
| id | NAME  | money |
+----+-----+-----+
| 1  | kobe  | 1000  |
| 2  | curry | 1000  |
+----+-----+-----+
2 rows in set (0.00 sec)

mysql> select * from account;
+----+-----+-----+
| id | NAME  | money |
+----+-----+-----+
| 1  | kobe  | 1000  |
| 2  | curry | 1000  |
+----+-----+-----+
2 rows in set (0.00 sec)
```

10.7.5 幻读演示

幻读: select 某记录是否存在, 不存在, 准备插入此记录, 但执行 insert 时发现此记录已存在, 无法插入, 此时就发生了幻读。

- 打开 A B 窗口, 选择数据库 开启事务

```
mysql> use db_mysql;  
Database changed  
mysql> start transaction;  
Query OK, 0 rows affected (0.00 sec)  
mysql>
```

A窗口, 选择数据库, 开启事务

```
mysql> use db_mysql;  
Database changed  
mysql> start transaction;  
Query OK, 0 rows affected (0.00 sec)  
mysql>
```

B窗口, 选择数据库, 开启事务

- A 窗口 先执行一次查询操作

```
-- 添加一条id为3的数据, 在添加之前先判断是否存在  
select * from account where id = 3;
```

```
mysql> use db_mysql;  
Database changed  
mysql> start transaction;  
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> select * from account where id = 3;  
Empty set (0.00 sec)
```

```
mysql>
```

不存在, 可以插入数据

- B 窗口 插入一条数据 提交事务

```
INSERT INTO account VALUES(3, 'James', 1000);  
-- 提交  
commit;
```

```
mysql> INSERT INTO account VALUES(3, 'James', 1000);  
Query OK, 1 row affected (0.00 sec)
```

```
mysql> commit;  
Query OK, 0 rows affected (0.00 sec)
```

B窗口插入一条, 主键id为3的数据

- A 窗口执行 插入操作, 发现报错. 出现幻读

```
mysql> select * from account where id = 3;  
Empty set (0.00 sec)
```

报错, 主键重复

```
mysql> INSERT INTO account VALUES(3, 'James', 1000);  
ERROR 1062 (23000): Duplicate entry '3' for key 'PRIMARY'  
mysql>
```


10.7.6 解决幻读问题

- 将事务隔离级别设置到最高 serializable，以挡住幻读的发生。

注意

如果一个事务，使用了SERIALIZABLE——可串行化隔离级别时，在这个事务没有被提交之前，其他的线程，只能等到当前操作完成之后，才能进行操作，这样会非常耗时，而且影响数据库的性能，数据库不会使用这种隔离级别。

- 恢复数据

```
DELETE FROM account WHERE id = 3;
```

- 打开A 窗口 将数据隔离级别提升到最高

```
set global transaction isolation level SERIALIZABLE;
```

```
mysql> set global transaction isolation level SERIALIZABLE;
Query OK, 0 rows affected (0.00 sec)

mysql> select @@tx_isolation;
+-----+
| @@tx_isolation |
+-----+
| SERIALIZABLE   |
+-----+
1 row in set, 1 warning (0.00 sec)
```

- 打开A B 窗口, 选择数据库 开启事务

```
mysql> use db_mysql;
Database changed
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)
mysql>
```

A窗口，选择数据库，开启事务

```
mysql> use db_mysql;
Database changed
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)
```

B窗口，选择数据库，开启事务

- A窗口 先执行一次查询操作

```
SELECT * FROM account WHERE id = 3;
```

- B 窗口插入一条数据

```
INSERT INTO account VALUES(3,'james',1000);
```

- A 窗口执行 插入操作, 提交事务 数据插入成功。

```
INSERT INTO account VALUES(3,'james',1000);
commit;
```

```
mysql> INSERT INTO account VALUES(3,'james',1000);
ERROR 1062 (23000): Duplicate entry '3' for key 'PRIMARY'
mysql> INSERT INTO account VALUES(3,'james',1000);
ERROR 1062 (23000): Duplicate entry '3' for key 'PRIMARY'
mysql>
```

- B窗口在A窗口提交事务之后, 再执行,但是主键冲突出现错误

```
mysql> INSERT INTO account VALUES(3,'james',1000);
ERROR 1062 (23000): Duplicate entry '3' for key 'PRIMARY'
mysql> INSERT INTO account VALUES(3,'james',1000);
ERROR 1062 (23000): Duplicate entry '3' for key 'PRIMARY'
mysql>
```

总结: serializable 串行化可以彻底解决幻读,但是事务只能排队执行,严重影响效率,数据库不会使用这种隔离级别。

11- 多表操作

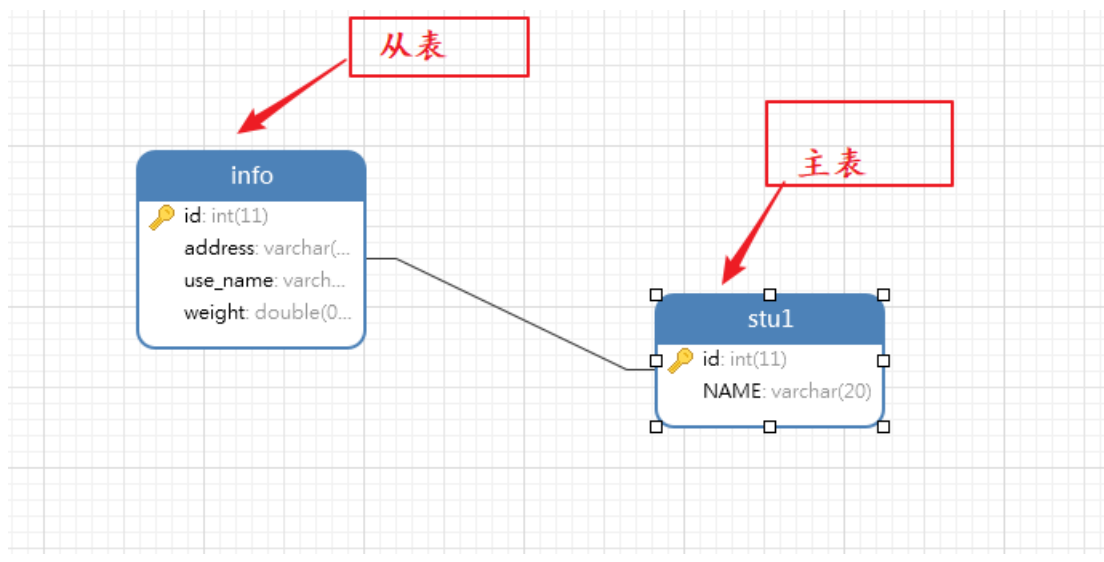
11.1 多表关系设计

11.1.1 一对一

A表和B表：A表中一条记录对应B表中一条记录。如：身份证 - 护照

一对一的建表原则	说明
外键唯一	将从表的外键添加唯一约束，变成了一对一的关系。其实是一个特殊的一对多的关系。
外键是主键	外键唯一 主表的主键和从表的外键（唯一），形成主外键关系，外键唯一 UNIQUE

架构图



案例语句

```

-- 创建数据库
CREATE DATABASE test2;

# 使用数据库
use db_mysql;

# 一对一的关系

-- 主表
CREATE TABLE stu1(
    id INT PRIMARY KEY, -- 主键
    NAME VARCHAR(20)
);

-- 从表
CREATE TABLE info(
    id INT PRIMARY key, -- 主键
    address VARCHAR(20),
    use_name VARCHAR(10),
    weight double,
    -- 创建外键,外键又是主键
    FOREIGN KEY(id) REFERENCES stu1(id)
);

```

11.1.2 一对多关系

表与表之间的关系	记录之间的对应关系
一对多	A表一方，B表多方： A表中一条记录对应了B表中多条记录。如：部门 - 员工 B表中一条记录对应了A表中一条记录
一对多建表原则	在从表(多方)创建一个字段,字段作为外键指向主表(一方)的主键。 在多方建立外键，指向一方的主键。

实现一个"线路分类"中有多个"旅游线路"的一对多的关系。(category 分类, route 线路)

跟团游

➡

线路分类

➡

旅游线路

[查看更多跟团游产品>](#)



桂林+阳朔+漓江3日2晚跟团游(4钻)·【国庆大促-100...

¥600起/份



惠州市+双月湾2日1晚跟团游(5钻)·【享海酒店】海景...

¥249起/份



桂林+阳朔+漓江3日2晚跟团游(4钻)·国庆大促-1200...

¥600起/份



广州长隆+珠海长隆5日4晚跟团游(5钻)·【官方自营 2...

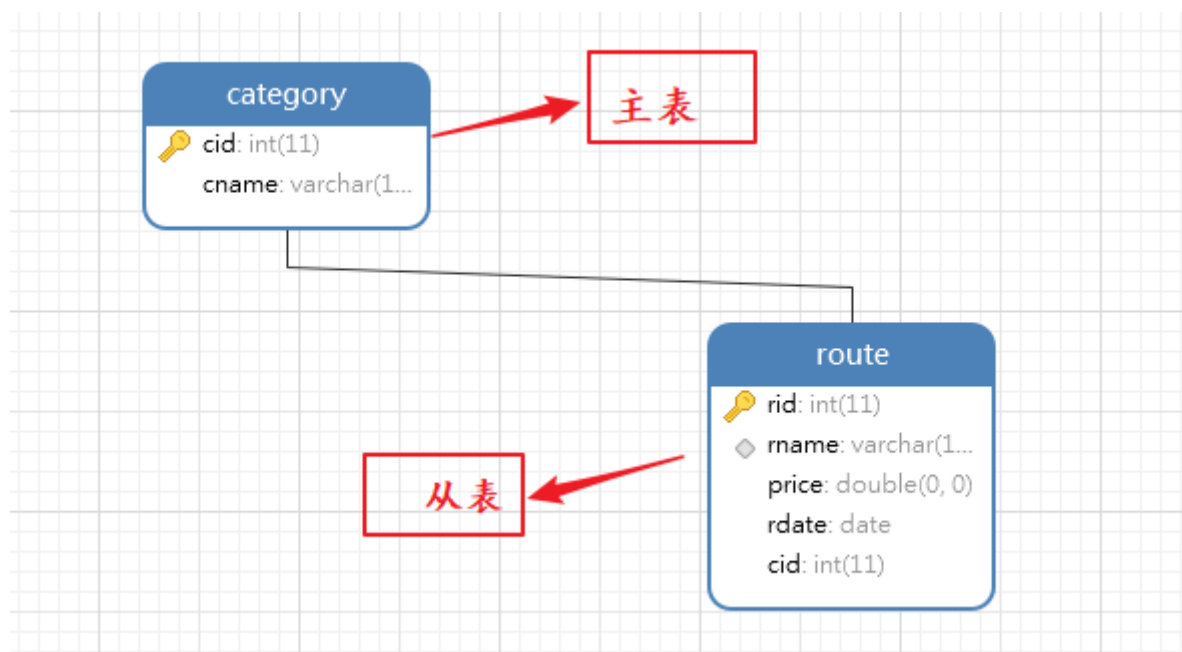
¥2149起/份



广东清远2日1晚跟团游·【国庆】住清新景泉温泉山庄...

¥328起/份

架构图



案例语句

```
# 一对多,一个分类对应多条线路

/*
  创建旅游线路分类表category
*/
create table category(
  cid int primary key auto_increment, -- cid旅游线路分类主键, 自动增长
  cname varchar(100) not null unique -- cname旅游线路分类名称非空, 唯一, 字符串100
);

drop table category;
drop table route;

/*
  创建旅游线路表route
*/
create table route(
  rid int primary key auto_increment, -- rid旅游线路主键, 自动增长
  rname varchar(100) not null unique, -- 旅游路线名称
  price double, -- 路线价格
  rdate date, -- 日期时间
  cid int, -- 添加外键约束
  constraint fk_cid foreign key(cid) references category(cid)
);

-- 删除外键约束
alter table route drop foreign key fk_cid;
```

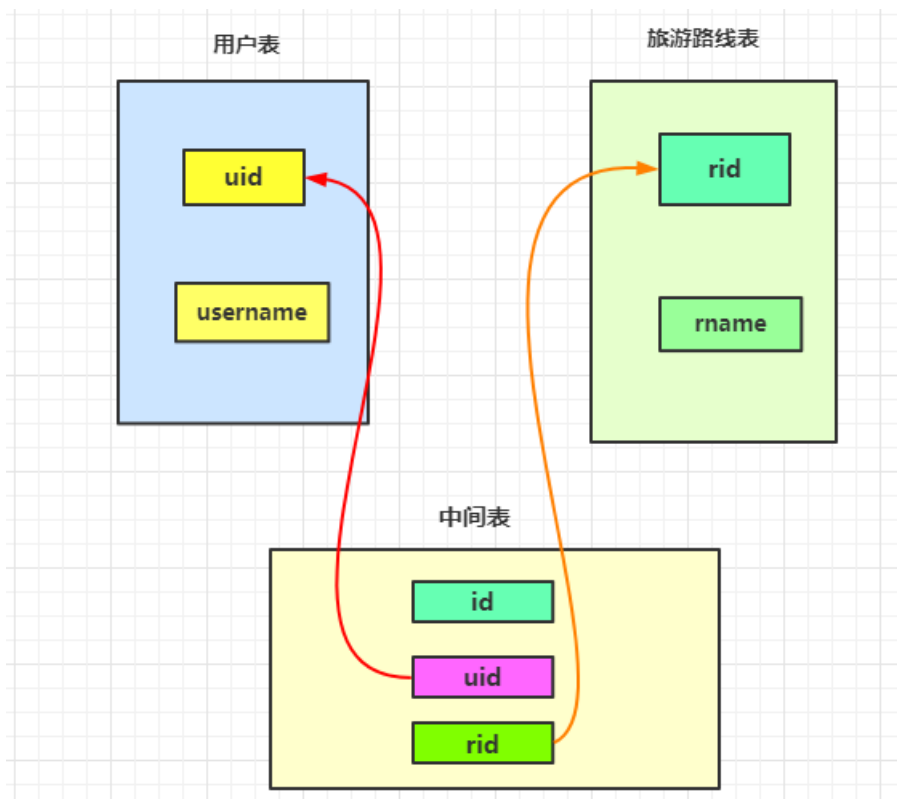
11.1.3 多对多关系

表与表之间的关系	记录之间的对应关系
多对多	A表中一条记录对应了B表中多条记录 B表中一条记录对应了A表中多条记录

多对多关系建表原则

- 需要创建第三张表，中间表中至少两个字段，这两个字段分别作为外键指向各自一方的主键。

图示



sql语句

一个用户收藏多个线路，一个线路被多个用户收藏，建表体现线路与用户之间的关系。用户与线路之间是多对多的关系。

```
/*
    创建旅游线路分类表category
*/
create table category(
    cid int primary key auto_increment, -- cid旅游线路分类主键，自动增长
    cname varchar(100) not null unique -- cname旅游线路分类名称非空，唯一，字符串100
);

/*
    创建旅游线路表route
*/
create table route(
    rid int primary key auto_increment, -- rid旅游线路主键，自动增长
    rname varchar(100) not null unique, -- 旅游线路名称
    price double, -- 路线价格
    rdate date, -- 日期时间
    cid int, -- 添加外键约束
    constraint fk_cid foreign key(cid) references category(cid)
);

/*
    用户表user
*/
create table USER(
    uid int primary key auto_increment, -- uid用户主键，自增长
```

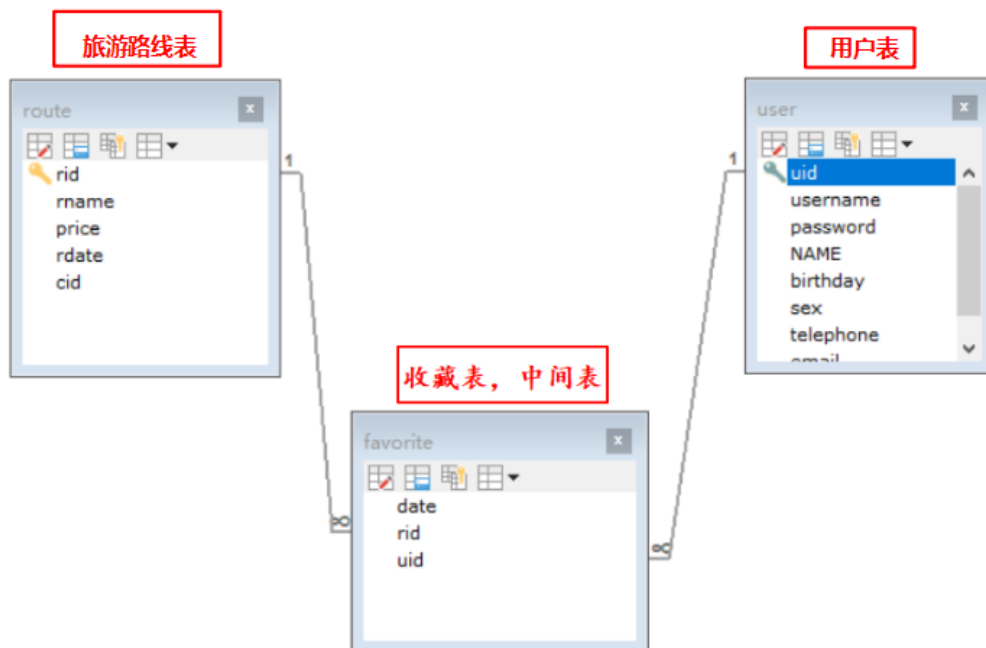
```

username varchar(100) unique not null, -- 用户名长度100, 唯一, 非空
password varchar(30) not null, -- 密码长度30, 为空....
name varchar(100), -- 姓名长度
birthday date, -- 生日
sex CHAR(1), -- sex性别, 定长字符串1
telephone varchar(11), -- 手机号, 字符串11
email varchar(100) -- email邮箱, 字符串长度100
);

/*
收藏表favorite
*/
create table favorite(
    date datetime, -- 收藏时间
    rid int, -- 路线id的外键
    constraint fk_rid foreign key(rid) references route(rid), -- 关联了线路的主键
    uid int, -- 用户id的外键
    constraint fk_uid foreign key(uid) references USER(uid) -- 关联了用户的主键
);

```

架构图



11.2 内连接

内连接的特点:

- 通过指定的条件去匹配两张表中的数据, 匹配上就显示, 匹配不上就不显示。
- 比如通过: 从表的外键 = 主表的主键 方式去匹配。

11.2.1 隐式内联接

- from子句 后面直接写 多个表名 使用where指定连接条件的 这种连接方式是 隐式内连接.
- 使用where条件过滤无用的数据

语法:

```
select 列名 from 左表,右表 where 从表.外键=主表.主键
```

11.2.2 显示内联接

使用 inner join ...on 这种方式, 就是显式内连接。

语法:

```
-- 显示内连接, on后面就是表连接的条件
select 列名 from 左表 inner join 右表 on 从表.外键 = 主表.主键
```

案例语句

```
-- 笛卡尔积和内连接

-- 创建部门表
create table dept(
    id int primary key auto_increment, -- 部门主键id,自增长
    name varchar(20) -- 部门name长度
);

-- 插入数据
insert into dept(name) values("开发部"),("市场部"),("财务部");

-- 创建员工表
create table emp(
    id int primary key auto_increment,
    name varchar(10), -- 员工姓名长度
    gender char(1), -- 性别
    salary double, -- 工资
    join_date date, -- 入职日期
    dept_id int, -- 外键
    constraint fk_dept_id foreign key(dept_id) references dept(id) -- 关联了部门的主键
);

-- 插入数据
insert into emp(name,gender,salary,join_date,dept_id) values('侯大利','男',7200,'2013-02-24',1);
insert into emp(name,gender,salary,join_date,dept_id) values('龚建明','男',3600,'2010-12-02',2);
insert into emp(name,gender,salary,join_date,dept_id) values('朱琳','男',9000,'2008-08-08',2);
insert into emp(name,gender,salary,join_date,dept_id) values('田甜','女',5000,'2015-10-07',3);
insert into emp(name,gender,salary,join_date,dept_id) values('杨红','女',4500,'2011-03-14',1);

### 需求: 查询所有的员工和所有的部门
```

```

select * from emp, dept; -- (产生笛卡尔积)
## 消除笛卡尔积

-- 隐式内连接(消除笛卡尔积) 从表.外键 = 主表.主键
select * from emp, dept where emp.dept_id = dept.id;
-- 给表起别名
select * from emp E, dept D where E.dept_id = D.id;

-- 显示内连接,on后面就是表连接的条件
select * from emp E inner join dept D on E.dept_id = D.id;

```

需求实现

```

## 1. 只显示两列
select E.name 员工名, D.name 部门名 from emp E, dept D where E.dept_id = D.id and E.id = 1;

## 2. 查询朱琳在哪个部门名字
select * from emp E, dept D where E.dept_id = D.id and E.id = 3;

## 3. 查询田甜的信息, 显示员工id, 姓名, 性别, 工资和所在的部门名称
-- 3.1. 确定查询哪些表
select * from emp E inner join dept D;
-- 3.2. 确定表连接的条件
select * from emp E inner join dept D on E.dept_id = D.id;
-- 3.3. 如果有其它的查询条件, 添加条件
select * from emp E inner join dept D on E.dept_id = D.id where E.`name` = "田甜";
-- 3.4. 确定查询列
select E.id 编号, E.`name` 姓名, E.gender 性别, E.salary 工资, D.`NAME` 部门名
from emp E inner join dept D on E.dept_id = D.id where E.`name` = "田甜";

```

11.3 外连接

11.3.1 左连接

左外连接 , 使用 `left outer join` , `outer` 可以省略

左外连接的特点

- 以左表为基准, 匹配右边表中的数据, 如果匹配的上, 就展示匹配到的数据。
- 如果匹配不到, 左表中的数据正常展示, 右边的展示为null。

语法

```

select 列名 from 左表 left join 右表 on 从表.外键=主表.主键

```

案例语句

需求：在部门表中增加一个销售部,将部门表设置成左表,员工表设置成右表。

```
select * from dept;
-- 插入数据
insert into dept VALUES(null, "销售部");
-- 使用显示内连接查询
select * from dept D inner join emp E on D.id = E.dept_id;

-- 使用左外连接查询
select * from dept D left join emp E on D.id = E.dept_id;
```

11.3.2 右连接

右外连接，使用 `right outer join`，`outer` 可以省略。

右外连接的特点

- 以右表为基准，匹配左边表中的数据，如果能匹配到，展示匹配到的数据。
- 如果匹配不到，右表中的数据正常展示，左边展示为null。

语法

```
select 列名 from 左表 right join 右表 on 从表.外键=主表.主键
```

案例语句

```
### 显示全部员工信息

select * from emp;
-- 添加元素
insert into emp values (null, "王永强", "男", 666, "2013-02-24", null);

-- 内连接
select * from dept D inner join emp E on D.id = E.dept_id;
-- 右外连接
select * from dept D right join emp E on D.id=E.dept_id;
```

11.4 子查询

子查询 subQuery：一条select 查询语句的结果, 作为另一条 select 语句的一部分。

子查询的特点

- 子查询必须放在小括号中。
- 子查询一般作为父查询的查询条件使用。

子查询常见分类

- where型 子查询: 将子查询的结果, 作为父查询的比较条件。
- from型 子查询: 将子查询的结果, 作为一张表,提供给父层查询使用。
- exists型 子查询: 子查询的结果是单列多行, 类似一个数组, 父层查询使用 IN 函数,包含子查询的结果。

11.4.1 单行单列

如果子查询是单行单列，父查询使用比较运算符：> < =

语法：

```
SELECT 查询字段 FROM 表 WHERE 字段=（子查询）；
```

案例语句

```
## 子查询:单行单列的情况
select id from dept where name = "开发部";

-- 案例：查询工资最高的员工

-- 1. 查询最高的工资
select max(salary) from emp;
-- 2. 根据最高工资到员工表查询到对应的员工信息
select * from emp where salary = (select max(salary) from emp);

-- 查询工资大于"田甜"的员工

-- 1. 查询田甜的工资
select salary from emp where name = "田甜";
-- 2. 查询大于这个工资的员工
select * from emp where salary > (select salary from emp where name = "田甜");
```

11.4.2 单列多行

子查询的结果类似一个数组，父层查询使用 IN /any /all函数，包含子查询的结果

语法

```
SELECT 查询字段 FROM 表 WHERE 字段 IN （子查询）；
```

案例查询

```
## 多行单列的情况

### 查询工资大于5000的员工，来自于哪些部门，得到部门的名字

-- 1. 先查询大于5000的员工所在的部门id
select dept_id from emp where salary > 5000;
-- 2. 再查询在这些部门id中部门的名字
select * from dept where id=(select dept_id from emp where salary > 5000);
select * from dept where id in (select dept_id from emp where salary > 5000);

### 列出工资高于在1号部门工作的所有员工，显示员工姓名和工资、部门名称。

-- 1. 查询1号部门所有员工的工资，得到多行单列
select salary from emp where dept_id = 1;
-- any关键字使用，表示任何一个
select * from emp where salary > any (select salary from emp where dept_id = 1);
-- all关键字使用，all表示所有
select * from emp where salary > all (select salary from emp where dept_id = 1);
```

11.4.3 多行多列

语法

认为它是一张虚拟表，可以使用表连接再次进行多表查询

案例语句

```
### 查询出2011年以后入职的员工信息,包括部门名称

-- 1. 在员工表中查询2011-1-1以后入职的员工
select * from emp where join_date > "2011-1-1";
-- 2. 查询所有的部门信息,与上面的虚拟表中的信息组合,找出所有部门id等于的dept_id
## -- 显示内连接
select * from dept D inner join (select * from emp where join_date > "2011-1-1")
E on D.id = E.dept_id;
## -- 右连接
select * from dept D right join (select * from emp where join_date > "2011-1-1")
E on D.id = E.dept_id;
```

11.5 多表查询

11.5.1 准备数据

```
## 创建数据库
create database test3;

-- 使用数据库
use db_mysql;

## 部门表
create table department(
    id INT PRIMARY KEY auto_increment, -- 部门表主键
    dep_name VARCHAR(10), -- 部门名
    dep_location VARCHAR(20) -- 部门所在城市
);

select * from department;
## 添加四个部门
insert into department(id, dep_name, dep_location) VALUES
(1, "市场部", "江州"),
(2, "销售部", "阳州"),
(3, "广告部", "秦阳"),
(4, "研发部", "广州");

## 职务表
create table job(
    id INT PRIMARY KEY auto_increment, -- 职务表主键
    jb_name VARCHAR(20), -- 职务名称
    jb_description VARCHAR(50) -- 职务描述
);

## 添加4个职务
insert into job(id, jb_name, jb_description) VALUES
(01, "董事长", "管理整个公司"),
(02, "经理", "管理部门员工"),
```

```
(03, "销售员工", "推销产品"),
(04, "文员", "使用办公软件");
```

员工表

```
create table emp(
    id INT PRIMARY KEY auto_increment, -- 员工表主键
    ep_name varchar(50), -- 员工姓名
    mgr INT, -- 上级领导
    joindate DATE, -- 入职日期
    salary DECIMAL(7,2), -- 工资
    bonus DECIMAL(7,2), -- 奖金
    job_id INT, -- 外键,引用职务表中的主键
    constraint fk_job_id foreign key(job_id) references job(id),
    dep_id INT, -- 外键,引用部门表中的主键
    constraint fk_dep_id foreign key(dep_id) references department(id)
);
```

向员工表中添加数据

```
INSERT INTO emp(id,ep_name,job_id,mgr,joindate,salary,bonus,dep_id) VALUES
(1001,'丁晨光',4,1004,'2000-12-17','8000.00',NULL,2),
(1002,'黄大磊',3,1006,'2001-02-20','16000.00','3000.00',3),
(1003,'侯国龙',3,1006,'2001-02-22','12500.00','5000.00',3),
(1004,'朱林',2,1009,'2001-04-02','29750.00',NULL,2),
(1005,'侯大利',4,1006,'2001-09-28','12500.00','14000.00',3),
(1006,'田甜',2,1009,'2001-05-01','28500.00',NULL,3),
(1007,'杨红',2,1009,'2001-09-01','24500.00',NULL,1),
(1008,'葛朗台',4,1004,'2007-04-19','30000.00',NULL,2),
(1009,'宫建明',1,NULL,'2001-11-17','50000.00',NULL,1),
(1010,'丁莉',3,1006,'2001-09-08','15000.00','0.00',3),
(1011,'王永强',4,1004,'2007-05-23','11000.00',NULL,2),
(1012,'金传统',4,1006,'2001-12-03','9500.00',NULL,3),
(1013,'李武林',4,1004,'2001-12-03','30000.00',NULL,2),
(1014,'樊勇',4,1007,'2002-01-23','13000.00',NULL,1);
```

```
select * from emp;
```

工资等级表

```
create table salarygrade(
    grade int primary key, -- 级别
    low_salary int, -- 最低工资
    high_salary int -- 最高工资
);
```

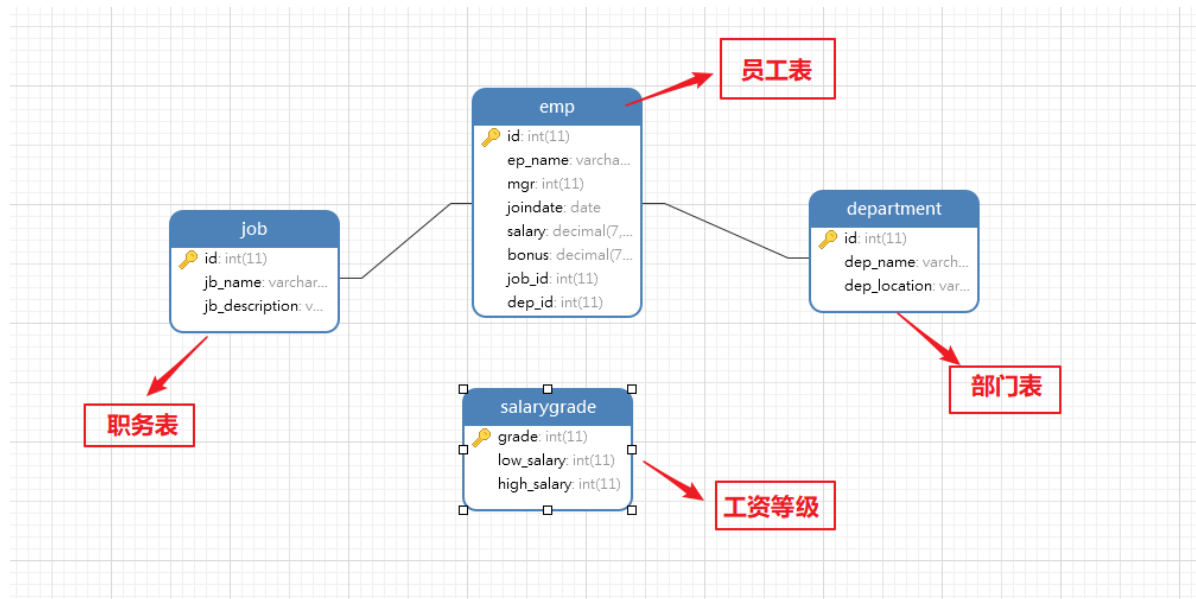
添加5个工资等级

```
insert into salarygrade(grade, low_salary, high_salary) VALUES
(1,7000,12000),
(2,12010,14000),
(3,14010,20000),
(4,20010,30000),
(5,30010,99990);
```

-- 查询操作

```
select * from department;
select * from emp;
select * from job;
select * from salarygrade;
```

架构图



11.5.2 实现需求1

```
### 查询所有员工姓名，工资，工资等级
-- 1. 确定查询哪些表
select
    *
from
    emp E
    inner join salarygrade S;

-- 2. 确定连接的条件
select
    *
from
    emp E
    inner join salarygrade S on E.salary BETWEEN S.low_salary
    and S.high_salary;

-- 3. 确定查询哪些列
select
    E.ep_name 姓名,
    E.salary 工资,
    S.grade 工资等级
from
    emp E
    inner join salarygrade S on E.salary BETWEEN S.low_salary
    and S.high_salary;
```

11.5.3 实现需求2

```
### 查询经理的信息。显示经理姓名，工资，职务名称，部门名称，工资等级
select
    E.ep_name 姓名,
    E.salary 工资,
    J.jb_name 职务,
    D.dep_name 部门名,
```

```

S.grade 工资等级
from
  emp E
  inner join department D on E.dep_id = D.id
  inner join job J on J.id = E.job_id
  inner join salarygrade S on E.salary BETWEEN S.low_salary
  and S.high_salary
where
  J.jb_name = "经理";

```

11.5.4 实现需求3

查询部门编号、部门名称、部门位置、部门人数.

```

### 查询部门编号、部门名称、部门位置、部门人数
select
  D.*,
  count( E.dep_id ) 部门人数
from
  department D
  left join emp E on D.id = E.dep_id
group by
  D.id;

```

11.5.5 实现需求4

```

### 列出所有员工的姓名及其直接上级的姓名。
select
  E.ep_name 员工名字,
  IFNULL( M.ep_name, 'BOSS' ) 上级名字
from
  emp E
  left join emp M on E.mgr = M.id;

```

11.5.6 实现需求5

查询工资高于公司平均工资的所有员工列:员工所有信息,部门名称,上级领导,工资等级。

```

-- 1. 查询公司的平均工资
select
  avg( salary )
from
  emp;

-- 2. 确定查询哪些表:员工表,员工表,部门表,工资等级
select
  *
from
  emp E
  left join emp M on E.mgr = M.id
  inner join department D on D.id = E.dep_id
  inner join salarygrade S on E.salary BETWEEN S.low_salary
  and S.high_salary;

-- 3. 确定查询哪些列

```

```

select
    E.*,
    D.dep_name 部门,
    ifnull( M.ep_name, "自己" ) 上级名字,
    S.grade 工资等级
from
    emp E
    left join emp M on E.mgr = M.id
    inner join department D on D.id = E.dep_id
    inner join salarygrade S on E.salary BETWEEN S.low_salary
    and S.high_salary
where
    E.salary > ( select avg( salary ) from emp );

```

12- 数据库设计

12.1 数据库三范式

基本概念

三范式就是设计数据库的规则。

- 为了建立冗余较小、结构合理的数据库，设计数据库时必须遵循一定的规则。在关系型数据库中这种规则就称为范式。

范式是符合某一种设计要求的总结。要想设计一个结构合理的关系型数据库，必须满足一定的范式。

- 满足最低要求的范式是第一范式（1NF）。在第一范式的基础上进一步满足更多规范要求的称为第二范式（2NF），其余范式以此类推。

一般说来，数据库只需满足第三范式(3NF)就行了。

12.1.1 第一范式 1NF

基本概念

- 原子性，做到列不可拆分第一范式是最基本的范式。
- 数据库表里面字段都是单一属性的，不可再分，如果数据表中每个字段都是不可再分的最小数据单元，则满足第一范式。

示例：

地址信息表中，contry这一列，还可以继续拆分，不符合第一范式。

id	country		id	country	city
1	中国广州		1	中国	广州
2	中国北京		2	中国	北京
3	中国南京		3	中国	南京

12.1.2 第二范式 2NF

基本概念

- 在第一范式的基础上更进一步，目标是确保表中的每列都和主键相关。
- 一张表只能描述一件事。

示例：

- 学员信息表中其实在描述两个事物，一个是学员的信息,一个是课程信息。
- 如果放在一张表中,会导致数据的冗余,如果删除学员信息,成绩的信息也被删除了。

id	name	sex	course	source
1	kobe	male	C++	100
2	lucy	female	Vue.js	88
3	james	male	Java	90

id	name	sex
1	kobe	male
2	lucy	female
3	james	male

id	course	source
1	C++	100
2	Vue.js	88
3	Java	90

12.1.3 第三范式 3NF

基本概念

- 消除传递依赖表的信息，如果能够被推导出来，就不应该单独的设计一个字段来存放。
- 示例通过number 与 price字段就可以计算出总金额,不要在表中再做记录。

total这个字段可以省略，节省数据空间

id	pname	price	number	total
1	华为	1000	10	10000
2	小米	2000	10	20000
3	苹果	3000	10	30000

12.2 数据库反三范式

12.2.1 基本概念

- 反范式化指的是通过增加冗余或重复的数据来提高数据库的读性能。
- 浪费存储空间,节省查询时间(以空间换时间)。

12.2.2 冗余字段

设计数据库时，某一个字段属于一张表，但它同时出现在另一个或多个表，且完全等同于它在其本来所属表的意义表示，那么这个字段就是一个冗余字段

12.2.4 反三范式示例

两张表，球员表、信息表，球员表中有字段name，而信息表中也存在字段name。

球员表			信息表			冗余字段
cid	name	age	pid	score	sex	name
1	kobe	21	1	33	男	kobe
2	james	19	2	27	男	james
3	curry	10	3	22	男	curry

使用操作

- 当需要查询“信息表”所有数据并且只需要“球员表”的name字段时, 没有冗余字段 就需要去join连接球员表。
- 假设表中数据量非常的大, 那么会这次连接查询就会非常大的消耗系统的性能。这时候冗余的字段就可以派上用场了, 有冗余字段我们查一张表就可以了。

12.2.5 基本总结

- 尽量遵循范式理论的规约, 尽可能少的冗余字段, 让数据库设计看起来舒适。
- 合理的加入冗余字段这个润滑剂, 减少join, 能让数据库执行性能更高更快。

13- MySQL索引和视图

13.1 MySQL索引

13.1.1 什么是索引

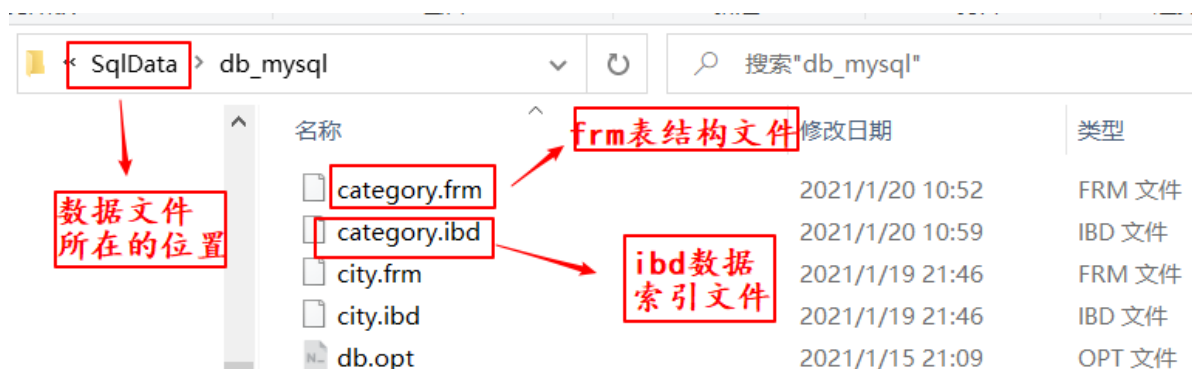
- 在数据库表中, 对字段建立索引可以大大提高查询速度。通过善用这些索引, 可以令MySQL的查询和运行更加高效。
- 如果合理的设计且使用索引的MySQL是一辆兰博基尼的话, 那么没有设计和使用索引的MySQL就是一个人力三轮车。拿汉语字典的目录页(索引) 打比方, 可以按拼音、笔画、偏旁部首等排序的目录(索引) 快速查找到需要的字。
- 索引分单列索引和组合索引。单列索引, 即一个索引只包含单个列, 一个表可以有多个单列索引, 但这不是组合索引。组合索引, 即一个索引包含多个列。

13.1.2 常见索引分类

索引名称	作用
主键索引 (primary key)	主键是一种唯一性索引,每个表只能有一个主键, 用于标识数据表中的每一条记录。
唯一索引 (unique)	唯一索引指的是 索引列的所有值都只能出现一次, 必须唯一。
普通索引 (index)	最常见的索引,作用就是 加快对数据的访问速度。

注意

MySQL将一个表的索引都保存在同一个索引文件中, 如果对中数据进行增删改操作,MySQL都会自动的更新索引。



11.1.3 主键索引 (PRIMARY KEY)

基本特点

- 主键是一种唯一性索引,每个表只能有一个主键,用于标识数据表中的某一条记录。
- 一个表可以没有主键,但最多只能有一个主键,并且主键值不能包含NULL。

1) 创建db_mysql2 数据库

```
create database db_mysql2 character set utf8;
```

2) 创建 user表

```
-- 创建数据表
create table USER(
  cid int,
  dname varchar(20),
  age int
);
```

3)基本语法格式

- 创建表的时候直接添加主键索引

```
CREATE TABLE 表名(
  -- 添加主键 (主键是唯一性索引,不能为null,不能重复,)
  字段名 类型 PRIMARY KEY,
);
```

- 修改表结构 添加主键索引

```
ALTER TABLE 表名 ADD PRIMARY KEY ( 列名 )
```

4) 为用户表添加主键索引

```
-- 主键索引
alter table user add primary key(cid);
```

结构信息

1 信息 2 表数据 3 信息

Format: ☒ HTML ☐ 文本/详细 刷新

表: user

Columns (3)

计算适合的数据类

通过读取现有数据查找该表的最佳数据类型。 [详细了解](#)

	Field	Type
	cid	int(11) NOT NULL
	dname	varchar(20) NULL
	age	int(11) NULL

Indexes (1)

找出多余索引

找到该表的冗余索引。 [了解详情](#)

	Indexes	Columns	Index Type
	PRIMARY	cid	Unique

主键索引

11.1.4 唯一索引(UNIQUE)

基本特点: 索引列的所有值都只能出现一次, 必须唯一。

注意事项

- 唯一索引可以保证数据记录的唯一性。
- 事实上, 在许多场合, 创建唯一索引的目的往往不是为了提高访问速度, 而只是为了避免数据出现重复。

1) 语法格式

- 创建表的时候直接添加主键索引

```
CREATE TABLE 表名(
    列名 类型(长度),
    -- 添加唯一索引
    UNIQUE [索引名称] (列名)
);
```

- 在已有的表上创建索引

```
create unique index 索引名 on 表名(列名(长度))
```

修改表结构添加索引

```
ALTER TABLE 表名 ADD UNIQUE ( 列名 )
```


2) 为 age 字段添加唯一索引

```
create unique index ind_age on USER(age);
```

Indexes (2)

找出多余索引

找到该表的冗余索引。 [了解详情](#)

	Indexes	Columns	Index Type
	PRIMARY	cid	Unique
	ind_age	age	Unique

3) 向表中插入数据

```
-- 插入数据
insert into user values(1, 'curry', 10);
```

11.1.5 普通索引 (INDEX)

普通索引（由关键字KEY或INDEX定义的索引）的唯一任务是加快对数据的访问速度。

因此，应该只为那些最经常出现在查询条件（WHERE column=）或排序条件（ORDERBY column）中的数据列创建索引。

1) 语法格式

- 使用create index 语句创建: 在已有的表上创建索引

```
create index 索引名 on 表名(列名[长度])
```

- 修改表结构添加索引


```
ALTER TABLE 表名 ADD INDEX 索引名 (列名)
```

2) 给 dname字段添加索引

```
# 给dname字段添加索引
alter table user add index dname_indx(dname);
```

找出多余索引

找到该表的冗余索引。 [了解详情](#)

	Indexes	Columns	Index Type
	PRIMARY	cid	Unique
	ind_age	age	Unique
	dname_indx	dname	

11.1.6 删除索引

由于索引会占用一定的磁盘空间，因此，为了避免影响数据库的性能，应该及时删除不再使用的索引。

1) 语法格式

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

2) 删除 user 表中名为 dname_indx 的普通索引。

```
alter table user drop index dname_indx;
```

11.1.7 索引的优缺点总结

- 添加索引首先应考虑在 where 及 order by 涉及的列上建立索引。
- 索引的优点
 1. 大大的提高查询速度
 2. 可以显著的减少查询中分组和排序的时间。
- 索引的缺点
 - 创建索引和维护索引需要时间，而且数据量越大时间越长。
 - 当对表中的数据进行增加，修改，删除的时候，索引也要同时进行维护，降低了数据的维护速度。

13.2 视图

13.2.1 什么是视图

1. 视图是一种虚拟表。
2. 视图建立在已有表的基础上，视图赖以建立的这些表称为基表。
3. 向视图提供数据内容的语句为 SELECT 语句，可以将视图理解为存储起来的 SELECT 语句。
4. 视图向用户提供基表数据的另一种表现形式

13.2.2 视图的作用

1) 权限控制时可以使用

- 某几个列可以运行用户查询,其他列不允许,可以开通视图 查询特定的列, 起到权限控制的作用

2) 简化复杂的多表查询

- 视图 本身就是一条查询SQL,我们可以将一次复杂的查询 构建成一张视图, 用户只要查询视图就可以获取想要得到的信息。
- 视图主要就是为了简化多表的查询

13.2.3 创建视图

1. 基本语法

```
create view 视图名 [column_list] as select语句;
view: 表示视图
column_list: 可选参数, 表示属性清单, 指定视图中各个属性的名称, 默认情况下, 与SELECT语句中查询的属性相同。
as : 表示视图要执行的操作
select语句: 向视图提供数据内容
```

2. 创建一张视图

```
-- 1. 分类表(主表)
create table category(
    #1.1 主键
    cid varchar(32) primary key,
    cname varchar(50)
);

-- 1.2 插入数据
INSERT INTO category(cid,cname) VALUES('c001','家电');
INSERT INTO category(cid,cname) VALUES('c002','鞋服');
INSERT INTO category(cid,cname) VALUES('c003','化妆品');
INSERT INTO category(cid,cname) VALUES('c004','汽车');

-- 2. 商品表(多表, 从表)
create table products(
    pid varchar(32) primary key,
    pname varchar(50),
    -- 价格
    price int,
    -- 2.1 是否标记为: 1表示上架、0表示下架
    flag varchar(2),
    -- 2.2 添加外键约束
    category_id varchar(32),
    -- 2.3 添加外键约束
    foreign key(category_id) references category(cid)
);

-- 插入数据
INSERT INTO products(pid, pname,price,flag,category_id) VALUES('p001','小米电视机',5000,'1','c001');
```

```

INSERT INTO products(pid, pname,price,flag,category_id) VALUES('p002','格力空调',3000,'1','c001');
INSERT INTO products(pid, pname,price,flag,category_id) VALUES('p003','美的冰箱',4500,'1','c001');

INSERT INTO products (pid, pname,price,flag,category_id) VALUES('p004','篮球鞋',800,'1','c002');
INSERT INTO products (pid, pname,price,flag,category_id) VALUES('p005','运动裤',200,'1','c002');
INSERT INTO products (pid, pname,price,flag,category_id) VALUES('p006','T恤',300,'1','c002');
INSERT INTO products (pid, pname,price,flag,category_id) VALUES('p007','冲锋衣',2000,'1','c002');

INSERT INTO products (pid, pname,price,flag,category_id) VALUES('p008','神仙水',800,'1','c003');
INSERT INTO products (pid, pname,price,flag,category_id) VALUES('p009','大宝',200,'1','c003');

-- 3.先查询所有商品和商品的对应分类信息
select * from products p left join category c on p.category_id = c.cid;

-- 4.基于查询语句,创建一张视图
create view products_category_view as select * from products p left join category c on p.category_id = c.cid;

-- 5.查询视图 ,当做一张只读的表操作就可以
select * from products_category_view;

```

信息	结果 1	剖析	状态			
pid	pname	price	flag	category_id	cid	cname
p001	小米电视机	5000	1	c001	c001	家电
p002	格力空调	3000	1	c001	c001	家电
p003	美的冰箱	4500	1	c001	c001	家电
p004	篮球鞋	800	1	c002	c002	鞋服
p005	运动裤	200	1	c002	c002	鞋服
p006	T恤	300	1	c002	c002	鞋服
▶ p007	冲锋衣	2000	1	c002	c002	鞋服
p008	神仙水	800	1	c003	c003	化妆品
p009	大宝	200	1	c003	c003	化妆品

13.2.4 通过视图进行查询

1. 查询各个分类下的商品的平均价格

```
-- 多表查询
select
    cname AS '分类名称',
    AVG(p.`price`) AS '平均价格'
from products p left join category c on p.category_id = c.cid
group by c.`cname`;

-- 视图查询
select
    cname as '分类名称',
    avg(price) as '平均价格'
from products_category_view group by cname;
```

2. 查询鞋服分类下最贵的商品的全部信息

```
## 1.1 求出鞋服分类下的最高商品价格
select
    max(price) as maxprice
from
products p left join category c on p.category_id = c.cid where c.cname = '鞋服';

## 1.2 将最高价格作为条件使用
select * from products p left join category c on p.category_id = c.cid where
c.cname = '鞋服' and p.price =
(select
    max(price) as maxprice
from
products p left join category c on p.category_id = c.cid where c.cname = '鞋服');
```

```
## 2. 通过视图查询
select * from products_category_view pv where pv.cname = '鞋服' and pv.price =
(
select
    max(price) as maxprice
from
products p left join category c on p.category_id = c.cid where c.cname = '鞋服'
)
```

13.2.5 视图与表的区别

- 视图是建立在表的基础上，表存储数据库中的数据，而视图只是做一个数据的展示。
- 通过视图不能改变表中数据（一般情况下视图中的数据都是表中的列 经过计算得到的结果,不允许更新）。
- 删除视图，表不受影响，而删除表，视图不再起作用。