

1、初识 MySQL

1.1、数据库分类

1.3、数据库分类

关系型数据库: (SQL)

- MySQL, Oracle, Sql Server, DB2, SQLlite
- 通过表和表之间，行和列之间的关系进行数据的存储，学员信息表，考勤表，.....

非关系型数据库: (NoSQL) Not Only

- Redis, MongoDB
- 非关系型数据库，对象存储，通过对对象的自身的属性来决定。

DBMS(数据库管理系统)

- 数据库的管理软件，科学有效的管理我们的数据。维护和获取数据；
- MySQL，数据库管理系统！

1.2、MySQL简介

1.4、MySQL简介

MySQL是一个**关系型数据库管理系统**

前世：瑞典MySQL AB 公司

今生：属于[Oracle](#)旗下产品

MySQL是最好的[RDBMS](#) (Relational Database Management System, 关系数据库管理系统) 应用软件之一。

开软的数据库软件~

体积小、速度快、总体拥有成本低，招人成本比较低，所有人必须会~

中小型网站、或者大型网站，集群！

官网：<https://www.mysql.com>

1.3、基本的命令操作

```
1 mysql -uroot -p123456 --连接数据库
2
3 update mysql.user set authentication_string=password('123456') where user='root' and Host =
'localhost'; -- 修改用户密码
4 flush privileges; -- 刷新权限
5
6 -----
7 -- 所有的语句都使用; 结尾
8 show databases; -- 查看所有的数据库
9
10 mysql> use school -- 切换数据库 use 数据库名
11 Database changed
12
13 show tables; -- 查看数据库中所有的表
14 describe student; -- 显示数据库中所有的表的信息
15
16 create database westos; -- 创建一个数据库
17
18 exit; --退出连接
19
20 -- 单行注释 (SQL 的本来的注释)
21 /* (sql的多行注释)
22 helloi
23 asdas
24 dasdas
25 */
```

994.j

数据库 xxx 语言 CRUD 增删改查！ CV 程序猿 API程序猿 CRUD 程序猿！ (业务！)

DDL 定义

DML 操作

DQL 查询

DCL 控制

2、操作数据库

2.1、操作数据库

2.1、操作数据库（了解）

1、创建数据库

```
1 CREATE DATABASE [IF NOT EXISTS] westos;
```

2、删除数据库

```
1 DROP DATABASE [IF EXISTS] westos
```

3、使用数据库

```
1 -- tab 键的上面,如果你的表名或者字段名是一个特殊字符,就需要带 ``  
2 USE `school`
```

4、查看数据库

```
1 SHOW DATABASES -- 查看所有的数据库
```

2.2、数据库的列类型

2.2、数据库的列类型

数值

- tinyint 十分小的数据 1个字节
- smallint 较小的数据 2个字节
- mediumint 中等大小的数据 3个字节
- int 标准的整数 4个字节 常用的 int
- bigint 较大的数据 8个字节
- float 浮点数 4个字节
- double 浮点数 8个字节 (精度问题!)
- decimal I 字符串形式的浮点数 金融计算的时候, 一般是使用decimal

字符串

- char 字符串固定大小的 0~255
- varchar 可变字符串 0~65535 常用的变量 String
- tinytext 微型文本 $2^{18} - 1$
- text 文本串 $2^{16} - 1$ 保存大文本

varchar(20), 是20字符, 无论是数字、字母还是UTF8汉字 (每个汉字3字节) , 都可存放20个。

时间日期

java.util.Date

- date YYYY-MM-DD , 日期格式
- time HH: mm: ss 时间格式
- **datetime YYYY-MM-DD HH: mm: ss 最常用的时间格式**
- **timestamp 时间戳， 1970.1.1 到现在的毫秒数！ 也较为常用！**
- year 年份表示

null

- 没有值, 未知
- 注意, 不要使用NULL进行运算, 结果为NULL

2.3、数据库的字段属性

2.3、数据库的字段属性（重点）

Unsigned :

- 无符号的整数
- 声明了该列不能声明为负数

zerofill:

- 0填充的
- 不足的位数，使用0来填充， int (3) , 5 --- 005

自增:

- 通常理解为自增，自动在上一条记录的基础上 + 1 (默认)
- 通常用来设计唯一的主键~ index，必须是整数类型
- 可以自定义设计主键自增的起始值和步长

非空 NULL not null

- 假设设置为 not null ，如果不给它赋值，就会报错！
- NULL ，如果不填写值，默认就是null！

默认:

- 设置默认的值！
- sex，默认值为男，如果不指定该列的值，则会有默认的值！

项目开发规范

```
/* 每一个表，都必须存在以下五个字段！未来做项目用的，表示一个记录存在意义！
 *
 id 主键
 `version` |乐观锁
 is_delete 伪删除
 gmt_create 创建时间
 gmt_update 修改时间
 */
```

2.4、创建数据库表

创建表

2.4、创建数据库表（重点）

```
1 -- 目标：创建一个school数据库
2 -- 创建学生表(列,字段) 使用SQL 创建
3 -- 学号int 登录密码varchar(20) 姓名,性别varchar(2),出生日期(datetime),家庭住址,email
4
5 -- 注意点，使用英文 ` `，表的名称 和 字段 尽量使用 `` 括起来
6 -- AUTO_INCREMENT 自增
7 -- 字符串使用 单引号括起来！
8 -- 所有的语句后面加 , （英文的），最后一个不用加
9 -- PRIMARY KEY 主键，一般一个表只有一个唯一的主键！
10 CREATE TABLE IF NOT EXISTS `student` (
11     `id` INT(4) NOT NULL AUTO_INCREMENT COMMENT '学号',
12     `name` VARCHAR(30) NOT NULL DEFAULT '匿名' COMMENT '姓名',
13     `pwd` VARCHAR(20) NOT NULL DEFAULT '123456' COMMENT '密码',
14     `sex` VARCHAR(2) NOT NULL DEFAULT '女' COMMENT '性别',
15     `birthday` DATETIME DEFAULT NULL COMMENT '出生日期',
16     `address` VARCHAR(100) DEFAULT NULL COMMENT '家庭住址',
17     `email` VARCHAR(50) DEFAULT NULL COMMENT '邮箱',
18     PRIMARY KEY(`id`)
19 )ENGINE=INNODB DEFAULT CHARSET=utf8
```

格式

```
1 CREATE TABLE [IF NOT EXISTS] `表名` (
2     '字段名' 列类型 [属性] [索引] [注释],
3     '字段名' 列类型 [属性] [索引] [注释],
4     .....
5     '字段名' 列类型 [属性] [索引] [注释]
6 ) [表类型] [字符集设置] [注释]
```

```
CREATE TABLE IF NOT EXISTS `student` (
    `id` INT NOT NULL AUTO_INCREMENT COMMENT '学号',
    `pwd` VARCHAR(10) NOT NULL DEFAULT '123456' COMMENT '密码',
    `name` VARCHAR(10) NOT NULL DEFAULT '匿名' COMMENT '名字',
    `sex` CHAR(2) NOT NULL DEFAULT '女' COMMENT '性别',
    `birthday` DATETIME DEFAULT NULL COMMENT '出生日期',
    `address` VARCHAR(100) NOT NULL COMMENT '地址',
    `email` VARCHAR(20) DEFAULT NULL COMMENT '邮箱',
    PRIMARY KEY (`id`)
) ENGINE=INNODB DEFAULT CHARSET=utf8;

DROP TABLE `student`
```

常用命令

常用命令

- 1 SHOW CREATE DATABASE school -- 查看创建数据库的语句
- 2 SHOW CREATE TABLE student -- 查看student数据表的定义语句
- 3 DESC student -- 显示表的结构

```
SHOW CREATE DATABASE school;
SHOW CREATE TABLE student;
DESC student;
```

2.5、数据库表的类型

MYISAM 与 INNODB区别

| | MYISAM (早些年使用的) | INNODB (默认使用) |
|--------|-----------------|-----------------|
| 事务支持 | 不支持 | 支持 |
| 数据行锁定 | 不支持 (表锁) | 支持 (行锁) |
| 外键约束 | 不支持 | 支持 |
| 全文索引 | 支持 | 支持 (mysql5.6后) |
| 表空间的大小 | 较小 | 较大 (约为2倍) |

- MYISAM 节约空间，速度较快
- INNODB 安全性高，支持事务，可以多表多用户操作

在物理空间存在的位置

所有的数据库文件都存在 data 目录下，一个文件夹就对应一个数据库
本质还是文件的存储！

MySQL 引擎在物理文件上的区别

- InnoDB 在数据库表中只有一个 *.frm 文件，以及上级目录下的 ibdata1 文件
- MYISAM 对应文件
 - *.frm 表结构的定义文件
 - *.MYD 数据文件 (data)
 - *.MYI 索引文件 (index)

设置数据库表的字符集编码

1、创建数据库表时设置

```
CHARSET=utf8
```

现在较新的版本默认值就是utf8

2、在my.ini 文件中配置默认的编码

```
character-set-server=utf8
```

2.6、修改和删除数据库表字段

所有的创建和删除操作尽量加上判断，以免报错~

注意点：

- `` 字段名，使用这个包裹！
- 注释 -- /**/
- sql 关键字大小写不敏感，建议大家写小写
- 所有的符号全部用英文！

```
-- 修改表名: alter table 旧表名 rename as 新表名
alter table student1 rename as student
-- 增加表的字段: alter table 表名 add 字段名 列属性
alter table student add age1 int(2)

-- 修改表的字段(重命名、修改约束)
-- modify 只能用来修改约束
-- change 可以用来重命名、修改约束
-- (change不能单独修改约束，但是在修改重命名字段的时候可以修改约束)
alter table student modify age1 varchar(2)
alter table student change age1 age int(2)

-- 删除表的字段:alter table 表名 drop 字段名
alter table student drop age

-- 删除表
drop table if exists student1
```

3、MySQL 数据管理

3.1、数据库级别的外键

在实际开发中一般不使用外键

其实这个话题是老生常谈，很多人在工作中确实也不会使用外键。包括在阿里的JAVA规范中也有下面这一条

【强制】不得使用外键与级联，一切外键概念必须在应用层解决。

但是呢，询问他们原因，大多是这么回答的

每次做DELETE 或者UPDATE都必须考虑外键约束，会导致开发的时候很痛苦，测试数据极为不方便。

3.2、DML语言

DML语言：数据操作语言

- insert 插入
- update 修改
- delete 删除

3.3、插入 insert

```
-- 插入(添加)
-- insert into 表名 ([字段名1,字段名2,字段名3,...]) values (值1,值2,值3),(值1,值2,值3)...

-- 插入多行
insert into `student` (`id`, `pwd`, `name`, `sex`, `birthday`, `address`, `email`)
values
(1, '666', 'name', '男', '1999-09-01', '空', '123123@123.com'),
(2, '666', 'name', '男', '1999-09-01', '空', '123123@123.com'),
(3, '666', 'name', '男', '1999-09-01', '空', '123123@123.com')
```

语法：

```
insert into 表名 ([字段名1,字段名2,字段名3,...]) values (值1,值2,值3),(值1,值2,值3)...
```

注意：

- 字段和字段之间使用逗号隔开
- 如果省略字段，后面的值必须要一一对应
- 可以同时插入多行数据

3.4、修改 update

```
-- update 表名 set 字段名=新的值,[字段名=新的值]... where [条件]
update `student` set `name`='哈哈', `age`=19 where `id`=1;

-- 不指定条件的话会修改所有行
update `student` set `name`='哈哈', `age`=19

-- 修改多个属性, 用逗号隔开

-- 语法:
-- update 表名 set 字段名=新的值,[字段名=新的值]... where [条件]
```

语法:

```
update 表名 set column_name=value,[column_name=value]... where [条件]
```

注意:

- value 可以是一个具体的值, 也可以是一个变量

```
update `student` set `name`='哈哈', `age`=19, `birthday`=CURRENT_TIME
```

| 操作符 | 含义 | 范围 | 结果 |
|---------------------|--------|-------------|-------|
| = | 等于 | 5=6 | false |
| <> 或 != | 不等于 | 5<>6 | true |
| > | | | |
| < | | | |
| <= | | | |
| >= | | | |
| BETWEEN ... and ... | 在某个范围内 | [2,5] | |
| AND | 我和你 && | 5>1 and 1>2 | false |
| OR | 我或你 | 5>1 or 1>2 | true |

```
1 -- 通过多个条件定位数据
2 UPDATE `student` SET `name`='长江7号' WHERE `name`='狂神44' AND sex='女'
```

语法: `UPDATE 表名 set column_name = value,[column_name = value,...] where [条件]`

注意:

- column_name 是数据库的列, 尽量带上``
- 条件, 筛选的条件, 如果没有指定, 则会修改所有的列
- value, 是一个具体的值, 也可以是一个变量
- 多个设置的属性之间, 使用英文逗号隔开

```
1 | UPDATE `student` SET `birthday` = CURRENT_TIME WHERE `name`='长江7号' AND sex='女'
```

3.5、删除 delete和 truncate

```
-- 删除数据（指定条件）
delete from `student` where `id` = 1;

delete from `student` -- 不会影响自增
truncate `student` -- 自增会归零
```

语法：

```
delete from 表名 where [条件]
```

delete和 truncate 区别：

- **相同点：**都能删除数据，都不会删除表结构
- **不同点：**
 - TRUNCATE 会重置自增列（计数器归零）
 - TRUNCATE 不会影响事务

delete删除的问题，重启数据库，现象：

- InnoDB 自增列会从 1开始（存在内存中的，断电即失，**已经于 8.0修复了此问题**）
- MYISAM 继续从上一个自增量开始（存在文件中的，不会丢失）

4、DQL 查询数据（最重点）

SELECT语法

```
1 SELECT [ALL | DISTINCT]
2 {* | table.* | [table.field1[as alias1][,table.field2[as alias2]][,...]]}
3 FROM table_name [as table_alias]
4     [left | right | inner join table_name2] -- 联合查询
5     [WHERE ...] -- 指定结果需满足的条件
6     ↪ [GROUP BY ...] -- 指定结果按照哪几个字段来分组
7     [HAVING] -- 过滤分组的记录必须满足的次要条件
8     [ORDER BY ...] -- 指定查询记录按一个或多个条件排序
9     [LIMIT {[offset,]row_count | row_countOFFSET offset}];
10    -- 指定查询的记录从哪条至哪条
```

注意：[]括号代表可选的，{}括号代表必选得

sql

4.1、DQL

(Data Query Language：数据查询语言)

- 所有的查询操作都用它：select
- 简单、复杂的查询它都能做
- 数据库中最核心最重要的语句
- 使用频率最高

```

create database if not exists `school`;
-- 创建一个school数据库
use `school`;-- 创建学生表
drop table if exists `student`;
create table `student`(
    `studentno` int(4) not null comment '学号',
    `loginpwd` varchar(20) default null,
    `studentname` varchar(20) default null comment '学生姓名',
    `sex` tinyint(1) default null comment '性别, 0或1',
    `gradeid` int(11) default null comment '年级编号',
    `phone` varchar(50) not null comment '联系电话, 允许为空',
    `address` varchar(255) not null comment '地址, 允许为空',
    `borndate` datetime default null comment '出生时间',
    `email` varchar(50) not null comment '邮箱账号允许为空',
    `identitycard` varchar(18) default null comment '身份证号',
    primary key (`studentno`),
    unique key `identitycard`(`identitycard`),
    key `email` (`email`)
)engine=myisam default charset=utf8;
-- 创建年级表
drop table if exists `grade`;
create table `grade`(
    `gradeid` int(11) not null auto_increment comment '年级编号',
    `gradename` varchar(50) not null comment '年级名称',
    primary key (`gradeid`)
) engine=innodb auto_increment = 6 default charset = utf8;

-- 创建科目表
drop table if exists `subject`;
create table `subject`(
    `subjectno` int(11) not null auto_increment comment '课程编号',
    `subjectname` varchar(50) default null comment '课程名称',
    `classhour` int(4) default null comment '学时',
    `gradeid` int(4) default null comment '年级编号',
    primary key (`subjectno`)
)engine = innodb auto_increment = 19 default charset = utf8;

-- 创建成绩表
drop table if exists `result`;
create table `result`(
    `studentno` int(4) not null comment '学号',
    `subjectno` int(4) not null comment '课程编号',
    `examdate` datetime not null comment '考试日期',
    `studentresult` int (4) not null comment '考试成绩',
    key `subjectno`(`subjectno`)
)engine = innodb default charset = utf8;

```

```

-- 插入学生数据 其余自行添加 这里只添加了2行
insert into `student`
(`studentno`, `loginpwd`, `studentname`, `sex`, `gradeid`, `phone`, `address`, `borndate`, `email`, `identitycard`)
values
(1000, '123456', '张伟', 0, 2, '13800001234', '北京朝阳', '1980-1-1', 'text123@qq.com', '123456198001011234'),

```

```

(1001,'123456','赵强',1,3,'13800002222','广东深圳','1990-1-
1','text111@qq.com','123456199001011233');

-- 插入成绩数据 这里仅插入了一组，其余自行添加
insert into `result`(`studentno`, `subjectno`, `examdate`, `studentresult`)
values
(1000,1,'2013-11-11 16:00:00',85),
(1000,2,'2013-11-12 16:00:00',70),
(1000,3,'2013-11-11 09:00:00',68),
(1000,4,'2013-11-13 16:00:00',98),
(1000,5,'2013-11-14 16:00:00',58);

-- 插入年级数据
insert into `grade`(`gradeid`, `gradename`) values(1,'大一'),(2,'大二'),(3,'大
三'),(4,'大四'),(5,'预科班');

-- 插入科目数据
insert into `subject`(`subjectno`, `subjectname`, `classhour`, `gradeid`)values
(1,'高等数学-1',110,1),
(2,'高等数学-2',110,2),
(3,'高等数学-3',100,3),
(4,'高等数学-4',130,4),
(5,'C语言-1',110,1),
(6,'C语言-2',110,2),
(7,'C语言-3',100,3),
(8,'C语言-4',130,4),
(9,'Java程序设计-1',110,1),
(10,'Java程序设计-2',110,2),
(11,'Java程序设计-3',100,3),
(12,'Java程序设计-4',130,4),
(13,'数据库结构-1',110,1),
(14,'数据库结构-2',110,2),
(15,'数据库结构-3',100,3),
(16,'数据库结构-4',130,4),
(17,'C#基础',130,1);

```

4.2、查询指定字段

简单查询

```

-- 查询全部字段
SELECT * FROM student;

-- 查询指定字段
SELECT `studentno`, `studentname` FROM `student`;

-- 给查询结果起别名 AS
SELECT `studentno` AS `学号`, `studentname` AS `学生名字` FROM `student` AS `s` 

-- 函数 Concat(a,b)
SELECT CONCAT('姓名: ', `studentname`) AS `新名字` FROM `student` 

```

语法:

```
SELECT 字段... FROM 表
```

有需要的情况下可以取别名：

字段名 AS 别名

去重 DISTINCT

```
-- 查询哪些同学参加了考试
SELECT * FROM result -- 查询全部考试成绩

SELECT `studentno` FROM `result`
-- 去重
SELECT DISTINCT `studentno` FROM `result`
```

数据库的列 (表达式)

```
SELECT VERSION() -- 查询系统版本
SELECT 100*3-1 AS '计算结果' -- 计算
SELECT @@auto_increment_increment -- 查询自增的步长
SELECT `studentno`, `studentresult`, `studentresult`+1 AS '提分后' FROM result -- 计算
```

数据库中的表达式：文本值、列、null、函数，计算表达式、系统变量

SELECT `表达式` FROM 表

4.3、条件查询 where

- 作用：检索数据中符合条件的值
- 搜索的条件由一个或多个表达式组成，结果为布尔值

逻辑运算符

(尽量使用英文字母)

| 运算符 | 语法 | 描述 |
|--------|--------------|-----|
| and && | a and b a&&b | 逻辑与 |
| or | a or b a b | 逻辑或 |
| not ! | not a !a | 逻辑非 |

-- 查询考试成绩在 95~100分之间

```
-- and
SELECT studentno,studentresult FROM result
WHERE studentresult >= 95 AND studentresult <= 100;

-- &&
SELECT studentno,studentresult FROM result
```

```

WHERE studentresult >= 95 && studentresult <= 100;

-- 区间查询, 模糊查询
SELECT studentno,studentresult FROM result
WHERE studentresult BETWEEN 95 AND 100;

-- 查询除了学号1000外的同学的成绩
-- !=
SELECT studentno,studentresult FROM result
WHERE studentno != 1000;

-- NOT
SELECT studentno,studentresult FROM result
WHERE NOT studentno = 1000;

```

模糊查询：比较运算符

| 运算符 | 语法 | 描述 |
|-------------|---------------------|---------------------------------|
| IS NULL | a is null | 如果操作符为 NULL, 结果为真 |
| IS NOT NULL | a is not null | 如果操作符不为 null, 结果为真 |
| BETWEEN | a between b and c | 若a 在 b 和c 之间, 则结果为真 |
| Like | a like b | SQL 匹配, 如果a匹配b, 则结果为真 |
| In | a in (a1,a2,a3....) | 假设a在a1, 或者a2.... 其中的某一个值中, 结果为真 |

```

-- 模糊查询

-- Like(代表 0到任意个字符)
-- _ (匹配任意一个字符)

-- 查询姓刘的同学
SELECT studentno,studentname FROM student
WHERE studentname LIKE '刘%'

-- 查询名字中有 嘉 字的同学
-- 数据库字符集为 ASCII时候一个汉字需要两个__ 当字符集为GBK时, 一个汉字需要一个_
-- 转义字符 ESCAPE
SELECT studentno,studentname FROM student
WHERE studentname LIKE '%嘉%'

-- IN (具体的一个或多个)
-- 查询指定学号的学员
SELECT studentno,studentname FROM student
WHERE studentno IN(1001,1002)

-- 查询在北京的学员
SELECT studentno,studentname,address FROM student
WHERE address LIKE ('北京%')

SELECT studentno,studentname,address FROM student
WHERE address IN ('北京朝阳')

-- NULL      NOT NULL

```

```
-- 查询地址为空的学生
```

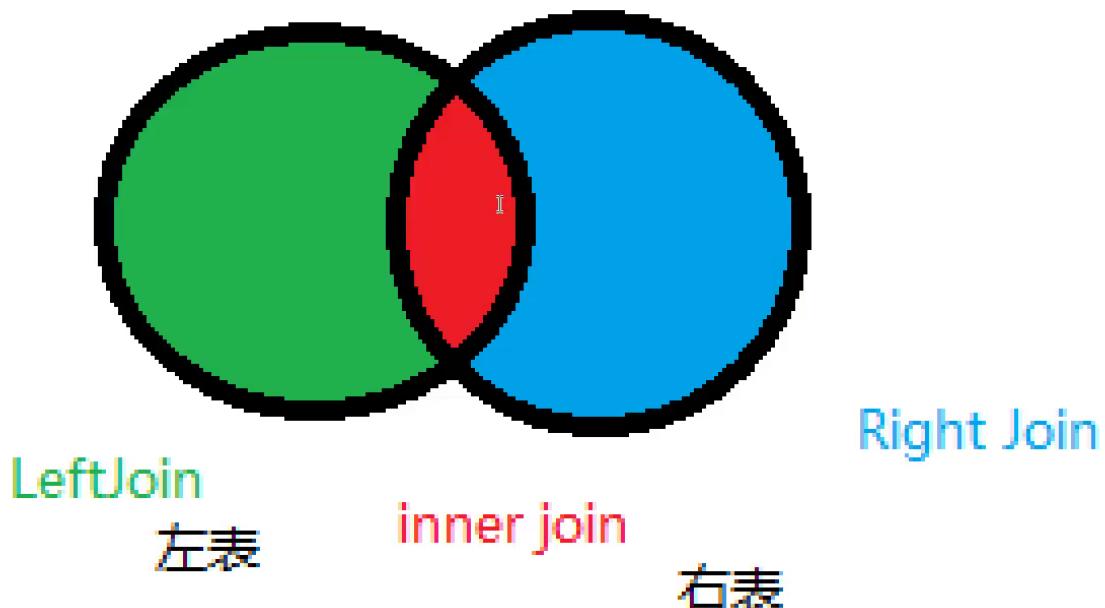
```
SELECT studentno,studentname,address FROM student  
WHERE address = '' OR address IS NULL;
```

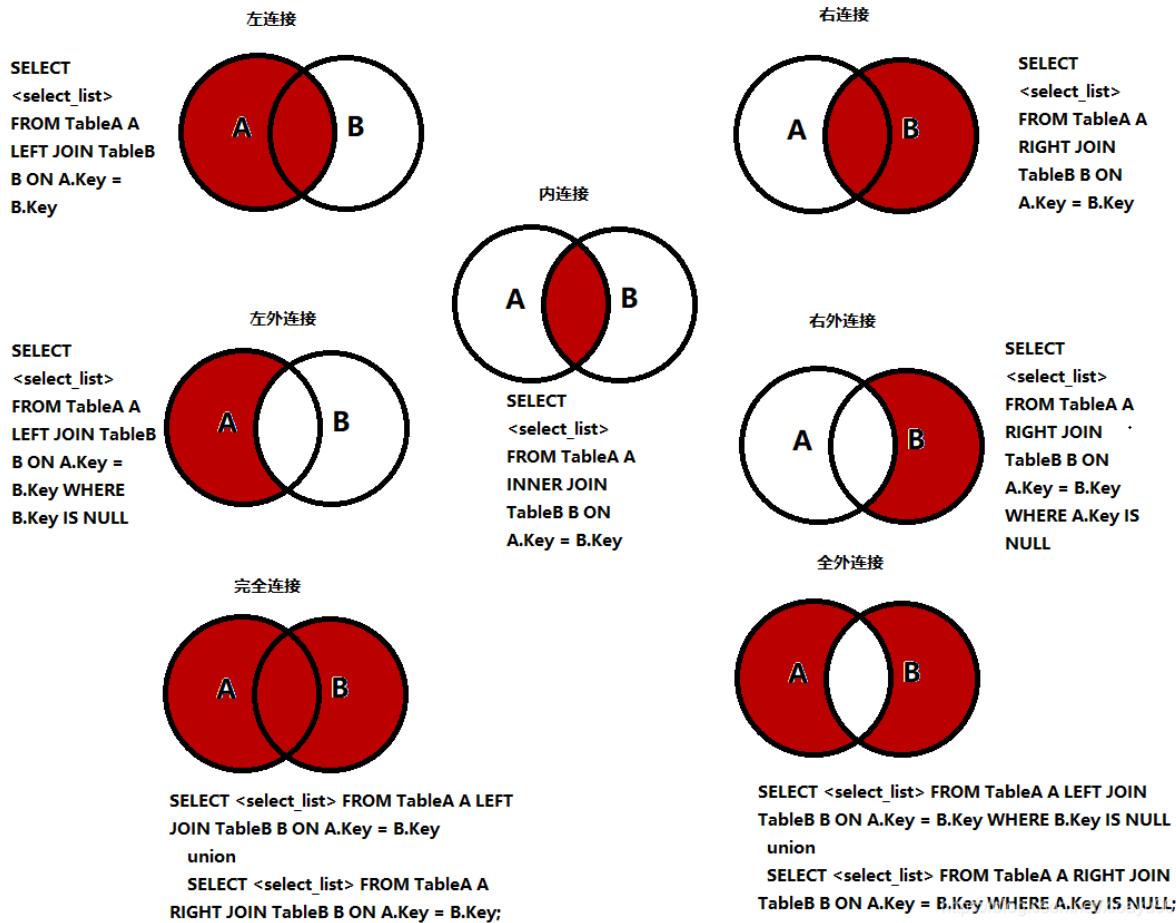
```
-- 查询没有手机号的同学
```

```
SELECT studentname,phone FROM student  
WHERE phone IS NULL OR phone = '';
```

4.4、联表查询 JOIN

联表查询





-- 思考题（查询了参加考试的同学信息： 学号，学生姓名，科目名，分数）
/* 思路
1. 分析需求，分析查询的字段来自哪些表， student、result、subject（连接查询）
2. 确定使用哪种连接查询？ 7种
确定交叉点（这两个表中哪个数据是相同的）
判断的条件： 学生表的中 studentNo = 成绩表 studentNo
*/

-- 联表查询
-- 查询参加了考试的同学的学号、姓名、科目编号、分数
/*
思路：
1. 分析查询的字段来自哪些表
2. 确定使用哪种连接查询
*/
SELECT * FROM student;
SELECT * FROM result;

-- 联表查询
SELECT s.studentno,studentname,subjectno,studentresult
FROM student AS s,result AS r
WHERE s.studentno = r.studentno;

SELECT s.studentno,studentname,subjectno,studentresult
FROM student AS s
INNER JOIN result AS r
ON s.studentno = r.studentno;

-- 查询缺考的同学
SELECT s.studentno,studentname,subjectno,studentresult
FROM student AS s
LEFT JOIN result AS r

```

ON s.studentno = r.studentno
WHERE studentresult IS NULL OR studentresult='';

-- 查询参加了考试的同学的学号、姓名、科目、分数(三表查询)
SELECT S.studentno,S.studentname,SS.subjectname,R.studentresult
FROM student AS S
INNER JOIN result AS R
ON S.studentno = R.studentno
INNER JOIN subject AS SS
ON R.subjectno = SS.subjectno;

```

自连接

自己的表和自己的表连接

```

CREATE TABLE `category`(
`categoryid` INT(10) UNSIGNED NOT NULL AUTO_INCREMENT COMMENT '主题id',
`pid` INT(10) NOT NULL COMMENT '父id',
`categoryname` VARCHAR(50) NOT NULL COMMENT '主题名字',
PRIMARY KEY (`categoryid`)
) ENGINE=INNODB AUTO_INCREMENT=9 DEFAULT CHARSET=utf8;

INSERT INTO `category`(`categoryid`, `pid`, `categoryname`)
VALUES ('2','1','信息技术'),
('3','1','软件开发'),
('5','1','美术设计'),
('4','3','数据库'),
('8','2','办公信息'),
('6','3','web开发'),
('7','5','ps技术');

```

```

-- 查询父子信息
SELECT b.categoryname AS '父栏目',a.categoryname AS '子栏目'
FROM category AS a,category AS b
WHERE a.pid = b.categoryid

```

4.5、排序和分页

排序

```

SELECT S.studentno,S.studentname,SS.subjectname,R.studentresult
FROM student AS S
INNER JOIN result AS R
ON S.studentno = R.studentno
INNER JOIN subject AS SS
ON R.subjectno = SS.subjectno
WHERE SS.subjectname = 'C语言-1'
ORDER BY R.studentresult ASC

```

分页

为什么要分页?

- 缓解数据库压力
- 给人好的体验 瀑布流

```
-- 分页
-- 语法: LIMIT 起始值索引（从0开始），页面的大小
-- 第n页: LIMIT (n-1)*pageSize,pageSize
SELECT S.studentno,S.studentname,SS.subjectname,R.studentresult
FROM student AS S
INNER JOIN result AS R
ON S.studentno = R.studentno
INNER JOIN subject AS SS
ON R.subjectno = SS.subjectno
WHERE SS.subjectname = 'C语言-1'
ORDER BY R.studentresult ASC
LIMIT 3,3
```

语法:

```
语法: LIMIT 起始值索引（从0开始），页面的大小
```

4.6、子查询

4.7、分组和过滤

```
select count(*) from student;
select count(1) from student;
select count('studentno') from student;

-- 查询不同课程的平均分、最高分、最低分，需要根据课程分组，平均分大于80
select `subject`.subjectname,AVG(studentresult) as '平均分',
MAX(studentresult) as '最高分',MIN(studentresult) as '最低分'
from result
inner join `subject`
on result.subjectno = `subject`.subjectno
group by result.subjectno
having AVG(studentresult) >= 80
```

4.8、SELECT 小结

顺序很重要：

```

1 select 去重 要查询的字段 from 表 (注意：表和字段可以取别名)
2 xxx join 要连接的表 on 等值判断
3 where (具体的值，子查询语句)
4 Group By (通过哪个字段来分组)
5 Having (过滤分组后的信息，条件和 where是一样的，位置不同)
6 Order By .. (通过哪个字段排序) [升序/降序]
7 Limit startIndex, pageSize

```

SELECT语法

```

1 SELECT [ALL | DISTINCT]
2 [* | table.* | [table.field1[as alias1][,table.field2[as alias2]][,...]]]
3 FROM table_name [as table_alias]
4 [left | right | inner join table_name2] -- 联合查询
5 [WHERE ...] -- 指定结果需满足的条件
6 [GROUP BY ...] -- 指定结果按照哪些字段来分组
7 [HAVING] -- 过滤分组的记录必须满足的次要条件
8 [ORDER BY ...] -- 指定查询记录按一个或多个条件排序
9 [LIMIT {[offset,]row_count | row_countOFFSET offset}]; 
10 -- 指定查询的记录从哪条至哪条

```

注意：[]括号代表可选的，{}括号代表必选得

sql

5、MySQL 函数

5.1、常用函数

```

SELECT ABS(-1) -- 绝对值
SELECT CEILING(9.4) -- 向上取整
SELECT FLOOR(9.4) -- 向下取整
SELECT RAND() -- 返回 0-1之间的随机数
SELECT SIGN(10) -- 判断一个数的符号 0 - 0, 负数 - -1, 正数 - 1

SELECT CHAR_LENGTH('真厉害') -- 字符串长度
SELECT CONCAT('长风','破流','都是浪') -- 拼接字符串
SELECT LOWER('Dou Shi Hao De') -- 小写
SELECT UPPER('Dou Shi Hao De') -- 大写
SELECT REVERSE('Dou Shi Hao De') -- 反转

```

5.2、聚合函数

| 函数名 | 描述 |
|---------|-----|
| COUNT() | 计数 |
| SUM() | 求和 |
| AVG() | 平均值 |
| MAX() | 最大值 |
| MIN() | 最小值 |

count(1)、count(*)与count(列名)的执行区别？

执行效果上：

count(*)包括了所有的列，相当于行数，在统计结果的时候，**不会忽略列值为NULL**

count(1)包括了忽略所有列，用1代表代码行，在统计结果的时候，**不会忽略列值为NULL**

count(列名)只包括列名那一列，在统计结果的时候，**会忽略列值为空** (这里的空不是只空字符串或者0，而是表示null) 的计数，即某个字段值为NULL时，不统计。

执行效率上：

1. 列名为主键，count(列名)会比count(1)快
2. 列名不为主键，count(1)会比count(列名)快
3. 如果表多个列并且没有主键，则 count (1) 的执行效率优于 count (*)
4. 如果有主键，则 select count (主键) 的执行效率是最优的
5. 如果表只有一个字段，则 select count (*) 最优

```
select count(*) from student;
select count(1) from student;
select count('studentno') from student;

-- 查询不同课程的平均分、最高分、最低分，需要根据课程分组，平均分大于80
select `subject`.subjectname, AVG(studentresult) as '平均分',
MAX(studentresult) as '最高分', MIN(studentresult) as '最低分'
from result
inner join `subject`
on result.subjectno = `subject`.subjectno
group by result.subjectno
having AVG(studentresult) >= 80
```

5.3、数据库级别的 MD5加密

5.3、数据库级别的MD5加密（扩展）

什么是MD5？

主要增强算法复杂度和不可逆性。

MD5 不可逆，具体的值的 md5 是一样的

MD5 破解网站的原理，背后有一个字典，MD5加密后的值，加密的前值

```

1 -- =====测试MD5 加密=====
2
3 CREATE TABLE `testmd5`(
4     `id` INT(4) NOT NULL,
5     `name` VARCHAR(20) NOT NULL,
6     `pwd` VARCHAR(50) NOT NULL,
7     PRIMARY KEY(`id`)
8 )ENGINE=INNODB DEFAULT CHARSET=utf8
9
10 -- 明文密码
11 INSERT INTO testmd5 VALUES(1,'zhangsan','123456'),(2,'lisi','123456'),(3,'wangwu','123456')
12
13 -- 加密
14 UPDATE testmd5 SET pwd=MD5(pwd) WHERE id = 1
15
16 UPDATE testmd5 SET pwd=MD5(pwd) -- 加密全部的密码
17
18 -- 插入的时候加密
19 INSERT INTO testmd5 VALUES(4,'xiaoming',MD5('123456'))
20
21 -- 如何校验：将用户传递进来的密码，进行md5加密，然后比对加密后的值
22 SELECT * FROM testmd5 WHERE `name`='xiaoming' AND pwd=MD5('123456')

```

sql

6、事务

6.1、什么是事务

要么都成功，要么都失败

在数据库系统中，一个事务是指：由一系列数据库操作组成的一个完整的逻辑过程。例如银行转帐，从原账户扣除金额，以及向目标账户添加金额，这两个数据库操作的总和，构成一个完整的逻辑过程，不可拆分。这个过程被称为一个事务，具有ACID特性。

实例：

银行转账，将一组SQL放在同一个批次去执行

-
- 1、SQL 执行 A给B转账 A 1000 --> 200 B 200
 - 2、SQL 执行 B收到A的转账 A 800 --> B 400
-

6.2、事务四大特性

参考博客：<https://blog.csdn.net/dengjili/article/details/82468576>

ACID，是指数据库管理系统（DBMS）在写入或更新资料的过程中，为保证事务（transaction）是正确可靠的，所必须具备的四个特性

1. **原子性（Atomicity）**：事务是最小的执行单位，不允许分割。**事务的原子性确保动作要么全部完成，要么完全不起作用；**
2. **一致性（Consistency）**：**执行事务前后，数据保持一致**，多个事务对同一个数据读取的结果是相同的；
3. **隔离性（Isolation）**：**并发访问数据库时，一个用户的事务不被其他事务所干扰**，各并发事务之间数据库是独立的；
4. **持久性（Durability）**：**一个事务被提交之后。它对数据库中数据的改变是持久的（事务一旦提交就不可逆了），即使数据库发生故障也不应该对其有任何影响**

6.3、并发事务带来哪些问题？

在典型的应用程序中，多个事务并发运行，经常会操作相同的数据来完成各自的任务（多个用户对同一数据进行操作）。并发虽然是必须的，但可能会导致以下的问题。

- **脏读 (Dirty read)** : 指一个事务读取了另外一个事务未提交的数据。
 - 当一个事务正在访问数据并且对数据进行了修改，而这种修改还没有提交到数据库中，这时另外一个事务也访问了这个数据，然后使用了这个数据。因为这个数据是还没有提交的数据，那么另外一个事务读到的这个数据是“脏数据”，依据“脏数据”所做的操作可能是不正确的。
- **丢失修改 (Lost to modify)** : 指在一个事务读取一个数据时，另外一个事务也访问了该数据，那么在第一个事务中修改了这个数据后，第二个事务也修改了这个数据。这样第一个事务内的修改结果就被丢失，因此称为丢失修改。例如：事务1读取某表中的数据A=20，事务2也读取A=20，事务1修改A=A-1，事务2也修改A=A-1，最终结果A=19，事务1的修改被丢失。
- **不可重复读 (Unrepeatable read)** : 在一个事务内读取表中的某一行数据，多次读取结果不同
 - 指在一个事务内多次读同一数据。在这个事务还没有结束时，另一个事务也访问该数据。那么，在第一个事务中的两次读数据之间，由于第二个事务的修改导致第一个事务两次读取的数据可能不太一样。这就发生了在一个事务内两次读到的数据是不一样的情况，因此称为不可重复读。
- **幻读 (Phantom read)** : 是指在一个事务内读取到了别的事务插入的数据，导致前后读取数量总量不一致。
 - 幻读与不可重复读类似。它发生在一个事务 (T1) 读取了几行数据，接着另一个并发事务 (T2) 插入了一些数据时。在随后的查询中，第一个事务 (T1) 就会发现多了一些原本不存在的记录，就好像发生了幻觉一样，所以称为幻读。

不可重复读和幻读区别：

不可重复读的重点是修改比如多次读取一条记录发现其中某些列的值被修改，幻读的重点在于新增或者删除比如多次读取一条记录发现记录增多或减少了。

看完上一P的人，记得去补充学习事务隔离级别，以及应用的业务场景！很重要！

6.4、事务隔离级别有哪些？MySQL的默认隔离级别是？

SQL 标准定义了四个隔离级别：

- **READ-UNCOMMITTED(读取未提交)**: 最低的隔离级别，允许读取尚未提交的数据变更，**可能会导致脏读、幻读或不可重复读**。
- **READ-COMMITTED(读取已提交)**: 允许读取并发事务已经提交的数据，**可以阻止脏读，但是幻读或不可重复读仍有可能发生**。
- **REPEATABLE-READ(可重复读)**: 对同一字段的多次读取结果都是一致的，除非数据是被本身事务自己所修改，**可以阻止脏读和不可重复读，但幻读仍有可能发生**。
- **SERIALIZABLE(可串行化)**: 最高的隔离级别，完全服从ACID的隔离级别。所有的事务依次逐个执行，这样事务之间就完全不可能产生干扰，也就是说，**该级别可以防止脏读、不可重复读以及幻读**。

| 隔离级别 | 脏读 | 不可重复读 | 幻读 |
|------------------|----|-------|----|
| READ-UNCOMMITTED | √ | √ | √ |
| READ-COMMITTED | ✗ | √ | √ |
| REPEATABLE-READ | ✗ | ✗ | √ |
| SERIALIZABLE | ✗ | ✗ | ✗ |

MySQL InnoDB 存储引擎的默认支持的隔离级别是 **REPEATABLE-READ (可重读)**

这里需要注意的是：与 SQL 标准不同的地方在于 InnoDB 存储引擎在 **REPEATABLE-READ (可重读)** 事务隔离级别下使用的是 Next-Key Lock 锁算法，因此可以避免幻读的产生，这与其他数据库系统(如 SQL Server) 是不同的。所以说InnoDB 存储引擎的默认支持的隔离级别是 **REPEATABLE-READ (可重读)** 已经可以完全保证事务的隔离性要求，即达到了 SQL 标准的 **SERIALIZABLE(可串行化)** 隔离级别。因为隔离级别越低，事务请求的锁越少，所以大部分数据库系统的隔离级别都是 **READ-COMMITTED(读取提交内容)**，但是你要知道的是InnoDB 存储引擎默认使用 **REPEATABLE-READ (可重读)** 并不会有性能损失。

InnoDB 存储引擎在 **分布式事务** 的情况下一般会用到 **SERIALIZABLE(可串行化)** 隔离级别。

6.5、四种隔离级别的应用场景

数据库提供的四种隔离级别：

- 01: Read uncommitted(读未提交): 最低级别，任何情况都会发生。
- 02: Read Committed(读已提交): 可避免脏读的发生。
- 03: Repeatable read(可重复读): 可避免脏读、不可重复读的发生。
- 04: Serializable(串行化): 避免脏读、不可重复读，幻读的发生。

注： 四种隔离级别最高： Serializable 级别，最低的是 Read uncommitted 级别； 级别越高，执行效率就越低； 隔离级别的设置只对当前链接有效，对 JDBC 操作数据库来说，一个 Connection 对象相当于一个链接，只对该 Connection 对象设置的隔离级别只对该 connection 对象有效，与其它链接 connection 对象无关。

01: Mysql 的默认隔离级别是： 可重复读： Repeatable read；

02: oracle 数据库中，只支持 serializable(串行化) 级别和 Read committed(); 默认的是 Read committed 级别；

下面就四种隔离级别进行场景设计：

01: Read uncommitted 读未提交： 公司发工资了，领导把5000元打到sing的账号上，但是该事务并未提交，而sing正好去查看账户，发现工资已经到账，是5000元整，非常高兴。可是不幸的是，领导发现发给sing的工资金额不对，是2000元，于是迅速回滚了事务，修改金额后，将事务提交，最后sing实际的工资只有2000元，sing空欢喜一场。

02: Read committed 读已提交： sing拿着工资卡去消费，系统读取到卡里确实有2000元，而此时她的老婆也正好在网上转账，把sing工资卡的2000元转到另一账户，并在sing之前提交了事务，当sing扣款时，系统检查到sing的工资卡已经没有钱，扣款失败，sing十分纳闷，明明卡里有钱，为何.....

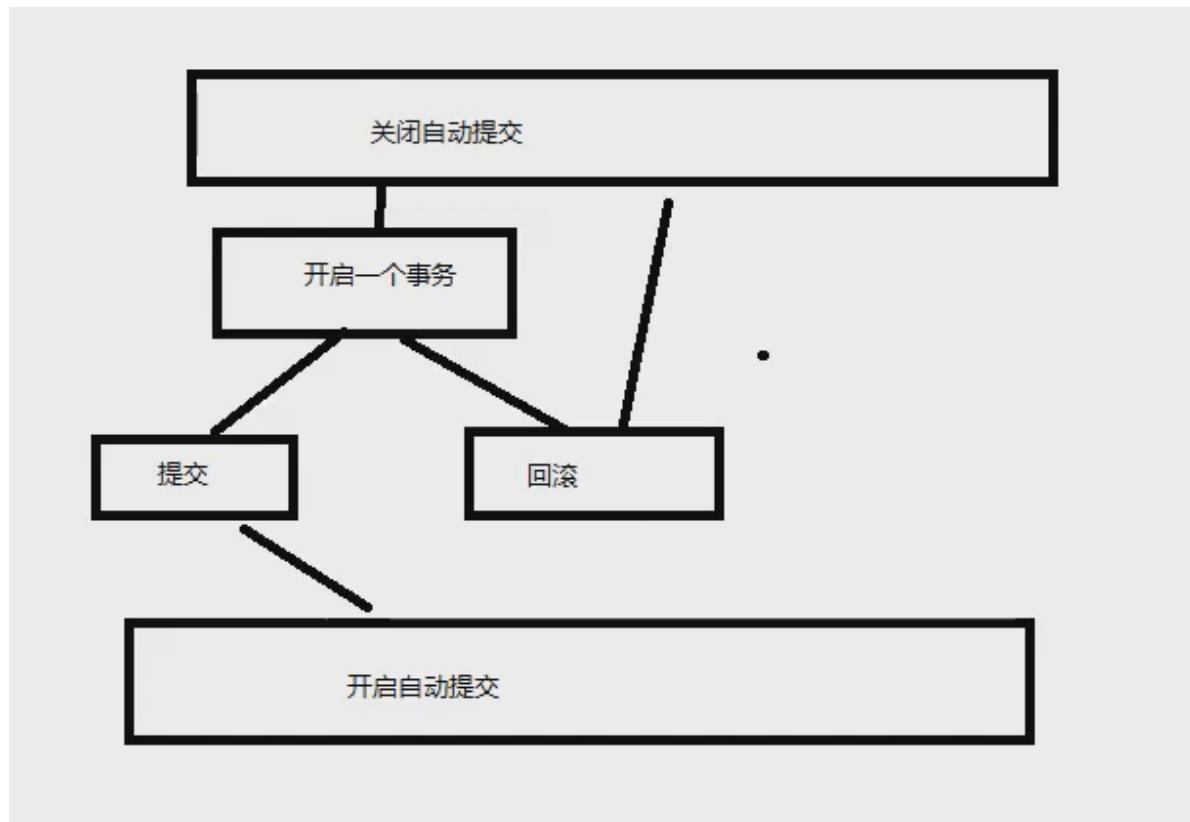
03: Repeatable read 重复读 当sing拿着工资卡去消费时，一旦系统开始读取工资卡信息（即事务开始），sing的老婆就不可能对该记录进行修改，也就是sing的老婆不能在此时转账。

04：重复读可能出现幻读：singgo的老婆工作在银行部门，她时常通过银行内部系统查看singgo的信用卡消费记录。有一天，她正在查询到singgo当月信用卡的总消费金额（`select sum(amount) from transaction where month = 本月`）为80元，而singgo此时正好在外面胡吃海塞后在收银台买单，消费1000元，即新增了一条1000元的消费记录（`insert transaction ...`），并提交了事务，随后singgo的老婆将singgo当月信用卡消费的明细打印到A4纸上，却发现消费总额为1080元，singgo的老婆很诧异，以为出现了幻觉，幻读就这样产生了。

Serializable:最高的事务隔离级别，代价花费最高，性能很低，很少使用，在此级别下，事务顺序执行，避免上述产生的情况。

6.6、实例，转账

事务的处理过程，如图：



-- mysql 默认时开启事务自动提交的

```
set autocommit = 0; -- 关闭  
set autocommit = 1; -- 开启
```

-- 手动处理事务

```
set autocommit = 0; -- 关闭事务自动提交  
-- 开启事务  
start transaction -- 标记一个事务的开始，从这个之后的sql都在同一个事务内
```

-- 提交：持久化

```
COMMIT
```

-- 回滚：回到事务开始前的状态

```
ROLLBACK  
-- 事务结束  
-- (COMMIT或ROLLBACK) 代表事务结束  
  
set autocommit = 1; -- 开启事务自动提交  
  
SAVEPOINT 【保存点名】-- 设置事务的保存点  
ROLLBACK TO SAVEPOINT 【保存点名】-- 回滚到保存点  
RELEASE 【保存点名】-- 释放保存点
```

```
create table `account`(  
`id` int PRIMARY KEY AUTO_INCREMENT COMMENT '账号',  
`name` varchar(20) COMMENT '名字',  
`money` decimal(9,2) COMMENT '余额'  
)engine=innodb default charset=utf8  
  
INSERT INTO account (`name`, `money`) values ('A', '2000.00'), ('B', 10000.00)
```

Navicat里面更新完数据，查看变化只能用查询语句才能看到变化

需要用这个语句SELECT * FROM `account`;

```
-- 模拟转账：事务  
set autocommit = 0;-- 关闭自动提交  
start transaction; -- 开启一个事务  
update account set money = money-500 where `name`='A'; -- A-500  
update account set money=money+500 where `name`='B';-- B+500  
  
commit; -- 提交事务，就被持久化了  
  
rollback; -- 回滚  
  
set autocommit=1; -- 恢复默认值
```

7、索引

文章：<https://blog.csdn.net/wangfeijiu/article/details/113409719>

索引在MySQL中也叫做“键”，是存储引擎用于快速找到记录的一种数据结构。索引对于良好的性能非常关键，尤其是当表中的数据量越来越大时，索引对于性能的影响愈发重要。

7.1、索引的分类

阿里开发规范不推荐联合主键，禁止使用数据库层面的外键，在web项目上层解决一致性问题

MySQL 的索引有两种分类方式：逻辑分类和物理分类。

逻辑分类

有多种逻辑划分的方式，比如按功能划分，按组成索引的列数划分等

按功能划分

- 主键索引：一张表只能有一个主键索引，不允许重复、不允许为 NULL；

- `ALTER TABLE TableName ADD PRIMARY KEY(column_list);`

- 唯一索引：数据列不允许重复，允许为 NULL 值，一张表可有多个唯一索引，索引列的值必须唯一，但允许有空值。如果是组合索引，则列值的组合必须唯一。

- `CREATE UNIQUE INDEX IndexName ON `TableName`(`字段名` (length));`
或者
`ALTER TABLE TableName ADD UNIQUE (column_list);`

- 普通索引：一张表可以创建多个普通索引，一个普通索引可以包含多个字段，允许数据重复，允许 NULL 值插入；

- `CREATE INDEX IndexName ON `TableName`(`字段名` (length));`
或者
`ALTER TABLE TableName ADD INDEX IndexName(`字段名` (length));`

- 全文索引：它查找的是文本中的关键词，主要用于全文检索（InnoDB 1.2.x 版本开始支持全文检索）

按列数划分

- 单例索引：一个索引只包含一个列，一个表可以有多个单例索引。
- 组合索引：一个组合索引包含两个或两个以上的列。查询的时候遵循 mysql 组合索引的“最左前缀”原则，即使用 where 时条件要按照建立索引的时候字段的排列方式放置索引才会生效。

物理分类

分为聚簇索引和非聚簇索引（有时也称辅助索引或二级索引）

聚簇索引：

聚簇是为了提高某个属性(或属性组)的查询速度，把这个或这些属性(称为聚簇码)上具有相同值的元组集中存放在连续的物理块。

- 聚簇索引（clustered index）不是单独的一种索引类型，而是一种数据存储方式。这种存储方式是依靠B+树来实现的，根据表的主键构造一棵B+树且B+树叶子节点存放的都是表的行记录数据时，方可称该主键索引为聚簇索引。聚簇索引也可理解为将数据存储与索引放到了一块，找到索引也就找到了数据。

非聚簇索引：

数据和索引是分开的，B+树叶子节点存放的不是数据表的行记录

虽然InnoDB和MyISAM存储引擎都默认使用B+树结构存储索引，但是只有InnoDB的主键索引才是聚簇索引，InnoDB中的辅助索引以及MyISAM使用的都是非聚簇索引。每张表最多只能拥有一个聚簇索引。

聚簇索引优缺点

优点：

- **数据访问更快**，因为聚簇索引将索引和数据保存在同一个B+树中，因此从聚簇索引中获取数据比非聚簇索引更快
- 聚簇索引对于主键的排序查找和范围查找速度非常快

缺点：

- **插入速度严重依赖于插入顺序**，按照主键的顺序插入是最快的方式，否则将会出现页分裂，严重影响性能。因此，对于InnoDB表，我们一般都会定义一个自增的ID列为主键（主键列不要选没有意义的自增列，选经常查询的条件列才好，不然无法体现其主键索引性能）
- **更新主键的代价很高**，因为将会导致被更新的行移动。因此，对于InnoDB表，我们一般定义主键为不可更新。
- **二级索引访问需要两次索引查找**，第一次找到主键值，第二次根据主键值找到行数据。

7.2 MySQL优化 — 看懂explain

博客链接：<https://blog.csdn.net/jiadajing267/article/details/81269067>

explain就是模拟sql优化器执行sql语句

什么是 explain？

explain模拟优化器执行SQL语句，在5.6以及以后的版本中，**除过select，其他比如insert, update和delete均可以使用explain查看执行计划**，从而知道mysql是如何处理sql语句，分析查询语句或者表结构的性能瓶颈。

作用：

- 1、表的读取顺序
- 2、数据读取操作的操作类型
- 3、哪些索引可以使用
- 4、哪些索引被实际使用
- 5、表之间的引用
- 6、每张表有多少行被优化器查询

explain 用法

explain+SQL语句即可！

执行计划包含的信息如下

| 信息 | 描述 |
|---------------|--|
| id | 查询的序号，包含一组数字，表示查询中执行select子句或操作表的顺序 两种情况 id相同，执行顺序从上往下 id不同，id值越大，优先级越高，越先执行 |
| select_type | 查询类型，主要用于区别普通查询，联合查询，子查询等的复杂查询 1、simple ——简单的select查询，查询中不包含子查询或者UNION 2、primary ——查询中若包含任何复杂的子部分，最外层查询被标记 3、subquery——在select或where列表中包含了子查询 4、derived——在from列表中包含的子查询被标记为derived（衍生），MySQL会递归执行这些子查询，把结果放到临时表中 5、union——如果第二个select出现在UNION之后，则被标记为UNION，如果union包含在from子句的子查询中，外层select被标记为derived 6、union result:UNION 的结果 |
| table | 输出的行所引用的表 |
| type | 显示联结类型，显示查询使用了何种类型，按照从最佳到最坏类型排序 1、system：表中仅有一行（=系统表）这是const联结类型的一个特例。 2、const：表示通过索引一次就找到，const用于比较primary key或者unique索引。因为只匹配一行数据，所以如果将主键置于where列表中，mysql能将该查询转换为一个常量 3、eq_ref:唯一性索引扫描，对于每个索引键，表中只有一条记录与之匹配。常见于唯一索引或者主键扫描 4、ref:非唯一性索引扫描，返回匹配某个单独值的所有行，本质上也是一种索引访问，它返回所有匹配某个单独值的行，可能会找多个符合条件的行，属于查找和扫描的混合体 5、range:只检索给定范围的行，使用一个索引来选择行。key列显示使用了哪个索引，一般就是where语句中出现了between,in等范围的查询。这种范围扫描索引扫描比全表扫描要好，因为它开始于索引的某一个点，而结束另一个点，不用全表扫描 6、index:index 与all区别为index类型只遍历索引树。通常比all快，因为索引文件比数据文件小很多。 7、all：遍历全表以找到匹配的行 注意:一般保证查询至少达到range级别，最好能达到ref。 |
| possible_keys | 指出MySQL能使用哪个索引在该表中找到行 |
| key | 显示MySQL实际决定使用的键(索引)。如果没有选择索引,键是NULL。查询中如果使用覆盖索引，则该索引和查询的select字段重叠。 |
| key_len | 表示索引中使用的字节数，该列计算查询中使用的索引的长度在不损失精度的情况下，长度越短越好。如果键是NULL，则长度为NULL。该字段显示为索引字段的最大可能长度，并非实际使用长度。 |
| ref | 显示索引的那一列被使用了，如果有可能是一个常数，哪些列或常量被用于查询索引列上的值 |
| rows | 根据表统计信息以及索引选用情况，大致估算出找到所需的记录所需要读取的行数 |

| 信息 | 描述 |
|-------|--|
| Extra | <p>包含不适合在其他列中显示，但是十分重要的额外信息</p> <p>1、Using filesort: 说明mysql会对数据适用一个外部的索引排序。而不是按照表内的索引顺序进行读取。MySQL中无法利用索引完成排序操作称为“文件排序”</p> <p>2、Using temporary: 使用了临时表保存中间结果，mysql在查询结果排序时使用临时表。常见于排序order by和分组查询group by。</p> <p>3、Using index: 表示相应的select操作使用覆盖索引，避免访问了表的数据行。如果同时出现using where，表名索引被用来执行索引键值的查找；如果没有同时出现using where，表名索引用来读取数据而非执行查询动作。</p> <p>4、Using where : 表明使用where过滤</p> <p>5、using join buffer: 使用了连接缓存</p> <p>6、impossible where: where子句的值总是false，不能用来获取任何元组</p> <p>7、select tables optimized away: 在没有group by子句的情况下，基于索引优化Min、max操作或者对于MyISAM存储引擎优化count (*)，不必等到执行阶段再进行计算，查询执行计划生成的阶段即完成优化。</p> <p>8、distinct: 优化distinct操作，在找到第一匹配的元组后即停止找同样值的动作。</p> |

SQL执行顺序

想要优化SQL，必须清楚知道SQL的执行顺序，这样再配合explain才能事半功倍！

extend

extended关键字：仅对select语句有效，在explain后使用extended关键字，可以显示filtered列显示了通过条件过滤出的行数的百分比估计值。

也可以通过show warnings显示扩展信息，输出中的 Message值SHOW WARNINGS显示优化程序如何限定SELECT语句 中的表名和列名， SELECT应用重写和优化规则后的外观，以及可能有关优化过程的其他说明。

实例

```
-- 索引的使用
-- 1.在创建表的时候给字段增加索引
-- 2.创建完毕后，增加索引

-- 显示所有的索引信息
SHOW INDEX FROM student;

-- 增加一个全文索引 (索引名) 列名
ALTER TABLE school.student ADD FULLTEXT INDEX `fulltext_studentname`(`studentname`);
```

```
CREATE TABLE `app_user` (
`id` BIGINT(20) UNSIGNED NOT NULL AUTO_INCREMENT,
`name` VARCHAR(50) DEFAULT '' COMMENT'用户昵称',
`email` VARCHAR(50) NOT NULL COMMENT'用户邮箱',
`phone` VARCHAR(20) DEFAULT '' COMMENT'手机号',
`gender` TINYINT(4) UNSIGNED DEFAULT '0' COMMENT '性别 (0: 男;1:女) ',
`password` VARCHAR(100) NOT NULL COMMENT '密码',
`age` TINYINT(4) DEFAULT '0' COMMENT '年龄',
```

```

`create_time` DATETIME DEFAULT CURRENT_TIMESTAMP,
`update_time` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
PRIMARY KEY (`id`)
) ENGINE=INNODB DEFAULT CHARSET=utf8 COMMENT = 'app用户表'

-- 插入100万数据.
DELIMITER $$

-- 写函数之前必须要写, 标志
CREATE FUNCTION mock_data()
RETURNS INT
DETERMINISTIC
BEGIN
DECLARE num INT DEFAULT 1000000;
DECLARE i INT DEFAULT 0;
WHILE i < num DO
    INSERT INTO
        app_user(`name`, `email`, `phone`, `gender`, `password`, `age`)VALUES(CONCAT('用
户',i),'24736743@qq.com',CONCAT('18',FLOOR(RAND()*99999999-
100000000)+100000000)),FLOOR(RAND()*2),UUID(),FLOOR(RAND()*100));
    SET i = i + 1;
END WHILE;
RETURN i;
END $$

SELECT mock_data(); -- 执行此函数 生成一百万条数据

```

```

-- 创建索引前查询
SELECT * FROM app_user WHERE name = '用户456';
SELECT * FROM app_user WHERE name = '用户9999';
SELECT * FROM app_user WHERE name = '用户7892';

EXPLAIN SELECT * FROM app_user WHERE name = '用户9999';

ALTER TABLE app_user ADD INDEX `index_name` (`name`);

-- 创建索引后查询
SELECT * FROM app_user WHERE name = '用户456';
SELECT * FROM app_user WHERE name = '用户9999';
SELECT * FROM app_user WHERE name = '用户7892';

EXPLAIN SELECT * FROM app_user WHERE name = '用户9999';

```

索引在数据量小的时候用处不大, 但在大数量的时候区别十分明显!

7.3、索引原则

- 索引不是多多好
- 不要对经常变动的数据加索引
- 数据量小的时候不需要加索引
- 索引一般加在需要经常查询的字段上
- 索引是存储在硬盘中的

索引的数据结构

- Hash类型的索引
- BTree: InnoDB 的默认数据结构

文章: <http://blog.codinglabs.org/articles/theory-of-mysql-index.html>

8、权限管理和数据库备份

9、数据库规约、三大范式

- 第一范式 (1NF) : 每一个分量必须是不可分的数据项
- 若 $R \in 1NF$, 且每一个非主属性完全函数依赖于任何一个候选码, 则 $R \in 2NF$
- 若 $R \in 3NF$, 则每一个非主属性既不传递依赖于码, 也不部分依赖于码

什么是数据依赖?

数据依赖是一个关系内部属性与属性之间的一种约束关系, 最重要的是函数依赖和多值依赖

三大范式

第一范式 (1NF)

原子性：保证每一列不可再分

第二范式 (2NF)

前提：满足第一范式

每张表只描述一件事情

第三范式 (3NF)

前提：满足第一范式 和 第二范式 I

第三范式需要确保数据表中的每一列数据都和主键直接相关, 而不能间接相关。

(规范数据库的设计)

规范性和性能的问题

关联查询的表不得超过三张表

- 考虑商业化的需求和目标, (成本, 用户体验!) 数据库的性能更加重要
- 在规范性能的问题的时候, 需要适当的考虑一下 规范性!
- 故意给某些表增加一些冗余的字段。 (从多表查询中变为单表查询)
- 故意增加一些计算列 (从大数据量降低为小数据量的查询: 索引)

10、JDBC (重点)

```
// 我的第一个JDBC程序
public class JdbcFirstDemo {
    public static void main(String[] args) throws ClassNotFoundException, SQLException {
        //1. 加载驱动
        Class.forName("com.mysql.jdbc.Driver"); // 固定写法，加载驱动

        //2. 用户信息和url
        // useUnicode=true&characterEncoding=utf8&useSSL=true
        String url = "jdbc:mysql://localhost:3306/jdbctest?useUnicode=true&characterEncoding=utf8&useSSL=true";
        String username = "root";
        String password = "123456";
        //3. 连接成功，数据库对象 Connection 代表数据库
        Connection connection = DriverManager.getConnection(url, username, password);
```

```
//4. 执行SQL的对象 Statement 执行sql的对象
Statement statement = connection.createStatement();

//5. 执行SQL的对象 去 执行SQL，可能存在结果，查看返回结果
String sql = "SELECT * FROM users";

ResultSet resultSet = statement.executeQuery(sql); //返回的结果集，结果集中封装了我们全部的查询出来的结果

while (resultSet.next()){
    System.out.println("id=" + resultSet.getObject( columnLabel: "id"));
    System.out.println("name=" + resultSet.getObject( columnLabel: "NAME"));
    System.out.println("pwd=" + resultSet.getObject( columnLabel: "PASSWORD"));
    System.out.println("email=" + resultSet.getObject( columnLabel: "email"));
    System.out.println("birth=" + resultSet.getObject( columnLabel: "birthday"));
}

//6、释放连接
resultSet.close();
statement.close();
connection.close();
```

步骤总结：

- 1、加载驱动
- 2、连接数据库 DriverManager
- 3、获得执行sql的对象 Statement
- 4、获得返回的结果集
- 5、释放连接

I

10.1、SQL注入

SQL注入即是指web应用程序对用户输入数据的合法性没有判断或过滤不严，攻击者可以在web应用程序中事先定义好的查询语句的结尾上添加额外的SQL语句，在管理员不知情的情况下实现非法操作，以此来实现欺骗数据库服务器执行非授权的任意查询，从而进一步得到相应的数据信息。

```
public class SQL注入 {
    public static void main(String[] args) {
        // Login("kuangshen", "123456");
        Login(username: "' or '1=1", password: "' or '1=1");
    }

    // 登录业务
    public static void login(String username, String password) {
        Connection conn = null;
        Statement st = null;
        ResultSet rs = null;
        try {
            conn = JdbcUtils.getConnection();
            st = conn.createStatement();

        } // SELECT * FROM users WHERE `Name` = 'kuangshen' AND `password` = '123456';

        // SELECT * FROM users WHERE `Name` = '' or '1=1' AND `password` = '' or '1=1';
    }
}
```

10.2、Statement 与 PreparedStatement

Statement 与 PreparedStatement 区别？

PreparedStatement 可以防止SQL注入并且效率更高。

```
// Login("Lisi", "123456");
Login(username: "' or 1=1", password: "123456"); //

}

// 登录业务
public static void login(String username, String password) {

    Connection conn = null;
    PreparedStatement st = null;
    ResultSet rs = null;
    try {
        conn = JdbcUtils.getConnection();
        // PreparedStatement 防止SQL注入的本质，把传递进来的参数当做字符串
        // 假设其中存在转义字符，比如说 ' 会被直接转义
        String sql = "select * from users where `NAME`=? and `PASSWORD`=?"; |

        st = conn.prepareStatement(sql);
        st.setString(parameterIndex: 1, username);
        st.setString(parameterIndex: 2, password);

        rs = st.executeQuery(); // 查询完毕会返回一个结果集
        while (rs.next()) {
            System.out.println(rs.getString(columnLabel: "NAME"));
            System.out.println(rs.getString(columnLabel: "password"));
            System.out.println("=====");
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

10.3、JDBC操作事务

代码实现

- 1、开启事务 `conn.setAutoCommit(false);`
- 2、一组业务执行完毕，提交事务
- 3、可以在catch语句中显示的定义回滚语句，但默认失败就会回滚

```
|  
try {  
    conn = JdbcUtils.getConnection();  
    // 关闭数据库的自动提交，自动会开启事务  
    conn.setAutoCommit(false); // 开启事务  
  
    String sql1 = "update account set money = money-100 where name = 'A'";  
    st = conn.prepareStatement(sql1);  
    st.executeUpdate();  
  
    String sql2 = "update account set money = money+100 where name = 'B'";  
    st = conn.prepareStatement(sql2);  
    st.executeUpdate();  
  
    // 业务完毕，提交事务  
    conn.commit();  
    System.out.println("成功！");  
  
} catch (SQLException e) {  
    try {  
        conn.rollback(); // 如果失败则回滚事务  
    } catch (SQLException e1) {  
        e1.printStackTrace();  
    }  
    e.printStackTrace();  
} finally {  
    JdbcUtils.release(conn, st, rs);  
}
```

10.4、数据库连接池

10.9、数据库连接池

数据库连接 --- 执行完毕 --- 释放

连接 -- 释放 十分浪费系统资源

池化技术：准备一些预先的资源，过来就连接预先准备好的

最小连接数： 10

最大连接数： 15

等待超时： 100ms

编写连接池，实现一个接口 `DataSource`

开源数据源实现

I
DBCP

C3P0

Druid：阿里巴巴

使用了这些数据库连接池之后，我们在项目开发中就不需要编写连接数据库的代码了！

mysql查询什么时候用on什么时候用where？

数据库在通过连接两张或多张表来返回记录时，都会生成一张中间的临时表，然后再将这张临时表返回给用户。

在使用left join时，on和where条件的区别如下：

- 1、on条件是在生成临时表时使用的条件，它不管on中的条件是否为真，都会返回左边表中的记录。
- 2、where条件是在临时表生成好后，再对临时表进行过滤的条件。这时已经没有left join的含义（必须返回左边表的记录）了，条件不为真的就全部过滤掉。

MySQL 临时表在我们需要保存一些临时数据时是非常有用的。临时表只在当前连接可见，当关闭连接时，Mysql会自动删除表并释放所有空间。

JOIN联表中ON,WHERE后面跟条件的区别？

在JOIN操作里，有几种情况。LEFT JOIN,RIGHT JOIN,INNER JOIN等。

说明：区分on和where首先我们将连接分为内部连接和非内部连接，内部连接时on和where的作用是一样的，通常我们分不清它们的区别说的是非内部连接

一般**on**用来连接两个表，只的是连接的条件，在内部连接时，可以省略**on**，此时它表示的是两个表的笛卡尔积；使用**on**连接后，mysql会生成一张临时表，而**where**就是在临时表的基础上，根据**where**子句来筛选出符合条件的记录，因此**where**是用来筛选的

内部连接(inner join)

说明：join默认为inner join，当为内部连接时，on和where的作用你可以看做是一样的

非内部连接(left join、right join、full join等)

一般分不清区别就是在使用非内部连接时，

MySQL中数据类型的长度问题总结

1) : int类型，对吧，经常像保存个时间戳啊、id啊、数量啊、排序啊都会用到

提问：INT (11) 是什么意思？最大保存十一位数么？

回答：不是愣个回事，听我给你摆，首先长度，也就是INT (M), M指示最大显示宽度，不是说限制你只能插入规定长度的数据，如果不信你试一下：INT(1),你可以插数字1，也可以插100,1000。那到底是撒子意思耶？MySQL手册上头说的清清楚楚：

M指示最大显示宽度，最大有效显示宽度是255。显示宽度与存储大小或类型包含的值的范围无关

好，这下晓得的撒？我们指定的INT (10)、INT (11) 是指定显示宽度，不是存储大小或者值的范围，再看一个例子：

INT (3) 的情况下插入数字20，和数字2以及数字200，显示结果为：

```
020;  
002;  
200;
```

MySQL还支持选择在该类型关键字后面的括号内指定整数值的显示宽度(例如，INT(3))。该可选显示宽度规定用于显示宽度小于指定的列宽度的值时从左侧填满宽度。显示宽度并不限制可以在列内保存的值的范围，也不限制超过列的指定宽度的值的显示。

说白了就是你设定最大宽度，插入的数据不够大，就从填充0，给你填满，够大就随便你，不得管你。这下就很通透了撒，哈哈~(但是强烈建议合理分配，过大过小都不好)

Tip: 对于int类型的，如果不需要存取负值，最好加上UNSIGNED；对于经常出现在WHERE语句中的字段，考虑加索引，整形尤其适合加索引。

2) : VARCHAR和CHAR 类型，先VARCHAR，经常用的比如用户名之类可变长度，那就说用户名吧

提问：你说这个用户名，VARCHAR (20) 是撒子意思噢？（当然自己把握，没说必须20），这是规定长度了吧？用没得限制哦？汉字英文都是一样的迈？

回答：对头，就是可变长度，你阔以叫渣渣辉，也阔以叫乌木喂喂威恩耶吞温威乌温穆扁欧萨斯，20个字符以内都阔以

说到是不是中英文都一样20个字符，在MySQL5.0之后：是的，一样。但在低版本中，英文肯定还是20，但中文要diao一点，一个占三个字节，所以存不了20个汉字，最多就6,7个？？

然后说CHAR，也就是固定长度了，比如密码，常见的MD5加密，就是CHAR (32) 对吧，多了少了都不行。

他们俩的区别就是一个可变长度一个固定长度，长度的区别CHAR是0-255，VARCHAR最大长度不固定(64K?反正一般文本够用)，存在像字符集不同会受影响之类的因素，感兴趣可以自己去查查，我.....晓得撒（懒）。大文本可以考虑TEXT，都是字符串类型的，最大到4G。

然后想说的是

1. 如果知道文本的长度，最好直接用定长CHAR。
2. CHAR>VARCHAR>TEXT。（具体以存储引擎而定）
3. CHAR和VARCHAR可以有默认值，TEXT不能指定默认值。
4. 尽可能节约空间使用更短的列，分配真正需要的空间，10和100都能存“你好”，懂得起撒

3) : 最后是数值型中的DECIMAL数据类型，看到一段描述：

float:浮点型，含字节数为4, 32bit, 数值范围为-3.4E38~3.4E38 (7个有效位)

double:双精度实型，含字节数为8, 64bit数值范围-1.7E308~1.7E308 (15个有效位)

decimal:数字型，128bit, 不存在精度损失，常用于银行帐目计算。 (28个有效位)

因为在建表时，我价格字段就用的DECIMAL，之前用的float,

提问：为什么不用float?

回答：float, double等非标准类型，在数据库中保存的是近似值，而DECIMAL是以字符串的形式保存数值。打个比方，我存的是整数的时候，他就整数给我处理了，（举一反三？？）我存0.00给我实际存个0，我存14.00实际给我存个14，是都可以存浮点数，但是涉及到钱，金额这方面还是给我精准着来。

DECIMAL(M,D)

a指定指定小数点左边和右边可以存储的十进制数字的最大个数，最大精度38。

b指定小数点右边可以存储的十进制数字的最大个数。小数位数必须是从0到a之间的值。默认小数位数是0。

也就是说，M就是总长度，D就是小数点后面的长度。比如：

DECIMAL (5,4) => 总长度不超过5位数字，并且小数点后头必须要4位数字：1.2345

DECIMAL (14,9) => 总长度5位数字，整数5位，小数点后9位：12345.123456789

注意：

超出范围或者长度不够会被截断或补位

example:DECIMAL (9,4)

1. insert 12.12=>12.1200 因为小数点后未满4位，补0。
2. insert 12.12345=>12.1235 小数点只留4位，多余的自动四舍五入截断
3. insert 1234567.12=> 失败，因为小数点未满4位，补0变成1234567.1200，超过了9位。严格模式下报错，非严格模式存成999999.999。
4. 若插入的值未指定小数部分或者小数部分不足D位则会自动补到D位小数，若插入的值小数部分超过了D位则会发生截断，截取前D位小数。M值得是整数部分加小数部分的总长度，也即插入的数字整数部分不能超过M-D位，否则不能成功插入，会报超出范围的错误。
5. 可能失误操作，感兴趣请自己尝试验证结果

```
create table Student(
Sno int COMMENT '学号',
Sname varchar(10) COMMENT '姓名',
Ssex varchar(2) COMMENT '性别',
```

```
Sage int COMMENT '年龄',
Sdept varchar(2) COMMENT '所在系',
PRIMARY KEY(Sno)
)
create table Course(
Cno int COMMENT '课程号',
Cname varchar(10) COMMENT '课程名',
Cpno int COMMENT '先行课',
Ccredit int COMMENT '学分',
PRIMARY KEY(Cno)
)
create table SC(
Sno int COMMENT '学号',
Cno int COMMENT '课程号',
Grade int COMMENT '成绩',
PRIMARY KEY(Sno,Cno)
)
```

存在量词 EXISTS:

```
select Sname
from Student
where exists(
select *
from SC
where Sno = Student.Sno and Cno = 1);
```