初识MySQL

前端：展示数据库中的数据

后台：连接点：连接数据库JDBC，连接前端（控制，控制试图跳转，给前端传递数据）

数据库：存数据

```
未来路线：
操作系统
数据结构与算法
离散数学
数字电路
体系结构
编译原理
。。。
```

1.1，为什么学习数据库

1. 岗位需求
2. 现在的世界，大数据时代
3. 被迫需求：存数据
4. 数据库是所有软件体系中最核心的存在

1.2，什么是数据库

数据库（DataBase）

概念：数据仓库，软件（安装在操作系统之上的），可以存储大量的数据

作用：存储数据，管理数据

1.3，DBMS

关系型数据库：（SQL）

- MySQL，Oracle，Sql Server
- 通过表和表之间，行和列之间的关系进行数据的存储：例如考勤表

非关系型数据库：（NoSQL-----Not Onlt Sql）

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

DBMS（数据库管理系统）

- 数据库管理软件，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

1.4,MySQL简介

MySQL是一个关系型数据库管理系统

开源的数据库软件

5.7稳定版

8.0逐渐趋于稳定了

安装建议：

尽量不要使用exe，安装会往注册表更改

尽可能用压缩包进行下载

1.5，安装MySQL

略

1.6，安装SQLyog

每一个SQLyog的操作，实际上就相当于执行了sql语句

1.7，连接数据库

```
mysql -uroot -p123456 --连接数据库

update mysql.user set authentication_string=password('123456') where user='root' and Host = 'localhost'; -- 修改用户密码
flush privileges;   -- 刷新权限

---------------------------------------
-- 所有的语句都使用; 结尾
show databases;  -- 查看所有的数据库

mysql> use school  -- 切换数据库 use 数据库名
Database changed

show tables; -- 查看数据库中所有的表
describe student; -- 显示数据库中所有的表的信息

create database westos; -- 创建一个数据库

exit;  --退出连接

-- 单行注释（SQL 的本来的注释）
/*    （sql的多行注释）
helloi
asdas
dasdas
*/
```

数据库xxx语言

DDL：数据库定义语言

DML：数据库管理语言

DQL：数据库查询语言

DCL：数据库控制语言

2，操作数据库

操作数据库--操作数据库中的表--操作数据库中表的数据

mysql的关键字不区分大小写

2.1，操作数据库（了解）

创建数据库

```
CREATE DATABASE IF NOT EXISTS westos
```

删除数据库

```
DROP DATABASE IF EXISTS westos
```

使用数据库

反引号表示里面的东西是一个字符串，防止与MySQL的关键字冲突

```
--如果表名或者字段名是一个关键字，则需要加入反引号``
USE `mipd-demo02`
```

查看所有的数据库

```
SHOW DATABASE
```

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**
- bigint 较大的数据 8个字节
- float 单精度（浮点数） 4个字节
- double 浮点数 8个字节（精度问题）
- decimal 字符串形式的浮点数 金融计算的是或一般是使用decimal

字符串

- char 字符串固定大小 0～255
- **varchar 可变字符串 0～65535**
- tinytext 微型文本 2^8-1 可以用来写博客
- text 文本串 2^16-1 可以用来存储大文本

时间日期

java.util.Date

- date YYYY-MM-DD 日期
- time HH:mm:ss 时间格式
- **datetime YYYY-MM-DD HH:mm:ss**
- **timestamp 时间戳 1970.1.1到现在的毫秒数！！！（全球是统一的～）**
- year 年份表示

null

- 没有值，未知
- 注意，不要使用NULL进行运算，结果必然为NULL

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
- 不足的位数是用0来填充

自增

- 通常理解为自增，自动在上一条记录的基础上加1
- 通常用来设计唯一的主键，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

7

默认

- 填写默认的值
- sex，默认值为男，如果不指定该列的值，则值为默认的值

2.4，创建数据库

```
-- auto increment:自增

CREATE TABLE IF NOT EXISTS `student`(
  `id` INT(4) NOT NULL AUTO_INCREMENT COMMENT '学号',
  `name` VARCHAR(30) NOT NULL DEFAULT '匿名' COMMENT '姓名',
  `pwd` VARCHAR(20) NOT NULL DEFAULT '123456' COMMENT '密码',
  `sex` VARCHAR(2) NOT NULL DEFAULT '女' COMMENT '性别',
  `birthday` DATETIME DEFAULT NULL COMMENT '出生日期',
  `address` VARCHAR(100) DEFAULT NULL COMMENT '家庭住址',
  `email` VARCHAR(50) DEFAULT NULL COMMENT '邮箱',
  PRIMARY KEY(`id`)
)ENGINE=INNODB DEFAULT CHARSET=utf8
```

格式

```
CREATE TABLE [IF NOT EXISTS] `表名`(
    `字段名` 列类型 [属性] [索引] [注释],
    `字段名` 列类型 [属性] [索引] [注释],
    `字段名` 列类型 [属性] [索引] [注释],
    `字段名` 列类型 [属性] [索引] [注释],
    `字段名` 列类型 [属性] [索引] [注释],
    `字段名` 列类型 [属性] [索引] [注释]
)[表类型][字符集设置][注释]
```

常用命令

```
-- 逆向操作
-- 查看创建数据库的语句
SHOW CREATE DATABASE school
-- 查看创建表的语句
SHOW CREATE DATABASE student
-- 显示表的结构
DESC student
```

2.5，关于数据库引擎

```
-- 关于数据库引擎
/*
INNODB 默认使用
MYISAM 早些年使用的

*/
```

	MYISAM	INNODB
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	不支持
表空间的大小	较小	较大，约为MYISAM的2倍

常规使用操作：

- MYISAM 节约空间，速度较快
- INNODB 安全性高 事务处理，多表多用户操作（因为支持外键约束）

```
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对应的文件：
  - *.frm -表结构的定义文件
  - *.MYD -数据文件（data）
  - *.MYI -索引文件（index）

```
CHARSET=utf8
```

不设置的话，会是Mysql默认的字符编码设置

Mysql的默认编码是Latin1，不支持中文

也可以在my.ini中加入编码设置（但是不利于跨平台）

也可以在创建表的时候加入编码设置

2.6，修改和删除数据表

2.6.1，修改

```
-- 修改表名  ALTER TABLE `旧表名` RENAME AS `新表名`
ALTER TABLE `student` RENAME AS `student1`
-- 增加表的字段  ALTER TABLE `表名` ADD 字段名 列属性
ALTER TABLE `student1` ADD teacher INT(11)


-- 修改表的字段
ALTER TABLE `student1` MODIFY `teacher` VARCHAR(11) -- modify 修改约束
ALTER TABLE `student1` CHANGE teacher parents VARCHAR(1)    -- change字段重命名
（？为什么change也可以修改约束？？）


-- 删除表的字段   ALTER TABLE `表名` drop `列名`
ALTER TABLE `student1` DROP `parents`
```

2.6.2，删除

```
-- 删除表(如果表存在再删除)
DROP TABLE IF EXISTS `student1`

-- 所有的创建和删除操作尽量加上判断，以免报错
```

注意点

- `` 所有的字段名，使用这个进行括起来，以免和关键字发生冲突
- 注释：-- /**/
- sql关键字大小写不敏感，建议写小写
- 所有的符号全部用英文

3，MySQL数据管理

3.1，外键（了解即可）

约束（Constraint）

方式一：

```
CREATE TABLE `grade`(
  `gradeid` INT(10) NOT NULL COMMENT '年级ID',
  `name` VARCHAR(20) NOT NULL COMMENT '年级姓名',
  PRIMARY KEY (`gradeid`)
)ENGINE=INNODB DEFAULT CHARSET=utf8

-- 学生表的gradeid字段，要去引用gradeid
-- 定义外键key
-- 给这个外键添加约束（执行引用） references引用

CREATE TABLE IF NOT EXISTS `student`(
  `id` INT(4) NOT NULL AUTO_INCREMENT COMMENT '学号',
  `name` VARCHAR(30) NOT NULL DEFAULT '匿名' COMMENT '姓名',
  `pwd` VARCHAR(20) NOT NULL DEFAULT '123456' COMMENT '密码',
  `sex` VARCHAR(2) NOT NULL DEFAULT '女' COMMENT '性别',
  `birthday` DATETIME DEFAULT NULL COMMENT '出生日期',
  `address` VARCHAR(100) DEFAULT NULL COMMENT '家庭住址',
  `email` VARCHAR(50) DEFAULT NULL COMMENT '邮箱',
  `gradeid` INT(10) NOT NULL COMMENT '老师ID',
  PRIMARY KEY(`id`),
  KEY `FK_gradeid` (`gradeid`),
  CONSTRAINT `FK_gradeid` FOREIGN KEY (`gradeid`) REFERENCES `grade`(`gradeid`)
)ENGINE=INNODB DEFAULT CHARSET=utf8
```

删除有外键关系的表的时候，必须要先删除引用别人的表，在删除被引用的表。

方式二：

```
-- 创建表的时候如果没有外键关系
ALTER TABLE `student`
ADD CONSTRAINT `FK_gradeid` FOREIGN KEY(`gradeid`) REFERENCES
`grade`(`gradeid`);

-- 格式
-- ALTER TABLE `表`
-- ADD CONSTRAINT `约束名` FOREIGN KEY(`作为外键的列`) REFERENCES `哪个表`(`哪个字段
`);
```

以上的操作都是物理外键，数据库级别的外键，我们不推荐使用！！（避免数据库过多造成困扰，这个了解即可）

最佳实践

- 数据库就是单纯的表，只用来存数据，只有行（数据）和列（字段）
- 我们想使用多张表的数据，想使用外键，用程序进行实现

3.2，DML语言（全部记住）

数据库意义：数据存储，数据管理

DML语言：数据操作语言

- Insert
- update
- delete

3.3，添加

Insert

```
-- 插入语句
-- 一般插入语句，数据和字段必须一一对应
INSERT INTO `grade`(`name`) VALUES('大四')

-- 插入多个字段
INSERT INTO `grade`(`gradeid`,`name`)
VALUES('1','大四'),('2','大三');
```

- 字段和字段之间使用的逗号要注意是英文的
- 字段是可以省略的，但是后面的值必须一一对应
- 可以同时插入多条数据，values后面的值，需要使用 “,” 隔开

3.4，修改

update

```
-- 修改语句
UPDATE `grade` SET `name`='test' WHERE `gradeid`=1;


-- 如果不加限制条件，将会修改整张表
UPDATE `grade` SET `name`='test111'

-- 语法
update 表名 set `列名`=值 where [条件]

--修改多个属性
update 表名 set `列名1`=值1 [, `列名2`=值2 , ......]where [条件]
```

条件：where字句运算符

其中 <>的意思和!=都表是不等于

between

```
UPDATE `grade` SET `name`='test' WHERE `gradeid` between 3 and 5;
-- 就是表示gradeid在3，5之中，用区间表示：[3,5]
```

注意点：

- 列名是数据库的列，尽量带上``
- 条件，筛选的条件，如果没有指定，则会修改所有的行
- value，是一个属性得知，也可以是一个变量
- 多个设置的属性之间，使用 "," 隔开

3.5，删除

delete

```
-- 删除指定数据
delete from `表名` where 列名=值

-- 删除一个数据库表
delete from `表名`
```

truncate：完全清空一个数据库表，表的结构和索引约束不会变（delete也不会）

```
truncate `表名`
```

delete和taruncate的区别

- 相同点：都能删除数据，都不会删除表结构
- 不同
  - truncate 会重新设置 自增列！！！！
  - 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（因为InnoDB是存在内存中的，断电即失）
- MyISAM 继续从上一个自增量开始（因为存在文件中，所以不会丢失）

4，DQL查询数据（最重点！！！）

4.1，DQL

Data Query Language:数据查询语言

- 所有的查询操作都用它，Select
- 简单的查询，复杂的查询他都能做
- 数据库中最核心的语言
- 使用频率最高的语句

4.2，select完整的语法：

```
select [all | distinct]
{* | table.* | [table.field1[as alias1][,table.field2[as alias2]][......]]}
from table_name [as table_alias]
    [left | right | inner join table_name2]-- 联合查询
    [where ...] -- 指定结果需满足的条件
    [group by ...] -- 指定结果按照哪几个字段来分组
    [having]  -- 过滤分组的记录必须满足的次要条件
    [order by] -- 指定查询记录按一个或多个条件排序
    [limit {[offset,]row_count | row_countOFFSET offset}];-- 指定查询的记录从哪条到
哪条
```

4.3，指定查询字段

```
select的一些简单的已经略过

-- concat函数，可以起到拼接字符串
SELECT CONCAT('id：',gradeid) FROM `grade`
```

concat的结果就是

concat('id：',gradeid)
id：1
id：2

```
-- 也可以用
SELECT CONCAT('id：',gradeid) AS 新名字 FROM `grade`
-- 来改一下列名
```

去重（去除select查询出来的结果中重复的数据）

```
-- 一般的查询 结果查看表格A
select `列名` from `表名`
-- 去重  结果查看表格B
select distinct `列名` from `表名`
```

A	B
1000	1000
1000	1001
1000	1002
1001	(null)
1002	(null)

数据库的列

例如，查询mysql的版本

```
-- 查看系统的版本  （函数）
SELECT VERSION()
-- 可以用来计算  （表达式）
SELECT 100*3-1 AS `result`
-- 查询自增的步长  （关键字）
SELECT @@auto_increment_increment



-- 学员考试成绩加一分后
SELECT `studentNo`,`studentGrade`+1 AS '提分后' FROM `result`
```

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

4.4，where条件子句

作用：检索数据中符合条件的值

4.5，模糊查询

```
-- %必须和like搭配使用

-- 查询型刘的
-- like结合  %（代表0到任意个字符）
select * from `表名`
where studentName like '刘%'

-- 查询姓刘的同学，名字后面只有一个字
select * from `表名`
where studentName like '刘_'
-- 如果是找后面有两个字的，那就两个_


-- 扩展
-- 查询1001，1002，1003学员
select * from `表名`
where studentNo in (1001,1002,1003)
-- 也可以起到类似于where的作用
```

4.6，联表查询

建表语句

```
-- 建立一个institute表，用来存储学院信息
CREATE TABLE `testjoin`.`institute`(
    `insId` INT(3) NOT NULL COMMENT '学院id',
    `insName` VARCHAR(10) COMMENT '学院名称',
    PRIMARY KEY (`insId`)
 ) ENGINE=INNODB CHARSET=utf8 COLLATE=utf8_general_ci;



-- 建立一个student表，在学院名称一列的列名与institute是一致的
CREATE TABLE `testjoin`.`student`(
    `stuId` INT(3) NOT NULL COMMENT '学生id',
    `stuName` VARCHAR(10) COMMENT '学生姓名',
    `insName` VARCHAR(10) COMMENT '所属学院',
    PRIMARY KEY (`stuId`)
) ENGINE=INNODB CHARSET=utf8 COLLATE=utf8_general_ci;


-- 建立一个student1表，因为他与institute并没有公共列
CREATE TABLE `testjoin`.`student1`(
```

```
    `stuId` INT(3) NOT NULL COMMENT '学生id',
    `stuName` VARCHAR(10) COMMENT '学生姓名',
    `stuIns` VARCHAR(10) COMMENT '所属学院',
    PRIMARY KEY (`stuId`)
) ENGINE=INNODB CHARSET=utf8 COLLATE=utf8_general_ci;


-- 给student1表插入数据
INSERT INTO `testjoin`.`student` (`stuId`, `stuName`, `insName`) VALUES ('1',
'张三', '计信学院');
INSERT INTO `testjoin`.`student` (`stuId`, `stuName`, `insName`) VALUES ('2',
'李四', '文学院');
INSERT INTO `testjoin`.`student` (`stuId`, `stuName`, `insName`) VALUES ('3',
'王五', '历史学院');
INSERT INTO `testjoin`.`student` (`stuId`, `stuName`, `insName`) VALUES ('4',
'赵六', '计信学院');
INSERT INTO `testjoin`.`student` (`stuId`, `stuName`, `insName`) VALUES ('5',
'钱七', '物电学院');
INSERT INTO `testjoin`.`student` (`stuId`, `stuName`, `insName`) VALUES ('6',
'孙八', '历史学院');


-- 给student1表插入数据
INSERT INTO `testjoin`.`student1` (`stuId`, `stuName`, `stuIns`) VALUES ('1',
'张三', '计信学院');
INSERT INTO `testjoin`.`student1` (`stuId`, `stuName`, `stuIns`) VALUES ('2',
'李四', '文学院');
INSERT INTO `testjoin`.`student1` (`stuId`, `stuName`, `stuIns`) VALUES ('3',
'王五', '历史学院');
INSERT INTO `testjoin`.`student1` (`stuId`, `stuName`, `stuIns`) VALUES ('4',
'赵六', '计信学院');
INSERT INTO `testjoin`.`student1` (`stuId`, `stuName`, `stuIns`) VALUES ('5',
'钱七', '物电学院');
INSERT INTO `testjoin`.`student1` (`stuId`, `stuName`, `stuIns`) VALUES ('6',
'孙八', '历史学院');



-- 给institutie插入数据
INSERT INTO `testjoin`.`institute` (`insId`, `insName`) VALUES ('1', '计信学院');
INSERT INTO `testjoin`.`institute` (`insId`, `insName`) VALUES ('2', '历史学院');
INSERT INTO `testjoin`.`institute` (`insId`, `insName`) VALUES ('3', '物电学院');
INSERT INTO `testjoin`.`institute` (`insId`, `insName`) VALUES ('4', '文学院');
INSERT INTO `testjoin`.`institute` (`insId`, `insName`) VALUES ('5', '生物学院');
INSERT INTO `testjoin`.`institute` (`insId`, `insName`) VALUES ('6', '音乐学院');
INSERT INTO `testjoin`.`institute` (`insId`, `insName`) VALUES ('7', '政治学院');
INSERT INTO `testjoin`.`institute` (`insId`, `insName`) VALUES ('8', '法学院');
```

4.6.1，内连接

4.6.1.1，自然连接与等值连接

自然连接不等同于笛卡尔积

> 当两个关系没有公共属性时，自然连接就转化成笛卡尔 积。

自然连接(Natural join)是一种特殊的等值连接，它要求两个关系中进行比较的分量必须是相同的属性组，并且在结果中把重复的属性列去掉。而等值连接并不去掉重复的属性列。

```
-- 笛卡尔积
SELECT * FROM `institute`,`student`


-- 等值连接
SELECT * FROM `student` s ,`institute` i WHERE s.insName=i.insName
-- 同样的效果
SELECT * FROM `student` s INNER JOIN `institute` i WHERE s.insName=i.insName

SELECT * FROM `student1` s INNER JOIN `institute` i WHERE s.stuIns=i.insName




-- 自然连接
-- 自然连接(Natural join)是一种特殊的等值连接，
-- 它要求两个关系中进行比较的分量必须是相同的属性组，
-- 并且在结果中把重复的属性列去掉。而等值连接并不去掉重复的属性列。

-- 测试什么叫重复的列（--》推测，重复的属性列应该为列名）
SELECT * FROM `institute` NATURAL JOIN `student`

-- 若没有相同的属性组，则跟笛卡尔积一样
SELECT * FROM `institute` NATURAL JOIN `student1`


-- 推测
-- 笛卡尔积就是将两个关系R与S进行操作，所得的元组个数正是两个关系中的元组个数之积
-- 而等值连接就相当于在笛卡尔积的基础上，根据WHERE 列名1=列名2 这样的条件进行筛选，若where子条件没有一条符合，则为空
-- 而自然连接就是在等值连接的基础上，将等值连接的结果中，去掉重复的属性列（？何为重复？）
```

4.6.2，外连接

外连不但返回符合连接和查询条件的数据行，还返回不符合条件的一些行

```
-- 左外连接/左连接
SELECT * FROM `institute` i LEFT JOIN `student` s ON i.`insName`=s.`insName`

SELECT * FROM `institute` i LEFT OUTER JOIN `student` s ON
i.`insName`=s.`insName`


-- 右外连接/右连接
SELECT * FROM `institute` i RIGHT JOIN `student` s ON i.`insName`=s.`insName`

SELECT * FROM `institute` i RIGHT OUTER JOIN `student` s ON
i.`insName`=s.`insName`
```

4.6.2.1，左外连接/左连接

LEFT JOIN或LEFT OUTER JOIN

左向外联接的结果集包括 LEFT OUTER子句中指定的左表的所有行，而不仅仅是联接列所匹配的行。如果左表的某行在右表中没有匹配行，则在相关联的结果集行中右表的所有选择列表列均为空值。

左外连接还返回左表中不符合连接条件单符合查询条件的数据行。

4.6.2.2，右外连接/右连接

RIGHT JOIN 或 RIGHT OUTER JOIN

右向外联接是左向外联接的反向联接。将返回右表的所有行。如果右表的某行在左表中没有匹配行，则将为左表返回空值。

右外连接还返回右表中不符合连接条件单符合查询条件的数据行。

4.6.3，sql查询原理

第一、 单表查询：根据where条件过滤表中的记录，形成中间表（这个中间表对用户是不可见的）；然后根据select的选择列选择相应的列进行返回最终结果。

第二、 两表连接查询：对两表求积（笛卡尔积）并用on条件和连接类型进行过滤形成中间表；然后根据where条件过滤中间表的记录，并根据select指定的列返回查询结果。

第三、 多表连接查询：先对第一个和第二个表按照两表连接做查询，然后用查询结果和第三个表做连接查询，以此类推，直到所有的表都连接上为止，最终形成一个中间的结果表，然后根据where条件过滤中间表的记录，并根据select指定的列返回查询结果。

```
-- sql查询的原理
-- 对两表求积（笛卡尔积）并用on条件和连接类型进行过滤形成中间表；然后根据where条件过滤中间表的记录，并根据select指定的列返回查询结果
SELECT * FROM `institute` i LEFT JOIN `student` s ON i.`insName`=s.`insName`
WHERE s.`insName`='历史学院'
```

4.7，分页和排序

4.7.1，limit

为什么要分页？----缓解数据库压力

给人的体验更好

例如每页只显示5条数据

```
语法：limit 0
SELECT s.studentNo,studentName
FROM student s
left join result r
on s.studentNo=r.studentNo
order by studentName ASC
limit 0,5

-- 第一页：limit 0,5
-- 第二页：limit 5,5
-- 第三页：limit 10，5
-- 第N页：limit （N-1）*pagesize，pagesize
```

4.7.2，orderby

升序ASC，降序DESC

order by通过哪个字段排序，怎么排

```
-- 查询的结果根据成绩降序，排序
SELECT s.studentNo,s.studentName
FROM student s
left join result r
on s.studentNo=r.studentNo
order by studentName ASC
```

4.8，子查询

两种写法

示例：

```
select studentNo,subjectNo,studentResult
from result
where subjectNo=()

select subjectNo from subject where subjectName='值'

-- 合并起来就是：
select studentNo,subjectNo,studentResult
from result
where subjectNo=(
    select subjectNo from subject
    where subjectName='值'
)
order by studentResult DESC
```

由里及外

```
select studentNo,studentName from student where studentNo in(
    select studentNo from result where studentResult>80 and subject=(
        select subjectNo from subject where subjectName='高等数学'
    )
)
```

4.9，分组和过滤

```
select subjectName,AVG(StudentResult) AS 平均分,MAX(studentResult) AS 最高
分,MIN(studentResult) AS 最低分
from result r
inner join subject sub
on r.subjectNo=sub.subjectNo
group by r.subjectNo
having 平均分>80
-- 记得，where后不能用group by
-- 得要使用 having
```

5，MySQL函数

5.1，常用函数

ABS(-8) 绝对值

CEILING(9.4) 向上取整

FLOOR(9.4) 向下取整

RAND() 随机数

SIGN(0) 判断一个数的符号 0-0 负数返回-1 正数返回1

字符串函数

CHAR_LENGTH('xxxxxxxxxxxx') 返回字符串长度

CONCAT('字符串1','字符串2','字符串3') 拼接字符串

INSERT('我爱编程helloworld',1,2,'超级热爱') --从某个位置开始替换某个长度 --结果：超级热爱编程helloworld

LOWER('字符串') --转成小写

UPPER('字符串') --转成大写

INSTR('字符串','某个字符') --返回某个字符再字符串中的位置

SUBSTR('字符串',4,6) --截取初始位置到指定位置的字符串

REVERSE('字符串') --反转

。。。。。。。

时间和日期函数

CURRENT_DATE()/CUR_DATE --获取当前日期

NOW() -获取当前的时间

LOCALTIME() --本地时间

SYSDATE() --系统时间

YEAR(NOW()) --获取当前时间的年份

。。。。。。 --其他同理

5.2，聚合函数

函数名称	描述
count()	计数
sum()	求和
avg()	平均值
max()	最大值
min()	最小值
...	...

count(字段) --会忽略所有的null值

count(*) --不会忽略null值 本质：计算行数

count(1) --不会忽略null值 本质：计算行数

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，背后是一个字典，MD5加密后的值，加密前的值（这种网站一般是骗人的）

```
-- 加密
update `user` set pwd=MD5(pwd) where id=1

-- 将用户传递进来的密码
-- 例如 123456加密后的是一串字符串，那么在加密就是另外一串字符
-- 如果 另外一个用户的密码也是123456，加密后跟上一行加密一次后的字符串是一样的
```

6，事务

6.1，什么是事务

1，sql执行：A给B转账 A有1000块钱，B有200块钱，此时A给B转200块钱

2，sql执行：B收到A的钱 A有800块钱 B有400块钱

事务就是将一组SQL放在一个批次中去执行!

事务的原则：ACID原则：原子性，一致性，隔离性，持久性 （脏读，幻读）

原子性（Atomicity）

要么都成功要么都失败

一致性（Consistency）

事务前后的数据完整性要保持一致

持久性（Durability）

事务一旦提交则不可逆，被持久化到数据库中

隔离性（Isolation）

事务的隔离性是要多个用户并发访问数据库的时候，数据库为每一个用户开启的事务，不能被其他事务操作的数据所干扰，多个事务之间要相互隔离

> 隔离所导致的一些问题

脏读:

不可重复读

虚读（幻读）

mysql是默认开启事务自动提交的

```
set autocommit =0 /*关闭*/
set autocommit =1 /*开启*/



-- 事务开启
start transaction -- 标记一个事务的开启，从这个之后的sql语句都在同一个事务当中

insert xxx
insert xxx

-- 提交：持久化（成功！）事务一旦提交就被持久化了，回滚也回滚不回去，因为回滚是当失败时才进行回滚，成功了就直接提交了
commit
-- 回滚：回到原来的样子（失败）
rollback


-- 事务结束




-- 上述的事务在现实中的开发是怎么样进行开发的呢？？？

java代码中：
try(){
    正常代码
    /*执行成功自然就提及了*/
    commit();
}catch(){
    /*执行失败，报错就在catch里面进行回滚*/
    rollback();
}
```

```
-- 了解
savepoint 保存点名 -- 设置一个事务的保存点（相当于存档）

rollback to savepoint 保存点名 -- 回滚到保存点（相当于读档）

release savepoint 保存点名 -- 删除保存点
```

7，索引

> MySQL官方对索引的定义为：索引（index）是帮助MySQL高效获取数据的数据结构
>
> 提取句子主干就是：索引是一种数据结构

7.1，索引的分类

- 主键索引 PRIMARY KEY
  - 唯一的标识，主键不可重复
- 唯一索引　Unique Key
  - 避免重复的行，唯一索引可以重复
- 常规索引　Key/Index
  - 默认的，index，key关键字来设置
- 全文索引　FullText
  - 在特定的数据库引擎才有，MyISAM（现在基本都有了）
  - 快速定位数据

> 索引的使用
>
> 1，在创建表的时候给字段增加索引
>
> 2，创建完毕之后增加索引
>
> 显示所有索引的信息：
>
> show index from `表`
>
> 增加一个索引
>
> alter table 数据库.表名 add 索引类型 `索引名` (列名)
>
> 增加索引

> create index 索引名 on 表(字段)
>
> 分析sql执行的情况
>
> explain select * from 表明

7.2，测试索引

索引在小数据量的时候，用处不大，但是在大数据的时候，区别十分明显

测试过程：略

7.3，索引原则

- 索引不是越多越好
- 不要对经常变动的数据加索引
- 小数据量的表不需要加索引
- 索引一般加在常用来查询的字段上

索引的数据结构！

> Hash类型的索引
>
> Btree：InnoDB的默认数据结构～
>
> 地址：http://blog.codinglabs.org/articles/theory-of-mysql-index.html

8，数据库备份

8.1，用户管理

> SQL yog管理

略

> SQL命令操作

> 本质上就是对User表进行增删改查

```
-- 创建用户 create user 用户名 by identified 密码
create user 用户名 identified by 密码

-- 修改密码(修改当前用户的密码)
set password=password(123456)

-- 修改密码(修改指定用户的密码)
set password for 用户名=password(密码)

-- 重命名
rename user 老的用户名 to 新的用户名

-- 用户授权all privileges，全部的权限，库，表（类似于administrator的权限）
-- 但他没有给别人授权的权限（这个权限是administrator才有的）
grant all privileges on *.* to 用户名


-- 查询权限
show grant for 用户名

-- 查看管理员的权限
show grant for root@localhost


-- 撤销权限，remove 哪些权限，给谁撤销
remove all privileges on *.* from 用户名

-- 删除用户
drop user 用户名
```

8.2，MySQL备份

为什么要备份

- 保证重要的数据不丢失
- 数据转移

MySQL数据库备份的方式

- 直接拷贝物理文件
- 在Sqlyog这种可视化工具中手动导出
- 使用命令行到处mysqldump命令行使用

```
-- 导出表
mysqldump -h主机 -u用户名 -p密码 数据库 表名 > 物理磁盘位置/文件名

-- 导出多张表
mysqldump -h主机 -u用户名 -p密码 数据库 表名1 表名2 ...  > 物理磁盘位置/文件名
```

导入

```
-- 先登录
-- 然后选择指定数据库（当导入表时）
-- 然后使用source命令
source 备份文件   -- 备份文件是一个物理磁盘位置
```

9，规范数据库设计

9.1，为什么需要设计

糟糕的数据库设计：

- 数据冗余，浪费空间
- 数据库插入和删除都会麻烦，异常（例如物理外键）
- 程序的性能差

良好的数据库设计：

- 节省内存空间
- 保证数据库的完整性
- 方便我们开发系统

软件开发中，关于数据库的设计

- 分析需求：分析业务和需要处理的数据库的需求
- 概要设计：设计关系图E-R图

设计数据库的步骤（以个人博客为例）

- 收集信息，分析需求
    - 用户表
    - 分类表
    - 文章表
    - 友情链接表
    - 自定义表
- 标识实体（把需求落实到每个字段）
- 标识实体之间的关系

9.2，三大范式

为什么要数据规范化

- 信息重复
- 更新异常
- 插入异常
    - 无法正常显示信息
- 删除异常
    - 丢失有效的信息

> 三大范式

10，数据库规约，三大范式

第一范式

原子性：保证每一列不可再分

要求数据库的每一列都是不可分割的原子数据项

第二范式

前提：满足第一范式

每张表只描述一件事情

（在第一范式的基础上消除部分依赖）

数据库的每一列都必须完全依赖于主键，而不能部分依赖于主键（针对联合主键）

第三范式

前提：满足第一范式和第二范式

（在第二范式的基础上消除传递依赖）

数据库的每一列都要与主键直接依赖，而不能传递依赖

但是三大范式也存在一些问题：按照三大范式设计的数据库，连接表查询时需要连接的表有时候会很多，这样就会导致性能的下降

规范性和性能的问题:

关联和查询的表最多不能超过三张表

- 考虑商业化的需求和目标
- 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字段，减少连接的表的数量
- 故意增加一些计算列（从大数据量减少为小数据量查询：索引）

11，JDBC（重点！！！）

10.1，数据库驱动

10.2，JDBC

SUN公司为了简化开发人员（对数据库的统一）操作，提供了（Java操作数据库）规范，俗称JDBC

这些规范的实现由具体的厂商去做；

对于开发人员来说，只需学习

java.sql

javax.sql

还需要导入MysqlConnector

10.3，第一个JDBC程序

1. 创建数据库
2. 创建项目
3. 导入驱动

自定义JDBC程序

```
public vlass JdbcFirstDemo(){

```

```
    public static void main(String[] args) throws
ClassNotFoundException,SQLException{
        //1，加载驱动
        //其源码内已经包括了DriverManager.registerDriver(new
com.mysql.jdbc.Driver());
        Class.forName("com.mysql.jdbc.Driver"); //固定写法
        // 原来的写法是
        //DriverManager.registerDriver(new com.mysql.jdbc.Driver());
        //=====================================================
        //2，用户信息和url
        //mysql的写法
        //mysql -- 默认端口号：3306
        //jdbc:mysql://主机地址:端口号/数据库名?参数1&参数2&参数3
        //oracle的写法
        //oracle -- 默认端口号：1521
        //jdbc:oracle:thin:@localhost:1521:sid

        String url="jdbc:mysql://localhost:3306/jdbcstudy?
userUnicode=true&characterEncoding=utf8&useSSL=true";
        String usernae="root";
        String password="123456";

        //=====================================================
        //3，连接成功，数据库对象,Connection代表数据库
        //获取数据库驱动对象，connection因为是代表数据库，所以可以设置自动提交，事务提交，事
务回滚
        //setAutoCommit自动提交，commit提交，rollback回滚
        Connection
connection=DriverManager.getConnection(url,username,password);

        //=====================================================
        //4，执行sql的对象
        //jdbc的statement对象用于向数据库发送SQL语句，想完成对数据库的增删改查，只需要通过
这个对象向数据库发送增删改查语句即可
        //Statement:执行SQL的对象
        //statement.executeQuery();//查询操作返回Resultset
        //statement.execute();//执行任何SQL，但是有一个判断的过程所以相应的效率会低一点
        //statement.executeUpdate();//更新，插入，删除都是用这个，返回一个受影响的行数

        //PrepareStatement:执行SQL的对象
        Statement statement=connection.creatStatement();

        //=====================================================
        //5，执行sql的对象去执行sql，可能存在的结果，查看返回的结果
        String sql="Select * from users";

        //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
        //但首先要获得指定的数据类型
        //例如:resultSet.getObject();//在不知道列类型的时候用这个
        //如果列的类型就是用指定的类型
        //resultSet.getString();
        //resultSet.getInt();
        //resultSet.getFloat();
        //resultSet.getDate();
        ResultSet resultSet= statement.executeQuery(sql);

        //遍历这个结果集，用到next()//移动到下一个数据
```

```
        //其他方法：beforeFirst()移动到最前面  afterLast()移动到最后面  previous()移动到前一个数据  absolute(row)移动到指定行
        while(result.next()){
            System.out.println("id="+resultSet.getObject("id"));
            System.out.println("id="+resultSet.getObject("Name"));
            System.out.println("id="+resultSet.getObject("Password"));
            System.out.println("id="+resultSet.getObject("email"));
            System.out.println("id="+resultSet.getObject("birthday"));
            System.out.println("==================================");
        }

        //=====================================================
        //6，释放连接     注意要关闭的顺序
        //释放资源必须做！！！
        resultSet.close();
        statement.close();
        connection.close();//很耗资源

    }

}
```

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

代码实现

jdbcUtils工具类（提取自定义JDBC程序中的公共类）

```
public class JdbcUtils(){

    private static String driver=null;
    private static String url=null;
    private static String username=null;
    private static String password=null;

    static{
        InputStream in
=JdbcUtils.class.getClassLoader().getResourceAsStream("db.properties");
        Properties properties=new Properties();
        properties.load(in);

        driver=properties.getProperties("driver");
        url=properties.getProperties("url");
        username=properties.getProperties("username");
        password=properties.getProperties("password");

        //1，驱动只用加载一次
        Class.forName(driver);

    }
```

```
    //获取连接
    public static void getConnection(){
        Connection
connection=DriverManager.getConnection(url,username,password);

    }

    //释放连接资源
    public static void release(Connection conn,Statement st,ResultSet rs){
        if(conn!=null){
            conn.close();
        }if(st!=null){
            st.close();
        }if(rs!=null){
            rs.close();
        }
    }
}
```

10.4，SQL注入

SQL会被拼接

sql存在漏洞，会被攻击导致数据泄露

```
select * from user where Name = userName and Password = password;
```

java代码中

```
String sql = "select * from user where Name= '"+userName+"' And Password
'"+password+"'"
```

而在实际中，我们userName和passWord传值过程中，如果传入的值是：

userName：' or 1=1 '

password：123456

结果的sql：

```
select * from user where Name ='' or '1=1' and Password = '123456';
```

结果会把所有的数据全都输出来

10.5，PrepareStatement对象

PrepareStatement可以防止SQL注入，效率更好

PrepareStatement防止sql注入的本质：把传递进来的参数当成字符串来处理

例如：上面的例子就会变成：

```
select * from user where Name =' ''  or '1=1' ' and Password = '123456';
```

就相当于把' or 1=1 '用双引号括起来，整体变成一个字符串

```
public class TestInsert(){

    public static void main(String[] args){
        Connection conn=null;
        PrepareStatement st=null;

        conn=JdbcUtils.getConnection();

        //区别：
        //使用?占位符
        String sql= "Insert into user('id','name') values(?,?)";

        //PrepareStatement防止sql注入的本质：把传递进来的参数当成字符串来处理
        //假设其中存在转义字符（例如引号），就会被直接转义了
        st=conn.prepareStatement(sql);//预编译sql，先写sql，然后不执行

        //手动给参数赋值
        st.setInt(1,4);//两个参数：第一个参数：参数的位置；第二个参数：参数的值
        st.setString(2,"123456");

        //注意点：st.setDate中，第二个参数的值，是数据库里面的时间(sql.Date)，而不是java
中的时间(util.Date)
        //所以应该写成：new java.sql.Date(new Date().getTime());

        //执行语句
        st.executeUpdate();
    }
}
```

10.6，事务

要么都成功，要么都失败

ACID原则

原子性：要么都完成，要么都不完成

一致性：总数不变

隔离性：多个进程互不干扰

持久性：一旦提交不可逆，持久化到数据库了

隔离性的问题：

脏读：一个事务读取了另一个没有提交的是事务

不可重复读：在同一个事务内，重复读取表中的数据，表的数据发生了改变

虚读（幻读）：在一个事务内，读取到了别人插入的数据，导致前后读出来的结果不一致

（不可重复读和虚读的区别推测是在于是不可重复读是两个人读的，虚读是自己读两次）

10.7，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

那么问题来了：到底要准备多少预先的资源

用常用连接数来设置最小连接数

常用连接数：10个

最小连接数：10个

最大连接数：100个 业务最高承载上限

排队等待

等待超时：100ms，超过这个时间就不会让他们继续等待

> 开放数据源实现

DBCP

C3P0

Druid:阿里巴巴的

是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

DBCP连接池

```
public class JdbcUtils(){

    private DataSource dataSource;

    static{
        InputStream in
=JdbcUtils.class.getClassLoader().getResourceAsStream("db.properties");
```

```
        Properties properties=new Properties();
        properties.load(in);

        //创建数据源 工厂模式--》创建
        dataSource=BasicDataSourceFactory.createDataSource(properties);

        //1，驱动只用加载一次
        Class.forName(driver);

    }

    //获取连接
    public static void getConnection(){
        Connection
connection=DriverManager.getConnection(url,username,password);

    }

    //释放连接资源
    public static void release(Connection conn,Statement st,ResultSet rs){
        if(conn!=null){
            conn.close();
        }if(st!=null){
            st.close();
        }if(rs!=null){
            rs.close();
        }
    }
}
```

C3P0连接池

```
public class JdbcUtils(){

    private static ComboPooledDataSource dataSource=null;
    static{
        //代码配置
        /*
        dataSource=new ComboPooledDataSource();
        dataSource.setDriverClass();
        dataSource.setUser();
        dataSource.setPasswor();
        dataSource.setJdbcUrl();


        dataSource.setMaxPoolSize();
        dataSource.setMinPoolSize();
        */

        //配置文件写法
        dataSource=new ComboPooledDataSource("MySQL")

    }
```

```
    //获取连接
    public static void getConnection(){
        Connection
connection=DriverManager.getConnection(url,username,password);

    }

    //释放连接资源
    public static void release(Connection conn,Statement st,ResultSet rs){
        if(conn!=null){
            conn.close();
        }if(st!=null){
            st.close();
        }if(rs!=null){
            rs.close();
        }
    }
}
```

> 结论：无论是用什么数据源，本质都还是一样的，DataSource接口不会变，方法就不会变

我们讲的属于业务级别的MySQL学习

运维MySQL学习涉及到底层