MySQL基础知识

1.初识数据库

数据库分类

关系型数据库： （SQL）

- MySQL，Oracle，Sql Server，DB2，SQLite
- 通过表和表之间，行和列之间的关系进行数据的存储

非关系型数据库： （No SQL）

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

管理员命令:

> sc delete mysql 清空服务，mysql安装出错时使用

一些数据库命令

```
mysql -u root -p    --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
show databases; --查看所有的数据库

use ''      --切换数据库命令 use 数据库名
Database changed

show tables;    --查看数据库中所有的表
describe student;   --显示数据库中某个表的所有信息

create database xxxx;   --创建一个数据库表

exit    --退出连接
```

2.操作数据库

> 操作数据库->操作数据库中的表->操作表中的数据

2.1对数据库进行简单的操作

1.创建数据库

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

2.删除数据库

```
DROP DATABASE IF EXISTS westos
```

3.使用数据库

```
use `school`
```

4.从表中选择某个属性

```
--如果你的表名或者字段名是一个特殊字符，需要带``[Tab键上面]
SELECT id FROM student
```

5.查看所所有数据库

```
SHOW DATABASES
```

2.2数据库的列类型

> 数值

- tinyint 十分小的数据 1个字节
- samllint 较小的数据 2个字节
- mediumint 中等大小的数据 3个字节
  int 标准整数 4个字节 常用类型
- bigint 较大的数据 8个字节
- float 浮点数 8个字节
- double 浮点数 16个字节（有精度问题）
- decimal 字符串形式的浮点数 适用于金融计算问题

> 字符串

- char 字符串固定大小 0-255
- varchar **可变字符串 0-65535 常用的String**
- tinytext 微型文本 2^8-1
- text 文本串 2^{16}-1 保存大文本

> 时间日期

- 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（重点）

`Unsigned`：

- 无符号整数
- 表示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
- 通常用来设计唯一的主键-index
- 可以自定义设计主键的起始值和步长

`非空`：

- 假设设置为not null ,如果不给它赋值，会报错！a
- 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 xx，如果不指定该列的值，则会有默认值

2.4使用SQL建表（重点）

```
-- 目标：创建一个School数据库
-- 创建学生表(列，字段) 使用SQL创建
-- 学号int 登陆密码varchar(20) 姓名 性别varchar，出生日期datetime，家庭住址,emial
--
-- 注意点：使用英文括号()，表的名称和字段尽量使用``
-- AUTO INCREMENT 自增
-- 字符串使用''括起来
-- 所有的语句后面加 ，　最后一个语句不用加
```

```
CREATE TABLE IF NOT EXISTS `student`(

    `id` INT(4) NOT NULL COMMENT '学号',
    `name` VARCHAR(20) NOT NULL COMMENT '学生姓名',
    `pwd` VARCHAR(20) NOT NULL DEFAULT '123456' COMMENT '密码',
    `sex` VARCHAR(2) NOT NULL DEFAULT '男' COMMENT '性别',
    `birthday` DATETIME DEFAULT NULL COMMENT '出生日期',
    `adress` 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` 字段类型 [属性] [索引] [注释],
    `字段名` 字段类型 [属性] [索引] [注释],
    `字段名` 字段类型 [属性] [索引] [注释],
    ...
   `字段名` 字段类型 [属性] [索引] [注释]
)ENGING=INNBOD DEFAULT CHARSET=utf8
```

SQL查看命令

```
SHOW CREATE DATABASE school --查看创建数据库的语句
SHOW CREATE TABLE student --查看创建数据库表的语句
DESC student --显示表的结构
```

2.5修改删除表

> 修改

```
-- 修改表名，ALTER TABLE 旧表名 RENAME AS 新表名
ALTER TABLE teacher RENAME AS teacher1

--增加表的字段 ALTER TABLE 表名 ADD 字段名 列属性
ALTER TABLE teacher1 ADD age INT(3)

-- 修改表的字段(重命名，修改约束)
-- ALTER TABLE  表名 MODIFY 字段名 列属性[]
ALTER TABLE teacher1 MODIFY age varchar(11) --修改列属性

--ALTER TABLE 表名 CHANGE 旧名字 新名字 列属性[]
ALTER TABLE teacher1 CHANGE age age1 INT(1) --修改字段名和属性

--modify和change区别：
change可以修改列名称，modify不能修改列名称

--删除表的字段
ALTER TABLE  teacher1 DROP age1
```

3.MySQL数据管理

3.1外键（了解）

外键的优点

一、数据一致性

由数据库自身保证数据一致性、完整性会更可靠，程序很难100%保证数据的一致性、完整性

二、ER图可靠性

有主外键的数据库设计可以增加ER图的可读性

外键的缺点

一、级联问题

阿里巴巴的开发手册中，就曾指出强制要求不允许使用外键，一切外键概念必须在应用层解决。因为每次级联delete或update的时候，都要级联操作相关的外键表，不论有没有这个必要，由其在高并发的场景下，这会导致性能瓶颈

二、增加数据库压力

外键等于把数据的一致性事务实现，全部交给数据库服务器完成，并且有了外键，当做一些涉及外键字段的增，删，更新操作之后，需要触发相关操作去检查，而不得不消耗资源

三、死锁问题

若是高并发大流量事务场景，使用外键还可能容易造成死锁

四、开发不方便

有外键时，无论开发还是维护，需要手工维护数据时，都不太方便，要考虑级联因素

总结

一、如是单机且低并发，也不需要性能调优，再或者不能用程序保证数据的一致性，完整性，可以使用外键。
二、**如果为了高并发，分布式，使系统性能更优，以及更好维护，则一定不能使用外键**

3.2DML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

添加

> 语法：insert into 表名(字段名1,字段名2,...) values ('值1','值2'),('值3','值4')...

```
-- 插入语句
INSERT INTO `grade`(`gradeName`)VALUES('大一')

--插入多个字段
```

```
INSERT INTO grade(`gradeName`) VALUES ('大二'),('大三')

--插入时需要一一对应
insert into `student` valuse ('大四') --报错

insert into `student`(`name`) VALUES ('张三')

--插入多条数据
INSERT INTO `student`(`id`,`name`,`pwd`)VALUES('1000','李四','123456'),('1001','万五','abcdefg')

--可以不写字段名，但是值必须都填写，并且一一对应
INSERT INTO `student` VALUES (1005,'老六','111111','男','2020-11-12','江西','email')
```

注意：

修改

> update语法: update 需要修改的表名 set 原来的值=新值 where 条件判断语句
>
> update修改表数据，alter修改表的结构

```
-- 修改学员名字，有条件
UPDATE `student` SET `name`='熊环环' where id=1

--修改学员名字，无条件,此时表中所有name都改变
UPDATE `student` SET `name`='熊环环'

--可以同时修改多个属性，使用英文逗号隔开
UPDATE `student` SET `name`='张三',`pwd`='xhh19990210' WHERE id=1001

--根据多个条件修改定位数据
UPDATE `student` SET `name`='小混混',`sex`='男' WHERE `name`='熊环环' AND `sex`='女'

-- values可以是变量
UPDATE `student` SET `birthday`='CURRENTTIME' WHERE id=1001
```

删除

> delete命令 语法： delete from 表名

```
--删除数据(不推荐写法),这样会删除整个表
DELETE FROM `student`

--推荐写法
DELETE FROM `student` where id=1;
```

> truncate命令 语法：TRUNCATE 表名

```
--清空student表
TRUNCATE `student`
```

delete和truncat的区别：

- 相同点：都能删除数据，都不会删除表结构
- 不同点：
  - TRUNCATE 重新设置 自增列 计数器会清零
  - TRUNCATE 不会影响事务

```
-- 新建表
CREATE TABLE `test` (
  `id` int(4) NOT NULL AUTO_INCREMENT,
  `name` varchar(20) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB  DEFAULT CHARSET=utf8


INSERT INTO `test`(`name`) VALUES ('张三'),('李四'),('王五')

DELETE FROM `test`  --不会影响自增

TRUNCATE TABLE `test`  --自增归零
```

了解:DELETE删除的问题，重启数据库现象

- InnoDB 自增列会从1开始(存在内存当中，断电丢失)
- MyISAM 继续上一个自增量开始(存在文件中，不会丢失)

4.DQL查询数据（最重点）

4.1DQL

(Date Query Language:数据查询语言)

- 所有的查询操作都用它 Select
- 简单的查询，复杂的查询都能进行
- 数据库中最核心的语言，最重要的语句
- 使用频率最高的语句

> select语法

```
select ... from ...
left/right/inner join          -- 联合查询
on                -- 等值判断
where               -- 指定结果需要满足的条件
group by                      -- 指定结果按照那几个字段来分组
having                         -- 过滤分组的记录必须满足的条件
order by                       -- 指定查询记录按一个或者多个条件查询
limit                          -- 指定查询记录从哪条到哪条
```

4.2指定查询字段

> 语法：SELECT 字段1，字段2，... FROM 表
>
> 起别名：字段名 AS 别名 表名 AS 别名

```
-- 查询所有学生 SELECT 字段 FROM 表
SELECT * FROM `student`

-- 查询指定字段
SELECT `studentno`,`studentname` FROM `student`
-- 别名，给结果起一个名字 AS

SELECT `studentno` AS 学生学号,`studentname` AS 学生姓名 FROM `student`

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

> 去重 distinct

作用：去除SELECT查询出来的结果中重复的数据，重复的数据显示一条

```
-- 查询一下有哪些同学进行了考试，成绩
SELECT * FROM `result`
-- 查询同学
SELECT `studentno` FROM `result`
-- 查询结果只显示一条重复数据
SELECT DISTINCT `studentno` from `result`
```

4.3where条件字句

> 检索数据中符合条件的值

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

> 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,a2,a3,...中，则结果为真

```
-- 模糊查询
-- -- 查询姓刘的同学 like结合 %(代表0-任意个字符) _（一个字符）
SELECT `studentno`,`studentname` FROM `student`
WHERE `studentname` LIKE '张%'
--
-- 查询名字后面只有一个字
SELECT `studentno`,`studentname` FROM `student`
WHERE `studentname` LIKE '张_'
--
-- 查询名字后面两个字
SELECT `studentno`,`studentname` FROM `student`
WHERE `studentname` LIKE '张__'
--
-- 查询名字中带某个字的同学
SELECT `studentno`,`studentname` FROM `student`
WHERE `studentname` LIKE '%张%'
--
--IN(里面必须是具体的值)
-- 查询1000，1001，1002号同学
SELECT `studentno`,`studentname` FROM `student`
WHERE `studentno` IN(1000,1001,1002)

-- 查询地址在北京的同学
-- SELECT `studentno`,`studentname` FROM `student`
-- WHERE `address` IN('北京')                                    查询失败

SELECT `studentno`,`studentname` FROM `student`
WHERE `address` LIKE '%北京%'
```

```
-- 查询地址为空的同学
SELECT `studentno`,`studentname` FROM `student`
WHERE `address`='' OR `address` IS NULL
```

4.4联表查询

> on和where的选择条件的区别：
>
> on后面的是连接条件， 代表两个表建立关系所遵循的规则
>
> where后面的可以看作是筛选条件，是对最终结果集进行过滤所遵循的规则

```
-- SELECT * FROM student
-- SELECT * FROM `result`
-- 联表查询
/*思路：
1.分析需求，分析查询的字段来自哪些表
2.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
*/
SELECT s.`studentno`,`studentname`,`subjectno`,`studentresult`
FROM student AS s
INNER JOIN result AS r
WHERE s.`studentno` = r.`studentno`
-- RIGHT JOIN方式
SELECT s.`studentno`,`studentname`,`subjectno`,`studentresult`
FROM student AS s
RIGHT JOIN result AS r
ON s.`studentno` = r.`studentno`
-- LEFT JOIN 方式
SELECT s.`studentno`,`studentname`,`subjectno`,`studentresult`
FROM student AS s
LEFT JOIN result AS r
ON s.`studentno` = r.`studentno`
```

操作	描述
inner join	如果表中至少有一个匹配，就返回行
left join	会从左表中返回所有的值，即使右表中没有匹配
right join	会从右表中返回所有的值，即使左表中没有匹配

```
-- 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）三表查询
SELECT s.`studentno`,`studentname`,`subjectname`,`studentresult`
FROM student AS s
   RIGHT JOIN result AS
on r.`studentno`=s.`studentno`
INNER JOIN `subject` AS sub
on r.subjectno = sub.subjectno
```

4.5分页和排序

> 排序

```
-- 升序 ASC 和降序 DESC
SELECT s.`studentno`,`studentname`,`subjectname`,`studentresult`
FROM `student` AS s
INNER JOIN `result` AS r
ON s.`studentno` = r.`studentno`
INNER JOIN `subject` AS sub
ON r.`subjectno` = sub.`subjectno`
where
ORDER BY `studentresult` DESC
limit
```

> 分页
>
> limit 语法： limit 起始值下标， 页面大小

```
-- 第二页 LIMIT 5,5
-- 第三页 LIMIT 10,5
-- 第n页 LIMIT (n-1)*pagesize,pagesize
```

4.6子查询

```
-- 1.查询 数据库结构-1的所有考试结果（学号，科目编号，成绩），降序排列
-- 方式1：使用连接查询
SELECT `studentno`,sub.`subjectno`,`studentresult`
FROM `subject` AS sub
INNER JOIN `result` AS r
ON sub.`subjectno` = r.`subjectno`
WHERE `subjectname` ='数据库结构-1'
ORDER BY studentresult DESC
-- 方式二：使用子查询（由里及外）
SELECT `studentno`,`subjectno`,`studentresult`
FROM `result`
WHERE subjectno=(
   SELECT `subjectno`  FROM `subject` WHERE subjectname='数据库结构-1'
)
ORDER BY studentresult DESC

-- 1.查询 数据库结构-1的所有考试结果（学号，科目编号，成绩），降序排列

SELECT `studentno`,`subjectno`,`studentresult`
FROM `result`
WHERE subjectno=(
   SELECT `subjectno`  FROM `subject` WHERE subjectname='数据库结构-1'
)

-- 2.查询学习了高等数学-2的同学学号和姓名，并且按照成绩高低排列，只显示80分以上人员
```

```
SELECT DISTINCT s.`studentno`,`studentname`
FROM `student` AS s
INNER JOIN `result` AS r
ON s.`studentno` = r.`studentno`
WHERE `studentresult` >= 80 AND `subjectno`=(
   SELECT `subjectno` FROM `subject` WHERE `subjectname`='高等数学-2'
)
ORDER BY studentresult DESC

-- 嵌套
SELECT `studentno`,`studentname` FROM `student` WHERE `studentno` IN (
   SELECT `studentno` FROM `result` WHERE studentresult>=80 AND `subjectno` = (
      SELECT `subjectno` FROM `subject` WHERE `subjectname` = '高等数学-2'
   )
)
```

5.MySQL函数

5.1聚合函数

```
-- 聚合函数
-- 计数COUNT
SELECT COUNT(studentname) FROM student   -- count(字段)，会忽略所有的null值
SELECT COUNT(*) FROM student -- count(*)，不会忽略null值，本质上计算行数
SELECT COUNT(1) FROM student -- count(1)，不会忽略null值，本质上计算行数

-- 查询不同课程的平均分，最高分，最低分
-- 根据不同课程分组

SELECT `subjectname`,AVG(studentresult) AS '平均分',MAX(studentresult) AS '最高
分',MIN(studentresult) AS '最低分'
FROM `subject` AS sub
INNER JOIN `result` AS r
ON sub.`subjectno` = r.`subjectno`
GROUP BY r.subjectno  -- 通过字段分组
HAVING AVG(studentresult)>80
```

6.事务

6.1什么是事务

将一组SQL放在一个批次中去执行，要么都成功，要么都失败

> 事务原则：ACID 原则 原子性，一致性，隔离性，持久性（脏读，幻读）

> 原子性：指事务是一个不可分割的工作单位，事务中的操作要么都发生，要么都不发生（要么都成功，要么都失败）
>
> 一致性：事务前后数据的完整性必须保持一致
>
> 隔离性：针对多个用户同时操作，主要是排除其他事务对本次事务的影响
>
> 持久性：事务结束后的数据不会随着外界原因导致数据丢失

6.2事务的隔离级别

> 事务并发可能出现的情况：

脏读：指一个事务读取了另外一个事务未提交的数据。

不可重复读：在一个事务内读取表中的某一行数据，多次读取结果不同。（不一定是错误，只是某些场合不对）

虚读/幻读：是指在一个事务内读取到了别的事务插入的数据，导致前后读取不一致。

> 4种事务的隔离级别：

读未提交：在读未提交隔离级别下，事务A可以读取到事务B修改过但未提交的数据。（可能造成脏读，不可重复读，幻读）

读已提交：在读已提交隔离级别下，事务B只能在事务A修改过并且已提交后才能读取到事务B修改的数据。（可能造成不可重复读，幻读）

可重复读（默认隔离级别）：在可重复读隔离级别下，事务B只能在事务A修改过数据并提交后，自己也提交事务后，才能读取到事务B修改的数据。（可能造成幻读问题）

可串化行：各种问题（脏读、不可重复读、幻读）都不会发生，通过加锁实现（读锁和写锁）。

> 执行事务

```
-- mysql 默认开启事务提交
SET autocommit = 0  -- 关闭
SET autocommit = 1 -- 开启

-- 手动处理事务
SET autocommit = 0 -- 关闭自动提交

-- 事务开启
START TRANSACTION -- 标记一个事务的开始，从这里开始之后的sql都在同一个事务中
1
-- 提交:持久化成功
COMMIT
-- 回滚：回到提交之前的状态
```

```
ROLLBACK
-- 事务结束

SET autocommit = 1 -- 开启自动提交
-- 了解
SAVEPOINT 保存点名 -- 设置一个事务的保存点
ROLLBACK TO SAVEPOINT 保存点名 -- 回滚到保存点
RELEASE SAVEPOINT 保存点名 -- 撤销保存点
```

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**

7.1索引的分类

- 主键索引（PRIMAR KEY）
  - 唯一的标识，主键不可重复，只能有一个列为主键
- 唯一索引（UNIQUE KEY）
  - 避免重复的列出现，唯一索引可以重复，多个列 都可以标识
- 常规索引（KEY/INDEX）
  - 默认的，index。key关键字来设置
- 全文索引（FULLText）
  - 在特定的数据库中引擎下
  - 快速定位数据

8.备份

使用命令行导出 mysqldump命令行使用

```
# mysqldump -h 主机 -u 用户名 -p密码  数据库> 物理磁盘位置/文件名
mysqdump -hlocalhost -uroot -p123456 school >d;/c.sql
```

9.规范数据库设计

9.1三大范式

> 三大范式

第一范式（1NF）： 要求数据库表的每一列都是不可分割的原子数据项

第二范式（2NF）： 在第一范式的基础上，第二范式需要确保数据库表中的每一列都和主键相关，而不能只与主键的某一部分相关

第三范式（3NF）： 在第二范式基础上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

10.JDBC

10.1第一个JDBC程序

> 创建数据库表

```
CREATE DATABASE jdbcStudy CHARACTER SET utf8 COLLATE utf8_general_ci;

USE jdbcStudy;

CREATE TABLE `users`(
    id INT PRIMARY KEY,
    NAME VARCHAR(40),
    PASSWORD VARCHAR(40),
    email VARCHAR(60),
    birthday DATE
);

INSERT INTO `users`(id,NAME,PASSWORD,email,birthday)
VALUES(1,'zhansan','123456','zs@sina.com','1980-12-04'),
(2,'lisi','123456','lisi@sina.com','1981-12-04'),
(3,'wangwu','123456','wangwu@sina.com','1979-12-04')
```

> 编写测试代码

```
import java.sql.*;

/**
 * @author xhh
 * @date 2020/11/17 0:58
 */
// 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");

        //2.用户信息和url
```

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


        //3.连接成功，数据库对象
        Connection connection = DriverManager.getConnection(url, username, password);


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


        //5.执行SQL对象，执行SQL，返回结果
        String sql = "select * from users";
        ResultSet resultSet = statement.executeQuery(sql);//返回的结果集，结果集中封装了查询出来的结
果
        while (resultSet.next()){
            System.out.println("id="+resultSet.getObject("id"));
            System.out.println("name="+resultSet.getObject("NAME"));
            System.out.println("pwd="+resultSet.getObject("PASSWORD"));
            System.out.println("email="+resultSet.getObject("email"));
            System.out.println("birth="+resultSet.getObject("birthday"));
        }
        //6.释放连接
        resultSet.close();
        statement.close();
        connection.close();
    }
}
```

> 加载驱动

```
Class.forName("com.mysql.jdbc,Driver");//固定写法，加载驱动
```

> url

```
String url = "jdbc:mysql://localhost:3306/jdbcstudy?
useUnicode=true&character=utf8&useSSL=true";
```