

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

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 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 2^8 - 1
- text 文本串 2^16 - 1 保存大文本

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（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!

```
/* 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!
1 id 主键
2 `version` 乐观锁
3 is_delete 伪删除
4 gmt_create 创建时间
5 gmt_update 修改时间
6 */
-- 注意点, 使用英文(), 表的名称 和 字段 尽量使用 `` 括起来
-- AUTO_INCREMENT 自增
-- 字符串使用 单引号括起来!
-- 所有的语句后面加 , (英文的), 最后一个不用加
-- PRIMARY KEY 主键, 一般一个表只有一个唯一的主键!
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

```
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]
```

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
```

```
3 INNODB 默认使用~  
4 MYISAM 早些年使用的  
5 */
```

	MYISAM	INNODB
事务支持	不支持	支持
数据行锁定	不支持	支持
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 2 倍

常规使用操作:

- MYISAM 节约空间, 速度较快
- INNODB 安全性高, 事务的处理, 多表多用户操作

修改

```
1 -- 修改表名 : ALTER TABLE 旧表名 RENAME AS 新表名  
2 ALTER TABLE teacher RENAME AS teacher1  
3 -- 增加表的字段 : ALTER TABLE 表名 ADD 字段名 列属性  
4 ALTER TABLE teacher1 ADD age INT(11)  
5  
6 -- 修改表的字段 (重命名, 修改约束!)  
7 -- ALTER TABLE 表名 MODIFY 字段名 列属性[]  
8 ALTER TABLE teacher1 MODIFY age VARCHAR(11) -- 修改约束  
9 -- ALTER TABLE 表名 CHANGE 旧名字 新名字 列属性[]  
10 ALTER TABLE teacher1 CHANGE age age1 INT(1) -- 字段重命名  
11  
12  
13 -- 删除表的字段: ALTER TABLE 表名 DROP 字段名  
14 ALTER TABLE teacher1 DROP age1
```

I

删除

```
1 -- 删除表 (如果表存在再删除)  
2 DROP TABLE IF EXISTS teacher1
```

==所有的创建和删除操作尽量加上判断, 以免报错==

3.1、外键（了解即可）

方式一、在创建表的时候，增加约束（麻烦，比较复杂）

```
1 | CREATE TABLE `grade` (
2 |   `gradeid` INT(10) NOT NULL AUTO_INCREMENT COMMENT '年级id',
3 |   `gradename` VARCHAR(50) NOT NULL COMMENT '年级名称',
4 |   PRIMARY KEY (`gradeid`)
5 | )ENGINE=INNODB DEFAULT CHARSET=utf8
6 |
7 |
8 | -- 学生表的 gradeid 字段 要去引用年级表的 gradeid
9 | -- 定义外键key
10 | -- 给这个外键添加约束（执行引用） references 引用
11 |
12 | CREATE TABLE IF NOT EXISTS `student` (
13 |   `id` INT(4) NOT NULL AUTO_INCREMENT COMMENT '学号',
14 |   `name` VARCHAR(30) NOT NULL DEFAULT '匿名' COMMENT '姓名',
15 |   `pwd` VARCHAR(20)NOT NULL DEFAULT '123456' COMMENT '密码',
16 |   `sex` VARCHAR(2) NOT NULL DEFAULT '女' COMMENT '性别',
17 |   `birthday` DATETIME DEFAULT NULL COMMENT '出生日期',
18 |   `gradeid` INT(10) NOT NULL COMMENT '学生的年级',
19 |   `address` VARCHAR(100) DEFAULT NULL COMMENT '家庭住址',
20 |   `email` VARCHAR(50) DEFAULT NULL COMMENT '邮箱',
21 |   PRIMARY KEY(`id`),
22 |   KEY `FK_gradeid` (`gradeid`),
23 |   CONSTRAINT `FK_gradeid` FOREIGN KEY (`gradeid`) REFERENCES `grade`(`gradeid`)
24 | )ENGINE=INNODB DEFAULT CHARSET=utf8
```

方式二：创建表成功后，添加外键约束

```
1 | CREATE TABLE `grade` (
2 |   `gradeid` INT(10) NOT NULL AUTO_INCREMENT COMMENT '年级id',
3 |   `gradename` VARCHAR(50) NOT NULL COMMENT '年级名称',
4 |   PRIMARY KEY (`gradeid`)
5 | )ENGINE=INNODB DEFAULT CHARSET=utf8
6 |
7 |
8 | -- 学生表的 gradeid 字段 要去引用年级表的 gradeid
9 | -- 定义外键key
10 | -- 给这个外键添加约束（执行引用） references 引用
11 | CREATE TABLE IF NOT EXISTS `student` (
12 |   `id` INT(4) NOT NULL AUTO_INCREMENT COMMENT '学号',
13 |   `name` VARCHAR(30) NOT NULL DEFAULT '匿名' COMMENT '姓名',
14 |   `pwd` VARCHAR(20)NOT NULL DEFAULT '123456' COMMENT '密码',
15 |   `sex` VARCHAR(2) NOT NULL DEFAULT '女' COMMENT '性别',
16 |   `birthday` DATETIME DEFAULT NULL COMMENT '出生日期',
17 |   `gradeid` INT(10) NOT NULL COMMENT '学生的年级',
18 |   `address` VARCHAR(100) DEFAULT NULL COMMENT '家庭住址',
19 |   `email` VARCHAR(50) DEFAULT NULL COMMENT '邮箱',
20 |   PRIMARY KEY(`id`)
21 | )ENGINE=INNODB DEFAULT CHARSET=utf8
22 |
23 | -- 创建表的时候没有外键关系
24 | ALTER TABLE `student`
25 | ADD CONSTRAINT `FK_gradeid` FOREIGN KEY(`gradeid`) REFERENCES `grade`(`gradeid`);
26 |
27 | -- ALTER TABLE 表 ADD CONSTRAINT 约束名 FOREIGN KEY(作为外键的列) REFERENCES 那个表(哪个字段)
```

insert

```
1 -- 插入语句（添加）
2 -- insert into 表名([字段名1,字段2,字段3])values('值1'),('值2'),('值3', ....)
3 INSERT INTO `grade`(`gradename`) VALUES('大四')
4
5 -- 由于主键自增我们可以省略（如果不写表的字段，他就会一一匹配）
6 INSERT INTO `grade` VALUES('大三')
7
8 -- 一般写插入语句，我们一定要数据和字段一一对应！
9
10 -- 插入多个字段
11 INSERT INTO `grade`(`gradename`)
12 VALUES('大二'),('大一')
13
14
15
16 INSERT INTO `student`(`name`) VALUES ('张三')
17
18
19 INSERT INTO `student`(`name`, `pwd`, `sex`) VALUES ('张三','aaaaaa','男')
20
21 INSERT INTO `student`(`name`, `pwd`, `sex`)
22 VALUES ('李四','aaaaaa','男'),('王五','aaaaaa','男')
```

3.4、修改

update 修改谁 (条件) set 原来的值 = 新值

```
1 -- 修改学员名字，带了简介
2 UPDATE `student` SET `name`='狂神' WHERE id = 1;
3
4 -- 不指定条件的情况下，会改动所有表！
5 UPDATE `student` SET `name`='长江7号'
6
7 -- 修改多个属性，逗号隔开
8 UPDATE `student` SET `name`='狂神', `email`='24736743@qq.com' WHERE id = 1;
9
10 -- 语法：
11 -- UPDATE 表名 set column_name = value,[column_name = value,...] where [条件]
```

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

3.5、删除

delete 命令

语法: **delete from 表名 [where 条件]**

```
1 -- 删除数据 (避免这样写, 会全部删除)
2 DELETE FROM `student`
3
4 -- 删除指定数据
5 DELETE FROM `student` WHERE id = 1;
```

TRUNCATE 命令

作用: 完全清空一个数据库表, 表的结构和索引约束不会变!

```
1 -- 清空 student 表
2 TRUNCATE `student`
```

delete 的 TRUNCATE 区别

- 相同点: 都能删除数据, 都不会删除表结构
- 不同:
 - TRUNCATE 重新设置自增列计数器会归零
 - TRUNCATE 不会影响事务

```
1 -- 测试delete 和 TRUNCATE 区别
2 CREATE TABLE `test`(
3   `id` INT(4) NOT NULL AUTO_INCREMENT,
4   `coll` VARCHAR(20) NOT NULL,
5   PRIMARY KEY (`id`)
6 )ENGINE=INNODB DEFAULT CHARSET=utf8
7
8
9 INSERT INTO `test`(`coll`) VALUES('1'),('2'),('3')
10
11 DELETE FROM `test` -- 不会影响自增
12
13 TRUNCATE TABLE `test` -- 自增会归零
```

```
1 -- 查询全部的学生    SELECT 字段 FROM 表
2 SELECT * FROM student
3
4 -- 查询指定字段
5 SELECT `studentNo`, `StudentName` FROM student
6
7 -- 别名, 给结果起一个名字 AS 可以给字段起别名, 也可以给表起别名
8 SELECT `StudentNo` AS 学号, `StudentName` AS 学生姓名 FROM student AS s
9
10 -- 函数 Concat(a, b)
11 SELECT CONCAT('姓名: ', StudentName) AS 新名字 FROM student
```

Select完整的语法:

```
1 SELECT
2     [ALL | DISTINCT | DISTINCTROW ]
3     [HIGH_PRIORITY]
4     [STRAIGHT_JOIN]
5     [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
6     [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
7     select_expr [, select_expr ...]
8     [FROM table_references
9         [PARTITION partition_list]
10        [WHERE where_condition]
11        [GROUP BY {col_name | expr | position}
12            [ASC | DESC], ... [WITH ROLLUP]]
13        [HAVING where_condition]
14        [ORDER BY {col_name | expr | position}
15            [ASC | DESC], ...]
16        [LIMIT {[offset,] row_count | row_count OFFSET offset}]
17        [PROCEDURE procedure_name(argument_list)]
18        [INTO OUTFILE 'file_name'
19            [CHARACTER SET charset_name]
20            export_options
21            | INTO DUMPFILE 'file_name'
22            | INTO var_name [, var_name]]
23        [FOR UPDATE | LOCK IN SHARE MODE]]]
```

去重 distinct

作用：去除SELECT查询出来的结果中重复的数据，重复的数据只显示一条

```
1 -- 查询一下有哪些同学参加了考试，成绩
2 SELECT * FROM result -- 查询全部的考试成绩
3 SELECT `studentNo` FROM result -- 查询有哪些同学参加了考试
4 SELECT DISTINCT `studentNo` FROM result -- 发现重复数据，去重
```

数据库的列（表达式）

```
1 SELECT VERSION() -- 查询系统版本（函数）
2 SELECT 100*3-1 AS 计算结果 -- 用来计算（表达式）
3 SELECT @@auto_increment_increment -- 查询自增的步长（变量）
4
5 -- 学员考试成绩 + 1分查看
6 SELECT `studentNo`, `studentResult`+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

1.锁 2.。3.所 4.缩 5.索 <>▼

逻辑运算符

运算符	语法		描述
and &&	a and b	a&&b	逻辑与，两个都为真，结果为真
or	a or b	a b	逻辑或，其中一个为真，则结果为真
Not !	not a	! a	逻辑非，真为假，假为真！

模糊查询：比较运算符

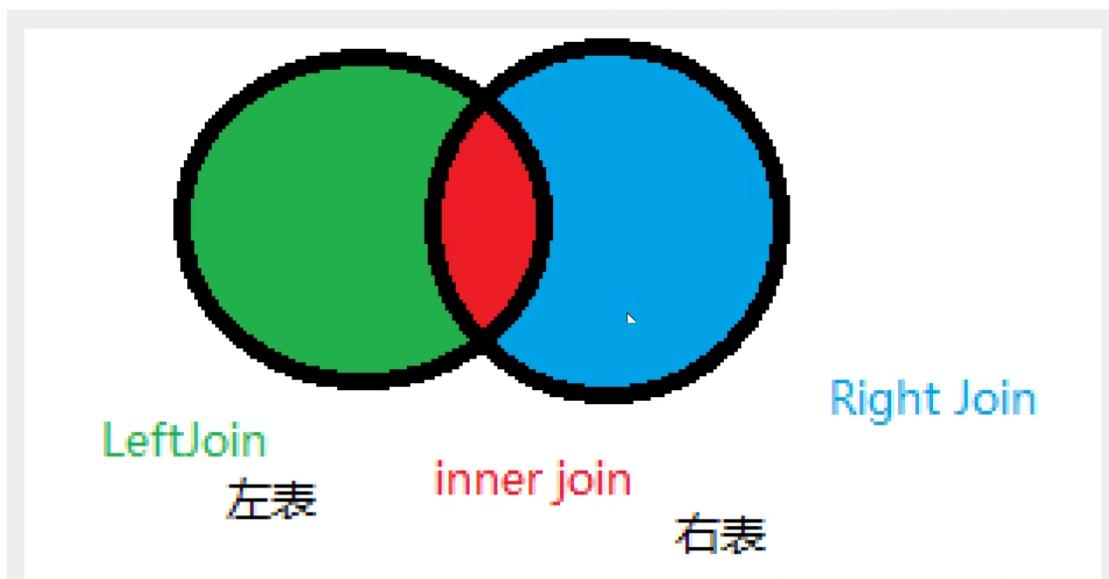
运算符	语法		描述
IS NULL	a is null		如果操作符为 Null, 结果为真
IS NOT NULL	a is not null		如果操作符不为 null, 结果为真
BETWEEN	a between b and c		若a在b和c之间，则结果为真
Like	a like b		SQL 匹配，如果a匹配b，则结果为真
In	a in (a1,a2,a3....)		假设a在a1, 或者a2.... 其中的某一个值中，结果为真

```
1 -- ===== 模糊查询 =====
2 -- 查询姓刘的同学
3 -- like结合 % (代表0到任意个字符) _ (一个字符)
4 SELECT `StudentNo`, `StudentName` FROM `student`
5 WHERE StudentName LIKE '刘%'
6
7 -- 查询姓刘的同学, 名字后面只有一个字的
8 SELECT `StudentNo`, `StudentName` FROM `student`
9 WHERE StudentName LIKE '刘_'
10
11 -- 查询姓刘的同学, 名字后面只有两个字的
12 SELECT `StudentNo`, `StudentName` FROM `student`
13 WHERE StudentName LIKE '刘__':
14
15 -- 查询名字中间有嘉字的同学 %嘉%
16 SELECT `StudentNo`, `StudentName` FROM `student`
17 WHERE StudentName LIKE '%嘉%'
18
19 -- ===== in (具体的一个或者多个值) =====
20 -- 查询 1001,1002,1003号学员
21 SELECT `StudentNo`, `StudentName` FROM `student`
22 WHERE StudentNo IN (1001,1002,1003);
23
24 -- 查询在北京的学生
25 SELECT `StudentNo`, `StudentName` FROM `student`
26 WHERE `Address` IN ('安徽','河南洛阳');
```

```
-- ===== null not null=====

-- 查询地址为空的学生 null ''
SELECT `StudentNo`, `StudentName` FROM `student`
WHERE address='' OR address IS NULL

-- 查询有出生日期的同学 不为空
SELECT `StudentNo`, `StudentName` FROM `student`
WHERE `BornDate` IS NOT NULL
-- 查询没有出生日期的同学 为空
SELECT `StudentNo`, `StudentName` FROM `student`
WHERE `BornDate` IS NULL
```



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

-- 查询父子信息：把一张表看为两个一模一样的表

```
SELECT a.`categoryName` AS '父栏目', b.`categoryName` AS '子栏目'
FROM `category` AS a, `category` AS b
WHERE a.`categoryId` = b.`pid`
```

```
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
```

排序

```
1 -- 排序 : 升序ASC , 降序DESC
2 -- ORDER BY 通过那个字段排序, 怎么排
3 -- 查询的结果根据 成绩降序 排序
4 SELECT s.`StudentNo`, `StudentName`, `SubjectName`, `StudentResult`
5 FROM student s
6 INNER JOIN `result` r
7 ON s.StudentNo = r.StudentNo
8 INNER JOIN `subject` sub
9 ON r.`SubjectNo` = sub.`SubjectNo`
10 WHERE subjectName = '数据库结构-1'
11 ORDER BY StudentResult ASC
```

```

15 INNER JOIN `subject` sub
16 ON r.`SubjectNo` = sub.`SubjectNo`
17 WHERE subjectName = '数据库结构-1'
18 ORDER BY StudentResult ASC
19 LIMIT 5,5
20
21 -- 第一页 limit 0,5      (1-1) *5
22 -- 第二页 limit 5,5      (2-1) *5
23 -- 第三页 limit 10,5     (3-1) *5
24 -- 第N页   limit 0,5      (n-1) * pagesize, pagesize
25 -- 【pagesize: 页面大小】
26 -- 【(n-1)* pagesize:起始值】
27 -- 【n : 当前页】
28 -- 【数据总数/页面大小 = 总页数】

```

语法：`**limit(查询起始下标, pagesize)**` |

4.6、子查询

where (这个值是计算出来的)

本质：在**where**语句中嵌套一个子查询语句

```

-- ===== 常用函数 =====

-- 数学运算
SELECT ABS(-8)    -- 绝对值
SELECT CEILING(9.4) -- 向上取整
SELECT FLOOR(9.4)  -- 向下取整
SELECT RAND()      -- 返回一个 0~1 之间的随机数
SELECT SIGN(10)    -- 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
SELECT CHAR_LENGTH('即使再小的帆也能远航') -- 字符串长度
SELECT CONCAT('我','爱','你们') -- 拼接字符串
SELECT INSERT('我爱编程helloworld',1,2,'超级热爱') -- 查询, 从某个位置开始替换某个长度
SELECT LOWER('KuangShen') -- 小写字母
SELECT UPPER('KuangShen') -- 大写字母
SELECT INSTR('kuangshen','h') -- 返回第一次出现的子串的索引
SELECT REPLACE('狂神说坚持就能成功','坚持','努力') -- 替换出现的指定字符串
SELECT SUBSTR('狂神说坚持就能成功',4,6) -- 返回指定的子字符串 (源字符串, 截取的位置, 截取的长度)
SELECT REVERSE('清晨我上马') -- 反转

-- 查询姓 周 的同学, 名字 邹
SELECT REPLACE(studentname,'周','邹') FROM student
WHERE studentname LIKE '周%'

-- 时间和日期函数 (记住)
SELECT CURRENT_DATE() -- 获取当前日期
SELECT CURDATE() -- 获取当前日期
SELECT NOW() -- 获取当前的时间
SELECT LOCALTIME() -- 本地时间
SELECT SYSDATE() -- 系统时间

-- 系统
SELECT YEAR(NOW())
SELECT MONTH(NOW())
SELECT DAY(NOW())
SELECT HOUR(NOW())
SELECT MINUTE(NOW())
SELECT SECOND(NOW())

```

5.2、聚合函数（常用）

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

```
-- ===== 聚合函数 =====
-- 都能够统计 表中的数据

SELECT COUNT(`BornDate`) FROM student; -- Count(字段), 会忽略所有的 null 值

SELECT COUNT(*) FROM student; -- Count (*), 不会忽略 null 值
SELECT COUNT(1) FROM result; -- Count (1), 不会忽略忽略所有的 null 值
```

4.7、分组和过滤

```
1 -- 查询不同课程的平均分, 最高分, 最低分, 平均分大于80
2 -- 核心: (根据不同的课程分组)
3 SELECT SubjectName, AVG(StudentResult) AS 平均分, MAX(StudentResult) AS 最高分, MIN(StudentResult) AS 最低分
4 FROM result r
5 INNER JOIN `subject` sub
6 ON r.`SubjectNo` = sub.`SubjectNo`
7 GROUP BY r.SubjectNo -- 通过什么字段来分组
8 HAVING 平均分>80
```

```
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')
```

6、事务

6.1、什么是事务

要么都成功，要么都失败

1、SQL 执行 A 给 B 转账 A 1000 -->200 B 200

2、SQL 执行 B 收到 A 的钱 A 800 --> B 400

将一组SQL放在一个批次中去执行~

事务原则：ACID 原则 原子性，一致性，隔离性，持久性 (脏读，幻读....)

```
5 CREATE TABLE `account`(
6     `id` INT(3) NOT NULL AUTO_INCREMENT,
7     `name` VARCHAR(30) NOT NULL,
8     `money` DECIMAL(9,2) NOT NULL,
9     PRIMARY KEY (`id`)
10 )ENGINE=INNODB DEFAULT CHARSET=utf8
11
12
13 INSERT INTO account(`name`, `money`)
14 VALUES ('A', 2000.00), ('B', 10000.00)
15
16 -- 模拟转账：事务
17 SET autocommit = 0; -- 关闭自动提交
18 START TRANSACTION -- 开启一个事务（一组事务）
19
20 UPDATE account SET money=money-500 WHERE `name` = 'A' -- A减500
21 UPDATE account SET money=money+500 WHERE `name` = 'B' -- A加500
22
23 COMMIT; -- 提交事务，就被持久化了！
24 ROLLBACK; -- 回滚
25
26 SET autocommit = 1; -- 恢复默认值
```

Java
方法(){
try(){
 正常的业务代码
 commit()
}catch(){
 rollback()
}
}

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。 0.5s 0.00001s

提取句子主干，就可以得到索引的本质：索引是数据结构。

7.1、索引的分类

在一个表中，主键索引只能有一个，唯一索引可以有多个

- 主键索引 (PRIMARY KEY)
 - 唯一的标识，主键不可重复，只能有一个列作为主键
- 唯一索引 (UNIQUE KEY)
 - 避免重复的列出现，唯一索引可以重复，多个列都可以标识位 唯一索引
- 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下才有，MyISAM
 - 快速定位数据

基础语法

```
1 -- 索引的使用
2 -- 1、在创建表的时候给字段增加索引
3 -- 2、创建完毕后，增加索引
4
5 -- 显示所有的索引信息
6 SHOW INDEX FROM student
7
8 -- 增加一个全文索引（索引名） 列名
9 ALTER TABLE school.student ADD FULLTEXT INDEX `studentName`(`studentName`);
10
11 -- EXPLAIN 分析sql执行的状况
12
13 EXPLAIN SELECT * FROM student; -- 非全文索引
14
15 EXPLAIN SELECT * FROM student WHERE MATCH(studentName) AGAINST('刘');
```

7.3、索引原则

- 索引不是越多越好
- 不要对进程变动数据加索引
- 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~

```
1 -- 创建用户 CREATE USER 用户名 IDENTIFIED BY '密码'
2 CREATE USER kuangshen IDENTIFIED BY '123456'
3
4 -- 修改密码 (修改当前用户密码)
5 SET PASSWORD = PASSWORD('123456')
6
7 -- 修改密码 (修改指定用户密码)
8 SET PASSWORD FOR kuangshen = PASSWORD('123456')
9
10 -- 重命名 RENAME USER 原来名字 TO 新的名字
11 RENAME USER kuangshen TO kuangshen2
12
13 -- 用户授权 ALL PRIVILEGES 全部的权限 , 库.表
14 -- ALL PRIVILEGES 除了给别人授权, 其他都能够干
15 GRANT ALL PRIVILEGES ON *.* TO kuangshen2
16
17 -- 查询权限
18 SHOW GRANTS FOR kuangshen2 -- 查看指定用户的权限
19 SHOW GRANTS FOR root@localhost
20
21 -- ROOT用户权限: GRANT ALL PRIVILEGES ON *.* TO 'root'@'localhost' WITH GRANT OPTION
22
23 -- 撤销权限 REVOKE 哪些权限, 在哪个库撤销, 给谁撤销
24 REVOKE ALL PRIVILEGES ON *.* FROM kuangshen2
25
26 -- 删除用户
27 DROP USER kuangshen
```

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

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！