SQL语句全集

用户管理

1. **新建用户**

```
CREATE USER  用户名 IDENTIFIED  BY '密码'
```

2. **更改密码**

```
SET PASSWORD FOR 用户名=PASSWORD(密码);
```

3. **权限管理**

```
//查看用户的权限
SHOW CRANTS FOR name;

//给予用户所有权限
GRANT SELECT ON db_name.* TO name;

//去除用户的权限
REVOKE SELECT ON db_name.* TO name;
```

数据库操作

1. **查看数据库**

```
SHOW DATABASES;
```

2. **创建数据库**

```
CREATE DATABASES 数据库名;
```

3. **使用数据库**

```
USE 数据库名;
```

4. **删除数据库**

```
DROP DATABASE 数据库名;
```

表基本操作

1. **创建表**

```
CREAT TABLE 表名( 字段名 类型 附加属性[NOT NULL
      AUTO_INCREMENT , DEFAULT, UNSIGNED]
                        ....
                    字段名 类型 附加属性,
                    PRIMARY KEY()
                )
```

2. **复制表**

```
CREAT TABLE 表名2 SELECT * FROM 表名2;
```

3. **创建临时表**

```
CREATE TEMPORARY TABLE 表名;
```

4. **查看可用表**

```
SHOW TABLES;
```

5. **查看表结构**

```
DESCRIBE 表名;

SHOW COLUMNS in 表名;
```

6. **删除表**

```
DROP [TEMPORARY] TABLE
[IF EXISTS]
表名1[,表名2...]
[CASCADE|RESTRICT];   -- 级联 强制
```

7. **表重命名**

```
RENAME TABLE 表老名字 TO 表新名字;
```

INDEX 索引

1. **分类**
   - 单列索引
   - 组合索引
2. **创建索引**

```
CREATE INDEX 索引名字 ON 表名    [ASC|DESC];  -- 普通索引
CREATE UNIQUE INDEX 索引名字 ON 表名    [ASC|DESC];  -- 唯一索引 但是可以为空
```

3. **删除索引**

```
DROP INDEX 索引名 ON 表名;
```

ALTER 表结构操作 [增加 删除 修改]字段

1. **增加列**

```
ALTER TABLE 表名 ADD COLUMN 列名 类型  属性;
```

2. **删除列**

```
ALTER TABLE 表名 DROP 列名;
```

3. **修改列**

```
ALTER TABLE CHANGE 旧列名 列名 类型 属性;
```

INSERT 插入数据操作

1. **直接插入数据**

```
INSERT INTO 表名 (列1, 列2...) VALUSE(数据1), (数据2);
```

2. **插入检索得到的数据**

```
INSERT INTO 表名 (列1, 列2...) SELECT 列1, 列2 FROM 另表名;
```

UPDATE 更新数据

1. **更新指定数据**

```
UPDATE 表名 SET 列名=值 WHERE 条件;
```

DELETE 删除数据

1. **删除指定数据**

```
DELETE FROM 表名 WHERE 条件
```

SELECT 查询数据

1. **基本形式**

```
SELECT 列名 FROM 表名 WHERE 条件;
```

2. **查询排序**

```
SELECT 列名 FROM 表名 WHERE ORDER BY 列名 [ASC升序|DESC降序];
```

3. **分组查询**

```
SELECT 列名 FROM 表名 [WHERE 条件] GROUP BY 列名  HAVEING 过滤条件;
```

4. **组合查询**

```
SELECT 列1, 列2 FROM 表1
UNION [ALL 保留重复行]
SELECT 列3, 列4 FROM 表2;
```

5. **子查询**

   - 子查询可以嵌套在主查询的所有位置
   - 分类
     - **相关子查询**
       - 依赖外部查询
       - 外查询一次 子查询一次
     - **非相关子查询**
       - 不依赖外部函数
       - 子查询总共执行一次

相关条件控制符

```
= > < <>  IN() BETWEEN x AND y    NOT AND  OR

like() % 任意字符 _ 匹配一个字符

IS NULL

EXISTS  NOT EXISTS
```

内置的函数

```
CONCAT() -- 字符串链接

AVG SUM MAX MIN COUNT([DISTINCT|ALL|*])  -- 数学函数

TRIM LOCATE UPPER LOWER SUBSTRING -- 文本处理函数

+ - * \   -- 运算符
```

SELECT 语句案例

```
/*计算班里有多少个学生*/
SELECT COUNT(*) FROM t_student

/*求班级中的女学生的数量*/
SELECT COUNT(*) FROM t_student WHERE sex="女";

/*计算班级的数量*/
SELECT COUNT(DISTINCT class) FROM t_student;

/*计算学生的年龄之和*/
SELECT SUM(age) FROM t_student;

/*统计每个班的人数*/
SELECT class, COUNT(ALL name) AS 总人数 FROM t_student GROUP BY class;

/*统计每个班20岁以上的学生数量*/
SELECT class, COUNT(ALL name) AS 总人数 FROM t_student WHERE age>20 GROUP BY class;

/*查询平均年龄大于20岁以上的班级*/
SELECT class, AVG(age) AS 平均年龄 FROM t_student GROUP BY class HAVING 平均年龄>20;

/*找出C语言成绩最高的学生信息*/
SELECT * FROM t_student WHERE subject='C语言' AND score>=ALL(SELECT score FROM
t_student WHERE subject='C语言');
```

VIEW 视图

1. **视图**
   - 虚拟表
   - 只供查询 不可修改
2. **视图优点**
   - 定制数据 【不同的用户可以用不同的视图】
   - 简化数据操作 【简化查询 不用聚合多个表等】
   - 安全性 【虚拟性 不可更改】
   - 兼容性 【业务扩张 或者业务调整】
3. **视图缺点**
   - 性能 【视图基于视图 速度慢】

- 表依赖关系 【更改表结构时候需要修改视图】

4. **创建视图**

```
CREATE VIEW 视图名  AS
SELECT 列名
FROM 表名
WHERE 条件
[GROUP BY]
[ORDER BY]
```

PROCEDURE 存储过程

1. **存储过程**
   - SQL语句集
   - 经过编译 存储到数据库中
   - 通过参数来调用
2. **存储过程优点**
   - 封装性 【隐藏商业逻辑】
   - 灵活性 【可以回传值 可以接受参数】
   - 用途性 【用于数据检验 强制实行商业逻辑】
3. **存储过程缺点**
   - 受限制于编程语言 【更换数据库系统时候需要重写原有存储过程】
   - 性能调校与编写限制与数据库系统
   - 不支持select
4. **存储过程创建**

```
DELIMITER $$  -- 自定义结束符号

CREATE
    PROCEDURE 存储过程名(
    [IN|OUT|INOUT]参数名 类型]...)
BEGIN
    SET 变量名=表达式值 -- 赋值
    SQL语句集合
END$$
```

5. **存储过程调用**

```
CALL 存储过程名(变量 @变量);
```

6. **存储过程其他操作**

```
SHOW PROCEDURE STATUS;    -- 显示当前的存储过程
DROP PROCEDURE 存储过程名;  -- 删除存储过程
```

CURSOR 游标

1. **游标**
    - 对查询数据库返回的记录进行遍历
    - 每次指向一行
2. **使用方法**
    1. 声明游标 【declare】
    2. 打开游标 【open】
    3. 获取下一行数据 【fetch】
    4. 执行语句
    5. 释放游标 【close】
3. **实例**

```
-- 定义变量
declare testrangeid BIGINT;
declare versionid BIGINT;
declare done int;
-- 创建游标，并存储数据
declare cur_test CURSOR for
   select id as testrangeid,version_id as versionid from tp_testrange;
-- 游标中的内容执行完后将done设置为1
 DECLARE CONTINUE HANDLER FOR NOT FOUND SET done=1;

-- 打开游标
open cur_test;
-- 执行循环
  posLoop:LOOP
-- 判断是否结束循环
        IF done=1 THEN
      LEAVE posLoop;
    END IF;
-- 取游标中的值
    FETCH  cur_test into testrangeid,versionid;
-- 执行更新操作
    update tp_data_execute set version_id=versionid where testrange_id =
testrangeid;
  END LOOP posLoop;
-- 释放游标
CLOSE cur_test;
```

TRIGGER 触发器

1. **触发器**
    - 语句之前或者之后
    - INSERT UPDATE DELETE三种语句
2. **触发器优点**

- 检查数据完整性
- 捕获业务逻辑错误
- 审核更改
- 提供运行计划任务的方法

3. **触发器注意点**

- 无法完成所有的验证
- 调试困难 无法弄清数据库中发生的情况
- 会增加数据库服务器的开销

4. **创建触发器**

```
DELIMITER $$  -- 自定义结束符号

CREATE TRIGGER 触发器名
[BEFORE|AFTER]           -- 触发事件
[INSERT|DELETE|UPDATE]   -- 触发事件
ON 表名
FOR EACH ROW
[FOLLOWS|PRECEDS]        -- 在所有触发器前触发 所有之后
BEGIN
     主体语句
END $$
```

5. **删除触发器**

```
DROP TRIGGER 触发器名;
```