表结构

```
CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50) NOT NULL,
    email VARCHAR(100) NOT NULL,
    age INT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

示例数据

```
INSERT INTO users (username, email, age) VALUES
('Alice', 'alice@example.com', 30),
('Bob', 'bob@example.com', 25),
('Charlie', 'charlie@example.com', 35),
('David', 'david@example.com', 40),
('Eva', 'eva@example.com', 28),
('Frank', 'frank@example.com', 22),
('Grace', 'grace@example.com', 31),
('Hannah', 'hannah@example.com', 29),
('Ian', 'ian@example.com', 33),
('Jack', 'jack@example.com', 27),
('Kathy', 'kathy@example.com', 26),
('Leo', 'leo@example.com', 34),
('Mia', 'mia@example.com', 24),
('Nina', 'nina@example.com', 36),
('Owen', 'owen@example.com', 37);
```

存储函数练习题

语法结构

```
DELIMITER //

CREATE FUNCTION function_name(parameter_list)
RETURNS return_data_type
[DETERMINISTIC | NOT DETERMINISTIC]  -- 指示函数返回值是确定性的还是不确定性的
BEGIN
    -- 函数体
    DECLARE variable_name data_type;  -- 声明变量（可选）

    -- 逻辑处理
    -- 例如，计算、条件判断、循环等

    RETURN value;  -- 返回值
END //

DELIMITER ;
```

DELIMITER:

- `DELIMITER //` 用于更改语句分隔符，防止在函数体内使用分号时提前结束函数定义。

CREATE FUNCTION：

- `CREATE FUNCTION function_name` 用于定义新的存储函数，`function_name` 是函数的名称。

parameter_list：

- 函数的参数列表，格式为 `parameter_name data_type`，可以有多个参数，用逗号分隔。

RETURNS：

- `RETURNS return_data_type` 用于指定函数返回值的数据类型。

DETERMINISTIC | NOT DETERMINISTIC：

- **DETERMINISTIC**：表示函数的返回值仅依赖于输入参数，对于相同的输入，总是返回相同的结果。可以在查询优化时使用。
- **NOT DETERMINISTIC**：表示函数的返回值可能依赖于其他因素（如当前时间、随机数等），对于相同的输入，可能返回不同的结果。默认情况下，如果不指定，函数被视为 `NOT DETERMINISTIC`。

BEGIN ... END：

- 函数的主体，包含了声明变量、逻辑处理和控制流语句等。

DECLARE：

- 用于声明局部变量（可选），可以在函数中用于存储临时数据。

RETURN：

- `RETURN value` 用于返回函数的结果。

注意:有多个返回结果的，拼接在一个字符串后返回

练习题

1. **创建用户**
   - 编写一个存储函数 `createUser`，接受 `username`、`email` 和 `age` 作为参数，向 `users` 表中插入一条新记录，并使用LAST_INSERT_ID()函数返回最近插入的主键自增值。
2. **获取用户信息**
   - 编写一个存储函数 `getUserById`，接受用户 ID 作为参数，返回该用户的所有信息，信息使用字符串拼接返回。
3. **更新用户信息**
   - 编写一个存储函数 `updateUser`，接受用户 ID、`username`、`email` 和 `age` 作为参数，更新对应用户的信息。
4. **删除用户**
   - 编写一个存储函数 `deleteUser`，接受用户 ID 作为参数，从 `users` 表中删除该用户。
5. **统计用户数量**
   - 编写一个存储函数 `countUsers`，返回 `users` 表中的用户总数。
6. **查找年龄大于指定值的用户**
   - 编写一个存储函数 `getUsersOlderThan`，接受一个年龄参数，返回所有年龄大于该参数的用户信息。

7. **获取最近创建的用户**
    - 编写一个存储函数 `getRecentUsers`，返回最近创建的 N 个用户信息。
8. **查找用户邮箱**
    - 编写一个存储函数 `findUserByEmail`，接受邮箱作为参数，返回该邮箱对应的用户信息。

循环结构练习题

```
WHILE 条件 DO
    -- 执行的语句
END WHILE;

REPEAT
    -- 执行的语句
UNTIL 条件 END REPEAT;

LOOP
    -- 执行的语句
    -- 可以使用 IF 条件来控制退出
    IF 条件 THEN
        LEAVE loop_label;  -- 退出循环
    END IF;
END LOOP;
```

LOOP：适合需要在循环中使用 `LEAVE` 语句来控制退出的情况。

WHILE：在条件为真时循环执行，适合需要先判断条件再执行的情况。

REPEAT：先执行一次循环体，再判断条件如果为真泽结束循环，适合至少需要执行一次的情况。

1. **批量创建用户**
    - 编写一个存储函数 `bulkCreateUsers`，接受一个以逗号分隔的字符串，包含多个用户的信息（例如 "username1,email1,age1;username2,email2,age2"）。使用循环结构解析字符串，并向 `users` 表中插入所有用户。
2. **查找并更新用户年龄**
    - 编写一个存储函数 `updateUsersAge`，接受一个年龄范围（最小年龄和最大年龄）和一个增加的年龄值。使用循环结构遍历 `users` 表中所有用户，更新年龄在指定范围内的用户。
3. **统计用户年龄总和**
    - 编写一个存储函数 `sumUsersAge`，使用循环结构遍历 `users` 表，计算所有用户的年龄总和，并返回这个值。
4. **获取所有用户信息**
    - 编写一个存储函数 `getAllUsersInfo`，使用循环结构遍历 `users` 表，构建一个字符串，包含所有用户的详细信息（ID、用户名、邮箱、年龄），并返回这个字符串。
5. **查找特定用户名的用户**
    - 编写一个存储函数 `findUsersByUsername`，接受一个用户名作为参数，使用循环结构遍历 `users` 表，返回所有匹配该用户名的用户信息。