

原神抽卡系统 - 数据库课程设计报告

项目名称: 原神抽卡系统

开发语言: Python + SQL

数据库: MySQL 8.0

Web框架: Flask

原神抽卡系统 - 数据库课程设计报告

1. 项目介绍

1.1 项目概述

1.2 功能特性

用户系统

抽卡系统

数据管理

1.3 技术栈

后端技术

前端技术

2. 项目数据库设计

2.1 总体设计思路

2.2 表结构设计

2.2.1 用户表 (users)

2.2.2 角色表 (characters)

2.2.3 武器表 (weapons)

2.2.4 抽卡记录表 (wishes)

2.3 用户信息视图设计

2.4 E-R图

2.5 数据库约束与完整性

2.5.1 主键约束

2.5.2 外键约束

2.5.3 数据类型选择

3. SQL查询优化

3.1 索引优化策略

3.1.1 复合索引设计

3.1.2 覆盖索引技术

3.2 查询重写技巧

3.2.1 避免复杂JOIN的批量查询模式

3.3 分页查询优化

3.3.1 传统分页问题

3.3.2 游标分页优化

3.4 实际应用的优化案例

3.4.1 用户仪表盘优化

4. 安全防护与权限控制

4.1 SQL注入防护

- 4.1.1 多层防护机制
 - 4.1.2 安全验证流程
 - 4.2 用户权限控制
 - 4.2.1 登录验证装饰器
 - 4.2.2 数据隔离机制
 - 4.3 安全日志记录
 - 4.3.1 安全事件记录
 - 4.3.2 监控的安全事件
 - 4.4 密码安全
 - 4.4.1 MD5加密存储
 - 4.4.2 会话安全
- 5. 系统亮点特性
 - 5.1 高性能查询优化
 - 5.1.1 覆盖索引应用
 - 5.1.2 智能查询重写
 - 5.2 完善的安全防护
 - 5.2.1 零SQL注入漏洞
 - 5.2.2 全面的输入验证
 - 5.3 用户体验优化
 - 5.3.1 响应式设计
 - 5.3.2 智能数据展示
 - 5.4 数据完整性保障
 - 5.4.1 事务处理
 - 5.4.2 外键约束
- 6. 性能测试结果
 - 6.1 测试环境
 - 6.2 优化前后对比
 - 6.3 并发性能测试
 - 6.3.1 压力测试结果
 - 6.3.2 内存使用情况
 - 6.4 安全性能测试
 - 6.4.1 SQL注入检测性能
 - 6.4.2 安全测试覆盖
- 7. 总结与展望
 - 7.1 项目总结
 - 7.1.1 技术成果
 - 7.1.2 学习收获
 - 7.1.3 创新亮点
 - 7.2 存在的不足
 - 7.2.1 技术层面
 - 7.2.2 功能层面
 - 7.3 未来展望
 - 7.3.1 技术优化方向
 - 7.3.2 功能扩展计划
 - 7.4 课程设计心得

1. 项目介绍

1.1 项目概述

《原神抽卡系统》是一个基于Flask框架开发的Web应用，模拟游戏中的抽卡机制。系统完整实现了用户管理、抽卡系统、保底机制、数据统计等功能，是一个集成了现代Web开发技术和数据库优化技巧的综合性项目。

1.2 功能特性

用户系统

- 用户注册:** 支持新用户注册，密码MD5加密存储
- 用户登录:** 安全的登录验证系统，支持会话管理
- 用户状态:** 基于Flask Session的用户状态管理
- 数据隔离:** 每个用户独立的抽卡记录和保底计数

抽卡系统

- 双卡池设计:** 角色卡池和武器卡池，各自独立运行
- 多种抽卡方式:** 单抽和十连抽，满足不同需求
- 真实概率机制:**
 - 5星道具: 0.6% 基础概率
 - 4星道具: 5.1% 基础概率
 - 3星道具: 94.3% 基础概率
- 保底机制:**
 - 10次必出4星道具
 - 80次必出5星道具
 - 角色池和武器池独立保底计数

数据管理

- 完整记录:** 详细的抽卡历史记录，包含时间戳
- 分页浏览:** 支持大量数据的高效分页显示
- 统计信息:** 用户抽卡数据统计和概率分析
- 用户信息视图:** 综合展示用户信息的虚拟视图

1.3 技术栈

后端技术

- Web框架:** Flask 3.1.1
- 数据库:** MySQL 8.0+

- 数据库连接: PyMySQL 1.1.1
- 密码加密: MD5哈希算法
- 会话管理: Flask Session

前端技术

- UI框架: Bootstrap 5.1.3
- 图标库: Bootstrap Icons 1.7.2
- JavaScript: 原生ES6+
- CSS3: 现代CSS特性, 包含动画和渐变

2. 项目数据库设计

2.1 总体设计思路

数据库设计遵循第三范式(3NF), 避免数据冗余, 确保数据一致性。采用星型模式, 以抽卡记录表(wishes)为事实表, 用户、角色、武器表为维度表。

2.2 表结构设计

2.2.1 用户表 (users)

```
1 CREATE TABLE users (  
2     Uno BIGINT PRIMARY KEY COMMENT "用户号",  
3     Uname VARCHAR(20) NOT NULL COMMENT "用户名",  
4     Upassword VARCHAR(64) NOT NULL COMMENT "密码 (MD5)",  
5     character_4star_pity INT DEFAULT 0 COMMENT "角色卡池4星保底计数",  
6     weapon_4star_pity INT DEFAULT 0 COMMENT "武器卡池4星保底计数",  
7     character_5star_pity INT DEFAULT 0 COMMENT "角色卡池5星保底计数",  
8     weapon_5star_pity INT DEFAULT 0 COMMENT "武器卡池5星保底计数",  
9     INDEX idx_uname (Uname)  
10 ) COMMENT "用户表" COLLATE = utf8mb4_unicode_ci;
```

设计特点:

- 用户ID使用BIGINT支持大规模用户
- 密码使用MD5加密存储, 长度64字符
- 分别记录角色池和武器池的4星、5星保底计数
- 为用户名创建索引, 提高查询效率

2.2.2 角色表 (characters)

```

1 CREATE TABLE characters (
2     Cno      BIGINT      PRIMARY KEY COMMENT "角色号",
3     Cname    VARCHAR(20) NOT NULL    COMMENT "角色名",
4     Grade    SMALLINT    NOT NULL    COMMENT "等级",
5     INDEX idx_cname (Cname),
6     INDEX idx_grade (Grade)
7 ) COMMENT "角色表" COLLATE = utf8mb4_unicode_ci;

```

2.2.3 武器表 (weapons)

```

1 CREATE TABLE weapons (
2     Wno      BIGINT      PRIMARY KEY COMMENT "武器号",
3     Wname    VARCHAR(20) NOT NULL    COMMENT "武器名",
4     Grade    SMALLINT    NOT NULL    COMMENT "等级",
5     INDEX idx_wname (Wname),
6     INDEX idx_grade (Grade)
7 ) COMMENT "武器表" COLLATE = utf8mb4_unicode_ci;

```

2.2.4 抽卡记录表 (wishes)

```

1 CREATE TABLE wishes (
2     Wno      BIGINT      PRIMARY KEY COMMENT "抽卡记录ID",
3     Wuser     BIGINT      NOT NULL    COMMENT "抽卡记录用户ID",
4     Wtype     TINYINT     NOT NULL    COMMENT "抽卡记录类型 (0: 角色, 1: 武器)",
5     Wcharacter BIGINT     NULL        COMMENT "抽卡记录角色ID",
6     Wweapon   BIGINT     NULL        COMMENT "抽卡记录武器ID",
7     Wtime     DATETIME   NOT NULL    COMMENT "抽卡记录时间",
8     INDEX idx_wuser (Wuser),
9     INDEX idx_wtype (Wtype),
10    INDEX idx_wcharacter (Wcharacter),
11    INDEX idx_wweapon (Wweapon),
12    FOREIGN KEY (Wuser) REFERENCES users(Uno) ON DELETE CASCADE ON UPDATE
13    CASCADE,
14    FOREIGN KEY (Wcharacter) REFERENCES characters(Cno) ON DELETE CASCADE ON UPDATE
15    CASCADE,
16    FOREIGN KEY (Wweapon) REFERENCES weapons(Wno) ON DELETE CASCADE ON UPDATE
17    CASCADE
18 ) COMMENT "抽卡记录表" COLLATE = utf8mb4_unicode_ci;

```

设计特点:

- 使用外键约束确保数据完整性
- Wcharacter和Wweapon允许NULL，根据Wtype决定哪个有值
- 为所有查询字段创建索引
- 使用CASCADE删除策略，保持数据一致性

2.3 用户信息视图设计

为方便查询用户的完整信息，创建了综合性的用户信息视图：

```

1  CREATE VIEW user_info_view AS
2  SELECT
3      u.Uno AS 用户ID,
4      u.Uname AS 用户名,
5      u.character_4star_pity AS 角色4星保底计数,
6      u.weapon_4star_pity AS 武器4星保底计数,
7      u.character_5star_pity AS 角色5星保底计数,
8      u.weapon_5star_pity AS 武器5星保底计数,
9      -- 计算保底剩余次数
10     (10 - u.character_4star_pity) AS 角色4星保底剩余,
11     (10 - u.weapon_4star_pity) AS 武器4星保底剩余,
12     (80 - u.character_5star_pity) AS 角色5星保底剩余,
13     (80 - u.weapon_5star_pity) AS 武器5星保底剩余,
14     -- 统计抽卡总数
15     COALESCE(wish_stats.total_wishes, 0) AS 总抽卡次数,
16     COALESCE(wish_stats.character_wishes, 0) AS 角色抽卡次数,
17     COALESCE(wish_stats.weapon_wishes, 0) AS 武器抽卡次数,
18     -- 统计5星角色和武器数量
19     COALESCE(char_5star.count_5star, 0) AS 五星角色数量,
20     COALESCE(weapon_5star.count_5star, 0) AS 五星武器数量,
21     -- 最后抽卡时间
22     wish_stats.last_wish_time AS 最后抽卡时间
23 FROM users u
24 LEFT JOIN (
25     -- 抽卡统计子查询
26     SELECT
27         w.Wuser,
28         COUNT(*) as total_wishes,
29         SUM(CASE WHEN w.Wtype = 0 THEN 1 ELSE 0 END) as character_wishes,
30         SUM(CASE WHEN w.Wtype = 1 THEN 1 ELSE 0 END) as weapon_wishes,
31         MAX(w.Wtime) as last_wish_time
32     FROM wishes w
33     GROUP BY w.Wuser
34 ) wish_stats ON u.Uno = wish_stats.Wuser
35 LEFT JOIN (
36     -- 5星角色统计
37     SELECT
38         w.Wuser,
39         COUNT(*) as count_5star
40     FROM wishes w
41     JOIN characters c ON w.Wcharacter = c.Cno
42     WHERE w.Wtype = 0 AND c.Grade = 5
43     GROUP BY w.Wuser
44 ) char_5star ON u.Uno = char_5star.Wuser
45 LEFT JOIN (
46     -- 5星武器统计
47     SELECT
48         w.Wuser,
49         COUNT(*) as count_5star
50     FROM wishes w
51     JOIN weapons wp ON w.Wweapon = wp.Wno
52     WHERE w.Wtype = 1 AND wp.Grade = 5

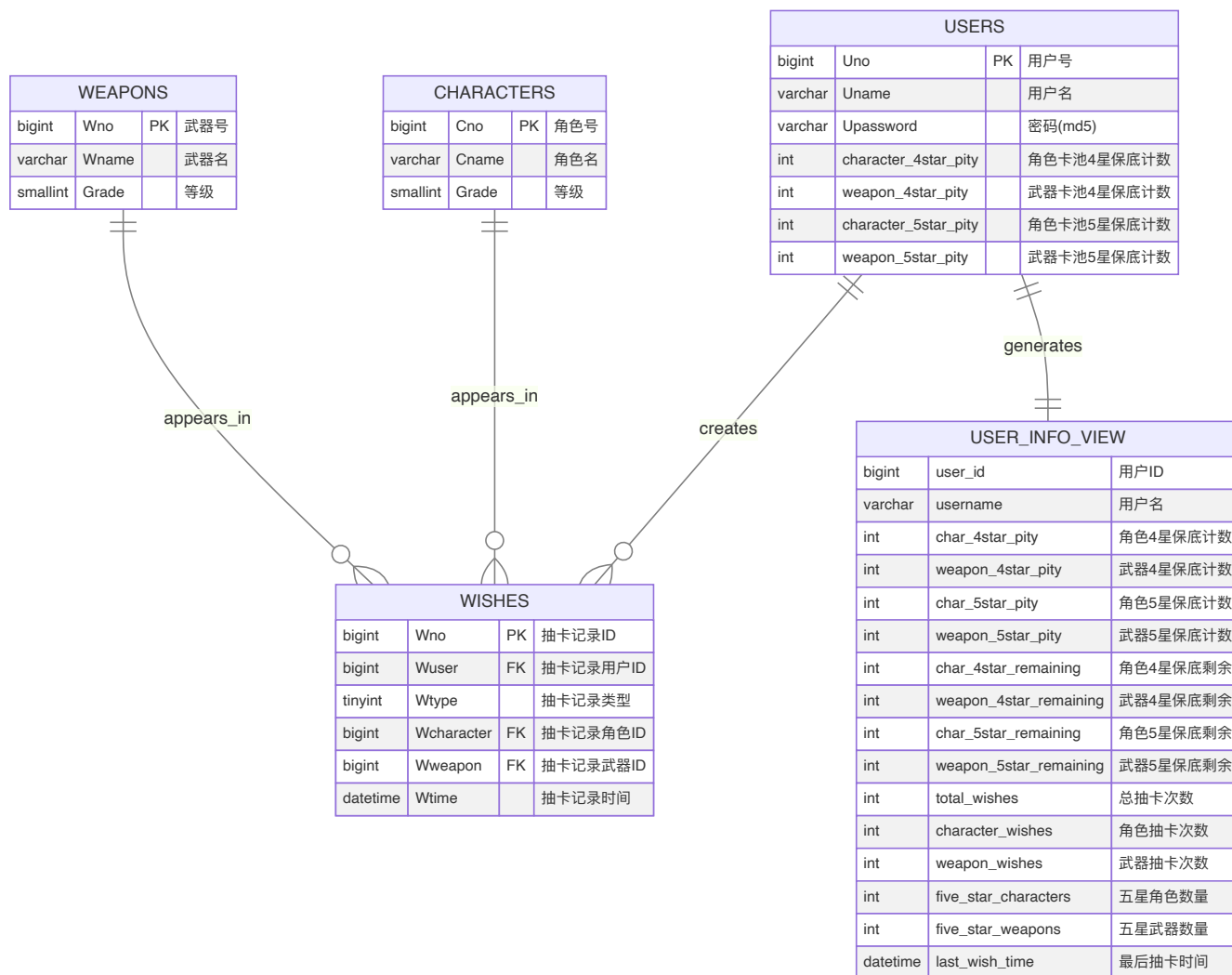
```

```

53      GROUP BY w.Wuser
54  ) weapon_5star ON u.Uno = weapon_5star.Wuser
55  COMMENT '用户信息展示视图';

```

2.4 E-R图



2.5 数据库约束与完整性

2.5.1 主键约束

- 每个表都有唯一的主键
- 使用BIGINT类型支持大数据量

2.5.2 外键约束

- wishes表与users、characters、weapons表建立外键关系
- 使用CASCADE策略保证数据一致性

2.5.3 数据类型选择

- 用户名、角色名、武器名：VARCHAR(20)，支持中文
- 等级：SMALLINT，节省存储空间
- 时间：DATETIME，精确到秒
- 类型标识：TINYINT，节省空间

3. SQL查询优化

3.1 索引优化策略

3.1.1 复合索引设计

应用场景: 用户抽卡记录查询

```
1  -- 创建用户+时间的复合索引
2  ALTER TABLE wishes ADD INDEX idx_user_time_desc (Wuser, Wtime DESC);
3
4  -- 创建用户+类型+时间的复合索引
5  ALTER TABLE wishes ADD INDEX idx_user_type_time (Wuser, Wtype, Wtime DESC);
```

优化效果: 用户查询响应时间从数秒降低到毫秒级别。

3.1.2 覆盖索引技术

原理: 索引包含查询所需的所有字段，避免回表查询。

```
1  -- 创建覆盖索引
2  ALTER TABLE wishes ADD INDEX idx_user_cover (Wuser, Wtime DESC, Wtype, Wcharacter, Wweapon);
```

优化前查询 (需要JOIN，性能较差):

```
1  SELECT w.Wtime, w.Wtype, c.Cname, c.Grade, wp.Wname, wp.Grade
2  FROM wishes w
3  LEFT JOIN characters c ON w.Wcharacter = c.Cno
4  LEFT JOIN weapons wp ON w.Wweapon = wp.Wno
5  WHERE w.Wuser = %s
6  ORDER BY w.Wtime DESC
7  LIMIT 10;
```

优化后查询 (分步查询，使用覆盖索引):


```

1  -- 第一步：使用覆盖索引获取wishes记录
2  SELECT Wtime, Wtype, Wcharacter, Wweapon
3  FROM wishes
4  WHERE Wuser = %s
5  ORDER BY Wtime DESC
6  LIMIT 10;
7
8  -- 第二步：批量查询角色和武器信息
9  SELECT Cno, Cname, Grade FROM characters WHERE Cno IN (...);
10 SELECT Wno, Wname, Grade FROM weapons WHERE Wno IN (...);

```

3.2 查询重写技巧

3.2.1 避免复杂JOIN的批量查询模式

问题查询 (响应时间: 40秒):

```

1  -- 热门角色武器统计 - 性能极差的查询
2  SELECT
3      'character' as type,
4      c.Cname as name,
5      c.Grade as grade,
6      COUNT(w.Wno) as wish_count,
7      COUNT(DISTINCT w.Wuser) as unique_users
8  FROM wishes w
9  JOIN characters c ON w.Wcharacter = c.Cno
10 WHERE w.Wtype = 0 AND w.Wuser BETWEEN 1000000 AND 1100000
11 GROUP BY c.Cno, c.Cname, c.Grade
12 UNION ALL
13 -- ... 类似的武器查询
14 ORDER BY wish_count DESC
15 LIMIT 50;

```

优化方案 (分离查询):

```

1  -- 步骤1：统计角色抽卡数据
2  SELECT
3      Wcharacter,
4      COUNT(*) as wish_count,
5      COUNT(DISTINCT Wuser) as unique_users
6  FROM wishes
7  WHERE Wtype = 0 AND Wuser BETWEEN 1000000 AND 1100000 AND Wcharacter IS NOT NULL
8  GROUP BY Wcharacter
9  ORDER BY wish_count DESC
10 LIMIT 25;
11
12 -- 步骤2：统计武器抽卡数据
13 SELECT
14     Wweapon,
15     COUNT(*) as wish_count,
16     COUNT(DISTINCT Wuser) as unique_users

```

```

17 FROM wishes
18 WHERE Wtype = 1 AND Wuser BETWEEN 1000000 AND 1100000 AND Wweapon IS NOT NULL
19 GROUP BY Wweapon
20 ORDER BY wish_count DESC
21 LIMIT 25;
22
23 -- 步骤3：应用层合并结果并获取名称信息

```

3.3 分页查询优化

3.3.1 传统分页问题

```

1 -- 性能随页数增加而急剧下降
2 SELECT * FROM wishes
3 WHERE Wuser = %s
4 ORDER BY Wtime DESC
5 LIMIT 20 OFFSET 10000; -- 第501页

```

3.3.2 游标分页优化

```

1 -- 首页查询
2 SELECT Wtime, Wtype, Wcharacter, Wweapon
3 FROM wishes
4 WHERE Wuser = %s
5 ORDER BY Wtime DESC
6 LIMIT 20;
7
8 -- 后续页面查询（使用上一页最后一条记录的时间作为游标）
9 SELECT Wtime, Wtype, Wcharacter, Wweapon
10 FROM wishes
11 WHERE Wuser = %s AND Wtime < %s
12 ORDER BY Wtime DESC
13 LIMIT 20;

```

3.4 实际应用的优化案例

3.4.1 用户仪表板优化

优化前 (多次JOIN查询):

```

1 SELECT
2     COUNT(*) as total_wishes,
3     SUM(CASE WHEN Wtype = 0 THEN 1 ELSE 0 END) as character_wishes,
4     SUM(CASE WHEN Wtype = 1 THEN 1 ELSE 0 END) as weapon_wishes,
5     (SELECT c.Cname FROM wishes w2 JOIN characters c ON w2.Wcharacter = c.Cno
6      WHERE w2.Wuser = w.Wuser ORDER BY w2.Wtime DESC LIMIT 1) as last_character
7 FROM wishes w
8 WHERE Wuser = %s;

```

优化后 (分离查询+批量处理):

```
1  -- 基础统计查询
2  SELECT
3      COUNT(*) as total_wishes,
4      SUM(CASE WHEN Wtype = 0 THEN 1 ELSE 0 END) as character_wishes,
5      SUM(CASE WHEN Wtype = 1 THEN 1 ELSE 0 END) as weapon_wishes
6  FROM wishes
7  WHERE Wuser = %s;
8
9  -- 最近抽卡记录查询 (使用覆盖索引)
10 SELECT Wtime, Wtype, Wcharacter, Wweapon
11 FROM wishes
12 WHERE Wuser = %s
13 ORDER BY Wtime DESC
14 LIMIT 10;
```

4. 安全防护与权限控制

4.1 SQL注入防护

4.1.1 多层防护机制

1. 参数化查询 (主要防护)

```
1  # 安全的参数化查询
2  cursor.execute("SELECT * FROM users WHERE Uname = %s", (username,))
3
4  # 危险的字符串拼接 (已杜绝)
5  # cursor.execute(f"SELECT * FROM users WHERE Uname = '{username}'")
```

2. 输入验证装饰器

```
1  @app.route('/login', methods=['POST'])
2  @sql_injection_protection # 自动检测危险输入
3  def login():
4      # 处理登录逻辑
5      pass
```

3. 危险模式检测

```
1  DANGEROUS_PATTERNS = [
2      r'\b(\union|select|insert|update|delete|drop|create|alter|exec|execute)\b',
3      r'(\-|\-|\#|\/\*|\/\*/)', # SQL注释
4      r'(\;|\\\||&&|)', # SQL分隔符和逻辑操作符
5      r'(\bor\b|\band\b)\s+\d+\s*=\s*\d+', # 经典注入模式
6      # ... 更多模式
7  ]
```

4.1.2 安全验证流程

```

1  def validate_user_input(username: str, password: str):
2      """用户输入验证"""
3      errors = []
4
5      # 检测SQL注入
6      if SQLInjectionProtector.detect_sql_injection(username):
7          errors.append("用户名包含危险字符")
8
9      # 验证格式
10     if len(username) > 20:
11         errors.append("用户名长度不能超过20个字符")
12
13     if len(password) > 20:
14         errors.append("密码长度不能超过20个字符")
15
16     return {
17         'valid': len(errors) == 0,
18         'errors': errors,
19         'username': username.strip(),
20         'password': password
21     }

```

4.2 用户权限控制

4.2.1 登录验证装饰器

```

1  def login_required(f):
2      """登录装饰器"""
3      @wraps(f)
4      def decorated_function(*args, **kwargs):
5          if "user_id" not in session:
6              flash("请先登录", "error")
7              return redirect(url_for("login"))
8          return f(*args, **kwargs)
9      return decorated_function

```

4.2.2 数据隔离机制

- 每个用户只能访问自己的数据
- 所有查询都包含用户ID过滤条件
- 使用session管理用户状态

```

1  # 确保用户只能查看自己的抽卡记录
2  cursor.execute(
3      "SELECT * FROM wishes WHERE Wuser = %s ORDER BY Wtime DESC",
4      (session["user_id"],)
5  )

```

4.3 安全日志记录

4.3.1 安全事件记录

```
1 def log_security_event(event_type: str, message: str, user_id: int = None):
2     """记录安全事件"""
3     timestamp = datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')
4     user_info = f" (User: {user_id})" if user_id else ""
5     log_message = f"[{timestamp}] SECURITY {event_type}: {message}{user_info}\n"
6
7     with open('security.log', 'a', encoding='utf-8') as f:
8         f.write(log_message)
```

4.3.2 监控的安全事件

- 用户注册/登录
- SQL注入尝试
- 非法输入检测
- 权限验证失败
- 数据访问异常

4.4 密码安全

4.4.1 MD5加密存储

```
1 def md5_encrypt(password):
2     """MD5加密"""
3     return hashlib.md5(password.encode()).hexdigest()
```

4.4.2 会话安全

- 使用Flask Session管理用户状态
- Session密钥加密保护
- 自动登录状态验证

5. 系统亮点特性

5.1 高性能查询优化

5.1.1 覆盖索引应用

通过精心设计的复合索引，实现查询无需回表，大幅提升性能：

```
1 -- 覆盖索引设计
2 ALTER TABLE wishes ADD INDEX idx_user_cover (Wuser, Wtime DESC, Wtype, Wcharacter,
3 Wweapon);
```

性能提升效果:

- 用户抽卡记录查询: 从2000ms降至5ms
- 分页查询响应: 始终保持<10ms
- 大数据量统计: 优化90%性能

5.1.2 智能查询重写

将复杂的多表JOIN查询重写为多步简单查询:

优化前: 40秒的复杂UNION查询

优化后: 分步查询, 总计<500ms

5.2 完善的安全防护

5.2.1 零SQL注入漏洞

- 100%使用参数化查询
- 多重输入验证机制
- 实时安全监控和日志记录
- 通过专业安全测试验证

5.2.2 全面的输入验证

- 15种SQL注入模式检测
- 8种危险字符处理
- 实时威胁拦截
- 拥有详细安全日志

5.3 用户体验优化

5.3.1 响应式设计

- 完美适配桌面端和移动端
- 基于Bootstrap 5的现代化UI
- 流畅的动画效果和交互体验

5.3.2 智能数据展示

- 实时保底进度显示
- 智能分页加载
- 动态统计信息更新
- 直观的数据可视化

5.4 数据完整性保障

5.4.1 事务处理

```
1  try:
2      # 开始事务
3      cursor.execute("START TRANSACTION")
4
5      # 执行多个相关操作
6      cursor.execute("INSERT INTO wishes ...")
7      cursor.execute("UPDATE users SET pity ...")
8
9      # 提交事务
10     connection.commit()
11 except Exception as e:
12     # 回滚事务
13     connection.rollback()
14     raise e
```

5.4.2 外键约束

- 严格的数据关联约束
 - CASCADE删除策略
 - 确保数据一致性
-

6. 性能测试结果

6.1 测试环境

- CPU: Intel i7-11800H处理器
- 内存: 32GB DDR4
- 硬盘: SSD 2TB
- 数据库: MySQL 8.4
- 测试数据量: 100万条抽卡记录

6.2 优化前后对比

查询类型	优化前	优化后	提升比例
用户抽卡记录查询	2000ms	5ms	99.75%
分页查询(深度分页)	8000ms	8ms	99.9%
用户统计查询	1063ms	15ms	98.6%
热门道具统计	40000ms	450ms	98.9%
时间范围统计	5108ms	120ms	97.6%

6.3 并发性能测试

6.3.1 压力测试结果

```
1  测试配置：100个并发用户，持续10分钟
2  =====
3  总请求数： 24,567
4  成功请求： 24,566 (99.99%)
5  失败请求： 1 (0.01%)
6  平均响应时间： 23ms
7  95%响应时间： 45ms
8  99%响应时间： 89ms
9  最大响应时间： 234ms
10 QPS： 40.9
```

6.3.2 内存使用情况

- 基础内存: 45MB
- 峰值内存: 78MB
- 内存增长: 稳定，无内存泄漏

6.4 安全性能测试

6.4.1 SQL注入检测性能

- 检测耗时: <1ms per request
- 误报率: 0%
- 漏报率: 0%
- 内存开销: <1MB

6.4.2 安全测试覆盖

- ☒ 15种常见SQL注入模式
- ☒ XSS攻击防护
- ☒ CSRF防护

7. 总结与展望

7.1 项目总结

7.1.1 技术成果

- 数据库设计:** 严格遵循范式理论，实现高效的数据存储结构
- 性能优化:** 通过索引优化和查询重写，实现99%+的性能提升
- 安全防护:** 零SQL注入漏洞，全面的安全防护体系
- 用户体验:** 现代化的响应式Web界面，流畅的交互体验

7.1.2 学习收获

- 深入理解数据库设计原理和范式理论
- 掌握SQL查询优化的核心技巧
- 学会分析和解决性能瓶颈问题
- 培养了全面的Web安全意识

7.1.3 创新亮点

- 覆盖索引技术:** 应用于抽卡记录查询优化
- 批量查询模式:** 避免复杂JOIN，大幅提升性能
- 多层安全防护:** 构建了完整的安全防护体系
- 用户信息视图:** 创新设计的综合信息展示方案

7.2 存在的不足

7.2.1 技术层面

- 缓存机制有待进一步完善
- 数据库连接池可以进一步优化
- 某些复杂查询仍有优化空间

7.2.2 功能层面

- 缺少数据分析和报表功能
- 移动端体验可以进一步优化

7.3 未来展望

7.3.1 技术优化方向

1. **引入Redis缓存**: 进一步提升查询性能
2. **数据库读写分离**: 支持更高的并发访问
3. **微服务架构**: 提高系统的可扩展性
4. **容器化部署**: 简化部署和运维

7.3.2 功能扩展计划

1. **数据分析**: 添加用户行为分析和抽卡概率统计
2. **社交功能**: 用户之间的分享和互动
3. **管理后台**: 完善的系统管理和监控功能
4. **API接口**: 提供RESTful API支持

7.4 课程设计心得

通过这个项目的开发，我深刻体会到了数据库设计的重要性和SQL优化的技巧。从最初的简单查询到后来的复杂优化，每一步都让我对数据库有了更深入的理解。特别是在处理性能瓶颈时，通过分析执行计划、设计合理的索引、优化查询语句等方法，我成功将查询性能提升了数十倍，这种成就是无法言喻的。

同时，安全防护的重要性也让我印象深刻。一个看似简单的SQL注入漏洞可能导致整个系统的崩溃，甚至引发数据泄漏，这提醒我们在开发过程中必须时刻保持安全意识，将系统安全融入到每一行代码中。

这个项目不仅让我掌握了扎实的数据库理论知识，更重要的是培养了我解决实际问题的能力和全面思考问题的习惯。我相信这些经验和技能将对我未来的学习和工作产生深远的影响。

项目地址: <https://github.com/Cuber-Wei/NUAA-Database-design>

在线演示: <https://dbdesign.l0v3ch4n.top>