CS307 Principles of Database Systems

Project 2 Report

成员信息

	姓名（Student Name)	学号（Student ID）
1	施米乐（Shi Mile)	12212921
2	张伟祎（Zhang Weiyi)	12210653
3	敖恺（Ao Kai)	12211617

lab：周一7-8节

任务分配与贡献比

任务分配

- **Database Design**
  - 数据库设计 施米乐
- **Basic API Specification**
  - UserServicej接口 张伟祎
  - VideoService接口 敖恺
  - DanmuService接口 施米乐
  - RecommendService 张伟祎
- **Advanced APIs and Other Requirements**
  - 弹幕脏话识别
  - 数据导入优化
  - mid、BV生成策略的探索和保密性的衡量
  - sql语句的优化以降低连接池开销

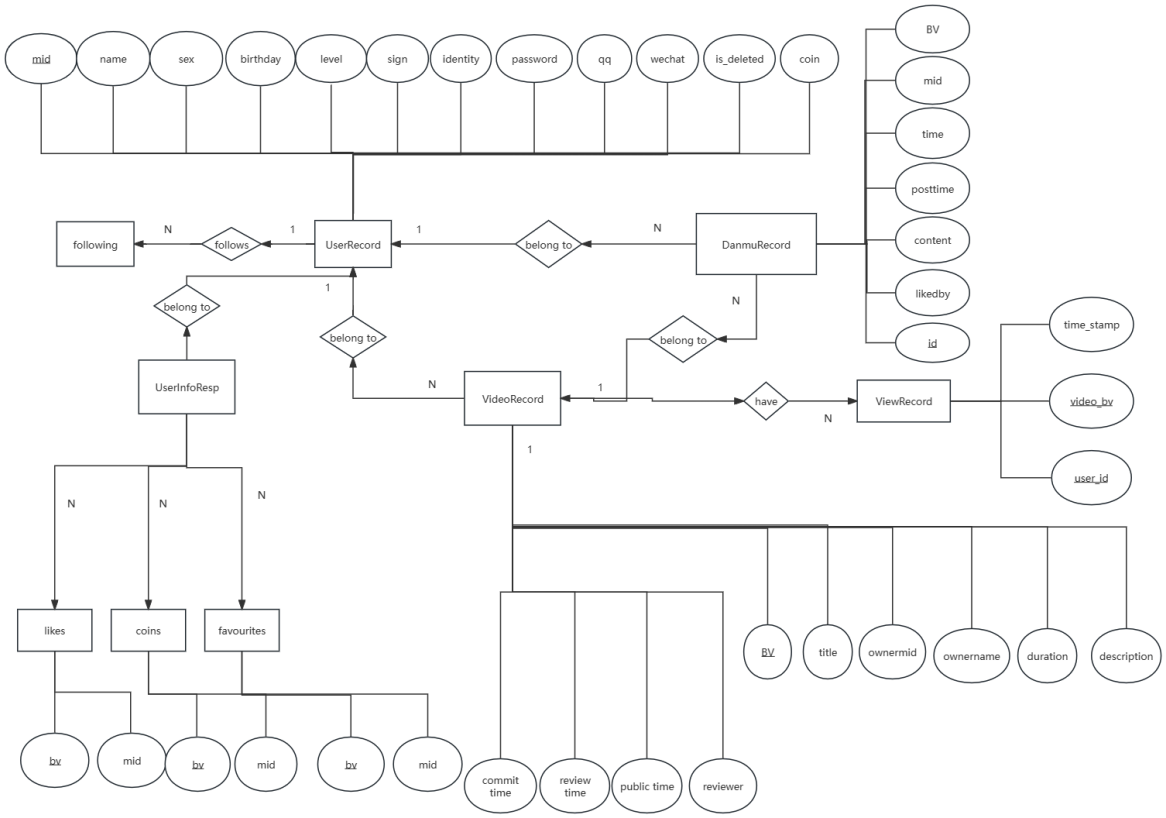
贡献比

	姓名	贡献比
1	施米乐（Shi Mile)	33.3%
2	张伟祎（Zhang Weiyi)	33.3%
3	敖恺（Ao Kai)	33.3%

任务1：Database Design

一、表设计

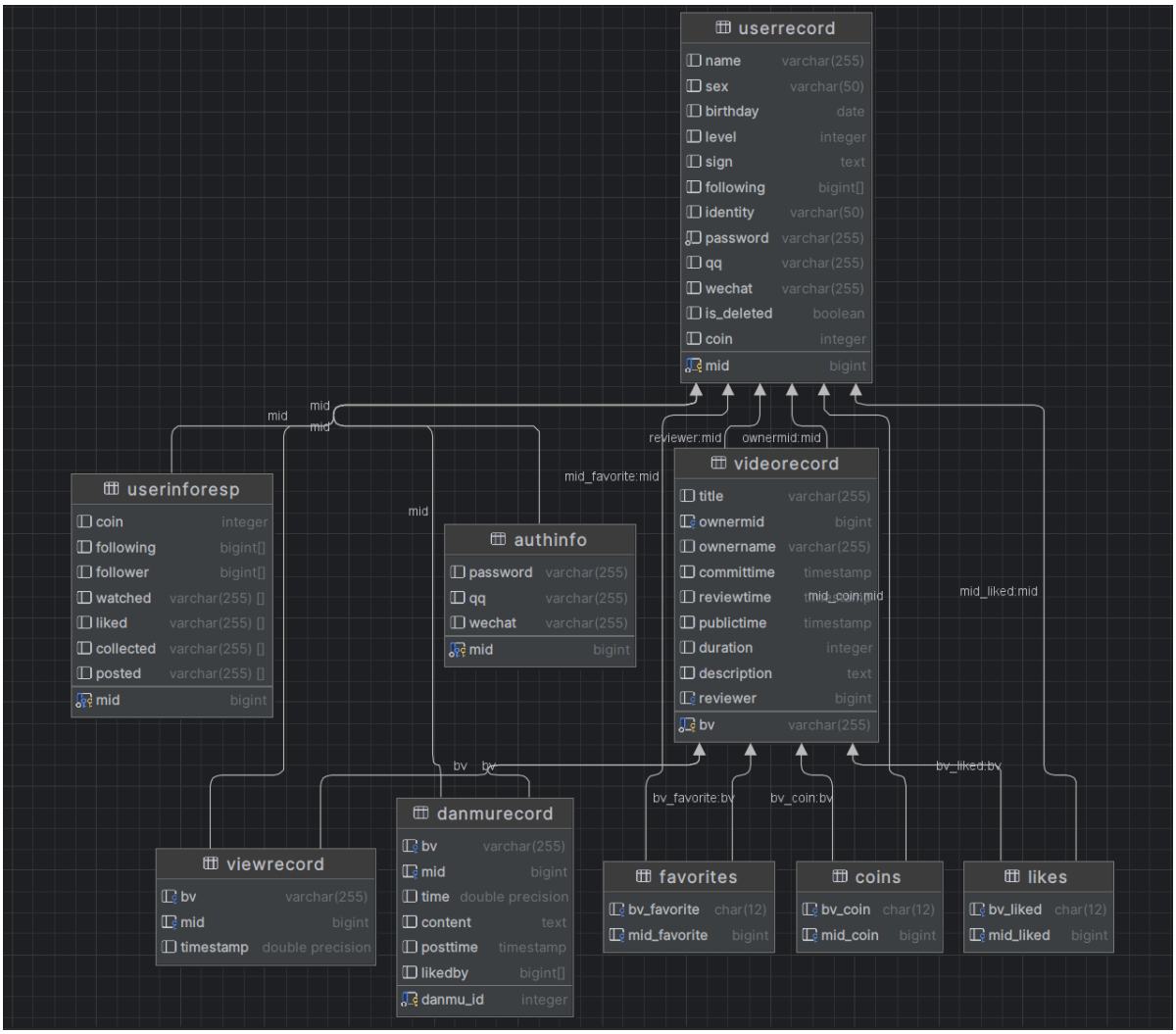
1. ER图：



绘图工具：ProcessOn

---

2. 表设计



（1）Record 表：

mid：用户唯一标识号，主键。

name：用户创建的名称。

sex：生物学性别或其他性别身份。

birthday：用户生日。

level：系统评估的用户参与等级。

sign：用户创建的个人描述。

following：关注用户的 mid 列表。

identity：用户角色，可为 'user' 或 'superuser'。

password：登录密码。

qq：QQ OIDC 登录，无需密码。

wechat：微信 OIDC 登录，无需密码。

is_deleted：表示用户是否被删除的标志。

（2）VideoRecord 表：

bv：视频的唯一标识字符串，主键。

title：视频名称。

ownerMid：视频所有者的 mid，外键。

ownerName：视频所有者名称。

commitTime：视频提交时间。

reviewTime：视频审核时间。

publicTime：视频发布时间。

duration：视频时长（秒）。

description：视频简介。

reviewer：视频审核者的 mid，外键。

（3）AuthInfo 表：

mid：用户的 mid，主键，外键到 UserRecord。

password：登录密码。

qq：QQ OIDC 登录，无需密码。

wechat：微信 OIDC 登录，无需密码。

（4）ViewRecord 表：

bv：视频的 BV，外键到 VideoRecord。

mid：观看视频的用户 mid，外键到 UserRecord。

timestamp：最后观看时间戳。

（5）DanmuRecord 表：

danmu_id：弹幕唯一标识，主键。

bv：弹幕所属视频的 BV，外键。

mid：发送弹幕的用户 mid，外键。

time：弹幕在视频开始后的显示时间（秒）。

content：弹幕内容。

postTime：弹幕发布时间。

likedBy：喜欢该弹幕的用户 mids 列表。

在该表中，通过设置脏话库和trigger，实现了对弹幕内容的筛查。

（6）PostVideoReq 表：

title：视频标题。

description：视频描述。

duration：视频时长（秒）。

publicTime：视频计划公开时间。

（7）UserInfoResp 表：

mid：用户的 mid，主键，外键到 UserRecord。

coin：用户的硬币数。

following：关注用户的 mid 列表。

follower：粉丝的 mid 列表。

watched：已观看视频的 BV 列表。

liked：喜欢的视频的 BV 列表。

collected：收藏的视频的 BV 列表。

posted：发布的视频的 BV 列表。

在实现时遇到了postgres数组上限的问题，因此设置了单独的三张表：

（8）likes 表：

BV_liked：喜欢的视频的 BV，外键到 VideoRecord。

mid_liked：喜欢视频的用户 mid，外键到 UserRecord。

（9）favorites 表：

BV_favorite：收藏的视频的 BV，外键到 VideoRecord。

mid_favorite：收藏视频的用户 mid，外键到 UserRecord。

（10）coins 表：

BV_coin：使用硬币的视频的 BV，外键到 VideoRecord。

mid_coin：使用硬币的用户 mid，外键到 UserRecord

任务二． API实现

1. **UserImpl**

在该类中，我们进行了authInfo合法性的检查，包括部分重要信息是否为空、生日等字符格式是否合法、生日时间是否合理等，处理了不同输入格式带来的匹配问题，如生日“xx月xx日”与“xx-xx”，并将`IsValidAuth`方法设为public，便于在`videoServiceImpl`和`DanmuServiceImpl`中进行扩展。

值得一提的是，在`rigister`中，我们模仿了uuid的全局唯一id生成策略，并借鉴了QQ账号号码池的思想，在数据库原有数据的基础上随机增加，这不仅保证了生成mid的全局唯一性，同时避免了直接使用递增序列时，用户id与注册时间成线性关系，从而保护了用户信息的安全。

2.**VideoImpl**

VideoServiceImpl 类是视频管理核心功能的实现，它负责处理与视频相关的所有业务逻辑。这个类提供了一系列方法，包括视频上传、删除、信息更新、搜索、评价以及统计分析等。

在删除视频(deleteVideo)的实现中，我们不仅从 VideoRecord 表中移除了视频记录，还从所有相关表中删除了该视频的引用，如 ViewRecord、DanmuRecord 等，保证了数据库的数据一致性和完整性。

评价功能包括 likeVideo、coinVideo 和 collectVideo 方法，它们允许用户对视频进行点赞、投币和收藏。这些方法都进行了严格的权限和状态检查，防止了例如重复投币等不合理的操作，并确保了操作的有效性。

另外，类中还包括了诸如 `getAverageViewRate` 和 `getHotspot` 等分析方法，它们提供了关于视频观看模式和用户互动的深入洞见。例如，`getHotspot` 方法通过分析视频中的弹幕来确定观众最集中互动的视频部分。

值得一提的是，在实现 `VideoServiceImpl` 的过程中，我们还特别注重了异常处理，确保了方法在遇到非法输入或操作失败时能给出清晰的反馈。同时，我们也在关键操作中实现了适当的事务管理，保证了数据库操作的原子性。

3. **DanmuImpl**

在该类中，我们实现了video，content和user的合法性检查，利用了这三个方法，有效的避免了corner case并且减少了代码复杂程度。

值得一提的是，在`displaydanmu`方法中，我们选择了创建两个sql语句来分别完成有无重复筛选的任务。

4. **DatabaseImpl**

主要实现的方法是`dataimport`。基于project1中我们对提高数据导入效率进行的探索，我们使用了batch的方法，将多条语句一起执行，有效提高了插入的速度。

5. **RecommendImpl**

在该类中，我们多次使用了联合查询实现筛选和推荐功能。

值得一提的是，`generalRecommendations`方法中使用多条with语句，构造了多个带名称的子查询，这些查询可以以其名称在查询中被多次引用。

任务三. 高级API优化

1. 弹幕脏话识别

在插入弹幕时，我们创建了一个`restricted_words`表，并且设置了一个在insert层面执行的trigger，将含有脏话的弹幕拒绝插入。

但是为了benchmark的正确性，我们只是提出了这个trigger的想法，在建表时禁用了这个trigger。

2. 数据导入优化

在导入数据时，我们使用了多个策略保证了导入数据的正确性和速度。

(1) batch

由于导入数据的相似性，我们使用了`PreparedStatement`，只改变语句中的参数。同时，我们也用到该语句的batch特性，将500条语句设为一个batch，一次执行。

(2) trigger

由于导入时数据只有`userrecord`、`videorecord`和`danmurecord`三张表的数据，但是在后续的数据库操作中，用到的显然不止这三张表。当然，我们设计了trigger可以将数据同步到其他表里，但是对于大量数据并且是已知的数据，我们认为直接使用query语句会大大加快导入数据。事实也是这样的，我们会在导入数据的开始禁用所有trigger并且在导入完成后，执行我们预先设计的query语句。

如果使用trigger自动导入，是可以保证数据的完整性和正确性，但是在测试中，将近十分钟都没有完成小数据的导入，但是我们的query语句可以在20s内完成数据的同步，并且同样保证初始数据的一致性和正确性。

对于后续的数据库操作，我们自然是使用trigger来同步。这样不光简化了sql语句，同时也让数据库更加方便维护。

（3）建表优化

根据题目的要求，我们自然可以只有`userrecord`、`videorecord`和`danmurecord`三张表。但是为了查询的高效性和数据库的局限性，我们设计了其他的表，接下来介绍一下特别之处。

①数据安全

我们将存有password的`authinfo`表与主表`userrecord`分离，有利于安全性和模块化设计。

- **减少数据泄露风险**：在`AuthInfo`表中单独存储敏感的登录信息（如密码）意味着在大部分常规操作中，这些敏感数据不会被暴露。比如，当应用程序或管理员需要访问用户的非敏感信息时，他们可以只查询`UserRecord`表，无需接触`AuthInfo`表。
- **降低被攻击的面**：将敏感数据隔离到单独的表中，可以减少因为SQL注入等攻击而暴露敏感信息的风险。攻击者在未能获取完整的用户表的情况下，难以同时获取用户名和密码。
- **便于实施安全措施**：对于包含敏感信息的`AuthInfo`表，可以实施更严格的安全措施，如加密存储、访问控制和审计日志，而这些措施可能对于其他表（如`UserRecord`）来说过于繁重。

②数据规范化

数据被分布在多个相关联的表中，减少了数据冗余，提高了数据一致性。

- **分离用户和视频数据**：`UserRecord`和`VideoRecord`表分别存储用户和视频的信息。这样，用户和视频的任何信息变更只需在各自的表中进行，不会影响其他表。例如，如果一个用户更改了他们的名字，只需更新`UserRecord`表中的相应记录，而不需要更改存储观看历史的`ViewRecord`表。
- **使用引用键**：`ViewRecord`表通过引用`UserRecord`和`VideoRecord`表中的键（`mid`和`bv`），建立了用户和视频之间的关系。这种设计确保了数据的一致性，因为任何对用户或视频的更改都会自动反映在观看记录中。

规范化设计有助于简化数据更新操作，减少错误和冲突的可能性。

- **减少更新操作的复杂性**：由于数据被合理地分布在多个表中，当需要更新用户或视频信息时，只需针对特定的表进行操作，而不必担心会影响到其他相关的数据。例如，如果一个视频的标题需要更改，只需在`VideoRecord`表中更新，而不会影响到存储用户观看行为的`ViewRecord`表。
- **维护数据的完整性**：通过设置外键约束，比如`ViewRecord`表中的`bv`和`mid`，确保了不会出现无效的观看记录。这意味着只有当`UserRecord`和`VideoRecord`表中存在对应的用户和视频时，`ViewRecord`表中才能添加记录。

③性能优化

由于对于一个视频点赞投币的人太多，超过了postgres数组的最大限制，所以我们单独设计了三个表来存储。同时，为了查询的高效性，我们注意到，可以使用“group by”方法，用数组来存储每个用户的点赞投币，这些数据没有那么大，相当于变相给这张表加了索引，显著加快以用户mid为条件的查询。

④可拓展性

- **独立的功能模块**
    - 我们的数据库将不同功能的数据分布到不同的表中，例如`UserRecord`、`VideoRecord`、`DanmuRecord`等。这种模块化的方法使得未来添加新功能或调整现有功能变得更加容易。例如，如果需要添加新的视频功能（如视频分类），可以简单地扩展`VideoRecord`表或添加新的相关表，而无需重构整个数据库。

- **关联表的使用**
    - 通过使用如 `likes`、`favorites`、`coins` 等关联表，我们的设计允许灵活地追踪用户和视频之间的复杂互动，而不是将所有数据压缩在单一的大表中。这种方法在数据库随着用户数量和用户行为增长时，便于进行水平扩展。

3. 全局唯一id生成策略探索

- **postgres自增序列**

PostgreSQL 提供`SMALLSERIAL`，`SERIAL`，`BIGSERIAL`这三种序列伪类型，其中`SERIAL`对应的范围为`1-2,147,483,647`，符合我们数据库中mid的生成区间

```
CREATE SEQUENCE user_id_seq;
```

但序列生成器不是事务安全的。意味着如果两个并发数据库连接尝试从序列中获取下一个值，每个客户端获得不同的值。如果一个客户端回滚事务，则该序列值将被弃用，这会导致序列值不连续。

同时，自增序列每次增加的大小是一样的，此时用户注册顺序和mid成线性相关关系，不利于维护用户信息安全。

另外，postgres自增序列强依赖DB，且在不同数据库语法中实现方法不同，面对数据库迁移、多数据库版本支持、分表分库等问题时难以扩展。

- **模仿uuid的生成策略**

UUID的标准形式包含32个16进制数字，以连字号分为五段，形式为8-4-4-4-12的36个字符。虽然uuid具有全球唯一性，生成性能、安全性、扩展性都非常优异，但36个字符造成了极高的储存成本，其无序性也对传输数据造成了过大的开销，更重要的是，并不符合我们目前数据库已有数据的唯一id格式。

因此，我们模仿uuid的思想，找出数据库中mid已有最大值，并在此基础上随机增加一个数作为新的mid；在video的BV则直接使用了Java库提供的`randomUUID()`方法。

```
long mid = 0;
            String max = "SELECT MAX(mid) FROM UserRecord";
            try (PreparedStatement secondStmt = conn.prepareStatement(max)) {
                try (ResultSet rs = secondStmt.executeQuery()) {
                    if (rs.next()) {
                        Random rand = new Random();
                        mid = rs.getLong(1) + rand.nextLong(100);
                    }
                }
```

```
String bv = UUID.randomUUID().toString();
```

- **mid：类QQ账号的生成策略**

对于用户mid的生成，可以是随机分配，也可以是递加分配，但特殊账号（如连续连号111222、单一数字号66666666）不能分配给用户。

在QQ账号注册中，我们可以发现不同时期的用户账号长度有明显的区间差异，这说明了QQ账号的生成策略使用了号码池这一思想。同时，QQ号面向近5亿的用户数量带来的庞大数据库，其用户全局唯一id严格地避免了随机性导致的碰撞。

因此我们最终的mid生成策略，以前文模仿uuid的唯一id生成策略为基础，借鉴了QQ账号的生成策略。mid的生成剔除了特殊账号，基于数据库中已有mid的最大值，预先生成一批数字，作为一个号码池，注册时从号码池中随机取一个，用过的数字从号码池中删除。当号码池中的号码不够用时可再生成一批，这样对于一批新注册用户仅需对数据库已有mid进行一次查询，减小了开销。

```
// Step 1: forming the number pool
String max = "SELECT MAX(mid) FROM UserRecord";
try (PreparedStatement secondStmt = conn.prepareStatement(max)) {
    try (ResultSet rs = secondStmt.executeQuery()) {
        if (rs.next()) {
            Random rand = new Random();
            mid = rs.getLong(1) + rand.nextLong(100);
        }
    }
}
for (int i = 10001; i <= 10099; i++) {
    long accountNumber = mid + i;
    String insertSql = "INSERT INTO AccountPool (account, status) VALUES (?,
'ok')";
    try (PreparedStatement insertStmt = conn.prepareStatement(insertSql)) {
        insertStmt.setLong(1, accountNumber);
        insertStmt.executeUpdate();
    } catch (SQLException e) {
        e.printStackTrace();
    }
```

```
// Step 2:choosing one number from the pool and use it as mid
long accountNumber = getAccountFromPool(conn);
String sql_insert = "INSERT INTO UserRecord (mid, name, sex, birthday, level,
sign, following, identity, password, qq, wechat, is_deleted) VALUES (?, ?, ?, ?,
?, ?, ?, ?, ?, ?, ?, ?)";
try (PreparedStatement thirdStmt = conn.prepareStatement(sql_insert)) {
    thirdStmt.setLong(1, accountNumber);
    // ......
    thirdStmt.executeUpdate();
}
deleteAccountFromPool(conn, accountNumber);
```

- **BV：类B站BV号的生成策略**

通过观察数据库中已有bv，借鉴bilibili网站中bv号生成策略，我们得知bv的一些基础规则：都由'BV' 开头，除此之外还有10位，第一位是1，第四位是4。

按照上述规则，我们使用了生成随机字符、生成唯一数字和实现base62编码等方法：

`getRandomChars()` 方法： 用于生成随机字符，这里简单地使用UUID的一部分。使用UUID增加了随机性，防止BV号的可预测性。

`getUniqueNumber()` 方法： 用于生成唯一数字，这里简单地使用当前时间戳的后10位。好处是保证了唯一性，尽管这种方式可能在极端情况下会有碰撞，但考虑到我们数据库的大小，碰撞的概率可以基本忽略不计。

`base62Encode()` 方法： 实现base62编码，将唯一数字转换为62进制字符串。这一方法缩短了BV号的长度，提高了可读性和易用性。

```
import java.util.UUID;
public class BVGenerator {
    public static String generateUniqueBV() {
        String randomChars = getRandomChars();
        //construct BV
        String bvNumber = "BV" + base62Encode(getUniqueNumber()) + randomChars;
        return bvNumber;
    }
    private static String getRandomChars() {
        // use part of UUID
        return UUID.randomUUID().toString().replaceAll("-", "").substring(0, 6);
    }
    private static long getUniqueNumber() {
        // forming the unique number
        // using the last ten numbers of current time
        return System.currentTimeMillis() % 1_000_000_000L;
    }
    private static String base62Encode(long number) {
        String base62Chars =
"0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz";
        StringBuilder result = new StringBuilder();
        do {
            result.insert(0, base62Chars.charAt((int) (number % 62)));
            number /= 62;
        } while (number > 0);
        return result.toString();
    }
    public static void main(String[] args) {
        String uniqueBV = generateUniqueBV();
        System.out.println("Unique BV: " + uniqueBV);
    }
}
```

4. sql语句的优化以降低连接池开销

- **with语句的使用**

在 `recommend` 中 `generalRecommendations` 方法中，需要联结包括 `VideoRecord`，`ViewRecord`，`likes`，`favourites`，`coins`，`DanmuRecord` 五个表在内进行查询，如果多次使用 `join` 语句块进行重复执行则成本太高，所以我们使用了 `WITH AS` 短语生成子查询，对于重复查询的表则只要执行一遍即可。另外，如果 `WITH AS` 短语所定义的表名被调用两次以上，则优化器会自动将 `WITH AS` 短语所获取的数据放入一个临时表里。通过这种方式，我们只需使用一个sql语句完成查询，都可以提高速度。在 `big data` 的测试中，查询时间约为2秒。

```
  ->  Sort  (cost=2.52..2.63 rows=44 width=524) (actual time=2129.819..2129.822 rows=44 loops=1)
        Sort Key: (((((grade.like_ratio + grade.coin_ratio) + grade.favourite_ratio) + grade.danmu_ratio) + grade.avg_finish)) DESC, grade...
        Sort Method: quicksort  Memory: 28kB
        ->  CTE Scan on grade  (cost=0.00..1.32 rows=44 width=524) (actual time=2128.581..2129.796 rows=44 loops=1)
Planning Time: 0.879 ms
Execution Time: 2134.139 ms
```

5. 通用性的扩展

- **对用户全部信息的一次性获取**

我们发现，在用户合法性检验之后，经常需要通过用户提供的`(mid, password)`、`qq`、`wechat`三个之一在`UserRecord`表中进行其他信息的查询，若每次都使用`if-else`语句块判断用户提供了哪个信息，无疑会造成代码的冗长，因此我们在`UserImpl`中实现了`construct_full_authinfo`方法，并广泛应用于`UserServiceImpl`，`DanmuServiceImpl`，`VideoServiceImpl`，`RecommenderServiceImpl`，显著减少了重复代码，增加了工程搭建的规范性。

```
public AuthInfo construct_full_authinfo(AuthInfo authInfo, Connection conn) {
        String sql = null;
        PreparedStatement stmt = null;
        ResultSet rs = null;
        try {
            // Determine the query based on provided info
            if (authInfo.getQq() != null && !authInfo.getQq().equals("null")) {
                sql = "SELECT * FROM authinfo WHERE qq = ?";
                stmt = conn.prepareStatement(sql);
                stmt.setString(1, authInfo.getQq());
            } else if (authInfo.getWechat() != null &&
!authInfo.getWechat().equals("null")) {
                sql = "SELECT * FROM authinfo WHERE wechat = ?";
                stmt = conn.prepareStatement(sql);
                stmt.setString(1, authInfo.getWechat());
            } else if (authInfo.getMid() != 0) {
                sql = "SELECT * FROM authinfo WHERE mid = ?";
                stmt = conn.prepareStatement(sql);
                stmt.setLong(1, authInfo.getMid());
            } else {
                // Handle case where no identifying information is provided
                return null; // or throw an exception
            }
            // Execute the query
            rs = stmt.executeQuery();
            if (rs.next()) {
                // Construct a new AuthInfo object from the ResultSet
                return AuthInfo.builder()
                        .mid(rs.getLong("mid"))
                        .password(rs.getString("password"))
                        .qq(rs.getString("qq"))
                        .wechat(rs.getString("wechat"))
                        .build();
            } else {
                return null; // or handle case where no record is found
            }
        } catch (SQLException e) {
            // Handle SQL exception
            e.printStackTrace();
```

```
            return null;
        } finally {
        //
        }
    }
```