一、需求分析

1.1 需求描述

1.1.1 背景调研

随着教育市场的不断发展，家教服务的需求日益增长。为了满足学生对高质量教育资源的需求，以及为家教提供更好的工作平台，我们团队开发了一个家教综合服务平台。该平台旨在整合学生和家教之间的资源，通过线上平台实现更高效、便捷的服务对接。以下是对当前教育市场现状的分析：

- **教育需求多样化**：不同年龄段的学生有不同的学习需求，从基础学科辅导到特长培养，从考试准备到职业规划，家教服务需要覆盖广泛的学习领域。
- **服务质量参差不齐**：市场上存在大量的家教机构和个人家教，但服务质量和服务标准差异较大，缺乏统一的评价体系。
- **信息不对称**：学生难以获取全面准确的家教信息，而家教也面临找到合适学生的挑战。
- **沟通成本高**：传统的家教对接方式往往依赖于电话、微信等即时通讯工具，效率低下且容易遗漏重要信息。

因此，建立一个集成化、功能强大、数据安全的家教服务平台显得尤为重要。该平台不仅可以解决上述问题，还可以为用户提供更加个性化的服务体验，提升整体教育服务质量。

1.1.2 用户构成

家教综合服务平台的用户为以下3类：

- 学生
- 家教
- 管理员

根据调研，总结初步需求如下：

学生	家教	管理员
个人资料管理	个人资料管理	发布公告
寻找家教	寻找学生	用户管理
管理自己的家教	管理自己的学生	帖子审核
获取学习资料	上传学习资料	
评价反馈	接收通知	
接收通知	浏览公告	
浏览公告		

1.1.3 需求总结

详细需求总结如下：

对于**学生**：

1、可以随时完善、修改自己的个人资料，包括性别、年龄、年级、邮箱、联系方式等。

2、可以根据自己对薄弱科目、时间安排、报酬等方面的因素的考虑发布招聘贴招聘家教，也可以去广场查看求聘贴寻找符合条件的家教，可以选择通过或拒绝想和自己建立师生关系的家教的请求。

3、对于已经建立师生关系的家教，学生可以选择与其解除师生关系。

4、可以获取家教上传的学习资料。

5、对于已经建立师生关系的家教，可以随时对其发表评价。

6、可以接收通知，包括发布的帖子是否审核通过、收到家教上传的学习资料等。

7、可以接收并浏览管理员发布的公告。

对于**家教**：

1、可以随时完善、修改自己的个人资料，包括性别、年龄、年级、邮箱、联系方式等。

2、可以根据自己对擅长科目、时间安排、期望薪资等方面的因素的考虑发布求聘贴寻找学生，也可以去广场查看招聘贴寻找符合条件的学生，可以选择通过或拒绝想和自己建立师生关系的学生的请求。

3、对于已经建立师生关系的学生，家教可以选择与其解除师生关系。

4、可以给学生上传学习资料。

5、可以接收通知，包括发布的帖子是否审核通过、收到学生的评价等。

6、可以接收并浏览管理员发布的公告。

对于**管理员**：

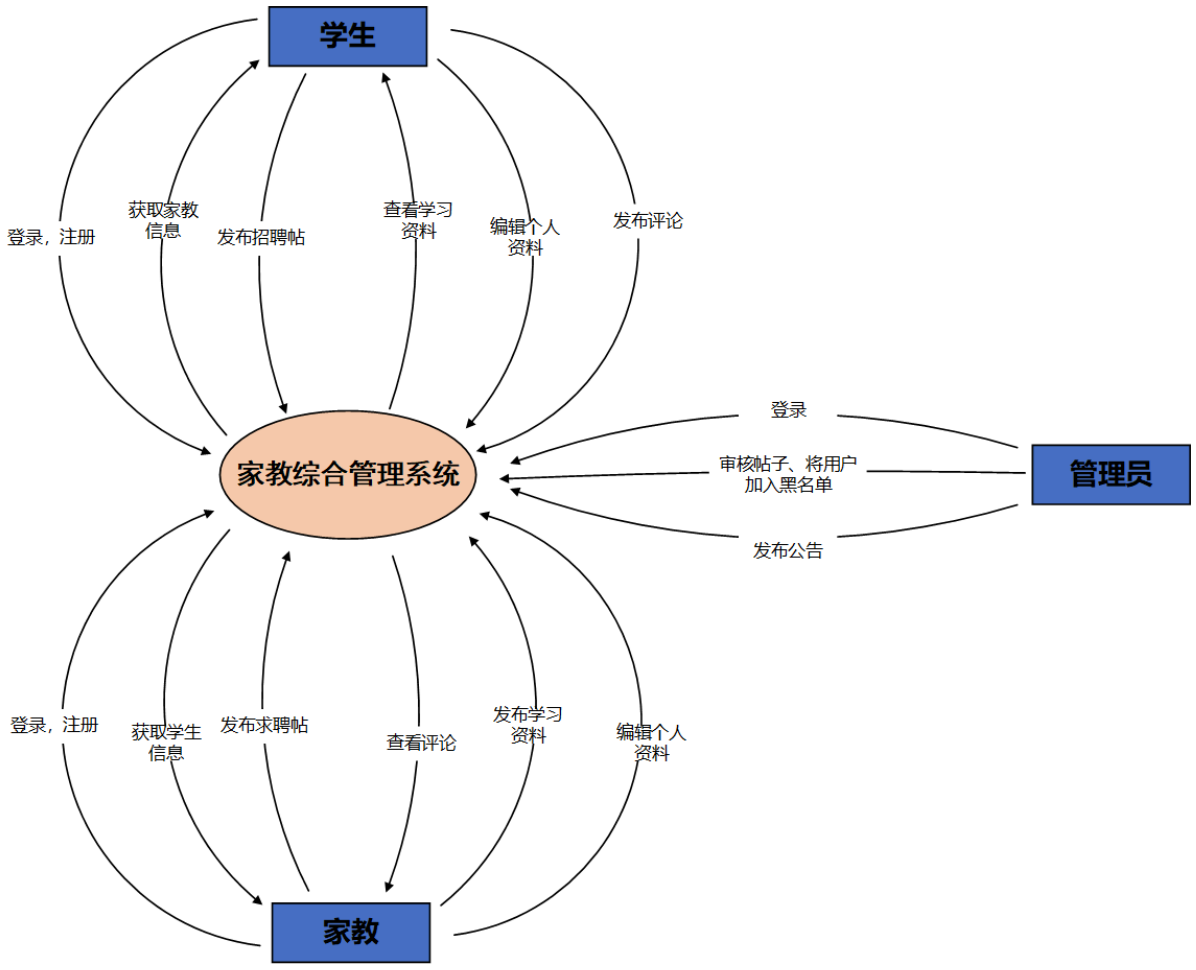
1、可以向全体学生和家教发布公告。

2、可以对用户进行管理，查看其个人主页或将其拉入黑名单。

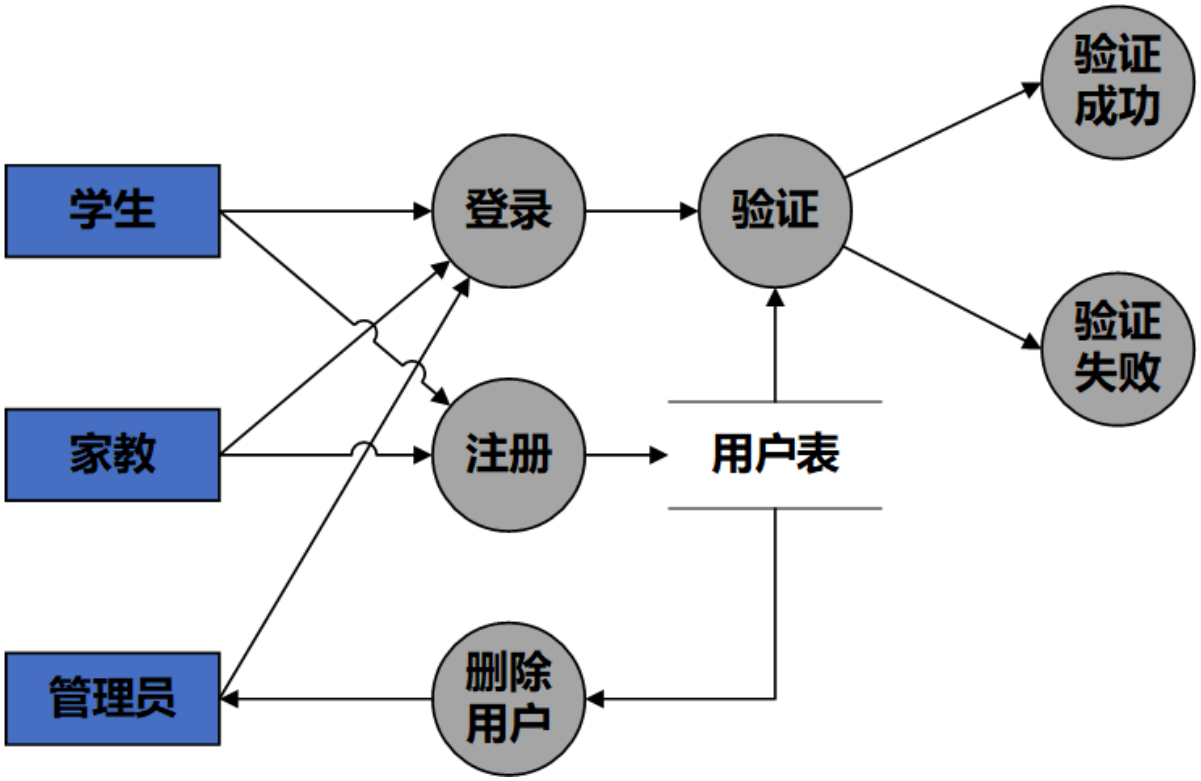
3、可以对未发布的帖子进行审核，只有审核通过的帖子才会发布到对应广场。

1.2 数据流图

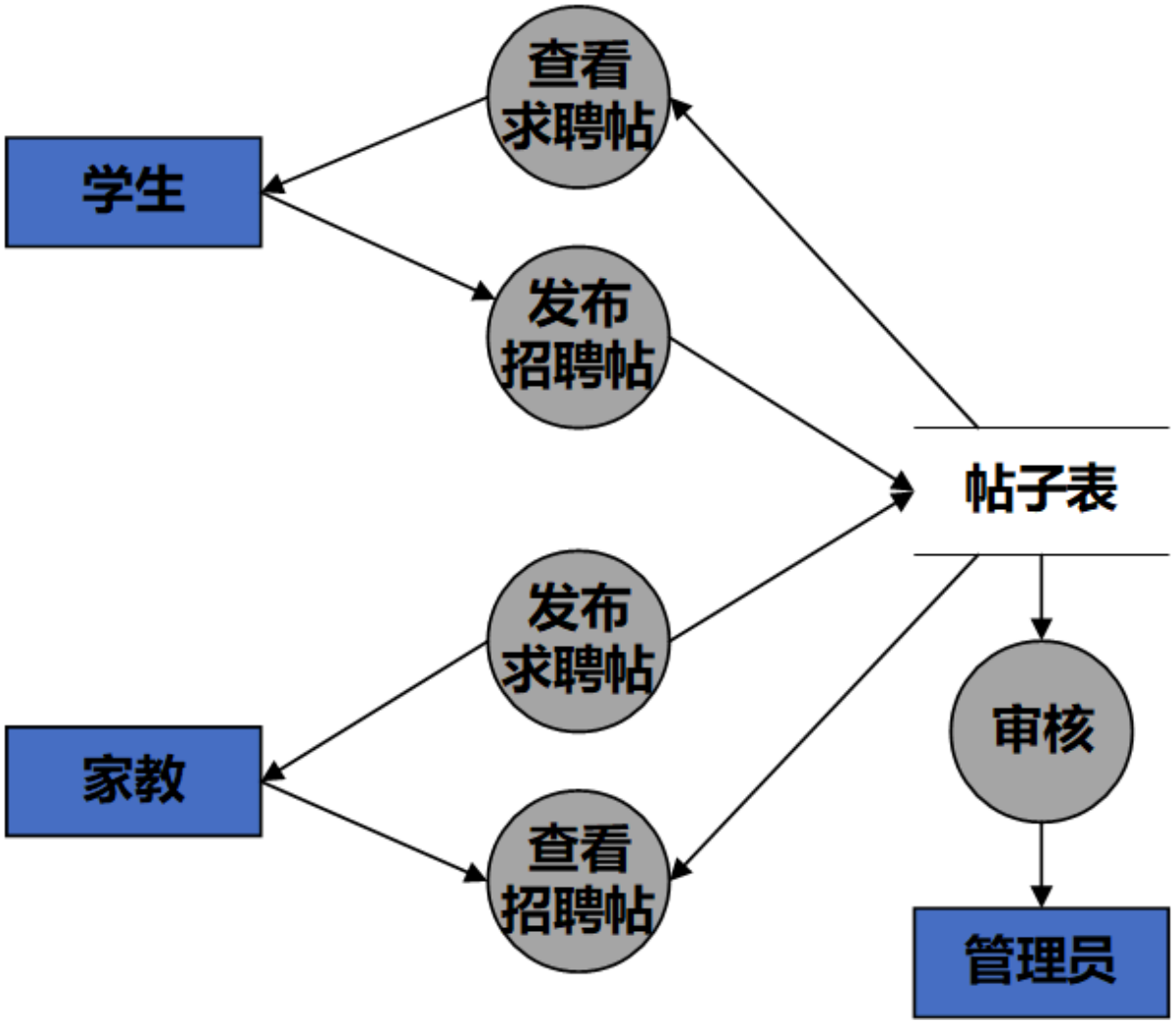
1.2.1 总体数据流图



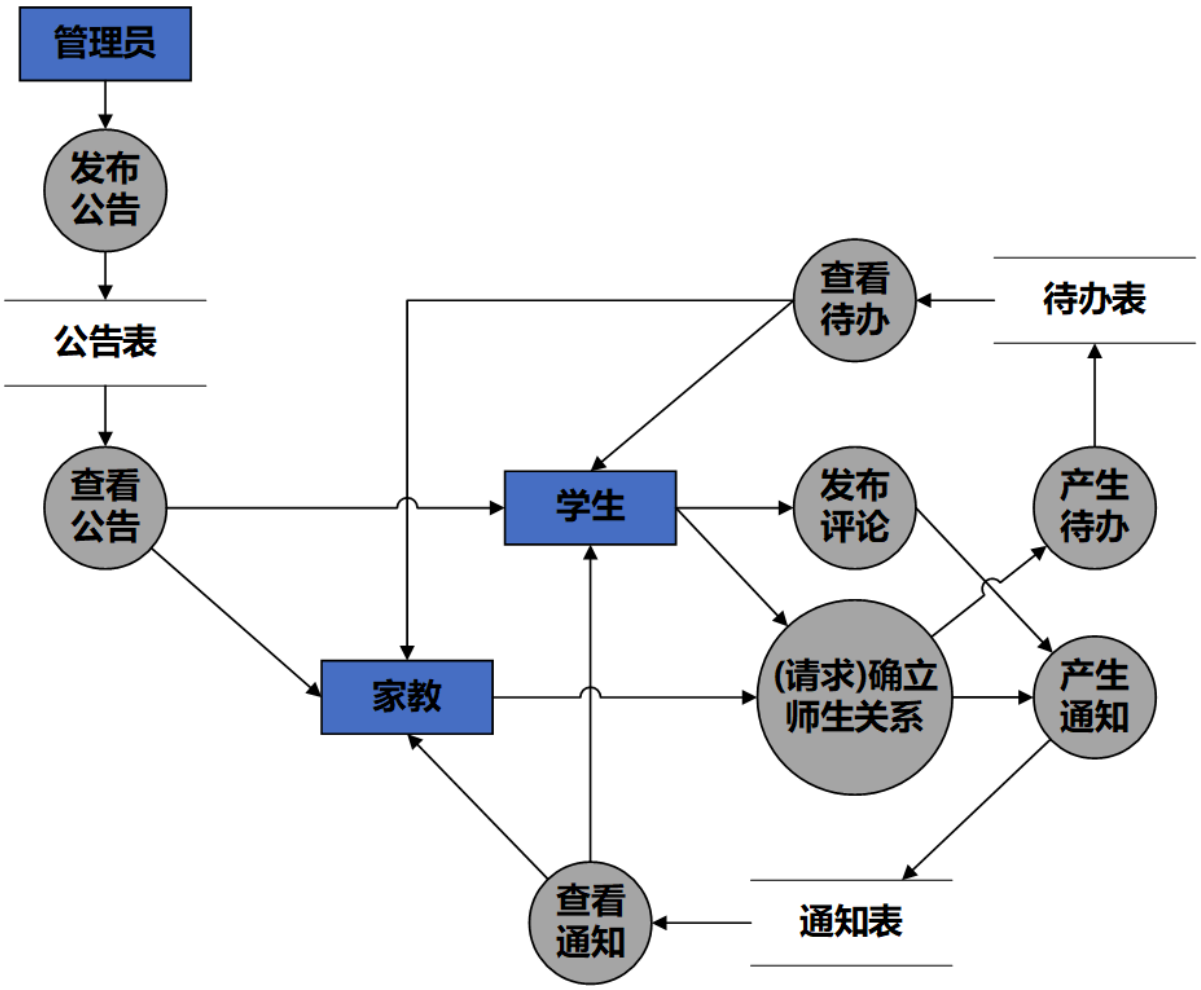
1.2.2 用户管理部分数据流图



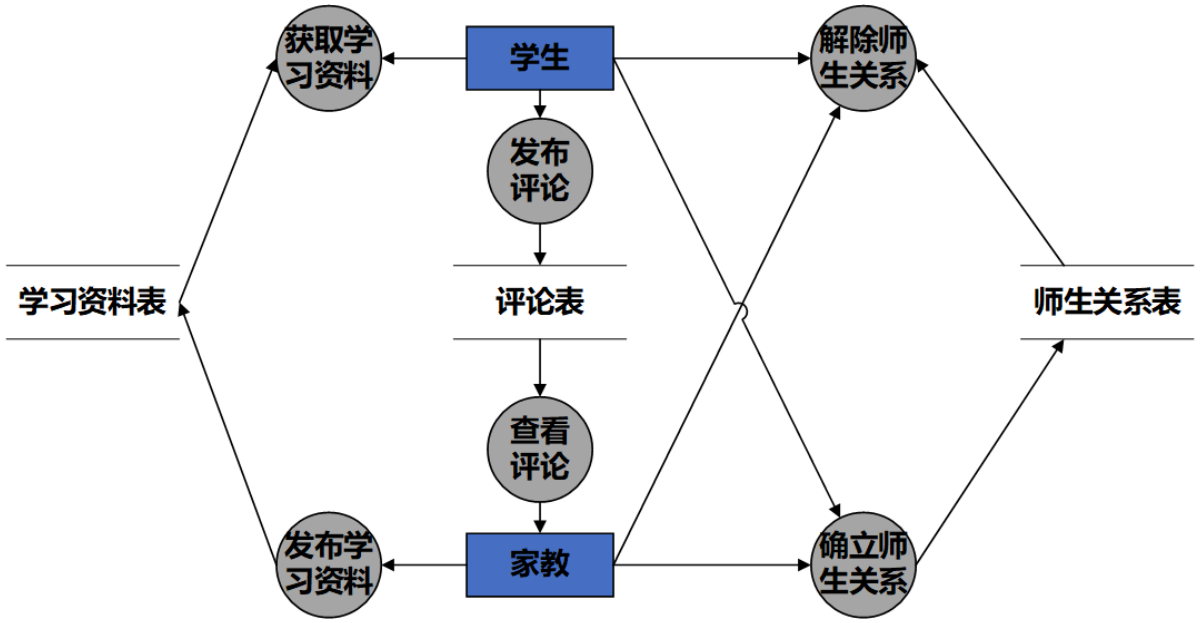
1.2.3 帖子部分数据流图



1.2.4 消息通知部分数据流图



1.2.5 师生互动部分数据流图



1.3 数据元素表

1.3.1 用户管理部分

User

属性名	数据类型	说明
user_id	int	用户编号
username	char	用户名
password	char	用户密码
identity	int	用户身份
registration_date	date	用户注册日期
avatar	char	用户头像

Student

字段名称	数据类型	说明
user_id	int	学生编号
age	int	学生年龄
gender	int	学生性别
contact	varchar	学生联系电话
email	email	学生邮箱
grade	int	学生年级

Tutor

字段名称	数据类型	说明
user_id	int	家教编号
age	int	家教年龄
gender	int	家教性别
contact	varchar	家教联系电话
email	email	家教邮箱
rating	int	家教评分

1.3.2 师生关系管理部分

Link

字段名称	数据类型	说明
link_id	int	师生关系编号
student_id	int	学生编号
tutor_id	int	家教编号

1.3.3 帖子管理部分

Post

字段名称	数据类型	说明
post_id	int	帖子编号
user_id	int	发帖人编号
title	varchar	帖子标题
post_date	date	发帖日期
start_date	date	帖子起始日期
end_date	date	帖子终止日期
content	text	帖子内容
is_completed	boolean	帖子是否被保存
is_approved	boolean	帖子是否被管理员通过

PostSubject

字段名称	数据类型	说明
post_id	int	帖子编号
subject	varchar	帖子包含的科目

1.3.4 其他事务部分

Notification

字段名称	数据类型	说明
notification_id	int	通知编号
user_id	int	接收通知用户编号
notification_date	date	发送通知日期
title	varchar	通知标题

字段名称	数据类型	说明
description	text	通知内容
is_read	boolean	通知是否已读

StudyMaterial‘

字段名称	数据类型	说明
material_id	int	学习材料编号
tutor_id	int	家教编号
student_id	int	学生编号
file_name	varchar	文件名
download_link	varchar	下载链接
upload_ate	date	学习材料上传日期

Review

字段名称	数据类型	说明
review_id	int	评价编号
student_id	int	学生编号
tutor_id	int	家教编号
rating	int	评分
content	text	评价内容
date	date	评价日期

Todo

字段名称	数据类型	说明
todo_id	int	待办编号
owner_id	int	发起者编号
accepter_id	int	接收者编号
accepter_post_id	int	接收者对应帖子的编号
is_completed	int	待办是否完成

Announcement

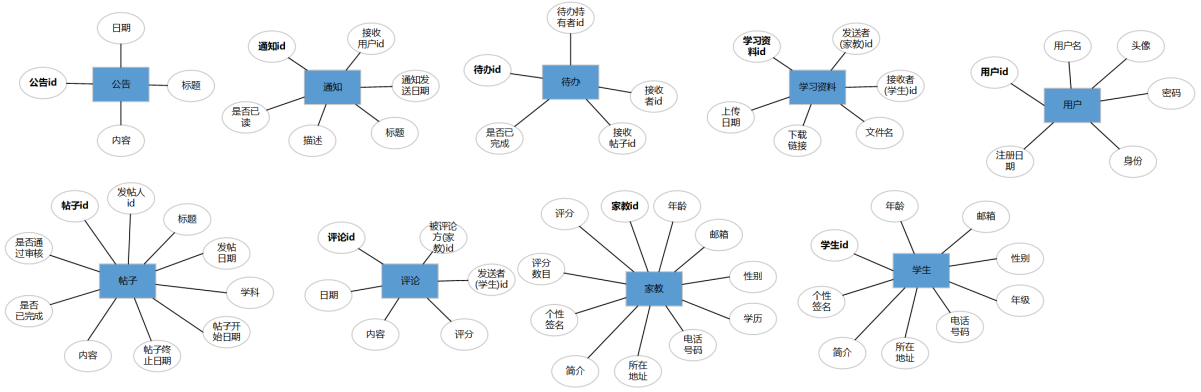
字段名称	数据类型	说明
announcement_id	int	公告编号
announcement_date	date	发布公告日期
title	varchar	公告标题
description	text	公告内容

二、数据库概念模式设计

2.1 系统初步ER图

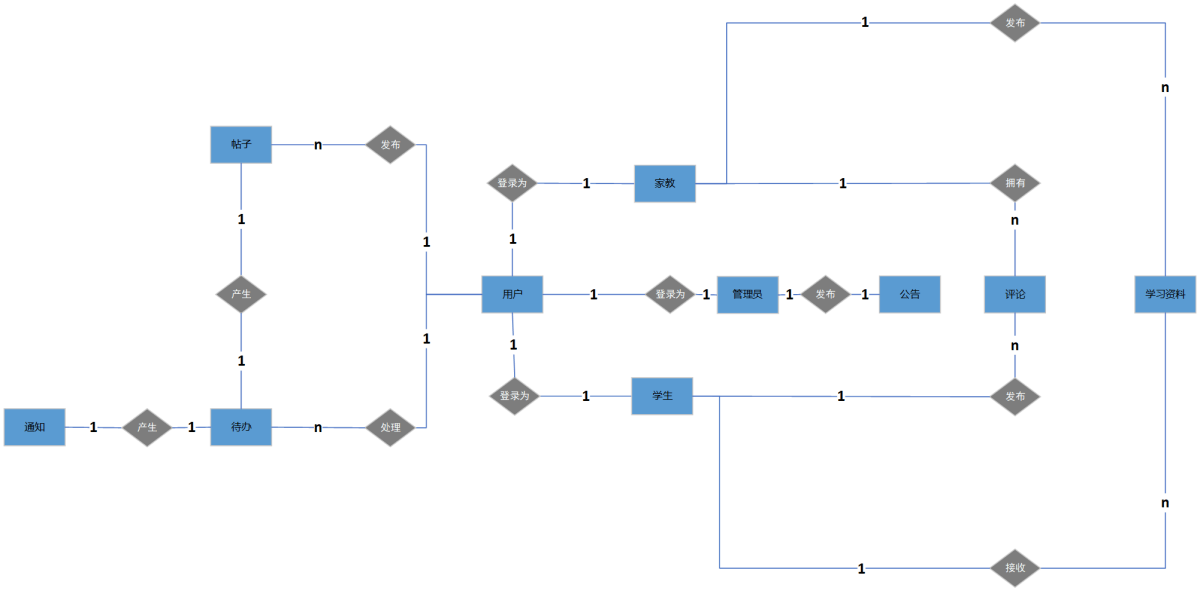
2.1.1 实体ER图

以下是包含主要实体属性的ER图



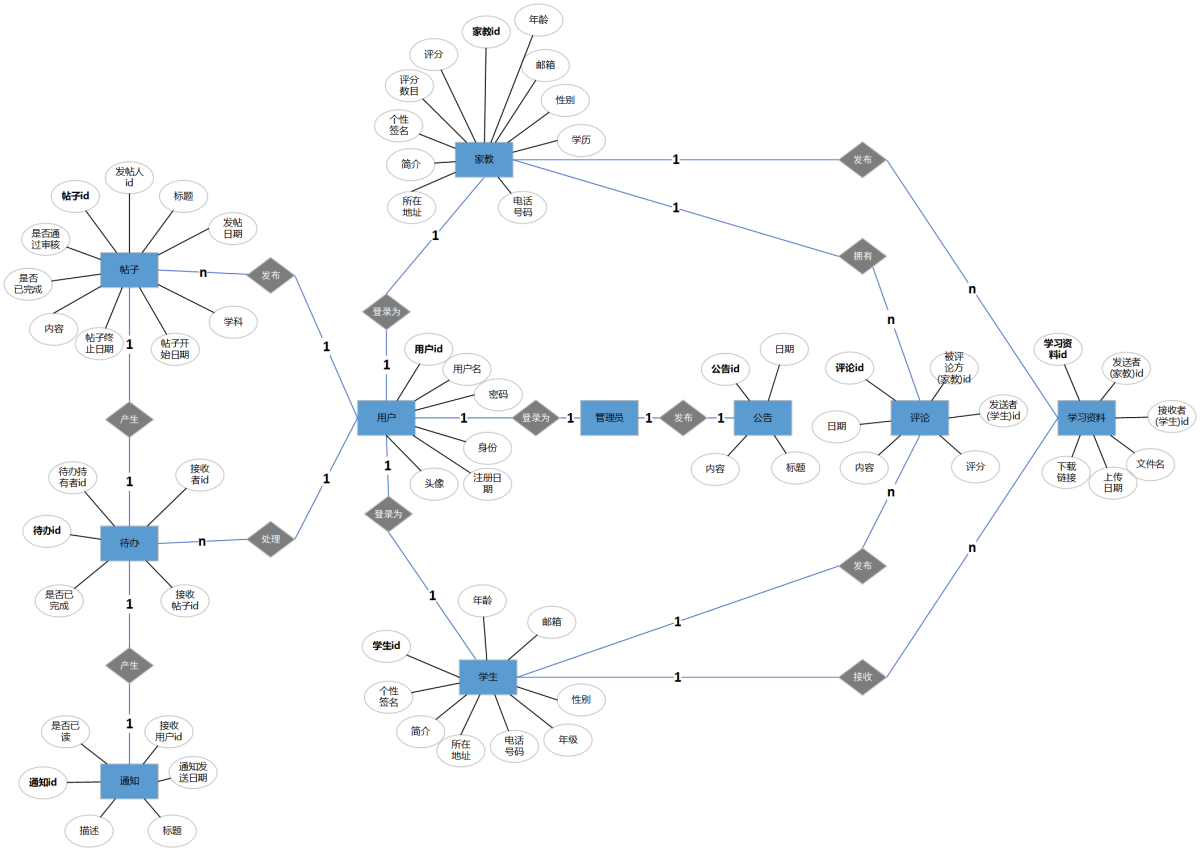
2.1.2 关系ER图

以下是包含主要实体关系的ER图



2.2 系统基本ER图

将实体ER图和关系ER图进行合并，消除冗余后，可以得到系统基本ER图



三、数据库逻辑模式设计与优化

1. 数据库关系模式定义

> **加粗**表示主码，*斜体*表示外码

1. User(**user_id**, user_name, password, identity, registration_date, avatar)
2. Student(**user_id**, age, gender, telephone, email, grade, address, intro, personal_signature)
3. Tutor(**user_id**, age, gender, telephone, email, rate, rate_num, address, intro, personal_signature, degree)
4. Post(**post_id**, *user_id*, title, post_date, start_date, end_date, content, is_completed, is_approved)
5. PostSubject(**post_id**, subject)
6. Notification(**notification_id**, *user_id*, notification_date, title, description, is_read)
7. StudyMaterial(**material_id**, *tutor_id*, *student_id*, file_name, download_link, upload_date)
8. Review(**review_id**, *student_id*, *tutor_id*, rating, content, date)
9. Link(**link_id**, *student_id*, *tutor_id*)
10. Todo(**todo_id**, *owner_id*, *accepter_id*, *accept_post_id*, is_completed)
11. Announcement(**announcement_id**, announcement_date, title, description)

2. 关系模式范式等级的判定与规范化

> 我们旨在将所有关系模式优化到BCNF，以做到规范度与冗余度之间的平衡。我们通过下面3个性质判定BCNF:
>
> - 所有非主属性都完全函数依赖于每个候选码
> - 所有主属性都完全函数依赖于每个不包含它的候选码
> - 没有任何属性完全函数依赖与非码的任何一组属性

- 为了避免冗余叙述，我们将类似的表统一叙述：
  - User, Student, Tutor, PostSubject, Announcement: 这5个关系模式的特点是只有**xx_id**这一个候选码，这个候选码也作为主码。其他的非主属性完全函数依赖于这个主码，且它们彼此之间没有绝对联系。综上所述，它们是BCNF。
  - Post, Notification: 这2个关系模式的特点是**xx_id**作为唯一候选码以及主码，同时有**user_id**作为外码，将user与该关系模式联系起来。我们以Post为例，虽然在发帖页面需要填写联系方式等字段以方便收帖人联系，但这些字段完全函数依赖于user，因此我们去除了这些冗余字段的存储。 剩余的其他属性完全函数依赖于唯一候选码。综上所述，它们是BCNF。
  - StudyMaterial, Review, Link, Todo: 这4个关系模式的特点是**xx_id**作为唯一候选码与主码，同时有**user1_id**, **user2_id**这两个外码，从而使关系模式与两个用户之间的联系关联起来。首先，所有非主属性都完全函数依赖于唯一候选码；其次，需要强调的是在我们的家教平台中师生关系是**多对多**关系，一个学生可以不只有一个老师，一个老师也可以不只有一个学生。因此它们之间也没有完全函数依赖关系。综上所述，它们是BCNF。

3. 数据库关系模式优化

> 在上一章节中我们已经论证了关系模式的规范性。在此基础上我们对于关系模式的设计秉持**简洁**原则，也就是不设计冗余表，不添加冗余属性。查询速度的优化通过合理设置索引实现。

- User只设置与用户**直接关联**的属性，避免在注册时需要填写大量非必要字段。
- Student和Tutor作为User的“子类”，不额外保留User中的属性，同时直接以user_id作为主码。
- Post中去除与User相关的属性，只保留与Post**直接关联**的属性。同时由于帖子的subject为集合，为规避“表中套表”的嫌疑，单独设置PostSubject表。
- 单独设置Link表描述师生关系，避免额外将Student集合设置为Tutor的属性，或将Tutor集合设置为Student属性。
- 合理使用外码，保证数据的关联性并提高访问速度，同时在使用过程中保证数据的参照完整性。在本项目的后端数据库中，各表之间具有完备的外码联系，通过遵循这种外码关系进行实际数据库的建立，将从逻辑的角度保证数据库的设计功能能够完全正确执行到位。

四、数据库物理设计

1. 存取方法

- 在数据存储方面，我们的项目逻辑中没有批量存储等复杂逻辑，因此可以直接使用django框架中自带的存储方法进行数据的存储。除此之外，我们在数据模型定义是注意模型之间的外码关联，使用`on_delete=models.CASCADE`等语句保证了删除数据时数据的一致性。具体示例如下：

```
user = User(username=username,
                    password=password,
                    identity=-1,

 registration_date=datetime.now(pytz.timezone('Asia/Shanghai')))
user.save()
```

```
# 待办事项表 BCNF
class Todo(models.Model):
    # 主码、主属性
    todo_id = models.BigAutoField(primary_key=True)
    # 外码
    owner_id = models.ForeignKey(User,
on_delete=models.CASCADE,related_name='owner_id_todo')
    accepter_id = models.ForeignKey(User,
on_delete=models.CASCADE,related_name='accepter_id_todo')
    accept_post_id = models.ForeignKey(Post,
on_delete=models.CASCADE,related_name='post_id_todo')
    is_completed = models.BooleanField()

    def __str__(self):
        return f"Todo {self.todo_id}"
```

- 在数据查询方面，我们使用索引提高查询速度，具体索引的建立在下一节讨论。除此之外，我们注意避免使用for循环等低效查询方式，普遍使用django提供的**filter**等函数进行数据查询，充分利用DBMS本身对于查询的优化。

```
request_id=int(request.GET.get('id'))
user=User.objects.get(user_id=request_id)
notifications=Notification.objects.filter(user_id=user)
```

2. 索引定义

- 考虑到大多数查询基于主码进行，我们对每个表的主码都建立了索引。
- 对于Post表，在我们的**帖子广场**上需要进行不同字段的频繁查询。因此，我们考虑到用户的查询偏好，就**用户id**，**标题**两个字段建立索引，提高查询速度。

```
# Post表 BCNF
class Post(models.Model):
    # 主码、主属性
    post_id = models.BigAutoField(primary_key=True)
    # 外码
    user_id = models.ForeignKey(User,
on_delete=models.CASCADE,related_name='user_id_post')
    title = models.CharField(max_length=255,blank=True,null=True)
    post_date = models.DateTimeField(blank=True,null=True)
    start_date = models.DateField(blank=True,null=True)
    end_date = models.DateField(blank=True,null=True)
    content = models.TextField(blank=True,null=True)
    is_completed = models.BooleanField()
    is_approved = models.BooleanField(default=False)

```

```
    def __str__(self):
        return self.title

    class Meta:
        indexes = [
            models.Index(fields=['post_id']),
            models.Index(fields=['user_id']),
            models.Index(fields=['title']),
        ]
```

- 有一些数据模型中的数据我们需要在插入时对其按照某种次序排列。例如Notification，用户希望最近的公告能够被最先看到，因此需要将其按照发布时间倒序排列。在这样的字段上建立索引可以极大地提升数据插入速度。

```
# 通知表 BCNF
class Notification(models.Model):
    # 主码、主属性
    notification_id = models.BigAutoField(primary_key=True)
    # 外码
    user_id = models.ForeignKey(User,
on_delete=models.CASCADE,related_name='user_id_notice')
    notificationDate = models.DateTimeField(blank=True,null=True)
    title = models.CharField(max_length=255,blank=True,null=True)
    description = models.TextField(blank=True,null=True)
    is_read = models.BooleanField()

    def __str__(self):
        return self.title

    class Meta:
        ordering = ['-notificationDate']
        indexes = [
            models.Index(fields=['notificationDate']),
        ]
```