

# 数据库工程作业

- 要求：
- 1. 完成一个小型的数据库信息管理系统（或部分功能），并填写工程作业报告；程序和报告请在规定时间之内上传。
  - 2. 开发模式（B/S 或 C/S）、开发高级语言任选，后台数据库使用大型数据库管理系统（SQL Server、Oracle、MySQL 等），不要使用桌面数据库。
  - 3. 报告中所列举的四种操作，每种操作举一个例子即可。
  - 4. 作业成绩按照报告中的标准评分，程序只实现报告中涉及的部分即可。
  - 5. 作业完成后，请将工程作业报告和程序打包提交给助教老师，并联系助教老师进行系统说明和演示，回答相关问题。

## 工程作业报告

### 1. 项目信息（10 分）

学号	2313119	姓名	申健强	专业	计算机科学与技术
项目名称	GenoBase:仿制 NCBI 的生物信息及文献数据库				
必备环境	Flask, python, MySQL				
系统主要功能简介（4 分）	向 reader 提供基本的数据查询功能（文献数据可上传和编辑），向 creator 提供生物信息数据增删查改功能，向 manager 提供 creator 的权限之外和用户状态管理功能（简称封号权限）				
系统主要页面截图（6 分）	<div>主界面：</div>  <div>查询界面：</div>				

基因组数据搜索

搜索

输入搜索关键词...

类别

全部

搜索

高级搜索提示

物种搜索

- 可以使用学名（如 "Escherichia coli"）
- 可以使用常用名（如 "大肠杆菌"）
- 可以使用分类ID

基因搜索

- 可以使用基因名称（如 "lacZ"）
- 可以使用基因符号

蛋白质搜索

- 可以使用蛋白质名称
- 可以使用UniProt ID

文献搜索

- 可以使用标题关键词
- 可以使用作者姓名
- 可以使用期刊名称
- 可以使用DOI标识符

管理界面：

GenoBase

首页

物种

基因

蛋白质

文献

实验数据

管理面板

基因活跃度

基因-物种-蛋白质数量统计

管理面板

用户管理

管理平台用户的权限与状态

进入

物种/基因/蛋白质

管理物种、基因和蛋白质数据

物种管理

基因管理

蛋白质管理

基因-物种-蛋白质数量统计

文献/实验数据

管理文献与实验数据

文献管理

实验数据管理

基因研究活跃度界面：

GenoBase

首页

物种

基因

蛋白质

文献

实验数据

管理面板

基因活跃度

基因-物种-蛋白质数量统计

基因研究活跃度

基因研究活跃度Top20

活跃度分数

Gene\_620

Gene\_571

Gene\_657

Gene\_640

Gene\_704

Gene\_983

Gene\_440

Gene\_534

Gene\_613

Gene\_61

Gene\_201

Gene\_229

Gene\_275

Gene\_320

Gene\_436

Gene\_515

Gene\_743

Gene\_773

Gene\_206

Gene\_532

15

基因ID	基因名称	物种	文献数	实验数	活跃度分数
621	Gene_620	Saccharomyces cerevisiae	8	4	20
572	Gene_571	Saccharomyces cerevisiae	8	2	18

2. 系统配置（10 分）

说明		(2分)请说明系统配置情况(后台数据库,高级语言); (8分)请使用连接串连接高级语言和数据库,并分析字符串的各个部分。			
配置 步骤 2分	DBMS	1. MySQL			
		2. ....			
	高级语言	1.python 作为主要开发语言,使用 flask 作为开发网页的框架, SQLAlchemy 链接数据库在高级语言中进行操作			
		2. ....			
连接串 分析 (6分)		序 号	名称	功能说明	取值
		1	mysql+mysqlconnector	数据库和数据库驱动	mysql+mysqlconnector
		2	Root+Sjq63100	数据库用户和数据库密码	Root+Sjq63100
		3	localhost+3306	数据库位置+端口名	localhost+3306
		4	genobase	数据库名	
连接串代码 (截屏) (2分)		<pre>class Config:     SECRET_KEY = os.environ.get('SECRET_KEY') or 'Sjq63100'     SQLALCHEMY_DATABASE_URI = os.environ.get('DATABASE_URL') or 'mysql+mysqlconnector://root:Sjq63100@localhost/genobase'  SQLALCHEMY_DATABASE_URI = os.environ.get('DATABASE_URL') or 'mysql+mysqlconnector://root:Sjq63100@localhost/genobase'</pre>			
备注					

### 3. 数据库设计 (14分)

说明	(10分)按照数据表的创建顺序,依次给出所涉及数据表的信息,其中参照字段以“(字段1, 字段2, ……, 字段n)”的形式给出,被参照字段以“表名(字段1, 字段2, ……, 字段n)”的形式给出; (4分)一般 DBMS 都可以为数据库生成关系图,请将该图片截屏并粘贴到表格中。				
数据表 (10)	创建顺序	数据表名称	主键	参照属性	被参照表及属性
	1	Users	user_id	-	-
	2	Creators	user_id	user_id	Users.user_id
	3	Managers	user_id	user_id	Users.user_id
	4	Readers	user_id	user_id	Users.user_id
	5	Species	species_id	-	-
	6	Genes	gene_id	species_id	Species.species_id
	7	Proteins	protein_id	gene_id	Genes.gene_id
	8	Publications	publication_id	-	-
	9	Gene_Publications	(gene_id, publication_id)	gene_id	Genes.gene_id
				publication_id	Publications.publication_id
	10	Experimental_Data	experiment_id	gene_id	Genes.gene_id
				publication_id	Publications.publication_id

关系图 (4)



备注 利用 datagrip 生成图，上图我们在生成时包含了表和生成的三个视图，后续我们会详细介绍

#### 4. 含有事务应用的删除操作（13 分）

说明	<p>（1 分）简要说明该操作所要完成的功能；</p> <p>（2 分）该操作会涉及的表（必须含有两张或两张以上的关系表，同时以“表名”的形式给出）</p> <p>（1 分）表连接涉及字段描述（描述方式为“表 1. 属性=表 2. 属性”）</p> <p>（1 分）删除条件涉及的字段描述（以“表名. 属性=? ”形式给出）</p> <p>（4 分）实现该操作的关键代码（高级语言、SQL），截图即可；（其中如果删除语句中不包含任何形式的事务应用将扣除 3 分）</p> <p>（4 分）如何执行该操作，按所述方法能够正常演示程序则给分。</p>	
功能描述 （1 分）	<p>删除指定 species_id 的物种及其所有相关数据：</p> <ul style="list-style-type: none"> <li>• 删除该物种所属的所有实验数据；</li> <li>• 删除该物种所属的所有基因与文献的关联；</li> <li>• 删除该物种所属的所有蛋白质；</li> <li>• 删除该物种所属的所有基因；</li> <li>• 最后删除该物种本身。</li> </ul> <p>整个过程在一个事务（transaction）中执行，保证所有子删除要么全部成功，要么全部回滚，不会留下孤立数据。</p>	
涉及的表 （2 分）	<ul style="list-style-type: none"> <li>• “Species”</li> <li>• “Genes”</li> <li>• “Proteins”</li> <li>• “Gene_Publications”</li> <li>• “Experimental_Data”</li> </ul>	
表连接涉及字段 （1 分）	<ul style="list-style-type: none"> <li>• “Genes.species_id = Species.species_id”</li> <li>• “Proteins.gene_id = Genes.gene_id”</li> <li>• “Gene_Publications.gene_id = Genes.gene_id”</li> <li>• “Experimental_Data.gene_id = Genes.gene_id”</li> </ul>	
删除条件字段描述 （1 分）	字段	规则
	Species.species_id	其中 ? 表示传入的目标 species_id 参数，删除 id 为 id 物种数据
	.....	.....

代码  
(4  
分)

```
1 create
2 definer = root@localhost procedure DeleteSpeciesWithRelatedData(IN p_species_id int)
3 BEGIN
4     DECLARE gene_count INT;
5
6     -- 检查物种是否存在
7     SELECT COUNT(*) INTO gene_count FROM Species WHERE species_id = p_species_id;
8
9     IF gene_count = 0 THEN
10         SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Species ID not found.';
11     ELSE
12         START TRANSACTION;
13
14         -- 删除该物种相关的实验数据
15         DELETE FROM Experimental_Data
16         WHERE gene_id IN (SELECT gene_id FROM Genes WHERE species_id = p_species_id);
17
18         -- 删除该物种基因与文献的关联
19         DELETE FROM Gene_Publications
20         WHERE gene_id IN (SELECT gene_id FROM Genes WHERE species_id = p_species_id);
21
22         -- 删除该物种相关的蛋白质
23         DELETE FROM Proteins
24         WHERE gene_id IN (SELECT gene_id FROM Genes WHERE species_id = p_species_id);
25
26         -- 删除该物种的基因
27         DELETE FROM Genes WHERE species_id = p_species_id;
28
29         -- 最后删除物种本身
30         DELETE FROM Species WHERE species_id = p_species_id;
31
32         COMMIT;
33
34         SELECT 'Species and all related data deleted successfully' AS result;
35     END IF;
36 END;
37
38 def delete_species(id):
39     check_admin_access()
40     if current_user.is_manager() and current_user.manager.access_level != 'full':
41         flash('您没有删除物种的权限')
42         return redirect(url_for('main.species_detail', id=id))
43
44     # 确保物种存在, 否则 404
45     Species.query.get_or_404(id)
46
47     try:
48         # 直接调用存储过程 (当前 Session 已经有事务, 这里直接 execute)
49         db.session.execute(
50             text("CALL DeleteSpeciesWithRelatedData(:sid)"),
51             {"sid": id}
52         )
53         # 存储过程内部要么 commit, 要么因 SIGNAL 抛异常
54         # 这里只再做一次 commit, 把实体 Session 的事务提交
55         db.session.commit()
56     except Exception as e:
57         # 如果有任何异常, 回滚当前 Session 的事务
58         db.session.rollback()
59         flash(f'删除失败: {str(e)}')
60         return redirect(url_for('main.species_detail', id=id))
61
62     flash('物种及其所有关联数据已删除')
63     return redirect(url_for('main.species_list'))
```

程序演示  
(4分)

ID	学名	常用名	分类ID	基因数量	操作
4	Drosophila melanogaster	Fruit fly	7227	196	<a href="#">查看</a> <a href="#">编辑</a>
1	Homo sapiens	Human	9606	212	<a href="#">查看</a> <a href="#">编辑</a>
2	Mus musculus	Mouse	10090	196	<a href="#">查看</a> <a href="#">编辑</a>
3	Rattus norvegicus	Rat	10116	213	<a href="#">查看</a> <a href="#">编辑</a>
5	Saccharomyces cerevisiae	Baker's yeast	4932	183	<a href="#">查看</a> <a href="#">编辑</a>
9	test_s	test	9999	1	<a href="#">查看</a> <a href="#">编辑</a>

test\_s

test

[编辑物种](#) [删除物种](#)

物种信息

<table><tr><td>ID</td><td>9</td></tr><tr><td>学名</td><td>test_s</td></tr><tr><td>常用名</td><td>test</td></tr><tr><td>分类ID</td><td>9999</td></tr><tr><td>添加时间</td><td>2025-06-01 11:53:19</td></tr><tr><td>更新时间</td><td>2025-06-01 11:53:19</td></tr></table>	ID	9	学名	test_s	常用名	test	分类ID	9999	添加时间	2025-06-01 11:53:19	更新时间	2025-06-01 11:53:19	<table><tr><td>描述</td></tr><tr><td>测试用物种</td></tr></table>	描述	测试用物种
ID	9														
学名	test_s														
常用名	test														
分类ID	9999														
添加时间	2025-06-01 11:53:19														
更新时间	2025-06-01 11:53:19														
描述															
测试用物种															

基因列表

[添加基因](#)

ID	基因名称	基因符号	染色体	蛋白质数量	操作
1003	gene_test	3	chr5	1	<a href="#">查看</a> <a href="#">编辑</a>

gene\_test

3

[编辑基因](#) [删除](#)

基因信息

<table><tr><td>ID</td><td>1003</td></tr><tr><td>基因名称</td><td>gene_test</td></tr><tr><td>基因符号</td><td>3</td></tr><tr><td>染色体</td><td>chr5</td></tr><tr><td>起始位置</td><td>998</td></tr><tr><td>终止位置</td><td>24435</td></tr><tr><td>链</td><td>+</td></tr><tr><td>所属物种</td><td>test_s</td></tr><tr><td>添加时间</td><td>2025-06-01 11:53:49</td></tr><tr><td>更新时间</td><td>2025-06-01 11:53:49</td></tr></table>	ID	1003	基因名称	gene_test	基因符号	3	染色体	chr5	起始位置	998	终止位置	24435	链	+	所属物种	test_s	添加时间	2025-06-01 11:53:49	更新时间	2025-06-01 11:53:49	<table><tr><td>序列</td></tr><tr><td>ATCGATCG</td></tr></table>	序列	ATCGATCG
ID	1003																						
基因名称	gene_test																						
基因符号	3																						
染色体	chr5																						
起始位置	998																						
终止位置	24435																						
链	+																						
所属物种	test_s																						
添加时间	2025-06-01 11:53:49																						
更新时间	2025-06-01 11:53:49																						
序列																							
ATCGATCG																							

相关蛋白质

ID	蛋白质名称	UniProt ID	操作
803	test_prot	test_id	<a href="#">查看</a>

相关文献

[AAA](#) (2013)

相关实验数据

[AAA](#)

原始情况，现在删除 test\_s 物种：

物种及其相关数据已成功删除

×

物种列表

添加物种

ID	学名	常用名	分类ID	基因数量	操作
4	Drosophila melanogaster	Fruit fly	7227	197	<div>查看编辑</div>
1	Homo sapiens	Human	9606	212	<div>查看编辑</div>
2	Mus musculus	Mouse	10090	196	<div>查看编辑</div>
3	Rattus norvegicus	Rat	10116	213	<div>查看编辑</div>
5	Saccharomyces cerevisiae	Baker's yeast	4932	183	<div>查看编辑</div>

其关联的基因和实验数据等都消失，值得注意的是：

文献详情

编辑文献

aaaa

作者：author

期刊：nature

发表年份：2051

DOI：xxxx

关联基因

Gene\_0

物种：Mus musculus

G0

编辑

删除

返回列表

一篇文献关联了这个物种涉及到的基因时，只会消除掉这个基因及其链接，但不会消除掉这个文献

备注



### 5. 触发器控制下的添加操作 (20 分)

说明	(1分) 简要说明该操作所要完成的功能; (2分) 简要说明该触发器所要完成的功能 (1分) 该操作会涉及的表(以“表名”的形式给出)。 (2分) 该操作输入数据以及输入数据应该满足的条件,如:数值范围、是否为空; (6分) 实现该操作的关键代码(高级语言、SQL), 截图即可; (8分) 如何执行该操作, 按所述方法能够正常演示程序则给分。	
功能描述 (1分)	在向 <b>Proteins</b> 或 <b>Experimental_Data</b> 表插入新记录时, 必须先校验所引用的 <code>gene_id</code> 是否在 <b>Genes</b> 表中真实存在; 若不存在, 则阻止该插入操作并报错。这样可保证数据库中所有蛋白质或实验数据都只关联到有效的基因, 而不会出现“关联了一个并不存在的基因 ID”的不一致情况。	
触发器描述 (2分)	<ul style="list-style-type: none"><li>• <b>before_protein_insert</b> (针对 <b>Proteins</b> 表)</li></ul> 在向 <b>Proteins</b> 中执行 <b>INSERT</b> 之前自动触发。 查询 <b>Genes</b> 表中是否存在 <code>NEW.gene_id</code> 对应的记录。 如果查询结果为 0, 即该基因 ID 不存在, 则执行 <code>SIGNAL SQLSTATE '45000'</code> 并抛出消息 “Cannot add protein: Associated gene_id does not exist.”, 阻止插入。 <ul style="list-style-type: none"><li>• <b>before_experiment_insert</b> (针对 <b>Experimental_Data</b> 表)</li></ul> 在向 <b>Experimental_Data</b> 表执行 <b>INSERT</b> 之前自动触发。 如果 <code>NEW.gene_id</code> 不为空, 则查询 <b>Genes</b> 表中是否存在相应基因 ID。 如果找不到, 则执行 <code>SIGNAL SQLSTATE '45000'</code> 并抛出消息 “Cannot add experiment: Associated gene_id does not exist.”, 阻止插入; 如果 <code>gene_id</code> 为空或存在, 则允许插入。	
涉及的表 (1分)	Proteins Genes Experimental_Data (针对实验数据插入场景) 注: 第一条触发器 ( <code>before_protein_insert</code> ) 涉及的表为 “Proteins” 与 “Genes”。 第二条触发器 ( <code>before_experiment_insert</code> ) 涉及的表为 “Experimental_Data” 与 “Genes”。	
输入数据 (2分)	字段	规则
	Proteins.gene_id	正整数, 外键约束指向 <code>`Genes(gene_id)`</code> , 必须在 <code>`Genes`</code> 表中存在, 否则触发器 <code>`before_protein_insert`</code> 拒绝插入
	Experimental_Data.gene_id	如果不为空必须是正整数外键约束指向 <code>`Genes(gene_id)`</code> 若不为空必须在 <code>`Genes`</code> 表中存在, 否则触发器 <code>`before_experiment_insert`</code> 拒绝插入
插入操作源码 (3分)	<pre>def add_experiment():     check_admin_access()     form = ExperimentalDataForm()     form.gene_id.choices = [(0, '-- 不关联基因 --')] + [(g.gene_id, f'{g.gene_name} ({g.gene_symbol})') for g in Genes.query.order_by(Genes.gene_name).all()]     form.publication_id.choices = [(0, '-- 不关联文献 --')] + [(p.publication_id, p.title) for p in Publications.query.order_by(Publications.title).all()]      if form.validate_on_submit():         try:             gene_id = form.gene_id.data if form.gene_id.data else None             # 这里不强制检查 gene_id 是否存在, 让触发器兜底             experiment = Experimental_Data(                 experiment_type=form.experiment_type.data,                 conditions=form.conditions.data,                 results=form.results.data,                 gene_id=gene_id,                 publication_id=form.publication_id.data if form.publication_id.data != 0 else None             )             db.session.add(experiment)             db.session.commit()             flash('实验数据添加成功', 'success')             return redirect(url_for('main.experiment_detail', id=experiment.experiment_id))         except (IntegrityError, DatabaseError) as e:             db.session.rollback()             error_msg = str(e.orig) if hasattr(e, 'orig') else str(e)             if 'Cannot add experiment: Associated gene_id does not exist' in error_msg:                 flash('无法添加实验数据, 所填基因ID不存在, 已被数据库触发器拦截。', 'danger')             else:                 flash(f'数据库错误: {error_msg}', 'danger')         except Exception as e:             db.session.rollback()             flash(f'发生未知错误: {str(e)}', 'danger')     return render_template('admin/experiment_form.html', title='添加实验数据', form=form) </pre>	

触发器 源码 (3分)	<pre>-- 1.1 在添加蛋白质前验证关联的基因是否存在 DELIMITER \$\$ CREATE TRIGGER before_protein_insert BEFORE INSERT ON Proteins FOR EACH ROW BEGIN     DECLARE gene_exists INT;     SELECT COUNT(*) INTO gene_exists FROM Genes WHERE gene_id = NEW.gene_id;     IF gene_exists = 0 THEN         SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Cannot add protein: Associated gene_id does not exist.';     END IF; END\$\$ DELIMITER ;  -- 1.2 在添加实验数据前验证关联的基因是否存在 DELIMITER \$\$ CREATE TRIGGER before_experiment_insert BEFORE INSERT ON Experimental_Data FOR EACH ROW BEGIN     IF NEW.gene_id IS NOT NULL THEN         DECLARE gene_exists INT;         SELECT COUNT(*) INTO gene_exists FROM Genes WHERE gene_id = NEW.gene_id;         IF gene_exists = 0 THEN             SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Cannot add experiment: Associated gene_id does not exist.';         END IF;     END IF; END\$\$ DELIMITER ;</pre>																																		
程序演示 (4分)	<div>说明：不违背触发器能够执行插入操作</div> <div><div><div>蛋白质添加成功</div><div>首页 / 蛋白质 / test_prot</div><div>test_prot</div><div>UniProt: test_id</div><div>蛋白质信息</div><table><tr><td>ID</td><td>808</td><td>氨基酸序列</td></tr><tr><td>蛋白质名称</td><td>test_prot</td><td>LLLLLLLL</td></tr><tr><td>UniProt ID</td><td>test_id</td><td></td></tr><tr><td>关联基因</td><td>Gene_99</td><td></td></tr><tr><td>添加时间</td><td>2025-06-01 14:11:35</td><td></td></tr><tr><td>更新时间</td><td>2025-06-01 14:11:35</td><td></td></tr></table></div><div><div>实验数据添加成功</div><div>首页 / 实验数据 / AAA</div><div>AAA</div><div>实验数据信息</div><table><tr><td>ID</td><td>2004</td></tr><tr><td>实验类型</td><td>AAA</td></tr><tr><td>实验条件</td><td>SSSSS</td></tr><tr><td>实验结果</td><td>SSSSSSSS</td></tr><tr><td>关联基因</td><td>Gene_8</td></tr><tr><td>关联文献</td><td>未设置</td></tr><tr><td>添加时间</td><td>2025-06-01 14:30:03</td></tr><tr><td>更新时间</td><td>2025-06-01 14:30:03</td></tr></table></div></div>	ID	808	氨基酸序列	蛋白质名称	test_prot	LLLLLLLL	UniProt ID	test_id		关联基因	Gene_99		添加时间	2025-06-01 14:11:35		更新时间	2025-06-01 14:11:35		ID	2004	实验类型	AAA	实验条件	SSSSS	实验结果	SSSSSSSS	关联基因	Gene_8	关联文献	未设置	添加时间	2025-06-01 14:30:03	更新时间	2025-06-01 14:30:03
ID	808	氨基酸序列																																	
蛋白质名称	test_prot	LLLLLLLL																																	
UniProt ID	test_id																																		
关联基因	Gene_99																																		
添加时间	2025-06-01 14:11:35																																		
更新时间	2025-06-01 14:11:35																																		
ID	2004																																		
实验类型	AAA																																		
实验条件	SSSSS																																		
实验结果	SSSSSSSS																																		
关联基因	Gene_8																																		
关联文献	未设置																																		
添加时间	2025-06-01 14:30:03																																		
更新时间	2025-06-01 14:30:03																																		
程序演示 (4分)	<div>说明：违背触发器要求，不能够执行插入操作，系统报错。</div> <div><div><div>无法添加蛋白质：所填基因ID不存在，已被数据库触发器拦截。</div><div>无法添加蛋白质：所填基因ID不存在，已被数据库触发器拦截。</div><div>添加蛋白质</div><div>蛋白质名称</div><div>test_prot</div><div>UniProt ID</div><div>test_id</div><div>关联基因</div><div>10086</div><div>氨基酸序列</div><div>ASDF</div><div>添加实验数据</div><div>实验类型</div><div>AAA</div><div>实验条件</div><div>AAAAA</div><div>实验结果</div><div>AAAAA</div><div>关联基因</div><div>10086</div><div>关联文献</div><div>...</div></div><div><div>数据库错误: 1452 (23000): Cannot add or update a child row: a foreign key constraint fails ('genobase','experimental_data', CONSTRAINT 'experimental_data_ibfk_1' FOREIGN KEY ('gene_id') REFERENCES 'genes' ('gene_id'))</div></div></div>																																		
备注																																			

## 6. 存储过程控制下的更新操作（18 分）

说明	<p>(1 分) 简要说明该操作所要完成的功能；</p> <p>(1 分) 简要说明该存储过程所要完成的功能；</p> <p>(2 分) 说明该操作涉及操作的表（必须包含两张或两张以上的关系表，以“表名形式”描述）</p> <p>(1 分) 表连接涉及字段描述（描述方式为“表 1. 属性=表 2. 属性”）</p> <p>(2 分) 该操作会修改字段（以“表名. 字段名”的形式给出），以及修改规则，如新数值的计算方法、在何种条件下予以修改等；</p> <p>(6 分) 实现该操作的关键代码（高级语言、SQL），截图即可；</p> <p>(5 分) 如何执行该操作，按所述方法能够正常演示程序则给分。</p>	
功能描述 (1 分)	通过存储过程，实现对特定生物实体（物种或基因）信息的原子性更新，并在更新失败时回滚事务，保证数据一致性。	
存储过程功能描述 (1 分)	<p>UpdateSpeciesInfo: 检索给定 species_id 对应的旧物种信息，若存在则按传入参数更新 Species 表中的 scientific_name、common_name、description 和 updated_at 字段；若对应记录不存在，则回滚并抛出错误。</p> <p>UpdateGeneAndProteinInfo: 检索给定 gene_id 对应的旧基因名称，若存在则更新 Genes 表中的 gene_name，并将已更新的基因名称信息附加到该基因关联的所有 Proteins 表记录的 protein_name 字段中；若基因不存在或更新失败，则回滚并抛出错误。</p>	
涉及的关系表 (2 分)	<p>Species</p> <p>Genes</p> <p>Proteins</p> <p>说明：上述三个表均为关系型表，其中：</p> <p>UpdateSpeciesInfo 仅操作 Species 表；</p> <p>UpdateGeneAndProteinInfo 同时操作 Genes 表和 Proteins 表。</p>	
表连接涉及字段(1)	Genes.gene_id = Proteins.gene_id	
更改字段 (2 分)	字段	规则
	Genes.gene_name	按输入参数 p_new_gene_name 更新，条件：查询到对应 gene_id，且原名称不为空。
	Proteins.protein_name	将所有当前与该 gene_id 关联的 protein_name 中末尾符合正则 \ (Gene renamed from .*\?) 的后缀去除；条件：上一步基因更新成功。
	Proteins.protein_name	再对上述更新后的 protein_name 追加字符串 (Gene renamed from <old_gene_name> to <new_gene_name>)；
	Species.scientific_name Species.common_name Species.description	若传入非空，则用该值替换；否则保留原值。
	Species.updated_at	无条件更新为 CURRENT_TIMESTAMP
更新代码	(截屏)	

(3  
分)

```
def edit_gene(id):
    check_admin_access()
    gene = Genes.query.get_or_404(id)
    form = GeneForm()
    form.species_id.choices = [(s.species_id, s.scientific_name) for s in Species.query.order_by(Species.scientific_name).all()]

    if form.validate_on_submit():
        # 清理序列
        sequence = re.sub(r'\s+', '', form.sequence.data)

        # 如果基因名称发生变化, 调用存储过程
        if gene.gene_name != form.gene_name.data:
            try:
                # 调用存储过程更新基因名称和关联的蛋白质信息
                result = db.session.execute(
                    text('CALL UpdateGeneAndProteinInfo(:gene_id, :new_gene_name)'),
                    {'gene_id': gene.gene_id, 'new_gene_name': form.gene_name.data}
                )
                db.session.commit()

                # 获取存储过程的返回结果
                for row in result:
                    flash(row[0], 'success')

            except DatabaseError as e:
                db.session.rollback()
                error_msg = str(e.orig) if hasattr(e, 'orig') else str(e)
                if 'Gene ID not found' in error_msg:
                    flash('错误: 找不到指定的基因ID', 'error')
                elif 'Failed to update gene' in error_msg:
                    flash('错误: 更新基因失败', 'error')
                else:
                    flash(f'数据库错误: {error_msg}', 'error')
                return render_template('admin/gene_form.html', title='编辑基因', form=form)
            except Exception as e:
                db.session.rollback()
                flash(f'发生未知错误: {str(e)}', 'error')
                return render_template('admin/gene_form.html', title='编辑基因', form=form)

        # 更新其他字段
        try:
            gene.gene_symbol = form.gene_symbol.data
            gene.sequence = sequence
            gene.chromosome = form.chromosome.data
            gene.start_position = form.start_position.data
            gene.end_position = form.end_position.data
            gene.strand = form.strand.data
            gene.species_id = form.species_id.data

            db.session.commit()
            flash('基因信息已更新', 'success')
            return redirect(url_for('main.gene_detail', id=gene.gene_id))
        except Exception as e:
            db.session.rollback()
            flash(f'更新基因信息时出错: {str(e)}', 'error')

    elif request.method == 'GET':
        form.gene_name.data = gene.gene_name
        form.gene_symbol.data = gene.gene_symbol
        form.sequence.data = gene.sequence
        form.chromosome.data = gene.chromosome
        form.start_position.data = gene.start_position
        form.end_position.data = gene.end_position
        form.strand.data = gene.strand
        form.species_id.data = gene.species_id
    return render_template('admin/gene_form.html', title='编辑基因', form=form)
```

```
def edit_species(id):
    check_admin_access()
    species = Species.query.get_or_404(id)
    form = SpeciesForm()
    if form.validate_on_submit():
        try:
            # 调用存储过程更新物种信息
            result = db.session.execute(
                text('CALL UpdateSpeciesInfo(:species_id, :scientific_name, :common_name, :description)'),
                {
                    'species_id': id,
                    'scientific_name': form.scientific_name.data,
                    'common_name': form.common_name.data,
                    'description': form.description.data
                }
            )
            db.session.commit()

            # 获取存储过程的返回结果
            for row in result:
                flash(row[0], 'success')

            return redirect(url_for('main.species_detail', id=species.species_id))

        except DatabaseError as e:
            db.session.rollback()
            error_msg = str(e.orig) if hasattr(e, 'orig') else str(e)
            if 'Species ID not found' in error_msg:
                flash('错误: 找不到指定的物种ID', 'error')
            else:
                flash(f'数据库错误: {error_msg}', 'error')
            return render_template('admin/species_form.html', title='编辑物种', form=form)
        except Exception as e:
            db.session.rollback()
            flash(f'发生未知错误: {str(e)}', 'error')
            return render_template('admin/species_form.html', title='编辑物种', form=form)

    elif request.method == 'GET':
        form.scientific_name.data = species.scientific_name
        form.common_name.data = species.common_name
        form.taxonomy_id.data = species.taxonomy_id
        form.description.data = species.description
        return render_template('admin/species_form.html', title='编辑物种', form=form)
```

(截屏)

```
DELIMITER $$
CREATE PROCEDURE UpdateGeneAndProteinInfo (
    IN p_gene_id INT,
    IN p_new_gene_name VARCHAR(255)
)
BEGIN
    DECLARE old_gene_name VARCHAR(255);
    DECLARE affected_rows INT;
    DECLARE base_protein_name VARCHAR(100);

    -- 开始事务
    START TRANSACTION;

    -- 获取旧的基因名称
    SELECT gene_name INTO old_gene_name FROM Genes WHERE gene_id = p_gene_id;

    IF old_gene_name IS NOT NULL THEN
        -- 更新基因名称
        UPDATE Genes SET gene_name = p_new_gene_name WHERE gene_id = p_gene_id;
        SET affected_rows = ROW_COUNT();

        IF affected_rows > 0 THEN
            -- 更新关联的蛋白质描述
            -- 首先获取蛋白质的基本名称 (去除之前的重命名信息)
            UPDATE Proteins
            SET protein_name = REGEXP_REPLACE(
                protein_name,
                '\\(Gene renamed from .*\\)$',
                ''
            )
            WHERE gene_id = p_gene_id;

            -- 然后添加新的重命名信息
            UPDATE Proteins
            SET protein_name = CONCAT(protein_name, ' (Gene renamed from ', old_gene_name, ' to ', p_new_gene_name, ')')
            WHERE gene_id = p_gene_id;

            COMMIT;
            SELECT CONCAT('Gene updated successfully. Affected proteins: ', ROW_COUNT()) AS result;
        ELSE
            ROLLBACK;
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Failed to update gene.';
        END IF;
    ELSE
        ROLLBACK;
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Gene ID not found.';
    END IF;
END$$
DELIMITER ;
```

创建  
存储  
过程  
源码  
(3  
分)

```
-- 2.2 更新物种信息并记录变更历史
DELIMITER $$
CREATE PROCEDURE UpdateSpeciesInfo (
    IN p_species_id INT,
    IN p_scientific_name VARCHAR(100),
    IN p_common_name VARCHAR(100),
    IN p_description TEXT
)
BEGIN
    DECLARE old_scientific_name VARCHAR(100);
    DECLARE old_common_name VARCHAR(100);

    -- 开始事务
    START TRANSACTION;

    -- 获取旧的物种信息
    SELECT scientific_name, common_name
    INTO old_scientific_name, old_common_name
    FROM Species
    WHERE species_id = p_species_id;

    IF old_scientific_name IS NOT NULL THEN
        -- 更新物种信息
        UPDATE Species
        SET
            scientific_name = IFNULL(p_scientific_name, scientific_name),
            common_name = IFNULL(p_common_name, common_name),
            description = IFNULL(p_description, description),
            updated_at = CURRENT_TIMESTAMP
        WHERE species_id = p_species_id;

        -- 这里可以添加记录变更历史的逻辑
        -- 例如插入到一个Species_Changes表（如果有的话）

        COMMIT;
        SELECT 'Species updated successfully' AS result;
    ELSE
        ROLLBACK;
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Species ID not found.';
    END IF;
END$$
DELIMITER ;
```

存储  
过程  
执行  
源码  
(1  
分)

(截屏)

```
# 调用存储过程更新基因名称和关联的蛋白质信息
result = db.session.execute(
    text('CALL UpdateGeneAndProteinInfo(:gene_id, :new_gene_name)'),
    {'gene_id': gene.gene_id, 'new_gene_name': form.gene_name.data}
)
db.session.commit()

result = db.session.execute(
    text('CALL UpdateSpeciesInfo(:species_id, :scientific_name, :common_name, :description)'),
    {
        'species_id': id,
        'scientific_name': form.scientific_name.data,
        'common_name': form.common_name.data,
        'description': form.description.data
    }
)
)
```

程序  
演示  
(2  
分)

说明：不违背存储过程，能够执行

蛋白质列表 [添加蛋白质](#)

ID	蛋白质名称	UniProt ID	关联基因	操作
801	pro10001 (Gene renamed from gene_test4 to gene_test41)	p11111111111	gene_test41	<a href="#">查看</a> <a href="#">编辑</a>
1	Protein_0 (Gene renamed from Gene_4388 to Gene_438)	P83952	Gene_438	<a href="#">查看</a> <a href="#">编辑</a>
2	Protein_1	P32806	Gene_61	<a href="#">查看</a> <a href="#">编辑</a>
11	Protein_10	P62005	Gene_312	<a href="#">查看</a> <a href="#">编辑</a>
101	Protein_100	P24797	Gene_994	<a href="#">查看</a> <a href="#">编辑</a>
102	Protein_101	P16019	Gene_846	<a href="#">查看</a> <a href="#">编辑</a>
103	Protein_102	P82695	Gene_18	<a href="#">查看</a> <a href="#">编辑</a>
104	Protein_103	P58407	Gene_755	<a href="#">查看</a> <a href="#">编辑</a>
105	Protein_104	P94683	Gene_342	<a href="#">查看</a> <a href="#">编辑</a>
106	Protein_105	P75672	Gene_448	<a href="#">查看</a> <a href="#">编辑</a>

上一页 1 2 3 ... 81 下一页

更新操作完成，可以看到被更改了依赖基因的但被指命名和关联基因的名称有所改变，改为了标定的模式

[illegible]

## 7. 含有视图的查询操作（15 分）

说明	<p>(1 分) 简要说明该操作所要完成的功能；</p> <p>(1 分) 简要说明建立的该视图的功能；</p> <p>(2 分) 简要说明该操作涉及的关系数据表（以“表名”的形式给出）</p> <p>(1 分) 简要说明表连接涉及的字段（以“表 1. 属性=表 2. 属性”）</p> <p>(6 分) 实现该操作的关键代码（高级语言、SQL），截图即可；</p> <p>(4 分) 如何执行该操作，按所述方法能够正常演示程序则给分。</p>
操作功能描述 (1 分)	<p>将底层多表关联与聚合查询逻辑封装起来，方便后续对“基因 - 物种 - 蛋白质数量”、“基因研究活跃度”以及“用户角色信息”等常用统计/查询需求的直接调用，无需每次都重复复杂的 JOIN 与聚合语句。</p>
视图功能描述 (1 分)	<p><b>GeneProteinCountBySpecies 视图：</b> 统计每条基因 (Genes.gene_id, gene_name) 所在的物种 (Species.scientific_name) 以及该基因对应的蛋白质数量 (COUNT(Proteins.protein_id))，用于快速查询“某基因在某物种下有多少条蛋白质”。</p> <p><b>GeneResearchActivity 视图：</b> 统计每条基因 (Genes.gene_id, gene_name) 所关联的文献数量 (Gene_Publications) 和实验记录数量 (Experimental_Data)，并计算一个“研究活跃度分数 (activity_score) = 2 × publication_count + 1 × experiment_count”，以支持按研究活跃度快速排序或筛选最热门的基因。</p> <p><b>UserRoleInfo 视图：</b> 将用户表 (Users) 与三类角色表 (Creators, Managers, Readers) 通过 LEFT JOIN 关联，依据 Users.user_type 字段动态抽取“affiliation”（隶属机构/部门/组织）和“role_detail”（研究领域/访问级别/订阅类型），统一输出“用户 → 所属机构 → 角色细节”三者信息，方便一次性查看平台中各类用户及其身份属性。</p>
涉及的关系表 (2 分)	<p><b>GeneProteinCountBySpecies 视图：</b> Genes、Species、Proteins</p> <p><b>GeneResearchActivity 视图：</b> Genes、Species、Gene_Publications、Experimental_Data</p> <p><b>UserRoleInfo 视图：</b> Users、Creators、Managers、Readers</p>
表连接字段 (1 分)	<p><b>GeneProteinCountBySpecies：</b> Genes.species_id = Species.species_id Genes.gene_id = Proteins.gene_id （使用 LEFT JOIN，以保证即使某基因暂无蛋白质也能显示）</p> <p><b>GeneResearchActivity：</b> Genes.species_id = Species.species_id （LEFT JOIN，可出现基因无物种，显示 'Unknown'） Genes.gene_id = Gene_Publications.gene_id （LEFT JOIN，用于统计文献数） Genes.gene_id = Experimental_Data.gene_id （LEFT JOIN，用于统计实验数）</p> <p><b>UserRoleInfo：</b> Users.user_id = Creators.user_id （LEFT JOIN，仅当 user_type='creator' 时，此表有值）</p>



	<p>Users.user_id = Managers.user_id (LEFT JOIN, 仅当 user_type='manager' 时, 此表有值)</p> <p>Users.user_id = Readers.user_id (LEFT JOIN, 仅当 user_type='reader' 时, 此表有值)</p>
创建视图代码 (3分)	<p>(截屏)</p> <pre> -- 3.1 基因-物种-蛋白质数量视图 CREATE OR REPLACE VIEW GeneProteinCountBySpecies AS SELECT     g.gene_id,     g.gene_name,     s.scientific_name AS species_name,     COUNT(p.protein_id) AS protein_count FROM     Genes g JOIN     Species s ON g.species_id = s.species_id LEFT JOIN     Proteins p ON g.gene_id = p.gene_id GROUP BY     g.gene_id, g.gene_name, s.scientific_name;  -- 3.2 基因研究活跃度视图 (基于文献和实验数量) CREATE OR REPLACE VIEW GeneResearchActivity AS SELECT     g.gene_id,     g.gene_name,     COALESCE(s.scientific_name, 'Unknown') AS species_name,     COUNT(DISTINCT gp.publication_id) AS publication_count,     COUNT(DISTINCT e.experiment_id) AS experiment_count,     (COUNT(DISTINCT gp.publication_id) * 2 + COUNT(DISTINCT e.experiment_id)) AS activity_score FROM     Genes g LEFT JOIN Species s ON g.species_id = s.species_id LEFT JOIN Gene_Publications gp ON g.gene_id = gp.gene_id LEFT JOIN Experimental_Data e ON g.gene_id = e.gene_id GROUP BY     g.gene_id, g.gene_name, s.scientific_name ORDER BY     activity_score DESC;  -- 3.3 用户角色信息视图 CREATE OR REPLACE VIEW UserRoleInfo AS SELECT     u.user_id,     u.username,     u.email,     u.user_type,     u.is_active, -- 新增     CASE         WHEN u.user_type = 'creator' THEN c.institution         WHEN u.user_type = 'manager' THEN m.department         WHEN u.user_type = 'reader' THEN r.organization         ELSE NULL     END AS affiliation,     CASE         WHEN u.user_type = 'creator' THEN c.research_field         WHEN u.user_type = 'manager' THEN m.access_level         WHEN u.user_type = 'reader' THEN r.subscription_type         ELSE NULL     END AS role_detail FROM     Users u LEFT JOIN     Creators c ON u.user_id = c.user_id LEFT JOIN     Managers m ON u.user_id = m.user_id LEFT JOIN     Readers r ON u.user_id = r.user_id; </pre>
查询代码 (3分)	<p>(截屏)</p> <pre> @login_required def gene_protein_count():     check_admin_access()     page = request.args.get('page', 1, type=int)     per_page = 20 # 每页显示20条, 可根据需要调整     pagination = GeneProteinCountBySpecies.query.paginate(page=page, per_page=per_page, error_out=False)     records = pagination.items     return render_template(         'admin/gene_protein_count.html',         title='基因-物种-蛋白质数量统计',         records=records,         pagination=pagination     )  @bp.route('/gene_activity') def gene_activity():     result = db.session.execute(text(         "SELECT gene_id, gene_name, species_name, publication_count, experiment_count, activity_score "         "FROM GeneResearchActivity ORDER BY activity_score DESC LIMIT 20"     ))     activity_data = [dict(row._mapping) for row in result]     return render_template('main/gene_activity.html', activity_data=activity_data) </pre>

程序演示  
(4分)

用户管理

ID	用户名	邮箱	用户类型	状态	所属机构/部门/组织	角色细节	操作
1	user_0	user_0@example.com	reader	禁用	Biotech Company B	premium	切换状态
2	user_1	user_1@example.com	reader	激活	Medical Center D	premium	切换状态
3	user_2	user_2@example.com	creator	激活	MIT	Systems Biology	切换状态
4	user_3	user_3@example.com	reader	禁用	University Lab C	premium	切换状态
5	user_4	user_4@example.com	creator	激活	Stanford University	Computational Biology	切换状态
6	user_5	user_5@example.com	reader	禁用	Research Institute A	premium	切换状态
7	user_6	user_6@example.com	manager	激活	Computational Biology	limited	切换状态
8	user_7	user_7@example.com	manager	激活	Genomics	full	切换状态

基因研究活跃度



基因ID	基因名称	物种	文献数	实验数	活跃度分数
620	Gene_620	Saccharomyces cerevisiae	8	4	20
571	Gene_571	Saccharomyces cerevisiae	8	2	18
857	Gene_857	Saccharomyces cerevisiae	6	6	18

基因-物种-蛋白质数量统计

基因ID	基因名称	物种名称	蛋白质数量
3	Gene_2	Drosophila melanogaster	0
5	Gene_4	Drosophila melanogaster	0
8	Gene_7	Drosophila melanogaster	0
9	Gene_8	Drosophila melanogaster	0
15	Gene_14	Drosophila melanogaster	3
16	Gene_15	Drosophila melanogaster	3
22	Gene_21	Drosophila melanogaster	0
25	Gene_24	Drosophila melanogaster	0
33	Gene_32	Drosophila melanogaster	1
40	Gene_39	Drosophila melanogaster	1

三个创建的视图的标展展示如上，其实现原理相似，其中基因表达活跃度我们查看视图并选取 top20 进行可视化和展示。

备注