天津大学

《数据库实践》课程设计文档

Github 数据库设计

学　　院　智能与计算学部

专　　业　计算机科学与技术

年　　级　2019 级

小组成员　刘梦迪（3019244420）

贾天玉（3019244416）

王志鸣（3019244395）

任课老师　陈世展

2021 年 10 月 3 日

一、前言

（一） 实践目的

本次数据库实践主要围绕两方面内容：数据库设计与基本数据库编程实践。通过本次实验，掌握数据库的设计方式及数据库设计的基本步骤；通过设计数据库系统的应用课题，提高查阅文献资料、编程开发以及文档书写的能力，提高分析问题和解决问题的能力。

（二） GitHub 数据库设计意义

Github 是世界上最大的代码托管平台，除了 Git 代码仓库托管以及基本的 Web 管理界面外，还提供了订阅，讨论组，代码片段分享等等功能。目前，其注册用户已经超过 350 万，托管版本数量也是非常之多，其中不乏知名开源项目 Ruby on Rails，jQuery，python 等，在 IT 行业有着举足轻重的地位，于是我们小组便基于 Github 网站，进行相应的本地数据库的设计，对其进行逆向工程，最大程度地还原 GitHub 数据库。

二、实践概要

我们小组选择利用网络爬虫技术抓取 github.org 数据并参考 Github 网站进行相应的本地数据库的设计。

1. 需求分析：通过分析 Github 网站、查阅资料、小组内讨论进行用户需求分析，包括数据、功能和性能需求，绘制数据流图（dfd 图）进行需求分析。

2. 概念结构设计：根据用户需求描述构建 E-R 模型，并绘制 E-R 图。

3. 逻辑结构设计：采用 MySQL 数据库作为实验用 DBMS，根据转换规则将 E-R 图转换成表，实现 E-R 模型与关系数据模型的转换。并以规范化理论为指导，对关系数据模型进行优化，规范到第三范式。

4. 数据库物理设计与实施：主要包括创建数据库；创建基本表，设置约束条件，包括主键与外键的键约束、基于元组与属性的约束、触发器等；创建和管理索引

（DBMS 会为主键自动建立索引以提高查询效率）；创建和管理视图；向数据库中插入数据；用 SQL 语句实现对数据查询、修改、删除等操作；编写触发器、存储过程等，并调试通过等工作。

三、系统需求分析

参考 Github 网站所具有的功能，包括：用户注册，代码托管，给项目 star，clone 代码，发布新版本等等功能设计表，并理清表与表之间的关系，设计并构建一个能够实现 Github 网站主体功能的数据库。

为 Github 平台设计后台数据库，将系统划分为三个子系统（用户系统、仓库系统以及用户与仓库关系系统）并分别进行需求分析。

（一） 数据分析

1. 用户系统

（1）用户(users)分别为个人用户(individual user)和组织(organization)，用户可以安装应用（app）和模块服务（action），用户可以点赞支持模块服务（action),用户可以关注和点赞用户。用户属性有:用户账号名(account name)、密码（password）、电子邮箱（email）、住址（address）、用户类型（type）。
（2）个人用户（individual user）可以创建或加入组织（organization），个人用户可以参与活动（event）,个人用户可以对模块服务（action）做贡献。个人用户除继承用户属性还包括:昵称(name)、个人自传(bio)、个人 URL(URL)、Twitter 用户名（twitter username）、所属公司（company）。
（3）组织(organization)中可以建立小团队(team),组织可以开发应用(app)。组织除继承用户属性还包括:昵称(Organization account name)、网站(Websit)、是否是官方认证的（Isverified）、组织简要介绍（Profile）。
（4）团队（team）属性有：团队名称（team_name）、团队描述（description）、团队对组织是否可见（visibility）。
（5）活动（event）属性包括活动名称（name）、开始日期（data_beginning）、结束日期（data_ending）、活动描述（description）。
（6）应用(app)属性有:应用名称(name)、简介(introduction)、价格(price)、编程语言（languages）、安装数量（install_num）。
（7）模块服务(aciton)属性有:应用名称(name)、价格(price)、版本(version)、

点赞数量（stars_number）。

2. 仓库系统

（1）仓库（repositories）包括需筹资仓库（sponsor_repositories）。仓库中包括代码（code），仓库中可以进行问题交流（issues）。仓库可以发出拉取请求（pull requests）。仓库与主题（topic)或集合（collections）相符合。仓库可以有发行版（releases）和发行包（packages）。仓库属性有：仓库名称（repository name）、仓库拥有者（owner）、仓库描述（description）、是否公开(public or private）、是否需要筹资（sponor_or_not）、最近更新时间（updated on）、点赞数量（star_number）、查看数量（watch_number）、复制数量（fork_number）。
（2）需筹资仓库（sponsor_repositories）属性除继承的属性还有：筹资目标（sponor_goal）。
（3）代码信息（code）属性有：仓库拥有者（owner）、仓库名称（repository name)、提交数量(commit_num)、分支数量(branch_num)、发布数量(tags_num)、最后一次更新时间(last_update)
（4）问题(issues)属性有：仓库拥有者(owner)、仓库名称(repository name)、评论用户账号名（account name）、序号（number）、评论（comment）、日期（data）、是否公开可见（open or close）。
（5）拉取请求（pull requests）属性有：仓库拥有者（owner）、仓库名称（repository name）、拉取用户账号名（account name）、序号（number）、评论（comment）、日期（data）、是否公开可见（open or close）。
（6）话题（topic）属性有：话题名称（topic name）、描述（description）、收纳仓库数量（number of repositories）
（7）集合（collections）属性有：集合名称（collection name）、描述（description）、收纳仓库数量（number of repositories）
（8）发行版(releases)属性有：仓库拥有者(owner)、仓库名称(repository name)、发行日期（release date）、发行版本（tag）、发行者（publisher)、下载地址(download url)。
（9）包(packages)属性有：仓库拥有者(owner)、仓库名称(repository name)、包名称（name）、包类型（type）、发行日期（release date）。

3. 用户与仓库关系系统

（1）热点仓库（popular_repositories）属性有：仓库拥有者（owner）、仓库名称（name）、编程语言（language）、日期范围（date range）（可选日推荐、周推荐、月推荐）。
（2）热门开发者（popular_users）属性有：用户账号名（account name）、日期范围（date range）（可选日推荐、周推荐、月推荐）。

（二） 基础功能分析

1. 用户系统

（1）用户可以安装应用或模块服务，一个用户可以安装多个应用或模块服务，一个，应用或模块服务可以被多个用户安装，安装需要记录安装时间（data）。
（2）用户可以点赞支持模块服务，一个用户可以为多个模块服务点赞，一个模块服务可以被多个用组织户点赞，点赞需要记录点赞时间（data）。
（3）用户可以点赞和关注用户，一个用户可以被多个用户关注和点赞，需记录时间（date）。
（4）个人用户可以创建或加入组织，一个个人用户可以创建或加入多个组织，一个组织只能被一个个人用户创建，一个组织可以被多个用户加入，创建或加入组织需要记录个人用户在组织中的角色（role）。
（5）个人用户可以参与活动，一个用户可以参与多个活动，一个活动可以被多个用户参与。
（6）个人用户可以对模块服务做贡献，一个个人用户可以对多个模块服务做贡献，一个模块服务可以被多个个人用户做贡献。
（7）组织中可以建立小团队，一个组织中可以有多个团队，一个团队只能所属于一个组织。
（8）个人用户可以加入多个团队，一个团队可以有多个个人用户，加入团队需要记录个人用户在团队中的角色（role）。
（9）组织可以开发应用，一个组织可以开发多个应用，一个应用只能被一个组织开发。

2. 仓库系统

（1）仓库中包含代码信息（code），一个仓库包含一个代码信息，一个代码信息包含于一个仓库。
（2）仓库中可以进行问题交流（issues），一个仓库可以发出一个问题交流请

求，一个问题交流请求属于一个仓库。
（3）仓库可以发出拉取请求（pull requests），一个仓库可以发出一个拉取请求，一个拉取请求只能由一个仓库发出。
（4）仓库与主题（topic)或集合（collections）相符合，一个仓库可以属于多个主题或集合，一个主题或集合可以包含多个仓库或集合。
（5）仓库可以有发行版（releases）和发行包（packages），一个仓库可以发行多个发行版本或发行包，一个发行版本或发行包只属于一个仓库。

3. 用户与仓库关系系统

（1）用户（users）可以给筹资仓库（sponsor repositories）捐钱，捐款需要记录捐款钱数（money）、筹资仓库（sponsor repositories）、筹资仓库拥有者(sponsor repositories owner）、捐款频率（frequency）。
（2）用户可以建立仓库（repositories）。
（3）用户可以浏览、复制、点赞、表明观看公开仓库，浏览、复制、点赞、表明观看公开仓库均需记录时间（date）。
（4）用户可以向自己仓库提交代码（code），用户提交代码需记录时间（date）。
（5）用户可以评论其他用户的仓库也可以回复其他人对自己仓库的评论（issues），需记录回复的日期（date）、回复的评论（issue number）、回复内容（response）。
（6）用户可以拉取其他用户的仓库并进行修改，需记录修改文件名（file）、修改行数（changed line num）、修改行的内容（changed line content）。
（7）用户可以对主题（topic）和集合（collection）点赞，需记录时间（date）。

（三） 扩展功能分析

接下来，我们将功能进行拓展，并使用数据流图的方式表示，。数据流图，简称 DFD，是 SA 方法中用于表示系统逻辑模型的一种工具，它以图形的方式描绘数据在系统中流动和处理的过程，由于它只反映系统必须完成的逻辑功能，所以它是一种功能模型。

GitHub 的功能:用户注册和登录系统：用户注册账号（输入用户名账号不得重复）、修改密码（需要输入原来密码和修改之后的密码）、创建仓库、pull 代码、push 代码、commit 代码、pull request 请求、对仓库提出 issue、回复

issue、创建 team、加入组织

（1） 标准的 Git 命令进行访问和操作。同时，所有的 Git 命令都可以用到 GitHub 项目上面
（2） 社交网络具有的功能，例如赞(star)、关注(follow)、评论。用户可以通过复刻(fork)他人项目的形式参与开发，并可通过协作示意图来查看有多少开发者参与了开发并追踪最新的复刻版本。
（3） 允许注册用户和非注册用户在网页中浏览项目，使用 zip 打包下载，但是用户必须允许注册用户和非注册用户在网页中浏览项目

（四） 性能分析

1. 数据精确度

保证查询的查全率和查准率为 100%，所有在相应域中包含查询关键字的记录都能查到，所有在相应域中不包含查询关键字的记录都不能查到。

2. 系统响应时间

（1）单个记录查询时间少于 3 秒
（2）多个记录查询时间少于 6 秒
（3）更新/保存记录时间少于 2 秒

3. 适应性

满足运行环境在允许操作系统之间的安全转换和与其他应用软件的独立运行要求。

4. 故障处理

正常使用时不应出错，若运行时遇到不可恢复的系统错误，也必须保证数据库完好无损。

四、概念结构设计

（一）数据库设计规则

- 主键外键关系、表间关系、表中字段是不可再分的属性；
- 表的功能划分合理；
- 字段的命名、类型和长度合理；
- 数据表均以小写命名，比较直观；
- 对于联系两张表的中间数据表，名字以下划线间隔（表一_两者关系_表二），如：user_issue_repositories。

（二）概念设计

1. 按照**系统功能**划分，该系统可以划分七个系统：用户信息管理系统、应用管理系统、用户活动系统、仓库信息管理系统、用户开发系统、仓库筹资系统、热门推荐系统，分别介绍如下系统的 ER 图：

(1) 用户信息管理系统

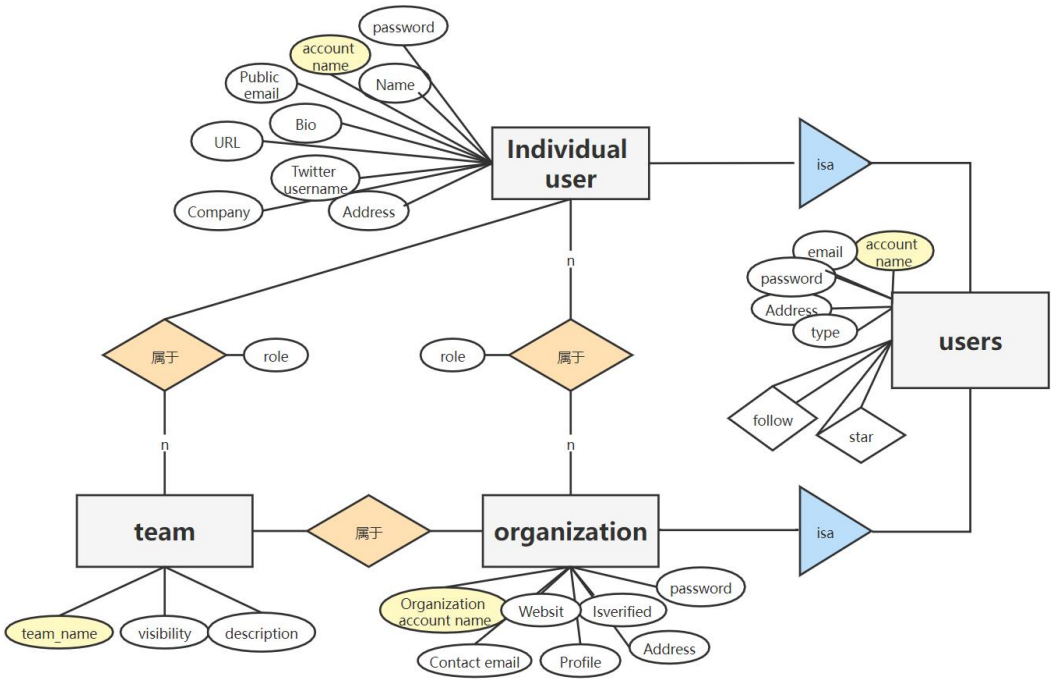

图 1：用户信息管理系统 ER 图

(2) 应用管理系统

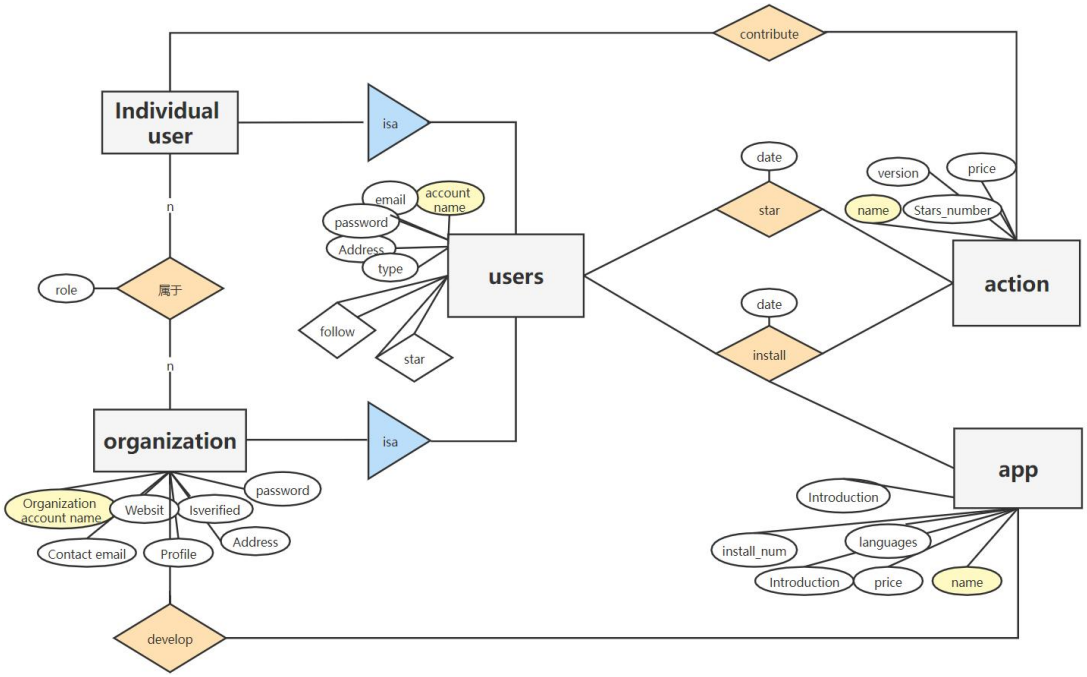

图 2：应用管理系统 ER 图

(3) 用户活动系统

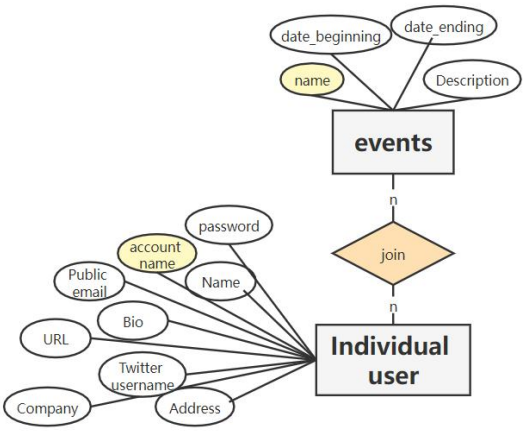

图 3：用户活动管理系统 ER 图

(4) 仓库信息管理系统

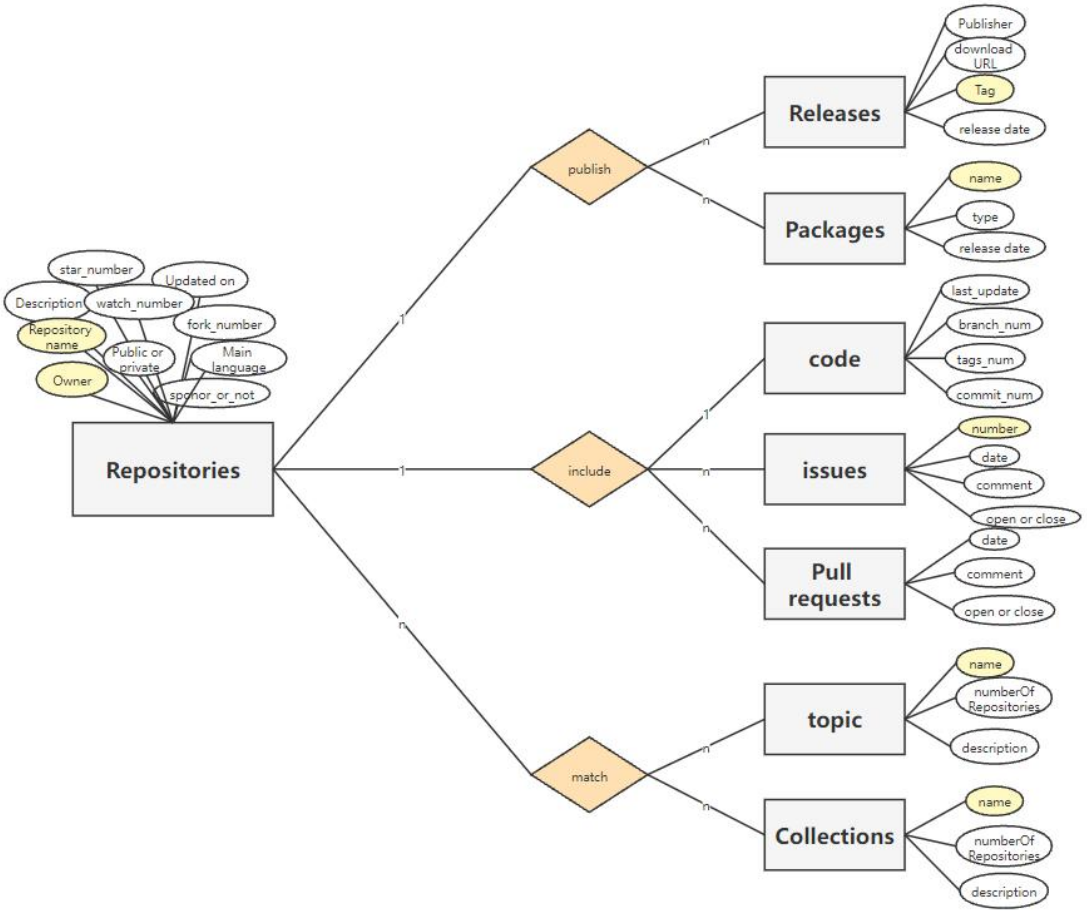

图 4：仓库信息管理系统 ER 图

(5) 用户开发系统

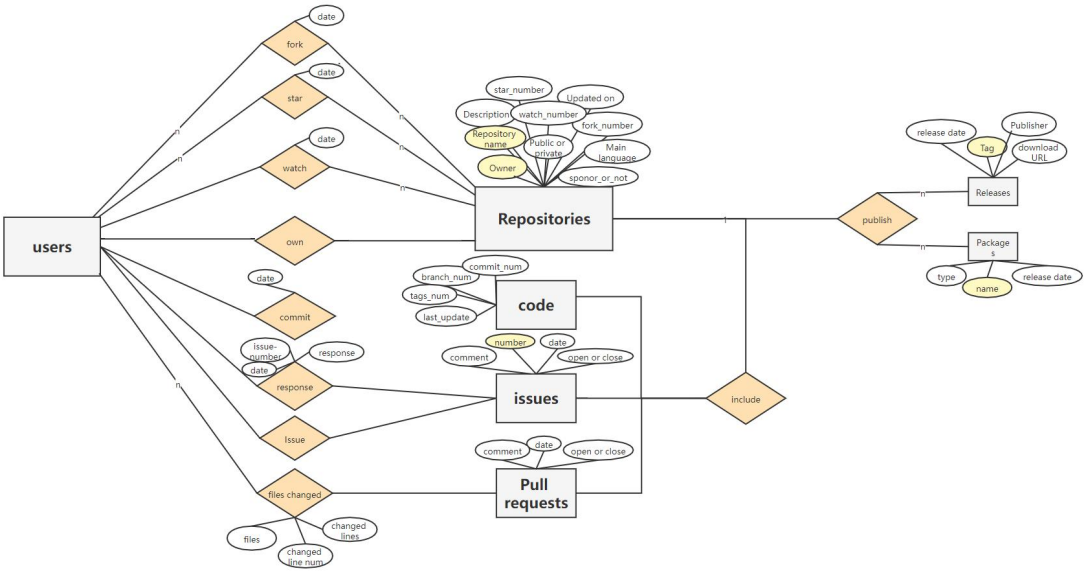

图 5：用户开发系统 ER 图

(6) 仓库筹资系统

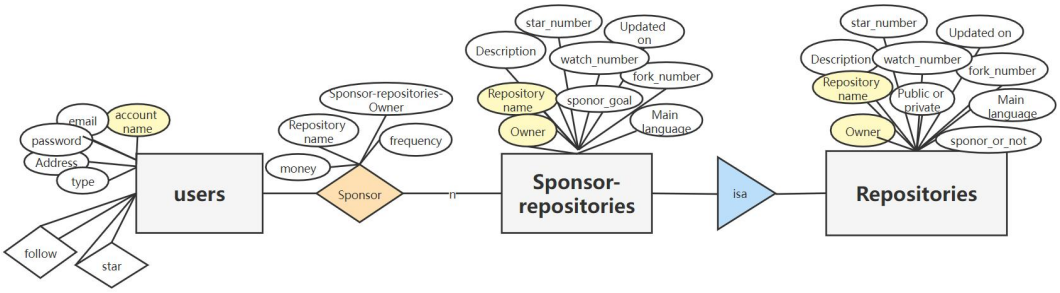

图 6：仓库筹资系统 ER 图

(7) 热门推荐系统

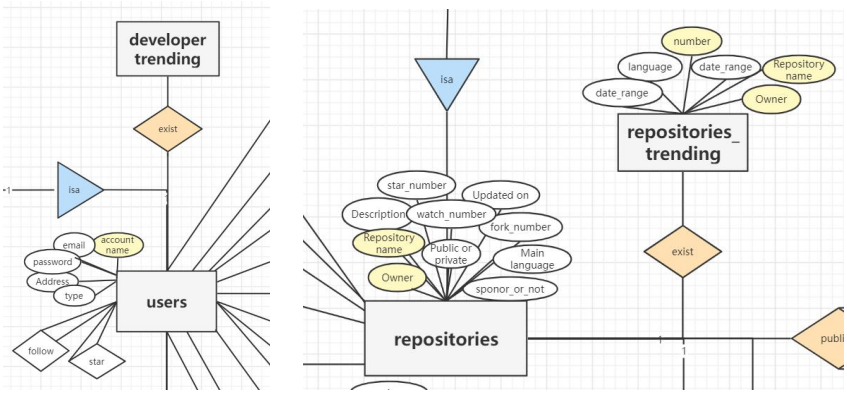

图 7：热门推荐系统 ER 图（左：developer，右：repositories）

因为热门推荐系统需要实时更新，所以我们建立视图存储所有开发者和仓库的排名，根据其 star、fork 等个数进行更新，每天更新 8 次。

2. 数据库各个实体表示

由此可知，在我们的数据库系统中共有 17 个实体，分别为：users、individual user、organization、team、events、app、action、repositories、sponsor-repositories、code、issues、pull requests、topic、collections、Trending、Releases、Packages。

这 17 个实体中，有两个实体最为重要，我们称之为中心实体——users（用户）和 repositories（仓库），其他实体和这两个实体都会有一些联系。其中，users 可以分为 individual user 和 organization 两类，在一个 organization 中可以有多个 team，用户参加 events，下载 app 和 action。在 repositories 端，一个仓库包含 code、issues、Pull requests 这些部分，所以将这些独立出来，形成实体；同时，在 GitHub 中，为帮助其他人找到并参与您的项目，可以为仓库添加主题（topic），这些主

题可以与项目的预期目的、学科领域、关联团队或其他重要特点相关；GitHub 还会对新兴行业、主题和社区的一些独到见解进行收集，称为 Collections，会收纳一些仓库，比如由世界各地的政府构建的网站、应用程序和工具，旨在让政府更好地协同工作。Sponsor-repositories 是 repositories 的一个子类，表示需要筹集的仓库。最后，repositories 可以发布为 Releases 和 Packages。

由此，得各个实体 E / R 图如下：

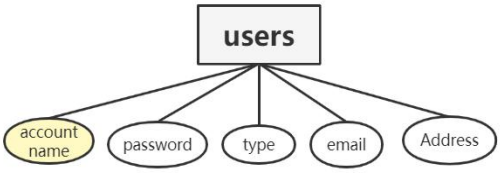

图 8：users 实体图

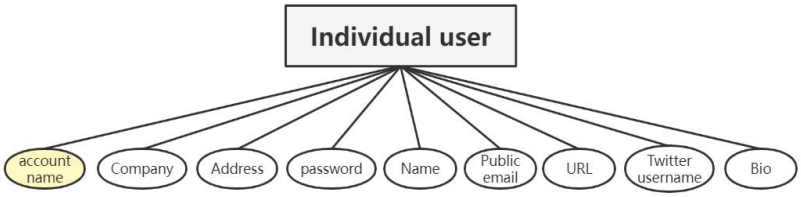

图 9：individual user 实体图

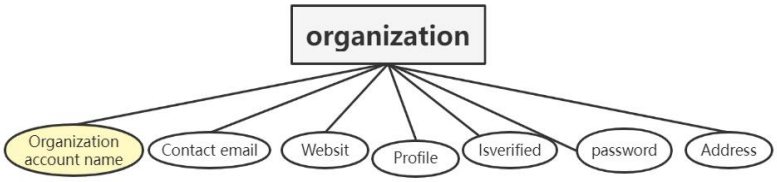

图 10：organization 实体图

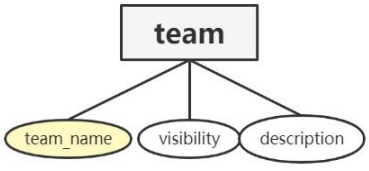

图 11：team 实体图

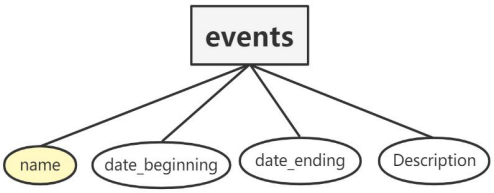

图 12：events 实体图

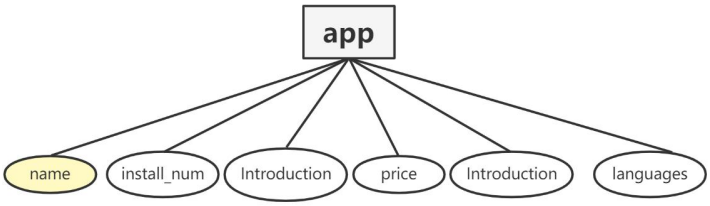

图 13：app 实体图

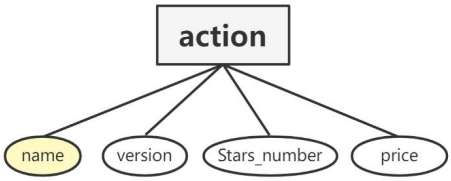

图 14：action 实体图

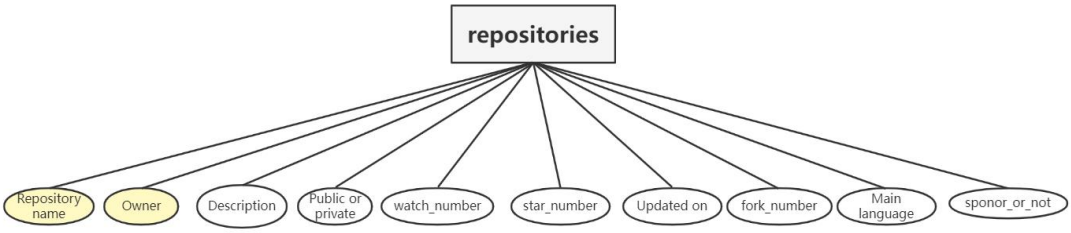

图 15：repositories 实体图

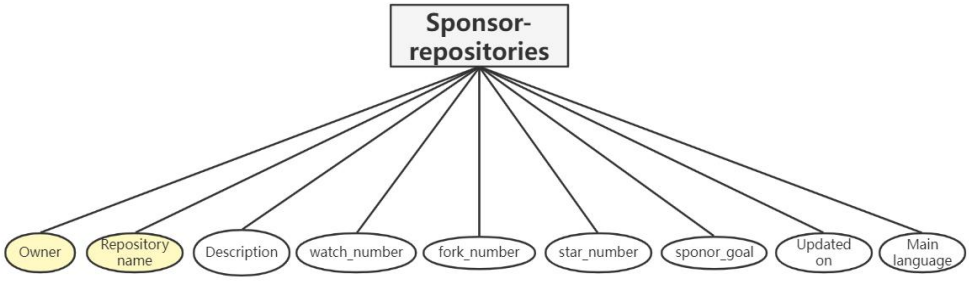

图 16：sponsor-repositories 实体图

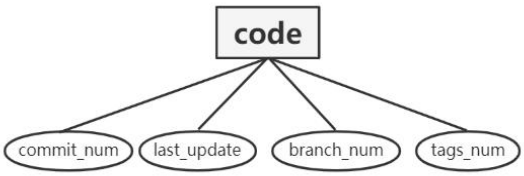

图 17：code 实体图

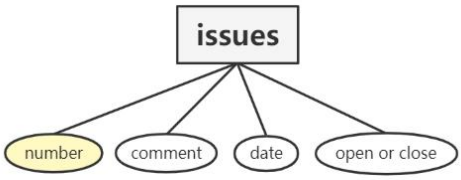

图 18：issues 实体图

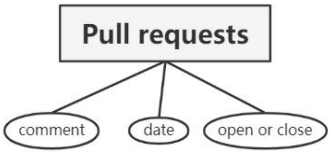

图 19：pull requests 实体图

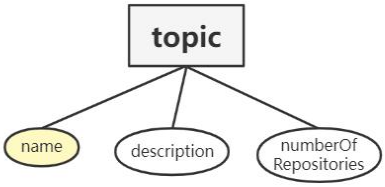

图 20：topic 实体图

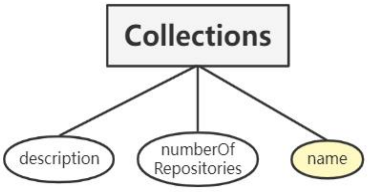

图 21：collections 实体图

图 22：Releases 实体图

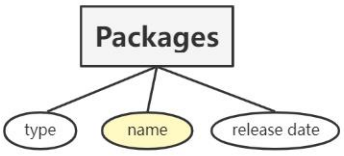

图 23：Packages 实体图

1. 系统整体 ER 图

由此，整个系统的 ER 图如下图所示：

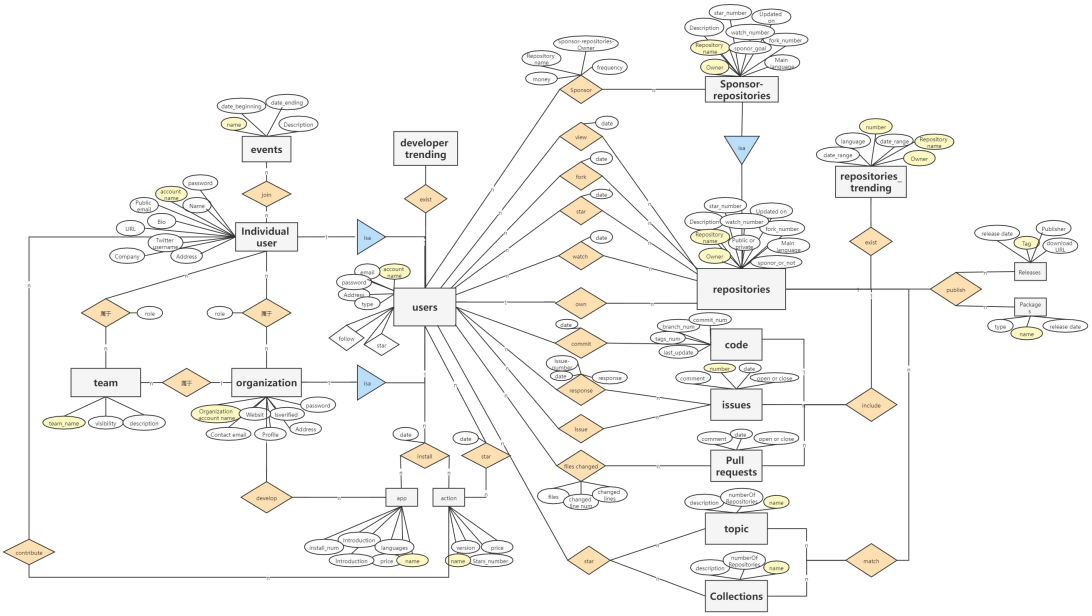

图 24：系统整体 ER 图

五、逻辑结构设计

（一） 逻辑转换

数据库总体可以分为三个部分——**用户系统**、**仓库系统**以及**用户与仓库系统**。

1. 用户系统

在用户系统中，一共有 7 个实体，分别为：

用户（用户账号名，密码，电子邮箱，住址，类型）
个人用户（用户账号名，昵称，密码，电子邮箱，个人自传，个人 URL，Twitter 用户名，所属公司，住址，类型）
组织（组织账号名，昵称，密码，电子邮件，网站，组织简要介绍，是否是官方认证的，住址，类型）
团队（组织账号名，团队名称，团队描述，团队对组织是否可见）
活动（活动名称，开始日期，结束日期，活动描述）
应用（应用名称，简介，价格，编程语言，安装数量）
模块服务（应用名称，价格，版本，点赞数量）

在用户系统中，一共 7 个实体之间关系表，分别为：

用户所属组织（用户账号名，组织账号名，用户角色）
用户所属团体（用户账号名，组织账号名，团队名称，用户角色）
用户参与活动（用户账号名，活动名称，开始日期）
用户安装应用（用户账号名，应用名称，日期）
用户安装模块服务（用户账号名，模块服务名称，日期）
组织开发应用（组织账号名，应用名称）
个人用户共享模块服务（用户账号名，模块服务名称）
用户关注用户（用户账号名，被关注用户账号名，日期）
用户点赞用户（用户账号名，被点赞用户账号名，日期）

用户、个人用户、组织、团队四个实体之间的关系：

（1） 用户分为个人用户和组织两类，所以在 E-R 图中使用 isa，表示“个人用户”和“组织”是“用户”的子类，继承“用户”表的所有属性。

（2） 每个用户可以创建多个组织，于是成为这些组织的拥有者，同时该用户还可以加入多个组织，成为这些组织的成员；于是个人用户和组织存在“属于”关系，该关系包含一个角色属性，表示该用户是组织的拥有者或是组织成员。

（3） 创建好组织后，组织内就存在一个用户——创建该组织的当前账号，该即为该组织的拥有者；同时，该用户还可以在组织内创建团队，并可以邀请组织内其他成员加入到团队中。下图为组织和团队的关系图。

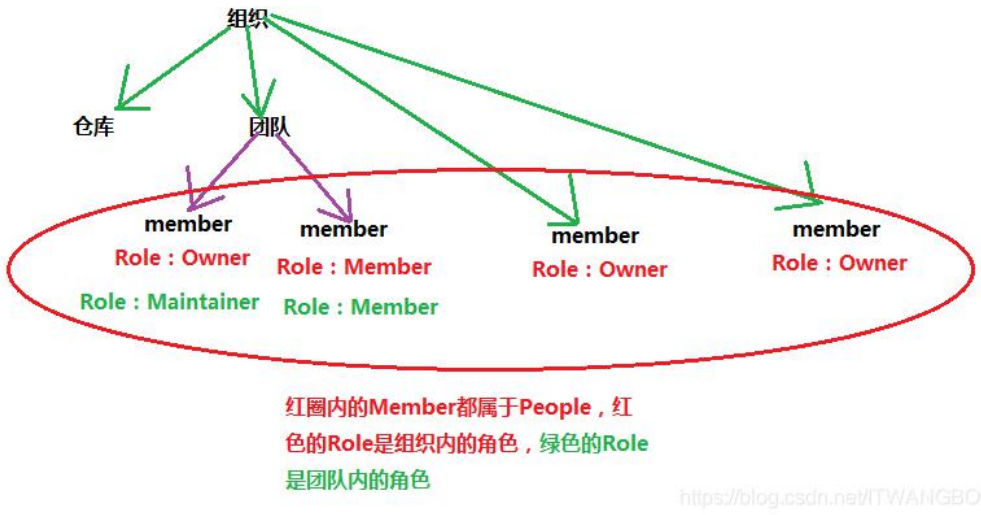

图 25：组织-团队关系图

（4） 约束：如果一个用户在组织内的角色是拥有者，那么他在团队内的角色就是 Maintainer；如果用户在组织内的角色是组织成员，那么他在团队内的角色也是 Member，于是据此设置该约束。

对活动表、应用表和模块服务表说明：

（1） 每个个人用户都可以参加 GitHub 举办的活动（Connect with the GitHub community at conferences, meetups, and hackathons around the world.），比如 GitHub Community Twitter Spaces Hacktoberfest 等。

（2） 同时用户还可以在应用商城中安装应用（分为应用软件和模块服务两类）（Find tools to improve your workflow），比如 CircleCI 可以帮助你自动化地快速地编译、测试并且部署你的项目。

2. 仓库系统

在仓库系统中，一共有 9 个实体，分别为：

仓库（仓库名称，仓库拥有者，仓库描述，是否公开，是否需要筹资，最近更新时间，点赞数量，查看数量，复制数量）
需筹资仓库（仓库名称，仓库拥有者，仓库描述，是否公开，是否需要筹资，最近更新时间，点赞数量，查看数量，复制数量，筹资目标）
代码信息（仓库拥有者，仓库名称，提交数量，分支数量，发布数量，最后一次更新时间）
问题（仓库拥有者，仓库名称，评论用户名，序号，评论，日期，是否公开可见）
拉取请求（仓库拥有者，仓库名称，拉取账号名，序号，评论，日期，是否公开可见）
话题（话题名称，描述，收纳仓库数量）
集合（集合名称，描述，收纳仓库数量）
发行版（仓库拥有者，仓库名称，发行日期，发行版本，发行者，下载地址）
发行包（仓库拥有者，仓库名称，发行日期，包名称，包类型）

在仓库系统中，一共有 2 个实体关系表，分别为：

仓库匹配话题（仓库拥有者，仓库名称，话题名称）
仓库匹配集合（仓库拥有者，仓库名称，集合名称）

对仓库的一些约束和说明：
（1） 约束：仓库属性中含有是否对外公开一项，所以如果该仓库是私密的，那么其他不属于该仓库的用户便不能对其进行表明观看（watch）、点赞（star）、复制（fork）等操作。
（2） 集资仓库是仓库的一个子类，通过仓库中集资的属性进行筛选，在 ER 图中，使用 isa 来表示，理论上继承仓库的所有属性，但是集资仓库一定都是公开的，并且在是否集资属性中为“是”，于是删除这两个属性。

对主题表的说明：
（1）每个仓库都可以选择和自己相关联的主题加入，比如话题 Android，有 87,780 个公共仓库加入该话题。在每一个主题中，还可以通过主要语言进行筛选，此外，还可选择按照点赞数量或复制数量进行排序，从而选出该主题中最受欢迎

的一个项目。

3. 用户与仓库关系系统

热门仓库（仓库拥有者，仓库名称，编程语言，日期范围）

热门开发者（用户账号名，日期范围）

捐款（筹资仓库，筹资仓库拥有者，捐款用户账号名，捐款钱数，捐款频率）

浏览（仓库拥有者，仓库名称，浏览用户账号名，时间）

复制（仓库拥有者，仓库名称，复制用户账号名，时间）

点赞（仓库拥有者，仓库名称，点赞用户账号名，时间）

表明观看（仓库拥有者，仓库名称，观看用户账号名，时间）

回复评论（仓库拥有者，仓库名称，回复评论用户名，回复评论序号，回复内容，回复时间）

拉取修改（仓库拥有者，仓库名称，拉取用户账号名，序号，修改文件名，修改行的内容，修改行数）

主题点赞（用户账号名，主题名称，时间）

集合点赞（用户账号名，集合名称，时间）

对热门推荐表的说明：

（1）**个性推荐功能**：Github 基于个人浏览、点赞和复制的仓库，通过特定算法为特定用户生成一些推荐仓库。（Based on repositories you’ve viewed / Based on repositories you’ve starred / Based on repositories you’ve forked）

（2）热门推荐包括热门仓库和热门开发者两类，分别表示 Github 社区对什么感兴趣（See what the GitHub community is most excited about today），和哪些开发者开发了热门的工具（These are the developers building the hot tools today）。无论是热门仓库还是热门开发者，都可以按照时间范围（天、周、月）和编程语言来重新筛选。这个排名是 Github 根据一系列数据（比如 **star 数、fork 数、提交数、follow 数以及项目页面浏览量**）进行统计的。Github 每天会进行 8 次统计，并根据结果刷新该页面。

（二） 细化表结构

为方便，根据上述文字描述，用英文简写为表和列取名，确定列的数据类型及必要的约束规则，给出如下所示数据库表的基本结构及说明：

1. 用户系统

（1）用户表（users）

列名	说明	数据类型	约束
account_name	用户账号名	VARCHAR(45)	NOT NULL，PRIMARY KEY
email	电子邮箱	VARCHAR(45)	NULL DEFAULT NULL，形式应符合“%@%”
address	住址	VARCHAR(145)	NULL DEFAULT NULL
type	类型	VARCHAR(45)	NOT NULL，类型为“个人用户”或“组织”
password	密码	VARCHAR(145)	NOT NULL，密码位数不应少于 5 位

（2）个人用户（individualuser）

列名	说明	数据类型	约束
account_name	用户账号名	VARCHAR(45)	NOT NULL，PRIMARY KEY
name	昵称	VARCHAR(45)	NULL DEFAULT NULL
bio	个人自传	VARCHAR(145)	NULL DEFAULT NULL
URL	个人 URL	VARCHAR(145)	NULL DEFAULT NULL
Twitter_username	Twitter 用户名	VARCHAR(45)	NULL DEFAULT NULL
company	所属公司	VARCHAR(145)	NULL DEFAULT NULL
email	电子邮箱	VARCHAR(45)	NULL DEFAULT NULL，形式应符合“%@%”
address	住址	VARCHAR(145)	NULL DEFAULT NULL
type	类型	VARCHAR(45)	NOT NULL，类型为“个人用户”
password	密码	VARCHAR(145)	NOT NULL，密码位数不应少于 5 位

（3）组织（organization）

列名	说明	数据类型	约束
Org_act_name	用户账号名	VARCHAR(45)	NOT NULL，PRIMARY KEY
name	昵称	VARCHAR(45)	NULL DEFAULT NULL
Profile	组织简要介绍	VARCHAR(145)	NULL DEFAULT NULL
Websit	个人 URL	VARCHAR(145)	NULL DEFAULT NULL，形式应符合“http%”
Isverified	是否官方认证	TINYINT	NULL DEFAULT NULL，为“1”或“0”(1 为 Isverified，0 为 Isnotverified)
email	电子邮箱	VARCHAR(45)	NULL DEFAULT NULL，形式应符合“%@%”
address	住址	VARCHAR(145)	NULL DEFAULT NULL
type	类型	VARCHAR(45)	NOT NULL，类型为“组织”
password	密码	VARCHAR(145)	NOT NULL，密码位数不应少于 5 位

（4）团队（team）

列名	说明	数据类型	约束
Org_act_name	组织账号名	VARCHAR(45)	NOT NULL，PRIMARY KEY
team_name	团队名称	VARCHAR(45)	NOT NULL，PRIMARY KEY
visibility	团队对组织是否可见	VARCHAR(45)	NOT NULL， 为“visible”或“secret”（表示可视性，其中 visible 表示 organization 内所有人都可以看到这个 team；secret 表示 team 内的人才可以看到这个 team’）
description	团队描述	VARCHAR(45)	NULL DEFAULT NULL

（5）活动（event）

列名	说明	数据类型	约束
name	活动名称	VARCHAR(45)	NOT NULL，PRIMARY KEY
date_beginning	开始日期	DATE	NOT NULL，PRIMARY KEY
date_ending	结束日期	DATE	NOT NULL
description	活动描述	VARCHAR(45)	NULL DEFAULT NULL

（6）应用（app）

列名	说明	数据类型	约束
name	应用名称	VARCHAR(45)	NOT NULL，PRIMARY KEY
Introduction	简介	VARCHAR(45)	NULL DEFAULT NULL
languages	编程语言	VARCHAR(45)	NULL DEFAULT NULL
price	价格	INT	NOT NULL
install_num	安装数量	INT	NOT NULL

（7）模块服务（action）

列名	说明	数据类型	约束
name	应用名称	VARCHAR(45)	NOT NULL，PRIMARY KEY
price	价格	INT	NOT NULL
version	版本	VARCHAR(45)	NULL DEFAULT NULL
stars_number	点赞数量	INT	NOT NULL

（8）用户所属组织（indvuser_belong_org）

列名	说明	数据类型	约束

account_name	用户账号名	VARCHAR(45)	NOT NULL，PRIMARY KEY
org_account_name	组织账号名	VARCHAR(45)	NOT NULL，PRIMARY KEY
role	角色	VARCHAR(45)	NOT NULL，为“owner”或“member”(表示每个成员在组织里面的角色，分为拥有者或成员)

（9）用户所属团体（indvuser_belong_team）

列名	说明	数据类型	约束
account_name	用户账号名	VARCHAR(45)	NOT NULL，PRIMARY KEY
org_account_name	组织账号名	VARCHAR(45)	NOT NULL，PRIMARY KEY
team_name	团队名称	VARCHAR(45)	NOT NULL，PRIMARY KEY
role	角色	VARCHAR(45)	NOT NULL，为“maintainer”或“member”(表示每个成员在团队里面的角色）

（10）用户参与活动（user_join_events）

列名	说明	数据类型	约束
account_name	用户账号名	VARCHAR(45)	NOT NULL，PRIMARY KEY
event_name	活动名称	VARCHAR(45)	NOT NULL，PRIMARY KEY
date_beginning	开始日期	DATE	NOT NULL，PRIMARY KEY

（10）用户安装应用（users_install_app）

列名	说明	数据类型	约束
account_name	用户账号名	VARCHAR(45)	NOT NULL，PRIMARY KEY
app_name	应用名称	VARCHAR(45)	NOT NULL，PRIMARY KEY
date	开始日期	DATE	NOT NULL，PRIMARY KEY

（11）用户安装模块服务（users_install_action）

列名	说明	数据类型	约束
account_name	用户账号名	VARCHAR(45)	NOT NULL，PRIMARY KEY
action_name	模块服务名称	VARCHAR(45)	NOT NULL，PRIMARY KEY
date	开始日期	DATE	NOT NULL，PRIMARY KEY

（12）组织开发应用（org_develop_app）

列名	说明	数据类型	约束
account_name	用户账号名	VARCHAR(45)	NOT NULL, PRIMARY KEY
app_name	应用名称	VARCHAR(45)	NOT NULL, PRIMARY KEY

（13）个人用户贡献模块服务（user_contribute_action）

列名	说明	数据类型	约束
account_name	用户账号名	VARCHAR(45)	NOT NULL, PRIMARY KEY
action_name	模块服务名称	VARCHAR(45)	NOT NULL, PRIMARY KEY

（14）用户关注用户（user1_follow_user2）

列名	说明	数据类型	约束
user1	用户账号名	VARCHAR(45)	NOT NULL, PRIMARY KEY
user2	被关注用户账号名	VARCHAR(45)	NOT NULL, PRIMARY KEY
date	日期	DATE	NULL DEFAULT NULL

（15）用户点赞用户（user1_star_user2）

列名	说明	数据类型	约束
user1	用户账号名	VARCHAR(45)	NOT NULL, PRIMARY KEY
user2	被关注用户账号名	VARCHAR(45)	NOT NULL, PRIMARY KEY
date	日期	DATE	NULL DEFAULT NULL

2. 仓库系统

（1）仓库（repositories）

列名	说明	数据类型	约束
repository_name	仓库名称	VARCHAR(45)	NOT NULL, PRIMARY KEY
owner	仓库拥有者	VARCHAR(45)	NOT NULL, PRIMARY KEY
description	仓库描述	VARCHAR(45)	NULL DEFAULT NULL
public_or_private	是否公开	TINYINT	NOT NULL,为“1”或“0”(1为 Public,0为 private)

sponor_or_not	是否需要筹资	TINYINT	NOT NULL，为“1”或“0”(1 为 sponor，0 为 not sponor)
updated_on	最近更新时间	DATE	NULL DEFAULT NULL
star_number	点赞数量	INT	NOT NULL
watch_number	查看数量	INT	NOT NULL
fork_number	复制数量	INT	NOT NULL

（2）需筹资仓库（sponsor_repositories）

列名	说明	数据类型	约束
repository_name	仓库名称	VARCHAR(45)	NOT NULL，PRIMARY KEY
owner	仓库拥有者	VARCHAR(45)	NOT NULL，PRIMARY KEY
description	仓库描述	VARCHAR(45)	NULL DEFAULT NULL
public_or_private	是否公开	TINYINT	NOT NULL，为“1”(1 为 Public)
sponor_or_not	是否需要筹资	TINYINT	NOT NULL，为“1”(1 为 sponor)
updated_on	最近更新时间	DATE	NULL DEFAULT NULL
star_number	点赞数量	INT	NOT NULL
watch_number	查看数量	INT	NOT NULL
fork_number	复制数量	INT	NOT NULL
goal	筹资目标	INT	NOT NULL，sponsor 的目标，比如每月 50 英镑

（3）代码信息（code）

列名	说明	数据类型	约束
owner	仓库拥有者	VARCHAR(45)	NOT NULL，PRIMARY KEY
repository_name	仓库名称	VARCHAR(45)	NOT NULL，PRIMARY KEY
commit_num	提交数量	INT	NULL DEFAULT 0
branch_num	分支数量	INT	NULL DEFAULT 0
tags_num	发布数量	INT	NULL DEFAULT 0
last_update	最近更新时间	DATE	NULL DEFAULT NULL

（4）问题（issues）

列名	说明	数据类型	约束
owner	仓库拥有者	VARCHAR(45)	NOT NULL，PRIMARY KEY
repository_name	仓库名称	VARCHAR(45)	NOT NULL，PRIMARY KEY
account_name	评论用户名	VARCHAR(45)	NOT NULL
number	序号	INT	NOT NULL，PRIMARY KEY

comment	评论	VARCHAR(145)	NULL DEFAULT NULL
date	日期	DATE	NULL DEFAULT NULL
open_or_close	是否公开可见	TINYINT	NOT NULL，为“1”或“0”（1 为 open，0 为 close）

（5）拉取请求（pull_requests）

列名	说明	数据类型	约束
owner	仓库拥有者	VARCHAR(45)	NOT NULL，PRIMARY KEY
repository_name	仓库名称	VARCHAR(45)	NOT NULL，PRIMARY KEY
account_name	拉取用户名	VARCHAR(45)	NOT NULL
number	序号	INT	NOT NULL，PRIMARY KEY
comment	评论	VARCHAR(145)	NULL DEFAULT NULL
date	日期	DATE	NULL DEFAULT NULL
open_or_close	是否公开可见	TINYINT	NOT NULL，为“1”或“0”（1 为 open，0 为 close）

（6）话题（topic）

列名	说明	数据类型	约束
topic_name	主题名称	VARCHAR(45)	NOT NULL，PRIMARY KEY
description	主题描述	VARCHAR(145)	NOT NULL
repositories_num	收纳仓库数量	INT	NULL DEFAULT 0

（7）集合（collections）

列名	说明	数据类型	约束
collection_name	集合名称	VARCHAR(45)	NOT NULL，PRIMARY KEY
description	集合描述	VARCHAR(145)	NOT NULL
repositories_num	收纳仓库数量	INT	NULL DEFAULT 0

（8）发行版（releases）

列名	说明	数据类型	约束
owner	仓库拥有者	VARCHAR(45)	NOT NULL，PRIMARY KEY
repository_name	仓库名称	VARCHAR(45)	NOT NULL，PRIMARY KEY

release_date	发行日期	DATE	NOT NULL
tag	发行版本	VARCHAR(45)	NOT NULL，PRIMARY KEY
publisher	发行者	VARCHAR(45)	NULL DEFAULT NULL
download_URL	下载地址	VARCHAR(145)	NULL DEFAULT NULL，要符合“http://%”或“https://%”形式

（9）发行包

列名	说明	数据类型	约束
owner	仓库拥有者	VARCHAR(45)	NOT NULL，PRIMARY KEY
repository_name	仓库名称	VARCHAR(45)	NOT NULL，PRIMARY KEY
release_date	发行日期	DATE	NOT NULL
packages_name	包名称	VARCHAR(45)	NOT NULL，PRIMARY KEY
type	包类型	VARCHAR(45)	NOT NULL，为“Docker”或“Apache Mave”或“NuGet”或“RubyGems”或“npm”或“Containers”

（10）仓库匹配话题（repositories_match_topic）

列名	说明	数据类型	约束
owner	仓库拥有者	VARCHAR(45)	NOT NULL，PRIMARY KEY
repository_name	仓库名称	VARCHAR(45)	NOT NULL，PRIMARY KEY
topic_name	话题名称	VARCHAR(45)	NOT NULL，PRIMARY KEY

（11）仓库匹配集合（repositories_match_collections）

列名	说明	数据类型	约束
owner	仓库拥有者	VARCHAR(45)	NOT NULL，PRIMARY KEY
repository_name	仓库名称	VARCHAR(45)	NOT NULL，PRIMARY KEY
collection_name	集合名称	VARCHAR(45)	NOT NULL，PRIMARY KEY

3. 用户与仓库关系系统

（1）热门仓库（popular_repositories）

列名	说明	数据类型	约束
owner	仓库拥有者	VARCHAR(45)	NOT NULL，PRIMARY KEY
repository_name	仓库名称	VARCHAR(45)	NOT NULL，PRIMARY KEY

language	编程语言	VARCHAR(45)	NULL DEFAULT NULL
date_range	日期范围	VARCHAR(45)	NOT NULL，PRIMARY KEY，比如日推荐、周推荐、月推荐

（2）热门开发者（popular_users）

列名	说明	数据类型	约束
account_name	用户账号名	VARCHAR(45)	NOT NULL，PRIMARY KEY
date_range	日期范围	VARCHAR(45)	NOT NULL，PRIMARY KEY，比如日推荐、周推荐、月推荐

（3）捐款（user_sponsor_repositories）

列名	说明	数据类型	约束
sponsor_repositories_owner	筹资仓库拥有者	VARCHAR(45)	NOT NULL，PRIMARY KEY
sponsor_repositories_name	筹资仓库名称	VARCHAR(45)	NOT NULL，PRIMARY KEY
account_name	捐款用户账号名	VARCHAR(45)	NOT NULL，PRIMARY KEY
money	捐款钱数	INT	NOT NULL
frequency	捐款频率	VARCHAR(45)	NOT NULL，为“Monthly”或“One-time”

（4）浏览（user_view_repositories）

列名	说明	数据类型	约束
owner	仓库拥有者	VARCHAR(45)	NOT NULL，PRIMARY KEY
repositories_name	仓库名称	VARCHAR(45)	NOT NULL，PRIMARY KEY
account_name	浏览用户账号名	VARCHAR(45)	NOT NULL，PRIMARY KEY
date	时间	DATE	NOT NULL

（5）复制（user_fork_repositories）

列名	说明	数据类型	约束
owner	仓库拥有者	VARCHAR(45)	NOT NULL，PRIMARY KEY
repositories_name	仓库名称	VARCHAR(45)	NOT NULL，PRIMARY KEY
account_name	复制用户账号名	VARCHAR(45)	NOT NULL，PRIMARY KEY

date	时间	DATE	NOT NULL

（6）点赞（user_star_repositories）

列名	说明	数据类型	约束
owner	仓库拥有者	VARCHAR(45)	NOT NULL，PRIMARY KEY
repositories_name	仓库名称	VARCHAR(45)	NOT NULL，PRIMARY KEY
account_name	点赞用户账号名	VARCHAR(45)	NOT NULL，PRIMARY KEY
date	时间	DATE	NOT NULL

（7）表明观看（user_watch_repositories）

列名	说明	数据类型	约束
owner	仓库拥有者	VARCHAR(45)	NOT NULL，PRIMARY KEY
repositories_name	仓库名称	VARCHAR(45)	NOT NULL，PRIMARY KEY
account_name	观看用户账号名	VARCHAR(45)	NOT NULL，PRIMARY KEY
date	时间	DATE	NOT NULL

（8）回复评论（user_respond_issue_repositories）

列名	说明	数据类型	约束
owner	仓库拥有者	VARCHAR(45)	NOT NULL，PRIMARY KEY
repositories_name	仓库名称	VARCHAR(45)	NOT NULL，PRIMARY KEY
account_name	回复评论用户账号名	VARCHAR(45)	NOT NULL，PRIMARY KEY
number	问题序号	INT	NOT NULL，PRIMARY KEY
response	回复内容	VARCHAR(45)	NOT NULL
date	回复时间	DATE	NOT NULL

（9）拉取修改（user_fileschangedin_repositories）

列名	说明	数据类型	约束
owner	仓库拥有者	VARCHAR(45)	NOT NULL，PRIMARY KEY
repositories_name	仓库名称	VARCHAR(45)	NOT NULL，PRIMARY KEY
account_name	拉取用户账号名	VARCHAR(45)	NOT NULL，PRIMARY KEY

number	序号	INT	NOT NULL, PRIMARY KEY
file	修改文件名	VARCHAR(45)	NOT NULL
changed_line_content	修改行内容	VARCHAR(145)	NOT NULL
changed_line_num	修改行数	INT	NOT NULL

（10）主题点赞（users_star_topic）

列名	说明	数据类型	约束
account_name	用户账号名	VARCHAR(45)	NOT NULL, PRIMARY KEY
topic_name	主题名称	VARCHAR(45)	NOT NULL, PRIMARY KEY
date	时间	DATE	NOT NULL

（11）集合点赞（users_star_collections）

列名	说明	数据类型	约束
account_name	用户账号名	VARCHAR(45)	NOT NULL, PRIMARY KEY
collection_name	集合名称	VARCHAR(45)	NOT NULL, PRIMARY KEY
date	时间	DATE	NOT NULL

六、数据库实施

本章节主要包含创建表、添加数据和创建必要的视图、触发器和存储过程等内容。

（一） 开发技术

1. 数据库开发技术

开发工具：MySQL Server 8.0 + MySQL Workbench 8.0 CE
开发语言：SQL
开发技术：数据库开发技术
面向对象：需求者

MySQL 8.0 是全球最受欢迎的开源数据库的一个非常令人兴奋的新版本，全面改进。其改进包括：可靠性 DDL 语句已变得原子性和崩溃安全，元数据存储在单个事务数据字典中，由 InnoDB 提供支持；性能 InnoDB 在读/写工作负载，IO 绑定工作负载和高争用“热点”工作负载方面明显更好。本实验中最终将使用 MySQL Server 8.0 数据库管理系统将我们设计的数据库实现。

2. 前端开发技术

开发工具：Vue + node + MySQL
开发语言：TypeScript SQL JavaScript Html Css
开发技术：前端开发 后端接口开发
面向对象：需求者

本项目的前端使用了 Vue+node+MySQL 的技术栈，vue 是一套用于构建用户界面的渐进式框架，只关注视图层，易于上手。使用 node 开发，其具有的优点包括：node 是基于 JavaScript，所以前后台的语言统一，便于全栈开发；并且由于 node 是通过异步函数来实现，所以比较适合处理 IO 密集型应用。

（二） 创建表

创建表的具体细节详见附录一

（三） 添加数据

GitHub 数据的爬取存在以下困难：

- Github 连接不稳定，有时需要外网才能连接；
- 爬取数据困难，数据库所需要数据种类较多，难以一次性爬取，需要多次进行自动化页面跳转操作，且信息分布较为分散，跳转操作无规律性可循；
- 一些用户的隐私信息，以及权限设置为 private 的仓库信息不能获取；
- 存在某些信息需要关键词的提取，比如仓库的最近一次更新时间，event 的开始时间和结束时间等，直接爬下来数据之后仍需要进一步关键词提取，任务量较大；

针对以上问题，我们采用 scrapy 框架+Selenium 进行爬虫，同时在此基础上使用 python 提取关键词等操作，进一步数据处理，最后进行人工补充缺失信息。

（四） 创建视图

（1） 一个用户有多少 follower
（2） 一个用户 following 了多少其他用户
（3） 有多少用户给这个用户 star
（4） 这个用户给了谁 star
（5） 含有 star 数量的用户统计表
（6） follower 数量的用户统计表
（7） 含有 star 数量的仓库统计表（总共/在某段时间内）
（8） fork 数量的仓库统计表（总共/在某段时间内）
（9） watch 数量的仓库统计表（总共/在某段时间内）
（10） view 数量的仓库统计表表（总共/在某段时间内）
（11） star、fork、watch、view 数量综合统计排名（总共/在某段时间内）

（五） 创建约束

约束可以分为键和外键、基于属性和元组上的约束（非空约束、CHECK 约束）、断言、触发器。

1. 键

各个表的主键在第五节逻辑结构设计部分已经详细说明，在这里不予赘述。

2. 外键

- 用户系统的外键

外键名称	引用表	引用列	被引用表	被引用列
FK_induser_user	individualuser	account_name	users	account_name
FK_inbeorg_org	indvuser_belong_org	org_account_name	organization	Org_act_name
FK_inbeorg_user	indvuser_belong_org	account_name	users	account_name

FK_team_org	team	organization_name	organization	Org_act_name
FK_indbeteam_user	indvuser_belong_team	account_name	users	account_name
FK_indbeteam_org	indvuser_belong_team	org_name	organization	Org_act_name
FK_indbeteam_team	indvuser_belong_team	team_name	team	team_name
FK_uje_user	user_join_events	account_name	users	account_name
FK_uje_event	user_join_events	event_name	events	name
FK_uia_user	users_install_app	account_name	users	account_name
FK_uia_app	users_install_app	app_name	app	name
FK_uiac_user	users_install_action	account_name	users	account_name
FK_uiac_action	users_install_action	action_name	action	name
FK_oda_user	org_develop_app	account_name	users	account_name
FK_oda_app	org_develop_app	app_name	app	name
FK_uca_user	user_contribute_action	account_name	users	account_name
FK_uca_action	user_contribute_action	action_name	action	name
FK_ufu_user	user1_follow_user2	user1	users	account_name
FK_ufu_user2	user1_follow_user2	user2	users	account_name
FK_usu_user	user1_star_user2	user1	users	account_name
FK_usu_user2	user1_star_user2	user2	users	account_name

- 仓库系统的外键

外键名称	引用表	引用列	被引用表	被引用列
FK_rep_user	repositories	owner	users	account_name
FK_srep_user	sponsor_repositories	owner	users	account_name
FK_rmt_rep	repositories_match_topic	Repository_name, Owner	repositories	Repository_name, Owner
FK_rmt_topic	repositories_match_topic	Repository_name, Owner	repositories	Repository_name, Owner
FK_rmt_topic	repositories_match_topic	topic_name	topic	topic_name
FK_rmc_rep	repositories_match_collections	Repository_name, Owner	repositories	Repository_name, Owner
FK_rmc_collections	repositories_match_collections	collection_name	collections	collection_name
FK_releases_rep	releases	Repository_name, Owner	repositories	Repository_name, Owner
FK_packages_rep	packages	Repository_name, Owner	repositories	Repository_name, Owner

FK_code_rep	code	Repository_name, Owner	repositories	Repository_name, Owner
FK_issues_rep	issues	Repository_name, Owner	repositories	Repository_name, Owner
FK_pullrequest_rep	pull_requests	Repository_name, Owner	repositories	Repository_name, Owner

- 用户与仓库关系系统的外键

外键名称	引用表	引用列	被引用表	被引用列
FK_usr_user	user_sponsor_repositories	account_name	users	account_name
FK_usr_rep	user_sponsor_repositories	repository_name	repositories	Repository_name
FK_uvr_rep	user_view_repositories	repository_name, owner	repositories	Repository_name, owner
FK_uvr_user	user_view_repositories	account_name	users	account_name
FK_ufr_rep	user_fork_repositories	repository_name, owner	repositories	Repository_name, owner
FK_ufr_user	user_fork_repositories	account_name	users	account_name
FK_ustr_rep	user_star_repositories	repository_name, owner	repositories	Repository_name, owner
FK_ustr_user	user_star_repositories	account_name	users	account_name
FK_uwr_rep	user_watch_repositories	repository_name, owner	repositories	Repository_name, owner
FK_uwr_user	user_watch_repositories	account_name	users	account_name
FK_udr_rep	user_develop_repositories	repository_name, owner	repositories	Repository_name, owner
FK_udr_user	user_develop_repositories	account_name	users	account_name
FK_urir_rep	user_respond_issue_repositories	repository_name, owner	repositories	Repository_name, owner
FK_urir_user	user_respond_issue_repositories	account_name	users	account_name
FK_ufcir_rep	user_fileschangedin_repositories	repository_name, owner	repositories	Repository_name, owner
FK_ufcir_user	user_fileschangedin_repositories	account_name	users	account_name
FK_ust_user	users_star_topic	account_name	users	account_name
FK_ust_topic	users_star_topic	topic_name	topic	topic_name
FK_usc_user	users_star_collections	account_name	users	account_name
FK_usc_collections	users_star_collections	collections_name	collections	collection_name
FK_usa_user	users_star_action	account_name	users	account_name
FK_usa_collections	users_star_action	action_name	action	name

FK_poprep_rep	popular_repositories	repository_name, owner	repositories	Repository_name, owner
FK_popusers_users	popular_users	account_name	users	account_name

维护引用完整性约束：在建表过程中，需要加上级联、置空。比如有三个表，表一为用户信息（主键为用户账号名），表二为仓库信息（主键为仓库名和仓库拥有者），表三为用户对仓库的 star 操作信息（设置外键为用户账号名指向表一，外键（仓库名，仓库拥有者）指向表二）。如果此时表一的主键用户账号名注销，那么此时表三的外键应该置为空；如果表一用户修改账号名，那么外键应该同时变化为修改后的名称。

3. 非空约束

在仓库表中，我们设置了仓库名称，仓库拥有者，仓库是公开的还是私有的等属性，这些属性都要求满足非空

4. CHECK 约束

各个表的 CHECK 在第五节逻辑结构设计部分已经详细说明，在这里不予赘述。

5. 触发器

（1） 团队成员必须在组织中

当对团队表进行插入操作时，先对要插入数据（元组）在组织表中进行检查，如果不在组织表中则拒绝。

（2） 如果一个 member 在组织内的角色是 Owner，那么他在团队内的角色就是 Maintainer；如果 member 在组织内的角色是 Member，那么他在团队内的角色也是 Member，于是据此设置该约束。

当对团队表进行插入操作时，对其“角色”属性进行约束检查，如果该成员在组织中的角色是 Owner，那么在团队内的角色应该是 Maintainer，否则就拒绝插入或者强制改变。同理第二点。

（3） 组织成员一定是 Individual user 表中元素。

当对组织表中插入操作，如果成员在 Individual user 表中，就插入并更新 indvuser_belong_org 表；否则，拒绝插入。

当对 Individual user 表进行插删除操作，如果成员在组织表中，就要同时也对其进行删除操作。

这两种情况概括为：如果 A 继承了 B（表 A 是表 B 的一部分），那么如果对表 A 进行插入操作时，表 B 也要相应的进行插入；如果对表 B 进行删除操作，相应的也要对 A 进行删除操作。

（4） 如果创建一个新用户：在 users 表中插入并检测该用户是 Individual user 还是 organization 类型（触发器），之后在对应表中进行插入操作。

当对 users 表进行插入、删除或者修改操作时，附带其他表的更新

（5） 如果更新一个用户的属性，那么会检测该元组的 type 属性是 Individual user 还是 organization 类型，对相应的表的属性对应修改。

总结：当用户发生注册、修改或者删除操作时，应当对 users、Individual user、organization 表均进行相应的操作。所以在这里的三个表，我们都设置相应的触发器。

（6） 如果修改用户名密码，需要先输入原始密码，如果和原先存储的密码相同，那么可以进行修改表的操作

当对用户表的密码属性进行修改时，同时输入一个字符串（表示之前的密码），设置触发器，如果和原密码相同则接收，否则就会拒绝。

因此，我们对以上三个表均设置了触发器，所以只要修改其中一个表另外一个表也会相应的进行改变

（7） 如果注销一个用户，需要对 users 表进行删除，同时检测该元组的 type 属性是 Individual user 还是 organization 类型，对相应表做删除操作。

（8） 当一个用户安装 app 时，该 App 的 install_num 属性会相应地发生变化，执行加一操作；同时会在 bill 表中插入购买了该 app。

当对 users_install_app 表进行插入元组时，会对 App 的 install_num 属性

（9） 当一个用户 star 一个 action 时，该 action 的 Stars_number 属性会相应地发生变化，执行加一操作。

当对 users_star_action 表进行插入或者删除元组操作时，会对 action 表的 Stars_number 属性做相应的修改。

（10） 如果是私有仓库，该仓库的 developer 不应超过五人

在对 user_develop_repositories 进行插入操作时，会检查当前仓库的开发者个数，如果超过五人就会拒绝。

（11） Star、fork 等操作表的插入会带来仓库表 star 数量属性的修改

在对表 user_fork_repositories 进行插入或删除操作时，会进而更新仓库表的 star、fork_num 的数量。

（12） Sponsor-repositories 是 Repositories 的子类，继承 Repositories 里的所有属性，同时增加 sponor_goal 额外属性，表示集资的目标。当 sponor 关系表中对该仓库的总集资达到该目标，那么提示用户已经达到集资目标。

在对表 sponsor-repositories 进行插入时，同时插入在表 Repositories；如果在表 Repositories 中删除某个元素，也要在表 sponsor-repositories 中相应的删除。

（13） 仓库属性中含有是否对外公开一项，所以如果该仓库是私密的，那么其他不属于该仓库的用户便不能对其进行 watch、star、fork、view、issue、Pull requests 等操作。

（14） 某用户一段时间（每月）内的花钱去向包括：Sponsor 和 install APP 和 Action，对每一项设置消费限制，如果这一段时间（每月）已花费金额超过设置的额度，那么就会阻止。

（15） topic 和 Collections 中的仓库必须是 Repositories 表中的元素

将上述 15 种触发器进行分类

类型一：子类关系——表 A 继承表 B（是 B 的子类）

对该类型触发器设计，如果 A 继承了 B（表 A 是表 B 的一部分），那么如果对表 A 进行插入操作时，表 B 也要相应的进行插入；如果对表 B 进行删除操作，相应的也要对 A 进行删除操作；对表 A 或者表 B 进行修改时，都要对另一个相应的表做出变化。

1. 团队成员必须在组织中

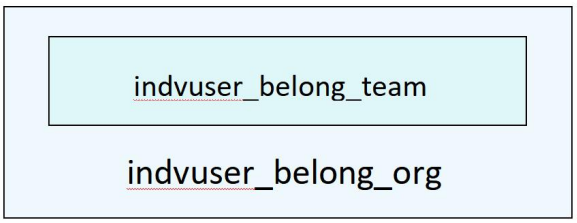

图 26：团队-组织 关系图

如果一个 member 在组织内的角色是 Owner，那么他在团队内的角色就是 Maintainer；如果 member 在组织内的角色是 Member，那么他在团队内的角色也是 Member，于是据此设置该约束。

2. Individual user 和 organization 是 users 的子类

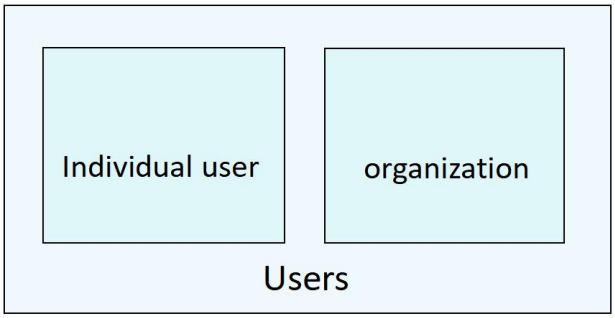

图 27：Individual user 和 organization 关系图

3. 组织成员一定是 Individual user 表元素

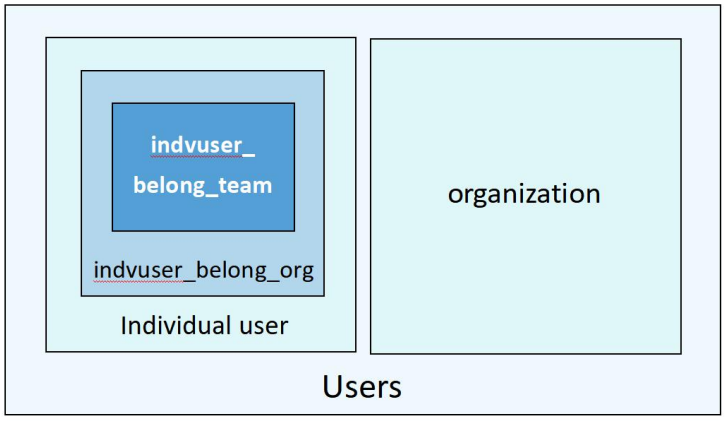

图 27：组织成员-Individual user 关系图

4. Sponsor-repositories 是 Repositories 的子类
5. topic 中的仓库必须是 Repositories 的子类
6. Collections 中的仓库必须是 Repositories 的子类

每两个有继承关系的表之间都应设置该触发器。上述共有七组继承关系，每组含有四个触发器，如下：

对该类型触发器设计，如果 A 继承了 B（表 A 是表 B 的一部分），那么如果对表 A 进行插入操作时，表 B 也要相应的进行插入；如果对表 B 进行删除操作，相应的也要对 A 进行删除操作；对表 A 或者表 B 进行修改时，都要对另一个相应的表做出变化。

所以共设计出 28 个触发器，比如：

```
DROP TRIGGER IF EXISTS `github_info`.`individualuser_BEFORE_INSERT`;

DELIMITER $$
```

```
USE `github_info`$$
CREATE DEFINER = CURRENT_USER TRIGGER
`github_info`.`individualuser_BEFORE_INSERT` BEFORE INSERT ON
`individualuser` FOR EACH ROW
BEGIN
IF EXISTS(select 1 from github_info.users A ,inserted B where A.GUID=B.GUID)
THEN
  UPDATE github_info.users set account_name=B.account_name,
email=B.public_email, address=B.Location,
    type='individual' ,password=B.password;
else
  insert into github_info.users select
account_name,public_email,Location,'individual',password from inserted;
END IF;
END$$
DELIMITER ;
```

类型二：一个表的插入或者删除会导致另一个表的某对应属性加减 1

1. 当一个用户安装 app 时，该 App 的 install_num 属性会相应地发生变化，执行加一操作；
2. 当一个用户 star 一个 action 时，该 action 的 Stars_number 属性会相应地发生变化，执行加一操作，同理取消 star
3. 当一个用户 star 一个仓库时，该仓库的 Stars_number 属性会相应地发生变化，执行加一操作，同理取消 star
4. 同理 fork 仓库

对于此类触发器，共有 4 组，每组分为插入元组和删除元组两种情况，所以共设置 8 个触发器。

比如：

```
 DROP TRIGGER IF EXISTS `github_info`.`users_install_app_AFTER_INSERT`;

 DELIMITER $$
 USE `github_info`$$
 CREATE DEFINER = CURRENT_USER TRIGGER
`github_info`.`users_install_app_AFTER_INSERT` AFTER INSERT ON
`users_install_app` FOR EACH ROW
 BEGIN
```

```
update app set install_num = install_num + 1
  where name = new.app_name;
END$$
DELIMITER ;
```

类型三：一些操作的限制

1. 仓库属性中含有是否对外公开一项，所以如果该仓库是私密的，那么其他不属于该仓库的用户便不能对其进行 watch、star、fork、view、issue、Pull requests 等操作。
2. 如果是私有仓库，该仓库的 developer 不应超过五人
3. 某用户一段时间（每月）内的花钱去向包括：Sponsor 和 install APP 和 Action，对每一项设置消费限制，如果这一段时间（每月）已花费金额超过设置的额度，那么就会阻止。
4. 如果修改用户名密码，需要先输入原始密码，如果和原先存储的密码相同，那么可以修改密码。

（六） 创建存储过程

存储过程就是有业务逻辑和流程的集合，可以在存储过程中创建表，更新数据，删除等等。

（1） **个性推荐**：Based on repositories you’ve viewed / Based on repositories you’ve starred / Based on repositories you’ve forked，通过特定算法生成一些推荐仓库。

（2） **Trending（热门推荐）**：无论是 Trending Repositories 还是 Trending Developers，都可以按照时间范围（天、周、月）和编程语言来重新筛选。这个排名是 Github 根据一系列数据（比如 star 数、fork 数、提交数、follow 数以及项目页面浏览量）进行统计的。其具体过程如下：

① 通过 star 数量、fork 数、提交数、follow 数以及项目页面浏览量，采用相应算法进行统计，统计出 Trending Repositories 和 Trending Developers

② 限制条件——固定的时间范围或者固定的编程语言

③ 每天会进行 8 次统计，8 次调用该存储过程，并根据结果刷新该页面。

（3） 对用户一段时间（每月）内的花钱账单进行统计，其花钱去向包括：Sponsor 和 install APP 和 Action，呈现出统计表，返回每个月的账单总金额。

（4） 某用户可以拥有多个 Sponsor-repositories 集资，也可以发布 app 和 action

挣钱，设置存储过程统计该用户在某段时间内集资挣钱的总钱数。

（5） 当 sponor 关系表中对该仓库的总集资达到该目标，那么提示用户已经达到集资目标。

七、数据库的维护与完善

一般来讲，数据库设计的范式越高，其系统性能会下降。因为范式是减少冗余，提高数据完整性。而提高数据库的性能，有时就是要增加冗余，这时候会和范式有矛盾。概括来说，在数据库设计中两者是空间与时间的平衡。

在这部分，我们对建立的数据库进行范式检验和性能测试，并相应地进行改善，以平衡数据库的范式约束和性能。

（一）功能测试

1. 前端页面展示

通过前端形象具体的展示设计的 E-R 图，并展示根据 E-R 图实现的表和数据以及视图，设计接口展示触发器触发之后对表产生的效果。

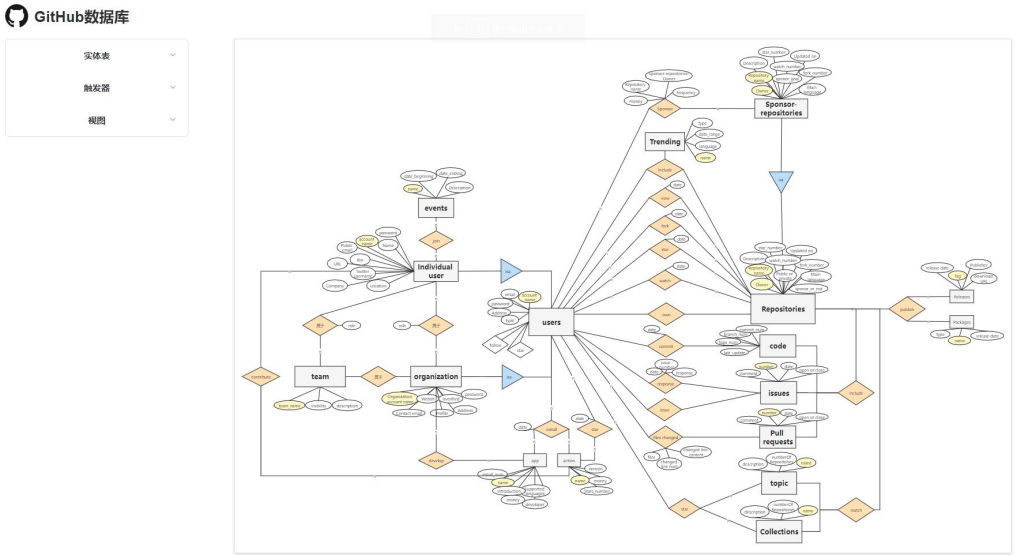

图 28：前端页面展示 E-R 图

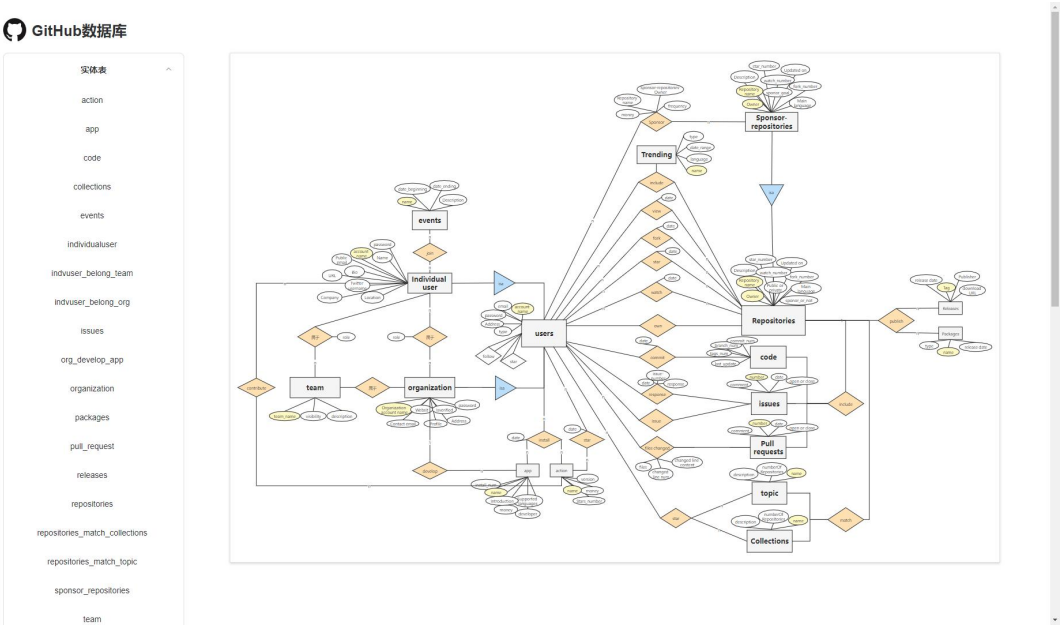

图 29：可跳转至展示任一实体表、触发器、视图

account_name	email	address	type	password
Bit	bit@gmail.com	China	organization	123456
Morning12138	morning@163.com	Tianjin	individualuser	123456
Niko	niko@gmail.com	China	organization	123456
Simple	simple@gmail.com	China	organization	123456

图 30：展示数据库中 users 表的数据

（二）三个范式和 BCNF 范式检验

1. 第一范式

如果一个关系模式 R 的所有属性都是不可分的基本数据项，则 R∈1NF（即

R 符合第一范式）。具体有以下两点：

（1） 每个字段都只能存放单一值

（2） 每个元组都要能利用一个唯一的主键来加以识别

比如表 APP 的 Supported languages 属性表示 APP 支持的开发语言，而我们已知一个 APP 语言有多种，这时每种语言都需要存为一个元组，而不能把所有属性都存入一个字段中。

2. 第二范式

若关系模式 R∈1NF（即 R 符合第一范式），并且每一个非主属性都完全依赖于 R 的码，则 R∈2NF（即 R 符合第二范式）。

3. 第三范式

若关系模式 R∈3NF（即 R 符合第三范式）,则每一个非主属性既不部分依赖于码也不传递依赖于码。

经过检验，我们的数据库满足第三范式

4. BCNF 范式

设关系模式 R<U，F>∈1NF，如果对于 R 的每个函数依赖 X→Y，若 Y 不属于 X，则 X 必含有超码，那么 R∈BCNF。满足 BCNF 条件有：所有非主属性对每一个候选键都是完全函数依赖； 所有的主属性对每一个不包含它的候选键，也是完全函数依赖；没有任何属性完全函数依赖于非候选键的任何一组属性。

虽然范式化的数据库具有如下优点：

1）范式化的数据库更新起来更快

2）范式化之后只有少量的重复数据，只需修改更少的数据

3）范式化的表更小，可以在内存中执行

4）很少的冗余数据，在查询的时候需要更少认得 distinct 或者 group by 语句。

但是也具有由于查询的时候需要很多关联，导致在稍微复杂一些的查询语句在查询范式的 schema 上可能需要多次的关联，这会增加查询的代价，也可能使得一些索引策略无效。所以，我们设计的 Github 数据库，没有一味的满足范式，

在可能需要大量查询的地方增加部分冗余信息，以便加速查找；在不需要大量查询的部分合理的使用范式，使得表的结构更加规范工整。

所以我们的数据库只满足到第三范式，并没有满足 BCNF 范式

（三）性能测试

1. 并发量测试

八、实践优点分析

（一） 工作量

1. 我们共建立了 **40 个表**，**24 个视图**，**45 个触发器**，**五类存储过程**，实践报告 **48 页**，并且表与表之间具有复杂的内部关系，使用 ER 图的方式进行呈现
2. 后端贡献代码数**千行**，同时，我们制作了**前端**，手撸代码约 600 行，能够呈现出 ER 图，list 各个表元素，并且后端插入元组后可以在前端呈现出变化
3. 我们共撰写 **55 页，共 18161 字的** 实践报告，**37 页** PPT
4. 我们爬取数据并对其进行提取关键词等数据分析，此外对于一些不可见数据我们对其虚拟化处理（手动编写）。

（二） 复杂度

1. 我们设计的数据库具有复杂的结构，其 ER 图表示如下

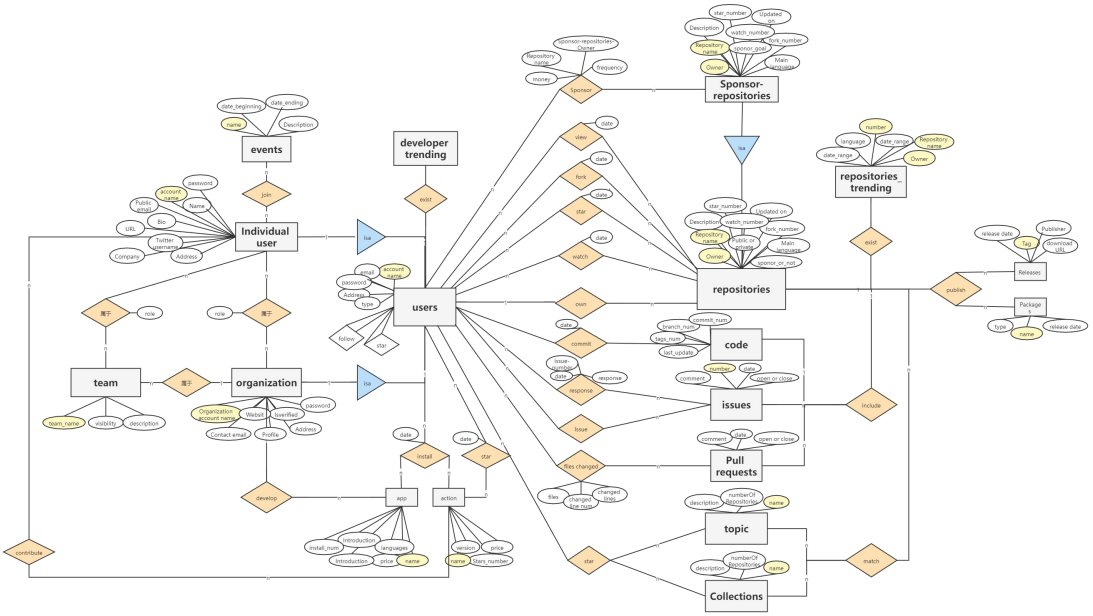

图 31：GitHub 数据库整体 ER 图

（三） 新颖性

1. 设计思路具有**严谨性**：数据库设计严格按照 需求分析——概念结构设计——逻辑结构设计——数据库物理设计与实施——数据库的维护与完善 五个步骤展开。

2. 设计思路运用了**分而治之**的思想：把大系统划分为小系统，先按照数据需求划分为用户系统、仓库系统以及用户与仓库关系系统**三个系统**；再按照功能需求进一步划分为**七个系统**

3. 我们对数据库进行了**范式检验和性能测试**：没有一味的满足范式，在可能需要大量查询的地方增加部分冗余信息，以便加速查找；在不需要大量查询的部分合理的使用范式，使得表的结构更加规范工整。

4. 我们成功使用视图和存储过程实现了**热门推荐系统和个性化推荐系统**

（四） 难度

1．关于 GitHub 的数据库后台设计，网上没有任何资料可以参考，只能根据 GitHub 网站提取信息，逐步分析，进行逆向工程，

2．我们使用了分而治之的思想，将 GitHub 系统化，从而全面完整的模拟了 GitHub 数据库，尽可能的包含了 GitHub 所有的数据和功能

九、小组总结

1. 实践中用到的知识

（1）MySQL 的基本语法
（2）函数依赖
（3）E/R 模型
（4）第一范式、第二范式、第三范式、BCNF 范式
（5）键与外键
（6）约束
（7）触发器
（8）视图
（9）索引
（10）html+css+js+Vue 的前端开发
（11）基于 node 的后端开发

2. 实践中遇到的困难和解决方法

（1）E/R 图的设计：由于 Github 是一个非常复杂的系统，且互联网上没有任何关于 Github 数据库的资料，所以整个数据库的构思是通过在 Github 上实际操作后归纳总结和结合其他用户对于 Github 的使用总结出的。因此很难一次性达到一个较为完整的状态，所以我们通过小组内的多次讨论和收集资料，不断的改进和完善设计，最终才能设计出一个较为完整且可行的 E/R 图。
（2）数据获取：由于 Github 上许多的数据不能直接通过爬虫获得，例如像是私人仓库、用户密码等，所以给我们的数据获取造成了较大的困难。所以我们最终选择自己设计一部分的数据。
（3）触发器的设置：因为我们设计的许多表与表之间都具有一定的关联，所以就需要使用大量的触发器来体现出这种关系，例如当对于 users 表进行修改，那么 individual_user 表也可能发生变化，我们也是集合小组的力量，理清表与表之间的关系，设计了许多必要的触发器，并对触发器进行分类设计，表达了不同的表与表之间的关系。
（4）前端设计：由于我们没有太多的前端开发经历，所以也是经过从小学期到现在的长期不断的学习，逐渐熟悉了前后端的一个开发流程，能够较为熟练的进行前后端的对接，并对我们设计的数据库进行测试。

3. 实践心得和收获

（1） 本次数据库实践设计的是一个系统，而非一两张表，因此在对待表与表之间的关系的时候要有一种宏观上的认识，这锻炼了我们对系统的设计能力。

（2） 本次数据库实践设计是通过小组合作的形式来一起完成一个数据库，所以非常考验小组成员之间的合作与配合，需要有着明确的分工和合作，只有每一个成员都各司其职，才能够圆满的完成这次实践。

（3） 一个完善的数据库不是一次性完成的。通过实践，我们体会到想要完成一个完善的数据库，需要经过多次的迭代和修改，并在实践中不断提高和修正，才能够逐渐完成一个好的数据库设计。

附录一 Create Table

```
--  drop table users;
--  drop table individualuser;
--  drop table organization;
--  drop table indvuser_belong_org;
--  drop table team;
--  drop table indvuser_belong_team;
--  drop table team_belong_org;
--  drop table events;
--  drop table user_join_events;
--  drop table app;
--  drop table action;
--  drop table users_install_app;
--  drop table users_install_action;
--  drop table org_develop_app;
--  drop table user_contribute_action;
--  drop table users_star_action;
--
--  drop table repositories;
--  drop table sponsor_repositories;
--
--  drop table topic;
--  drop table repositories_match_topic;
--  drop table collections;
--  drop table repositories_match_collections;
--  drop table releases;
--  drop table packages;
--  drop table code;
--  drop table issues;
--  drop table pull_requests;
--
--  drop table popular_repositories;
--  drop table popular_users;
--  drop table user_fork_repositories;
--  drop table user_develop_repositories;
--  drop table user_respond_issue_repositories;
--  drop table user_fileschangedin_repositories;
--  drop table users_star_topic;
--  drop table users_star_collections;
--  drop table user_sponsor_repositories;
--  drop table user_view_repositories;
--  drop table user_watch_repositories;
--  drop table user_star_repositories;
```

```sql
--  drop table user1_follow_user2;
--  drop table user1_star_user2;
--
--  drop table trending;



  --  建立 users 表,其中分为两个类，分别是 Individual user 和 organization
 CREATE TABLE `github_info`.`users` (
  `account_name` VARCHAR(45) NOT NULL,
  `email` VARCHAR(45) NULL DEFAULT NULL,
  `address` VARCHAR(45) NULL DEFAULT NULL,
  `type` VARCHAR(45) NOT NULL CHECK(`type` = "individualuser" or `type` =
"organization" ) ,
  `password` VARCHAR(145) NOT NULL CHECK( CHAR_LENGTH(`password`) >= 5 )
COMMENT '密码位数不应少于 5 位',
  PRIMARY KEY (`account_name`));

 --  建立 Individual user 表
 CREATE TABLE `github_info`.`individualuser` (
  `account_name` VARCHAR(45) NOT NULL,
  `name` VARCHAR(45) NULL DEFAULT NULL,
  `public_email` VARCHAR(45) NULL DEFAULT NULL,
  `bio` VARCHAR(145) NULL DEFAULT NULL,
  `URL` VARCHAR(145) NULL DEFAULT NULL,
  `Twitter_username` VARCHAR(45) NULL DEFAULT NULL,
  `Company` VARCHAR(145) NULL DEFAULT NULL,
  `Location` VARCHAR(145) NULL DEFAULT NULL,
  `password` VARCHAR(145) NOT NULL CHECK( CHAR_LENGTH(`password`) >= 5 )
COMMENT '密码位数不应少于 5 位',
  PRIMARY KEY (`account_name`));

--  建立 organization 表
 CREATE TABLE `github_info`.`organization` (
  `Org_act_name` VARCHAR(45) NOT NULL,
  `Contact_email` VARCHAR(45) NULL DEFAULT NULL,
  `Profile` VARCHAR(45) NULL DEFAULT NULL,
  `Websit` VARCHAR(45) NULL DEFAULT NULL CHECK(`Websit` like "http%" ) ,
  `Address` VARCHAR(45) NULL DEFAULT NULL,
  `Isverified` TINYINT COMMENT '1 为 Isverified，0 为 Isnotverified',
  `password` VARCHAR(145) NOT NULL CHECK( CHAR_LENGTH(`password`) >= 5 )
COMMENT '密码位数不应少于 5 位',
  PRIMARY KEY (`Org_act_name`));
```

```
  -- INSERT INTO `github_info`.`organization` (`Org_act_name`,
`Contact_email`,`Profile`,`Websit`) VALUES ('appsmith','14333','Build',
'http://www');


--  建立 indvuser_org 表，表示每个 organization 包含若干个 Individual_user
CREATE TABLE `github_info`.`indvuser_belong_org` (
  `account_name` VARCHAR(45) NOT NULL,
  `Org_account_name` VARCHAR(45) NOT NULL,
  `role` VARCHAR(45) NOT NULL CHECK( `role` = "owner" OR `role` = "Member" )
COMMENT '表示每个成员在 org 里面的角色，分为 owner 和 Member',
  PRIMARY KEY (`account_name`, `Org_account_name`));


 CREATE TABLE `github_info`.`team` (
  `organization_name` VARCHAR(45) NOT NULL,
  `team_name` VARCHAR(45) NOT NULL,
  `visibility` VARCHAR(45) NOT NULL CHECK( `visibility` = "visible" OR
`visibility` = "secret" ) COMMENT '表示可视性，其中 visible 表示 organization
内所有人都可以看到这个 team；secret 表示 team 内的人才可以看到这个 team',
  `description` VARCHAR(45) NULL,
  PRIMARY KEY (`organization_name`,`team_name`));

 CREATE TABLE `github_info`.`indvuser_belong_team` (
  `account_name` VARCHAR(45) NOT NULL,
  `Org_name` VARCHAR(45) NOT NULL,
  `team_name` VARCHAR(45) NOT NULL,
  `role` VARCHAR(45) NULL CHECK( `role` = "Maintainer" OR `role` = "Member" )
COMMENT '表示每个成员在 team 里面的角色，分为 Maintainer 和 Member',
  PRIMARY KEY (`account_name`, `Org_name`, `team_name`));


 CREATE TABLE `github_info`.`events` (
  `name` VARCHAR(45) NOT NULL,
  `date_beginning` DATE NOT NULL,
  `date_ending` DATE NOT NULL,
  `description` VARCHAR(45) NULL,
  PRIMARY KEY (`name`,`date_beginning`))
COMMENT = 'Connect with the GitHub community at conferences, meetups, and
hackathons around the world.';

CREATE TABLE `github_info`.`user_join_events` (
  `account_name` VARCHAR(45) NOT NULL,
  `event_name` VARCHAR(45) NOT NULL,
```

```
  `date_beginning` DATE NOT NULL,
  PRIMARY KEY (`account_name`,`event_name`,`date_beginning` ))
COMMENT = 'Individual user join in events';

-- install app and action

 CREATE TABLE `github_info`.`app` (
  `name` VARCHAR(45) NOT NULL,
  `Introduction` VARCHAR(45) NULL,
  `languages` VARCHAR(45) NULL,
  `price` INT NOT NULL,
  `install_num` INT NOT NULL,
  PRIMARY KEY (`name`))
COMMENT = 'Extend GitHub';

CREATE TABLE `github_info`.`action` (
  `name` INT NOT NULL,
  `Stars_number` INT NULL,
  `version` VARCHAR(45) NULL,
  `money` INT NOT NULL,
  PRIMARY KEY (`name`));

  CREATE TABLE `github_info`.`users_install_app` (
  `account_name` VARCHAR(45) NOT NULL,
  `app_name` VARCHAR(45) NOT NULL,
  `date` DATE NOT NULL,
  PRIMARY KEY (`account_name`, `app_name`,`date`));

  CREATE TABLE `github_info`.`users_install_action` (
  `account_name` VARCHAR(45) NOT NULL,
  `action_name` VARCHAR(45) NOT NULL,
  `date` DATE NOT NULL,
  PRIMARY KEY (`account_name`, `action_name`,`date`));

  CREATE TABLE `github_info`.`org_develop_app` (
  `account_name` VARCHAR(45) NOT NULL,
  `app_name` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`account_name`, `app_name`));

  CREATE TABLE `github_info`.`user_contribute_action` (
  `account_name` VARCHAR(45) NOT NULL,
  `action_name` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`account_name`, `action_name`));
```

```sql
 CREATE TABLE `github_info`.`user1_follow_user2` (
  `user1` VARCHAR(45) NOT NULL,
  `user2` VARCHAR(45) NOT NULL,
  `date` DATE NULL,
  PRIMARY KEY (`user1`, `user2`));

CREATE TABLE `github_info`.`user1_star_user2` (
  `user1` VARCHAR(45) NOT NULL,
  `user2` VARCHAR(45) NOT NULL,
  `date` DATE NULL,
  PRIMARY KEY (`user1`, `user2`));


--  Repositories

CREATE TABLE `github_info`.`repositories` (
  `Repository_name` VARCHAR(45) NOT NULL,
  `Owner` VARCHAR(45) NOT NULL,
  `Description` VARCHAR(45) NULL,
  `Public_or_private` TINYINT COMMENT '1 为 Public，0 为 private',
  `Updated_on` DATE NULL,
  `star_number` INT NOT NULL,
  `watch_number` INT NOT NULL,
  `fork_number` INT NOT NULL,
  `sponor_or_not` TINYINT COMMENT '1 为 sponor，0 为 not sponor',
  PRIMARY KEY (`Repository_name`, `Owner`));

 CREATE TABLE `github_info`.`sponsor_repositories` (
  `Repository_name` VARCHAR(45) NOT NULL,
  `Owner` VARCHAR(45) NOT NULL,
  `Description` VARCHAR(45) NULL,
  `Public_or_private` TINYINT COMMENT '1 为 Public，0 为 private',
  `goal` INT NOT NULL COMMENT 'sponsor 的目标，比如每月 50 英镑',
  `Updated_on` DATE NULL,
  `star_number` INT NOT NULL,
  `watch_number` INT NOT NULL,
  `fork_number` INT NOT NULL,
  PRIMARY KEY (`Repository_name`, `Owner`));

  CREATE TABLE `github_info`.`topic` (
  `topic_name` VARCHAR(45) NOT NULL,
  `description` VARCHAR(145) NOT NULL,
  `repositories_num` INT NULL,
  PRIMARY KEY (`topic_name`))
```

```
COMMENT = '每个 topic 会收纳一些 Repositories';

 CREATE TABLE `github_info`.`repositories_match_topic` (
  `Repository_name` VARCHAR(45) NOT NULL,
  `Owner` VARCHAR(45) NOT NULL,
  `topic_name` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`Repository_name`, `owner`, `topic_name`));

  CREATE TABLE `github_info`.`collections` (
  `collection_name` VARCHAR(45) NOT NULL,
  `description` VARCHAR(145) NOT NULL,
  `repositories_num` INT NULL,
  PRIMARY KEY (`collection_name`))
COMMENT = '每个 topic 会收纳一些 Repositories';



 CREATE TABLE `github_info`.`repositories_match_collections` (
  `Repository_name` VARCHAR(45) NOT NULL,
  `Owner` VARCHAR(45) NOT NULL,
  `collection_name` VARCHAR(45) NOT NULL,
PRIMARY KEY (`Repository_name`, `owner`, `collection_name`));

  CREATE TABLE `github_info`.`releases` (
  `Repository_name` VARCHAR(45) NOT NULL,
  `Owner` VARCHAR(45) NOT NULL,
  `Tag` VARCHAR(45) NOT NULL,
  `release_date` DATE NOT NULL,
  `Publisher` VARCHAR(45) NULL,
  `download_URL` VARCHAR(145) NULL CHECK(`download_URL` LIKE "http://%" or
`download_URL` LIKE "https://%"),
  PRIMARY KEY (`Repository_name`, `Owner`, `Tag`));

 CREATE TABLE `github_info`.`packages` (
  `Repository_name` VARCHAR(45) NOT NULL,
  `Owner` VARCHAR(45) NOT NULL,
  `Packages_name` VARCHAR(45) NOT NULL,
  `type` VARCHAR(45) NOT NULL CHECK (`type` = "Docker" or `type` = "Apache
Mave" or `type` = "NuGet" or `type` = "RubyGems" or `type` = "npm" or `type`
= "Containers" ),
  `release_date` DATE NULL,
  PRIMARY KEY (`Repository_name`, `Owner`,  `Packages_name`, `type`));

CREATE TABLE `github_info`.`code` (
```

```
  `Repository_name` VARCHAR(45) NOT NULL,
  `Owner` VARCHAR(45) NOT NULL,
  `commit_num` INT NULL DEFAULT 0,
  `branch_num` INT NULL DEFAULT 0,
  `tags_num` INT NULL DEFAULT 0,
  `last_update` DATE NULL,
  PRIMARY KEY (`Repository_name`, `Owner`));

CREATE TABLE `github_info`.`issues` (
  `Repository_name` VARCHAR(45) NOT NULL,
  `Owner` VARCHAR(45) NOT NULL,
  `number` INT NOT NULL,
  `account_name` VARCHAR(45) NOT NULL COMMENT '在这里加上了用户名字，由仓库
名和 issue 序号作为主键可以唯一确定是谁发出的 issue',
  `comment` VARCHAR(45) NULL,
  `date` DATE NULL,
  `open or close` TINYINT NULL COMMENT 'open is 1 and closed is 0',
  PRIMARY KEY (`Repository_name`, `Owner`, `number`));

  CREATE TABLE `github_info`.`pull_requests` (
  `Repository_name` VARCHAR(45) NOT NULL,
  `Owner` VARCHAR(45) NOT NULL,
  `number` INT NOT NULL,
  `comment` VARCHAR(45) NULL,
  `date` DATE NOT NULL,
  `open_or_close` TINYINT NOT NULL,
  PRIMARY KEY (`Repository_name`, `Owner`, `number`));


-- 一些操作

CREATE TABLE `github_info`.`user_sponsor_repositories` (
  `account_name` VARCHAR(45) NOT NULL,
  `repository_name` VARCHAR(45) NOT NULL,
  `sponsor-repositories-owner` VARCHAR(45) NOT NULL,
  `money` INT NOT NULL COMMENT 'Choose a custom amount.',
  `frequency` VARCHAR(45) NOT NULL CHECK (`frequency` = "Monthly" or
`frequency` = "One-time") COMMENT 'Monthly and One-time',
  PRIMARY KEY (`account_name`, `repository_name`,
`sponsor-repositories-owner` ));

CREATE TABLE `github_info`.`user_view_repositories` (
  `owner` VARCHAR(45) NOT NULL,
  `repository_name` VARCHAR(45) NOT NULL,
```

```
  `account_name` VARCHAR(45) NOT NULL,
  `date` DATE NOT NULL,
  PRIMARY KEY (`account_name`, `repository_name`, `owner`));

CREATE TABLE `github_info`.`user_fork_repositories` (
  `owner` VARCHAR(45) NOT NULL,
  `repository_name` VARCHAR(45) NOT NULL,
  `account_name` VARCHAR(45) NOT NULL,
  `date` DATE NOT NULL,
  PRIMARY KEY (`account_name`, `repository_name`, `owner`));

CREATE TABLE `github_info`.`user_star_repositories` (
  `owner` VARCHAR(45) NOT NULL,
  `repository_name` VARCHAR(45) NOT NULL,
  `account_name` VARCHAR(45) NOT NULL,
  `date` DATE NOT NULL,
  PRIMARY KEY (`account_name`, `repository_name`, `owner`));

CREATE TABLE `github_info`.`user_watch_repositories` (
  `owner` VARCHAR(45) NOT NULL,
  `repository_name` VARCHAR(45) NOT NULL,
  `account_name` VARCHAR(45) NOT NULL,
  `date` DATE NOT NULL,
  PRIMARY KEY (`account_name`, `repository_name`, `owner`));

 CREATE TABLE `github_info`.`user_develop_repositories` (
  `owner` VARCHAR(45) NOT NULL,
  `repository_name` VARCHAR(45) NOT NULL,
  `account_name` VARCHAR(45) NOT NULL,
  `role` VARCHAR(45) NOT NULL CHECK(`role` = "owner" or `role` =
"member" )COMMENT 'include owner and member',
  PRIMARY KEY (`account_name`, `repository_name`, `owner`));




  CREATE TABLE `github_info`.`user_respond_issue_repositories` (
  `owner` VARCHAR(45) NOT NULL,
  `repository_name` VARCHAR(45) NOT NULL,
  `account_name` VARCHAR(45) NOT NULL,
  `number` INT COMMENT 'the numbers of issues',
  `date` DATE NOT NULL,
  `response` VARCHAR(45) ,
  PRIMARY KEY (`account_name`, `repository_name`, `owner`, `number`));
```

```
 CREATE TABLE `github_info`.`user_fileschangedin_repositories` (
 `owner` VARCHAR(45) NOT NULL,
 `repository_name` VARCHAR(45) NOT NULL,
 `account_name` VARCHAR(45) NOT NULL,
 `number` INT NOT NULL,
 `files` VARCHAR(45) NOT NULL,
 `changed_line_content` VARCHAR(145),
 `changed_line_num` VARCHAR(45),
 PRIMARY KEY (`account_name`, `repository_name`, `owner`, `number`));


 CREATE TABLE `github_info`.`users_star_topic` (
 `account_name` VARCHAR(45) NOT NULL,
 `topic_name` VARCHAR(45) NOT NULL,
 `date` DATE NOT NULL,
 PRIMARY KEY (`account_name`, `topic_name`));

 CREATE TABLE `github_info`.`users_star_collections` (
 `account_name` VARCHAR(45) NOT NULL,
 `collections_name` VARCHAR(45) NOT NULL,
 `date` DATE NOT NULL,
 PRIMARY KEY (`account_name`, `collections_name`));

  CREATE TABLE `github_info`.`users_star_action` (
 `account_name` VARCHAR(45) NOT NULL,
 `action_name` VARCHAR(45) NOT NULL,
 `date` DATE NOT NULL,
 PRIMARY KEY (`account_name`, `action_name`));

CREATE TABLE `github_info`.`popular_repositories` (
 `owner` VARCHAR(45) NOT NULL,
 `repository_name` VARCHAR(45) NOT NULL,
 `language` VARCHAR(45) NOT NULL,
 `date_range` VARCHAR(45) NOT NULL,
 PRIMARY KEY (`owner`, `repository_name`, `date_range`));

 CREATE TABLE `github_info`.`popular_users` (
 `account_name` VARCHAR(45) NOT NULL,
 `date_range` VARCHAR(45) NOT NULL ,
 PRIMARY KEY (`account_name`,  `date_range`));

-- show create table users;
```