|  |  |  |  |
| --- | --- | --- | --- |
| **完整性**  **（40分）** | **深度**  **（40分）** | **创新性**  **（20分）** | **总分**  **（100分）** |
|  |  |  |  |

武汉理工大学

**2025-2026（1）**

**零工经济—业务分析与数据库设计和实现**

**目录**

**[一、 需求分析 1](#_Toc32327)**

**[1． 宏观背景与市场动因 1](#_Toc2697)**

**[2． 目标用户和核心功能需求 1](#_Toc18323)**

**[3． 平台级协同需求 2](#_Toc29748)**

**[4． 需求分析ER图 2](#_Toc26502)**

**[二、 数据库构建与操作 8](#_Toc29858)**

**[1． 数据库设计与完整性约束 8](#_Toc24586)**

**[2． 功能支撑与SQL实现 8](#_Toc19916)**

**[三、 性能分析 11](#_Toc31747)**

**[1． 规范化分析 11](#_Toc13459)**

**[2． 数据冗余问题识别 14](#_Toc24205)**

**[3． 恢复技术 15](#_Toc27613)**

**[4． 并发 16](#_Toc10261)**

**成员任务分布**

刘一语：支付与担保系统ER图绘制，SQL语句的攥写以及相应解释。

张楠婕：招聘者侧和总需求ER图绘制，电子版PPT制作。

龙姿男：应聘者侧ER图绘制，Word版报告制作,PPT讲解。

王雯霏：工作流程引擎ER图绘制，性能分析。

**正文**

# 需求分析

## 宏观背景与市场动因

零工经济的兴起并非偶然，而是社会、经济与技术多重因素共同作用的结果。首先，新一代就业观念的转变是核心驱动力，如今的年轻人更追求自由与灵活的工作方式，如90后、00后群体普遍青睐兼职零工。其次，从企业端看，成本考量促使众多企业，尤其是在业务旺季时，倾向于招聘零工以应对人力需求波动，快递行业便是典型例子。最后，互联网技术的飞速发展，特别是以Boss直聘APP为代表的平台，极大地提升了零工供需双方的对接效率，为零工经济的规模化发展提供了技术基石。

然而，零工经济在蓬勃发展的同时也暴露了其固有的弊端。数据显示，当前市场面临的主要挑战包括“人员配置不足”和“劳工信息不够透明”，这两大问题构成了行业发展的主要障碍。此外，“缺少信誉评价系统”等问题也亟待解决，这些痛点为我们设计系统提供了明确的方向。

## 目标用户和核心功能需求

我们将目标用户划分为两大核心角色：个体求职者（应聘者）与企业招聘方（招聘者）。

* + 1. **应聘者侧需求：**

应聘者的核心诉求是高效、可靠地获取工作机会并管理个人职业发展。

**个人资料与人才库管理：**需要一个完善的系统来创建、编辑和维护个人简历，并能展示技能、经验与作品集，构建个人品牌。

**工作机会探索与申请：**平台需提供强大的搜索与筛选功能，并支持一键申请，以降低求职门槛。

**工作流程与项目管理：**应聘者需要一个清晰的界面来跟踪申请进度、查看面试安排及管理正在进行的项目。

* + 1. **招聘者侧需求：**

招聘者的核心诉求是快速、精准地找到合适的人才并高效管理招聘流程。

**公司信息与招聘需求管理：**招聘者需要能够发布和更新公司介绍、文化福利，并灵活地创建、编辑和关闭职位。

**人才发现与候选人管理：**平台应具备智能的人才搜索、筛选与推荐功能，并提供简历库管理、标签分类和进度跟踪等工具，以沉淀人才资产。

**项目创建与流程管控：**对于短期或项目制用工，招聘者需要能够创建项目、定制招聘流程、分配任务并进行审批管理。

## 平台级协同需求

除了单侧用户需求，平台还需构建支撑双方高效协同的核心机制：

* + 1. **工作流程引擎：**为确保项目顺利推进，平台需建立项目状态追踪、里程碑与交付物管理以及任务分配与提醒系统。
    2. **支付与担保系统：**资金安全是零工经济的信任基石。平台必须建立安全可靠的资金托管机制，支持多种支付方式，并设计能根据项目进度自动触发的智能支付与结算系统。

## 需求分析ER图

* + 1. **应聘者侧ER图**

**功能：**用于管理应聘者的个人背景、求职意向及职位申请流程，记录应聘者的求职相关信息与申请状态。

**表数量：**5张表（应聘者表、工作经历表、教育经历表、求职意向表、申请表）。

**每张表的主键和属性：**

应聘者表：主键「应聘者ID」；属性含应聘者基础信息。

工作经历表：主键「工作经历ID」；属性含工作经历ID、应聘者ID、工作经历内容、公司、职位。

教育经历表：主键「教育经历ID」；属性含教育经历ID、应聘者ID、学校、学历、专业。

求职意向表：主键「求职意向ID」；属性含求职意向ID、应聘者ID、职位、薪资。

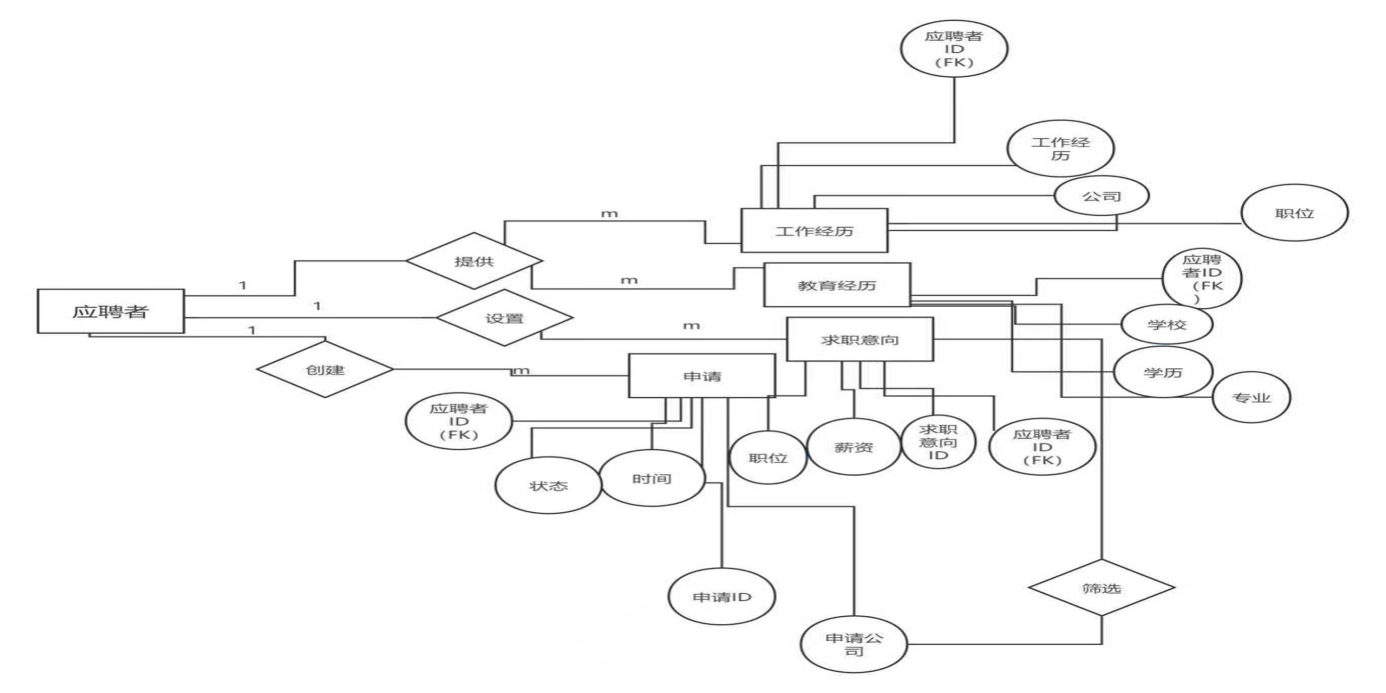
申请表：主键「申请 ID」；属性含申请ID、应聘者ID、时间、状态、申请公司。

**外键关系：**

工作/教育经历表、求职意向表、申请表的「应聘者ID」均引用应聘者表的「应聘者ID」。

**关系基数：**

应聘者与工作/教育经历/求职意向/申请：均为「1 对多」（1 个应聘者可对应多个工作经历、教育经历等）。

****

* + 1. **招聘者侧ER图**

**功能：**用于管理招聘者、公司信息、招聘职位发布，以及招聘者发现候选人的流程。

**表数量：**4 张表（招聘者表、公司表、职位表、候选人表）。

**每张表的主键和属性：**

招聘者表：主键「招聘者 ID」；属性含招聘者 ID、姓名、手机号、邮箱、公司 ID。

公司表：主键「公司 ID」；属性含公司 ID、名称、简介、营业执照。

职位（招聘需求）表：主键「职位 ID」；属性含职位 ID、名称、描述、公司 ID、招聘者 ID。

候选人表：主键「候选人 ID」；属性含候选人 ID、姓名、邮箱、手机号、简历。

**外键关系：**

招聘者表「公司 ID」引用公司表的「公司 ID」；

职位表「公司 ID」「招聘者 ID」分别引用公司表「招聘者表」的对应主键。

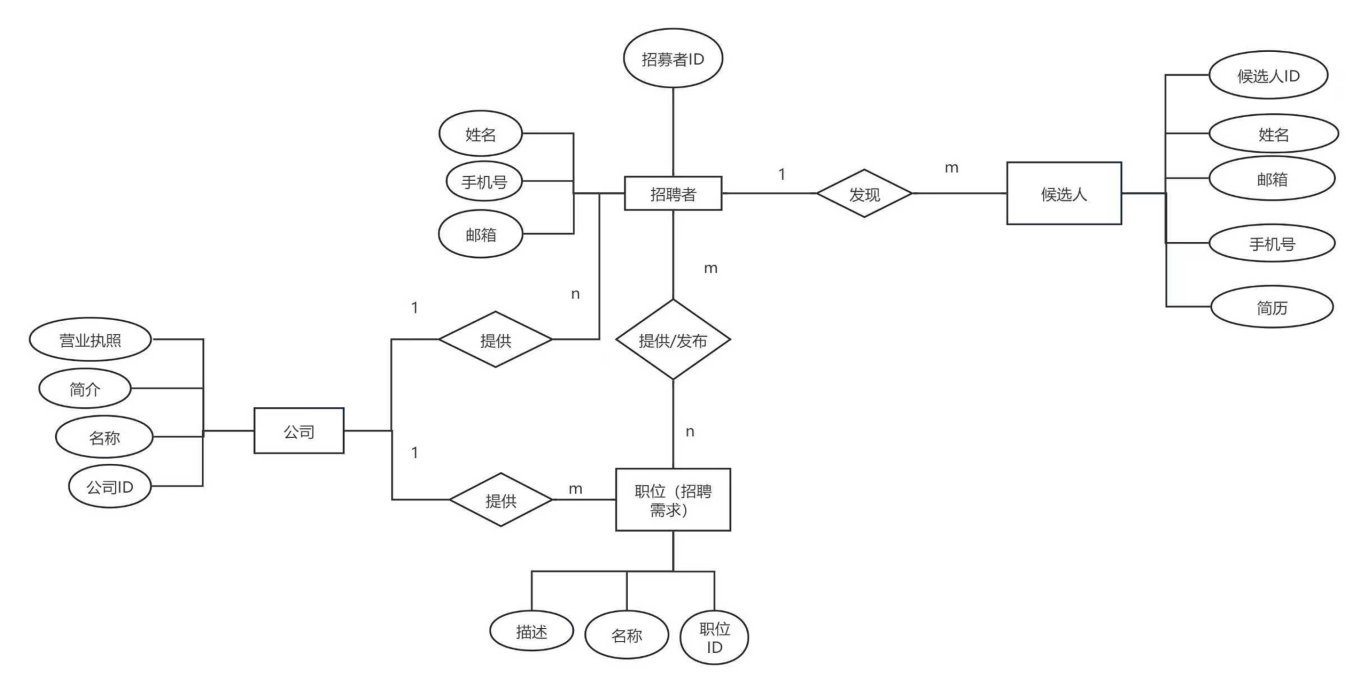
**关系基数：**

公司与招聘者：1 对多（1 家公司对应多个招聘者）；

招聘者与职位：1 对多（1 个招聘者发布多个职位）；

公司与职位：1 对多（1 家公司提供多个职位）；

招聘者与候选人：1 对多（1 个招聘者发现多个候选人）。



* + 1. **工作流程引擎ER图**

**功能：**用于跟踪项目发布、应聘者参与项目、支付流程及交付物管理，覆盖项目全生命周期的信息记录。

**表数量：**6张表（项目表、里程碑表、招聘者表、应聘者表、支付表、交付物表）。

**每张表的主键和属性：**

项目表：主键「项目 ID」；属性含项目 ID、名称、状态、描述、预算、招聘者 ID。

里程碑表：主键「里程碑 ID」；属性含里程碑 ID、项目 ID、名称、状态、时间。

招聘者表：主键「招聘者 ID」；属性含招聘者 ID、姓名、邮箱、电话。

应聘者表：主键「应聘者 ID」；属性含应聘者 ID、姓名、电话、邮箱。

支付表：主键「支付 ID」；属性含支付 ID、应聘者 ID、交付金额、状态、时间。

交付物表：主键「交付物 ID」；属性含交付物 ID、应聘者 ID、项目 ID、里程碑 ID、名称。

**外键关系：**

项目表「招聘者 ID」引用招聘者表主键；

里程碑表「项目 ID」引用项目表主键；

支付表「应聘者 ID」、交付物表「应聘者 ID / 项目 ID / 里程碑 ID」引用对应表主键。

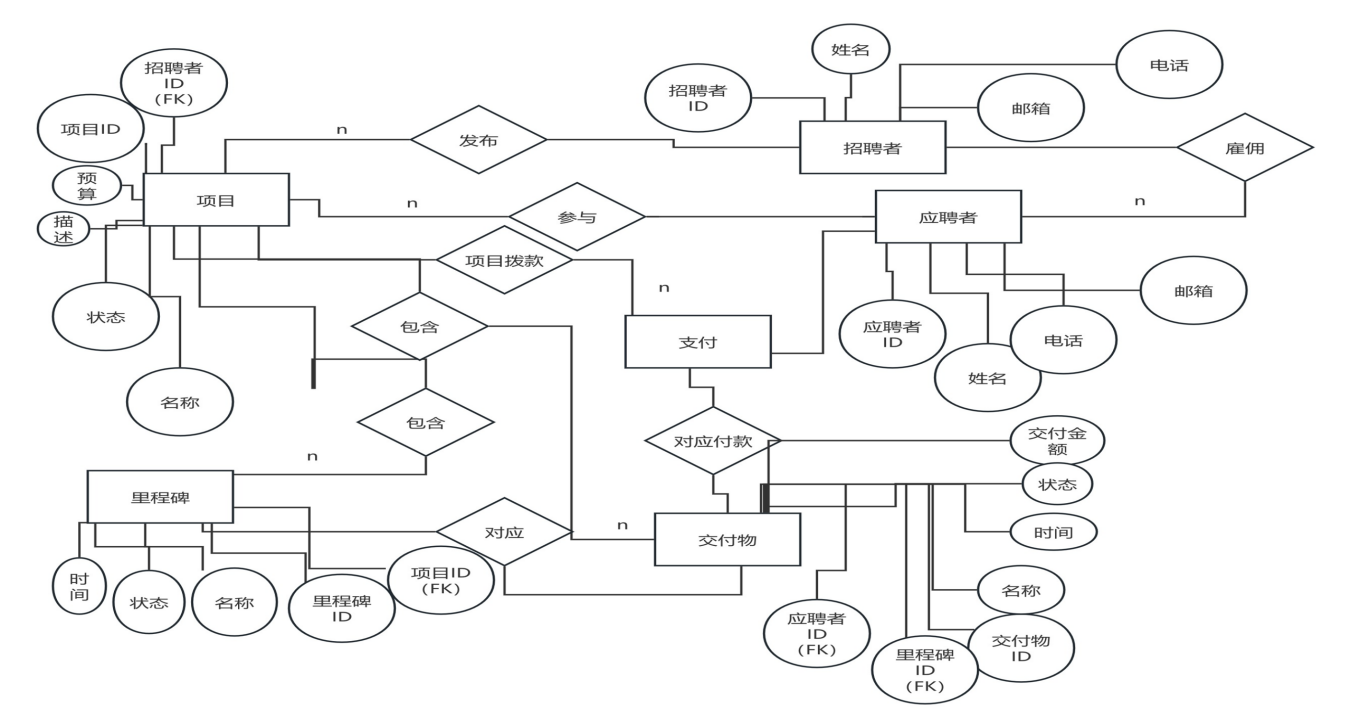
**关系基数：**

招聘者与项目：1 对多（1 个招聘者发布多个项目）；

项目与里程碑：1 对多（1 个项目含多个里程碑）；

招聘者与应聘者：1 对多（1 个招聘者雇佣多个应聘者）；

项目 / 应聘者与交付物：均为 1 对多（1 个项目/应聘者对应多个交付物）。



* + 1. **支付与担保系统ER图**

**功能：**用于管理招聘者的资产账户、资金托管订单、支付订单，以及应聘者的结算流程，处理招聘中的资金流转。

**表数量：**6 张表（招聘者表、资产账户表、托管订单表、支付订单表、结算单表、应聘者表）。

**每张表的主键和属性：**

招聘者表：主键「招聘者 ID」；属性含招聘者 ID、姓名、邮箱、联系电话。

资产账户表：主键「账户 ID」；属性含账户 ID、招聘者 ID、账户类型、状态、关联主体 ID、余额。

托管订单表：主键「托管订单 ID」；属性含托管订单 ID、招聘者 ID、金额、创建/到期时间、状态。

支付订单表：主键「支付订单 ID」；属性含支付订单 ID、托管订单 ID、金额、方式、状态、时间。

结算单表：主键「结算单 ID」；属性含结算单 ID、应聘者 ID、支付订单 ID、金额、类型、时间、状态。

应聘者表：主键「应聘者 ID」；属性含应聘者 ID、姓名、邮箱、联系电话、银行卡号。

**外键关系：**

资产账户表「招聘者 ID」引用招聘者表主键；

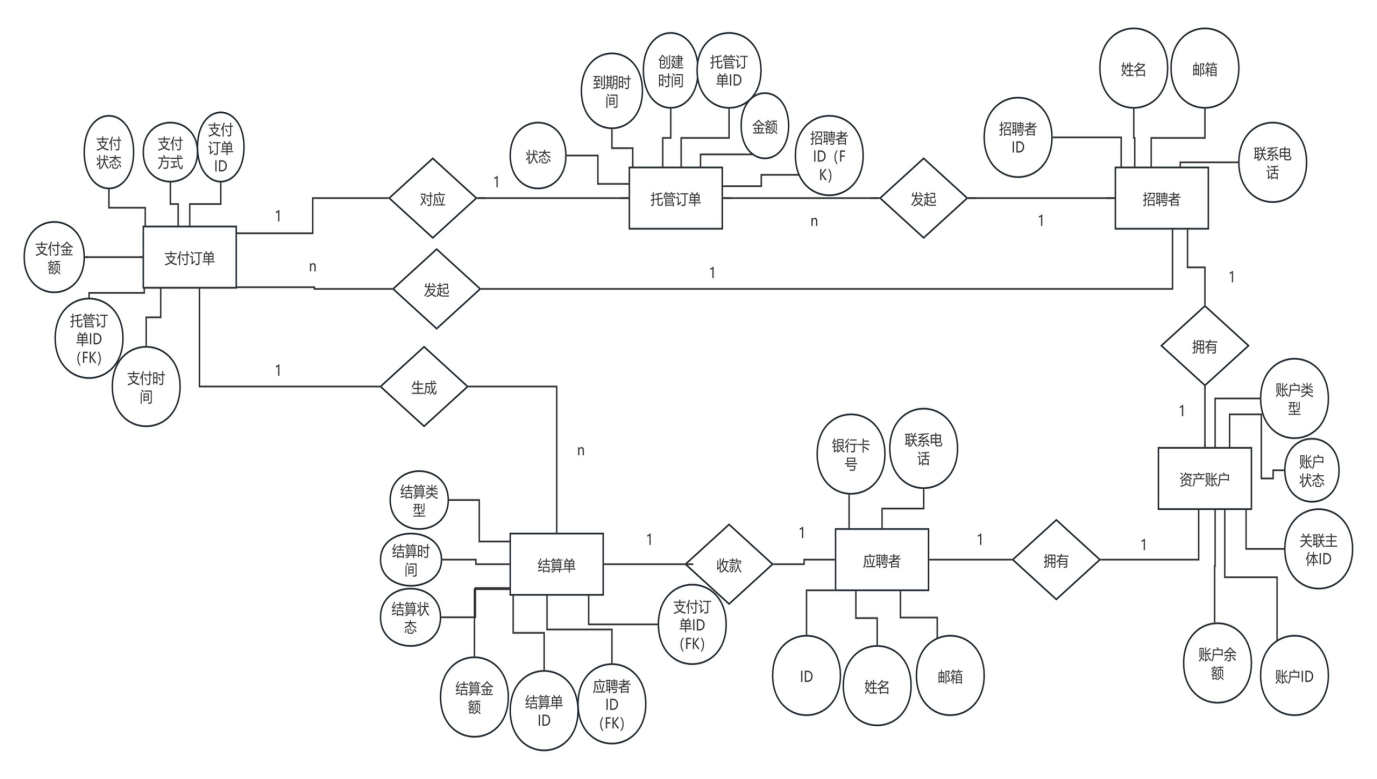
托管订单表「招聘者 ID」、支付订单表「托管订单 ID」、结算单表「应聘者 ID / 支付订单 ID」引用对应表主键。

**关系基数：**

招聘者与资产账户：1 对 1（1 个招聘者对应 1 个资产账户）；

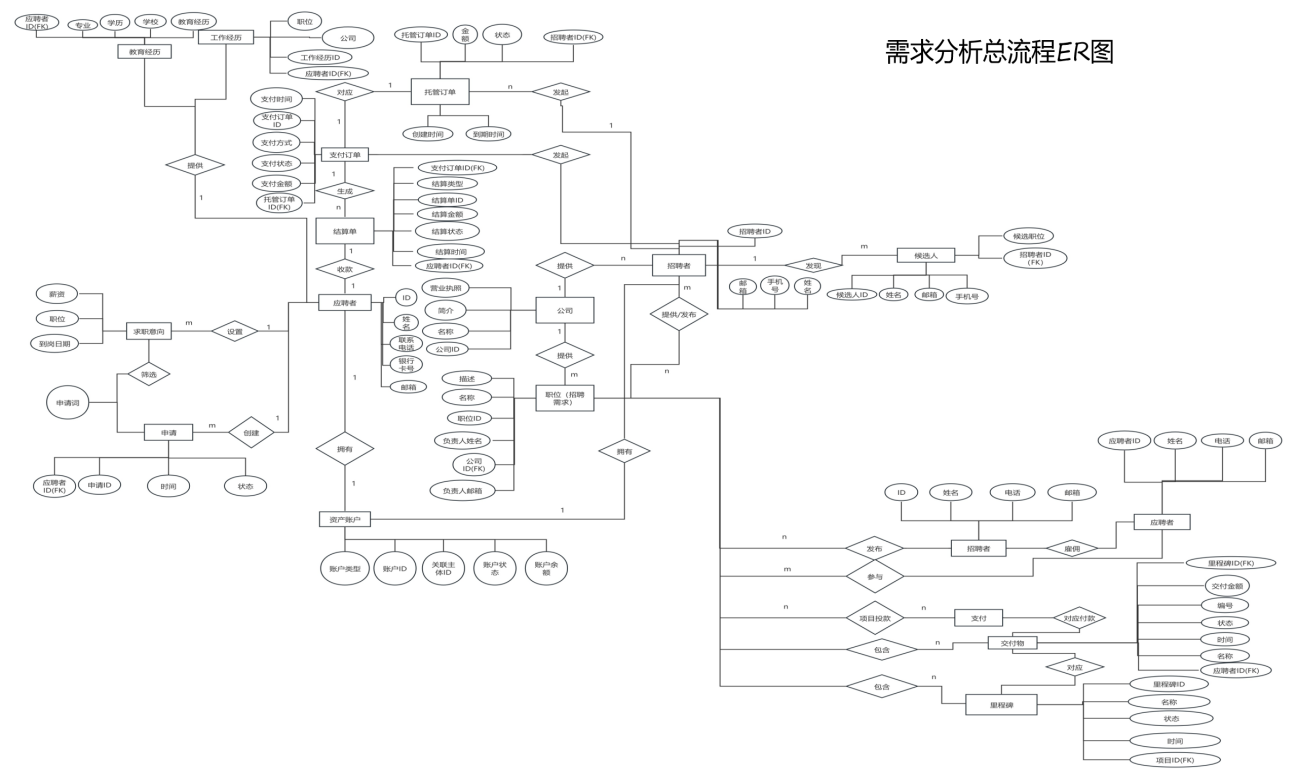
招聘者与托管订单：1 对多（1 个招聘者发起多个托管订单）；

托管订单与支付订单、支付订单与结算单：均为 1 对多。



* + 1. 总ER图

这张总需求ER图通过“招聘者-应聘者”作为核心串联点，左侧以应聘者为中心，关联申请、作品、资产账户等求职/资金模块；右侧以招聘者为中心，关联公司、职位、候选人等招聘模块；再通过“支付/结算”衔接招聘者的资金操作与应聘者的薪资流程，最终把招聘、求职、资金三大业务模块的实体通过外键和关系基数串联成完整的零工经济业务闭环。



# 数据库构建与操作

## 数据库设计与完整性约束

根据上述ER图，我们进行了数据库的逻辑结构设计，并通过SQL代码实现了完整的定义。数据库采用了模块化设计，通过不同的Schema（架构）来隔离应聘者、招聘者、工作流引擎和支付系统的业务逻辑，保证了系统的清晰性和可维护性。

**实体完整性：**所有核心表都定义了自增主键，例如应聘者表中的CandidateID INT PRIMARY KEY IDENTITY(1,1)，确保了每条记录的唯一性。

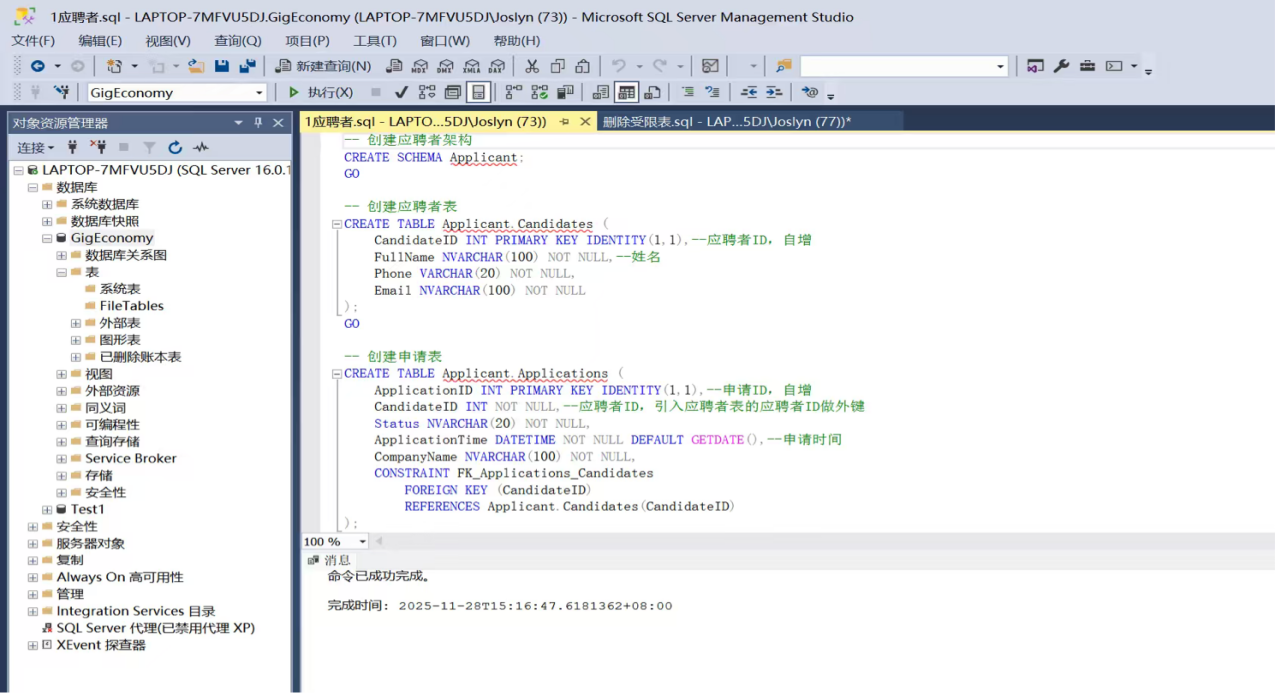
**参照完整性：**广泛使用了外键约束来保障数据的一致性。例如，申请表通过CONSTRAINT FK\_Applications\_Candidates FOREIGN KEY (CandidateID) REFERENCES Applicant.Candidates(CandidateID)确保每一条申请记录都必须对应一个有效的应聘者。同样，招聘者表通过外键关联到公司表，确保每个招聘者都属于一个合法的公司。

**用户定义完整性：** 通过NOT NULL约束（如应聘者的姓名、电话、邮箱）确保关键字段不为空，通过UNIQUE约束（如招聘者与职位的关系）防止重复关联，通过CHECK约束（如账户类型必须是'Recruiter'或'Candidate'）保证数据的有效性。

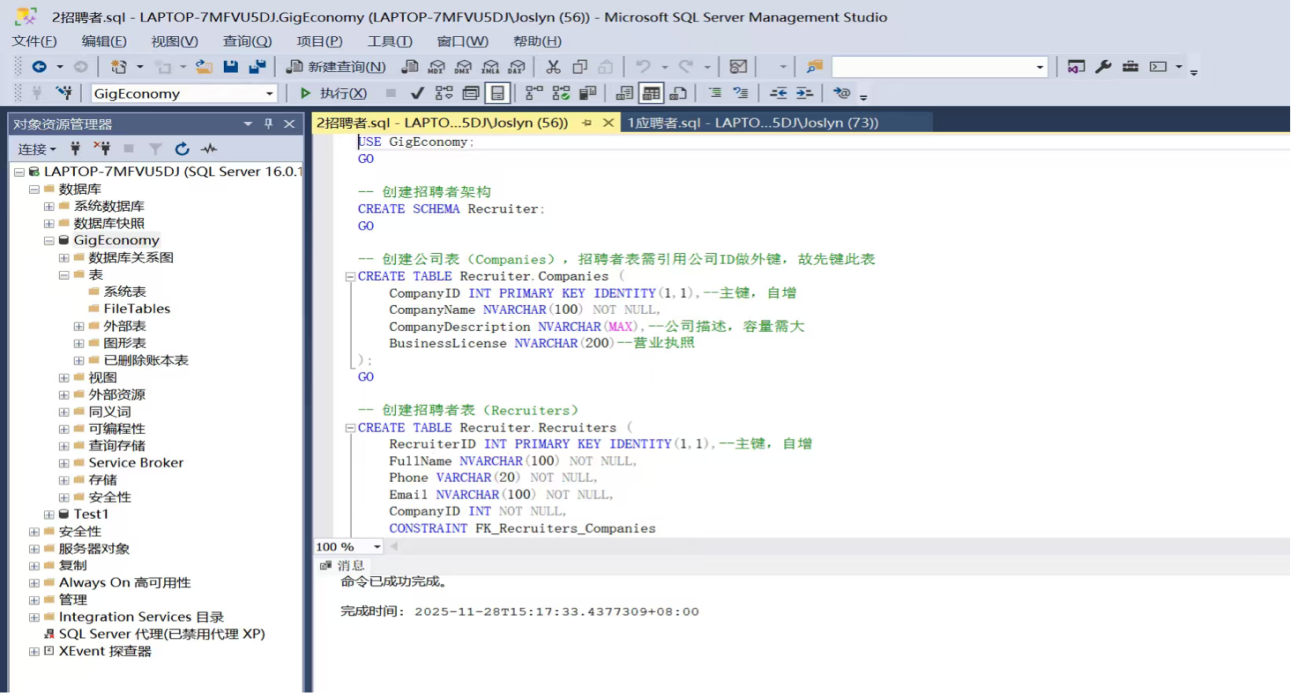
## 功能支撑与SQL实现

数据库的实现紧密围绕功能需求，通过精心设计的表结构和逻辑来支撑平台的各项核心功能。

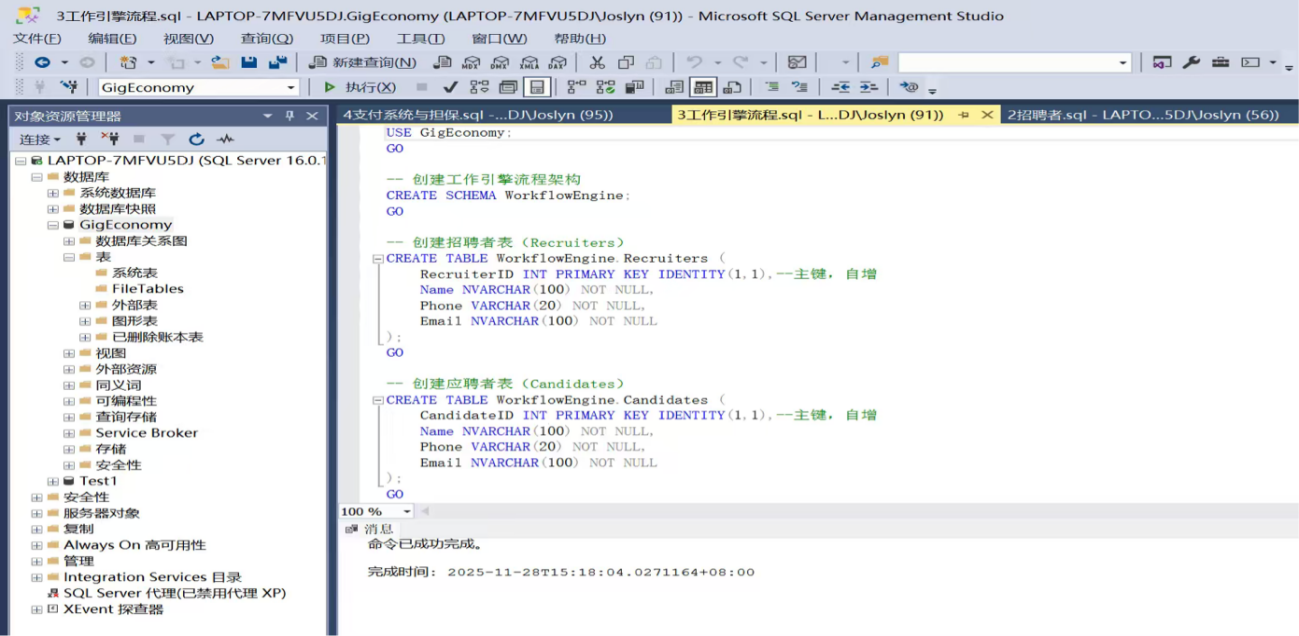
**应聘者模块：**该模块通过Applicant Schema实现，核心是Candidates表，它存储了应聘者的基本信息，如FullName, Phone, Email等，为个人资料管理提供了数据基础。Applications表则记录了应聘者的申请行为，通过外键关联Candidates表，实现了工作申请功能的追踪。此外，JobPreferences, WorkExperiences, Educations, Portfolios等表分别对应求职意向、工作经历、教育背景和作品集，共同构建了应聘者的完整画像。



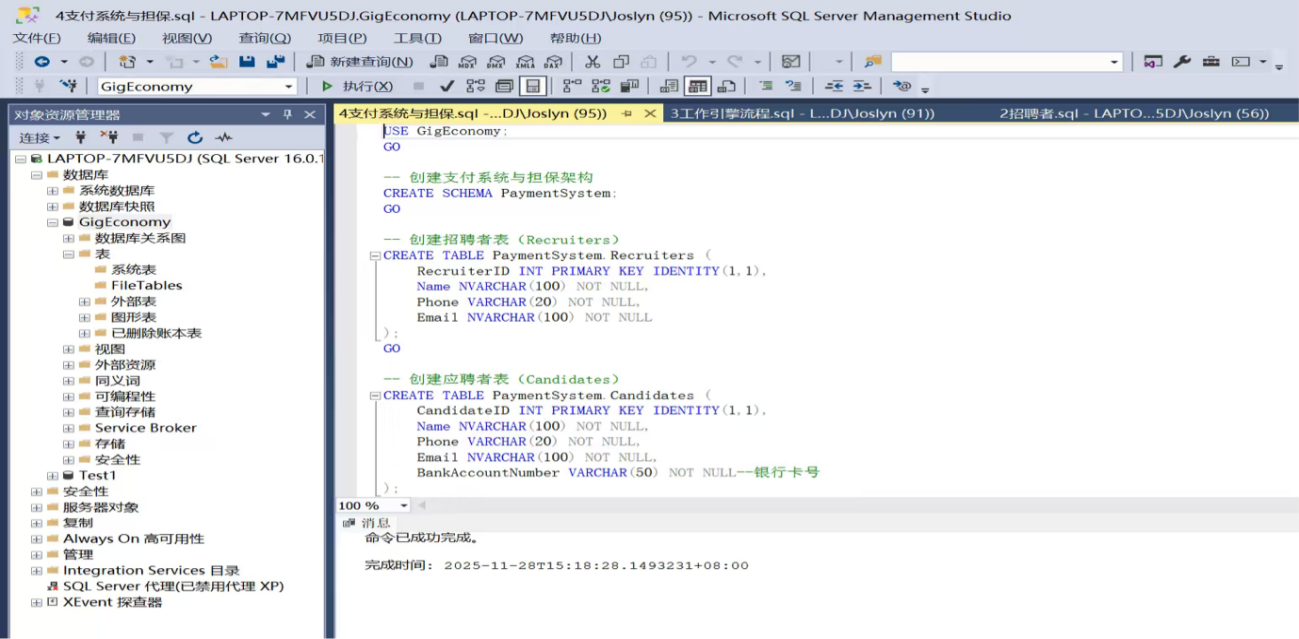
**招聘者模块：**该模块通过Recruiter Schema实现。Companies表存储公司信息，Recruiters表存储招聘者个人信息，并通过外键与Companies表关联，明确了招聘者的归属。JobRequirements表用于发布招聘需求，同样关联到公司，确保职位来源的合法性。特别值得一提的是RecruiterJobRelations表，它作为关联表，实现了招聘者与招聘需求之间的多对多关系，并添加了唯一约束以防止重复关联，体现了设计的严谨性。



**工作流引擎模块：**该模块通过WorkflowEngine Schema实现，旨在管理项目制零工。Projects表是核心，包含了项目名称、预算、描述和状态，并通过外键关联到负责的招聘者。Milestones（里程碑）和Deliverables（交付物）表则与项目关联，用于分解项目、跟踪进度和管理最终产出。ProjectCandidates关联表则清晰地定义了项目与应聘者之间的多对多协作关系。



**支付与担保系统模块：**这是保障平台信任的关键模块，通过PaymentSystem Schema实现。EscrowOrders（托管订单）表实现了资金冻结，确保在项目完成前资金安全。PaymentOrders（支付订单）与托管订单一对一关联，记录支付详情。SettlementOrders（结算单）则记录了资金最终流向应聘者的过程。



为了确保支付流程的绝对安全和可靠，我们设计了核心的存储过程PaymentSystem.ProcessPaymentSettlement。该过程通过BEGIN TRANSACTION开启事务，使用BEGIN TRY...CATCH进行错误捕获，并在发生错误时执行ROLLBACK TRANSACTION回滚所有操作，从而保证了资金转移的原子性：要么全部成功，要么全部失败。这一机制完美地解决了高并发场景下可能出现的“丢失更新”等问题，是系统健壮性的核心保障。

# 性能分析

## 规范化分析

* 1. 应聘者模块
     1. Applicant.Candidates 表函数依赖分析

主键：CandidateID

完全函数依赖：CandidateID → FullName, Phone, Email

部分函数依赖：无（单属性主键）

范式分析： 1NF-BCNF全部满足

* + 1. Applicant.Applications 表函数依赖分析

主键：ApplicationID

完全函数依赖：ApplicationID → CandidateID, Status, ApplicationTime, CompanyName

范式分析：1NF-BCNF全部满足

注意：CompanyName存在冗余，应该关联到JobRequirements表

* + 1. Applicant.JobPreferences 表函数依赖分析

主键：PreferenceID

完全函数依赖：PreferenceID → CandidateID, Position, ExpectedSalary

范式分析：1NF-BCNF全部满足

* + 1. Applicant.WorkExperiences 表函数依赖分析

主键：ExperienceID

完全函数依赖：ExperienceID → CandidateID, CompanyName, Position, StartDate, EndDate, Description

范式分析：1NF-BCNF全部满足

* + 1. Applicant.Educations 表函数依赖分析

主键：EducationID

完全函数依赖： EducationID → CandidateID, Degree, School, Major, StartDate, EndDate, GPA

范式分析：1NF-BCNF全部满足

* + 1. Applicant.Portfolios 表函数依赖分析

主键：PortfolioID

完全函数依赖：PortfolioID → CandidateID, PortfolioName, PortfolioDescription

范式分析：1NF-BCNF全部满足

* 1. 招聘者模块
     1. Recruiter.Companies 表函数依赖分析

主键：CompanyID

完全函数依赖：CompanyID → CompanyName, CompanyDescription, BusinessLicense

范式分析：1NF-BCNF：全部满足

* + 1. Recruiter.Recruiters 表函数依赖分析

主键：RecruiterID

完全函数依赖：RecruiterID → FullName, Phone, Email, CompanyID

传递函数依赖：RecruiterID → CompanyID → CompanyName（存在传递依赖，但不违反3NF，因为CompanyName不在此表中）

范式分析：1NF-BCNF全部满足

* + 1. Recruiter.JobRequirements 表函数依赖分析

主键：JobID

完全函数依赖：JobID → JobTitle, JobDescription, ContactPerson, ContactEmail, CompanyID

传递函数依赖：JobID → CompanyID → CompanyName（同上，不违反3NF）

范式分析：1NF-BCNF全部满足

* + 1. Recruiter.Candidates 表函数依赖分析

主键：CandidateID

完全函数依赖：CandidateID → FullName, Email, Phone, TargetPosition, RecruiterID

范式分析：1NF-BCNF全部满足

注意：此表与Applicant.Candidates存在重复，需要整合

* + 1. Recruiter.RecruiterJobRelations 表函数依赖分析

主键：RelationID

候选键：(RecruiterID, JobID)

完全函数依赖：RelationID → RecruiterID, JobID

(RecruiterID, JobID) → RelationID

范式分析： 1NF-BCNF全部满足

* 1. 工作引擎流程模块
     1. WorkflowEngine.Recruiters 表函数依赖分析

主键：RecruiterID

完全函数依赖：RecruiterID → Name, Phone, Email

范式分析：1NF-BCNF全部满足

* + 1. WorkflowEngine.Candidates 表函数依赖分析

主键：CandidateID

完全函数依赖：CandidateID → Name, Phone, Email

范式分析：1NF-BCNF全部满足

* + 1. WorkflowEngine.Projects 表函数依赖分析

主键：ProjectID

完全函数依赖：ProjectID → Name, Budget, Description, Status, RecruiterID, StartDate, EndDate

范式分析：1NF-BCNF全部满足

* + 1. WorkflowEngine.ProjectCandidates 表函数依赖分析

主键：(ProjectID, CandidateID) - 复合主键

完全函数依赖：(ProjectID, CandidateID) → Role, JoinDate, LeaveDate, IsActive

范式分析：1NF-BCNF满足（所有决定因素都是候选键）

* + 1. WorkflowEngine.Milestones 表函数依赖分析

主键：MilestoneID

完全函数依赖：MilestoneID → Name, Date, Status, ProjectID, TargetDate, CompletedDate, Priority, Description

传递函数依赖：MilestoneID → ProjectID → ProjectName（不违反3NF）

范式分析：1NF-BCNF全部满足

* + 1. WorkflowEngine.Deliverables 表函数依赖分析

主键：DeliverableID

完全函数依赖：DeliverableID → Name, Date, Status, PaymentAmount, MilestoneID, CandidateID

传递函数依赖：DeliverableID → MilestoneID → ProjectID（不违反3NF）

DeliverableID → CandidateID → CandidateName（不违反3NF）

范式分析：1NF-BCNF全部满足

* 1. 支付系统与担保模块
     1. PaymentSystem.Recruiters 表函数依赖分析

主键：RecruiterID

完全函数依赖：RecruiterID → Name, Phone, Email

范式分析： 1NF-BCNF全部满足

* + 1. PaymentSystem.Candidates 表函数依赖分析

主键：CandidateID

完全函数依赖：CandidateID → Name, Phone, Email, BankAccountNumber

范式分析：1NF-BCNF全部满足

* + 1. PaymentSystem.AssetAccounts 表函数依赖分析

主键：AccountID

候选键：(SubjectType, SubjectID) - 一对一关系

完全函数依赖：AccountID → Balance, SubjectType, SubjectID

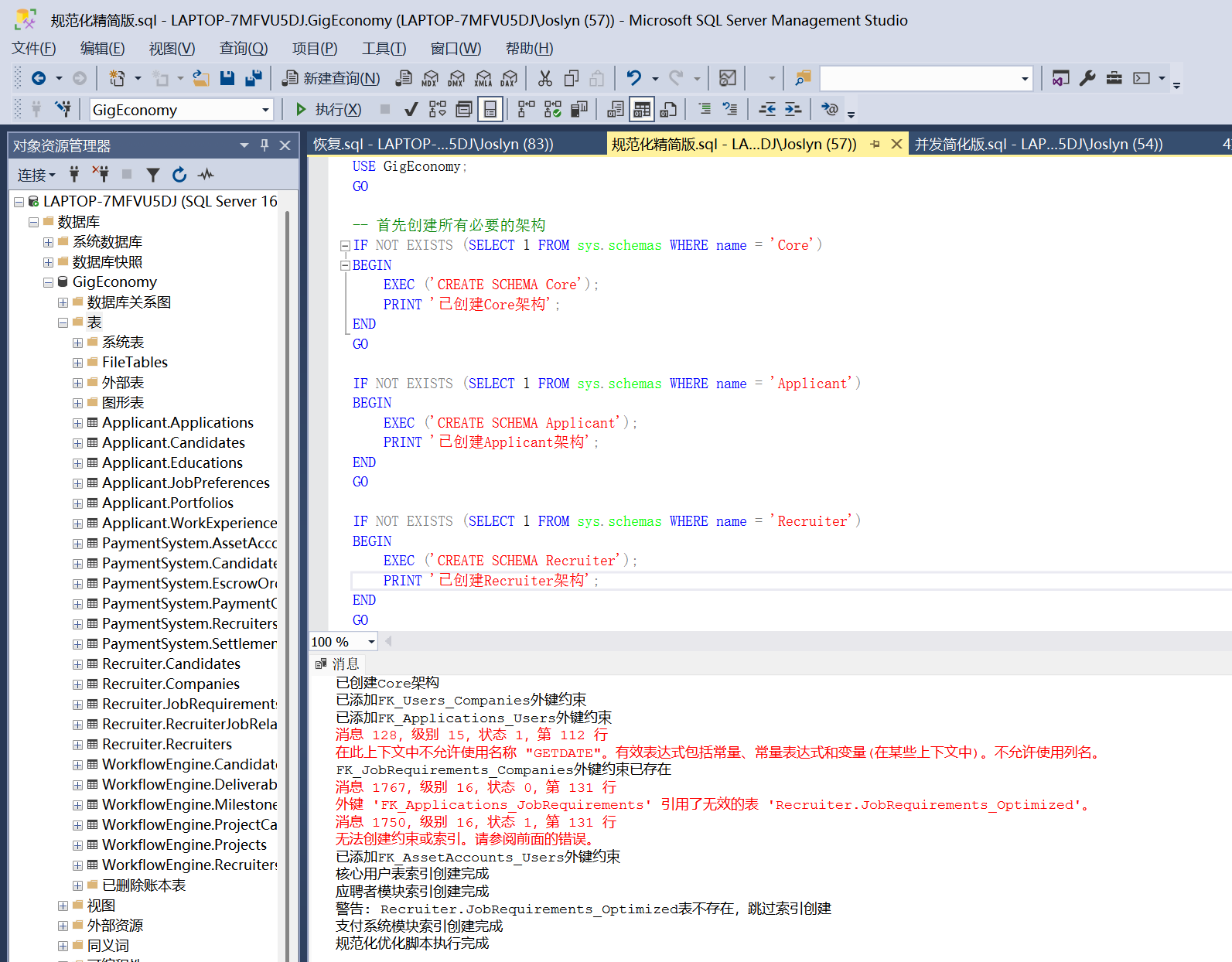
(SubjectType, SubjectID) → AccountID

范式分析：1NF-BCNF全部满足

## 数据冗余问题识别

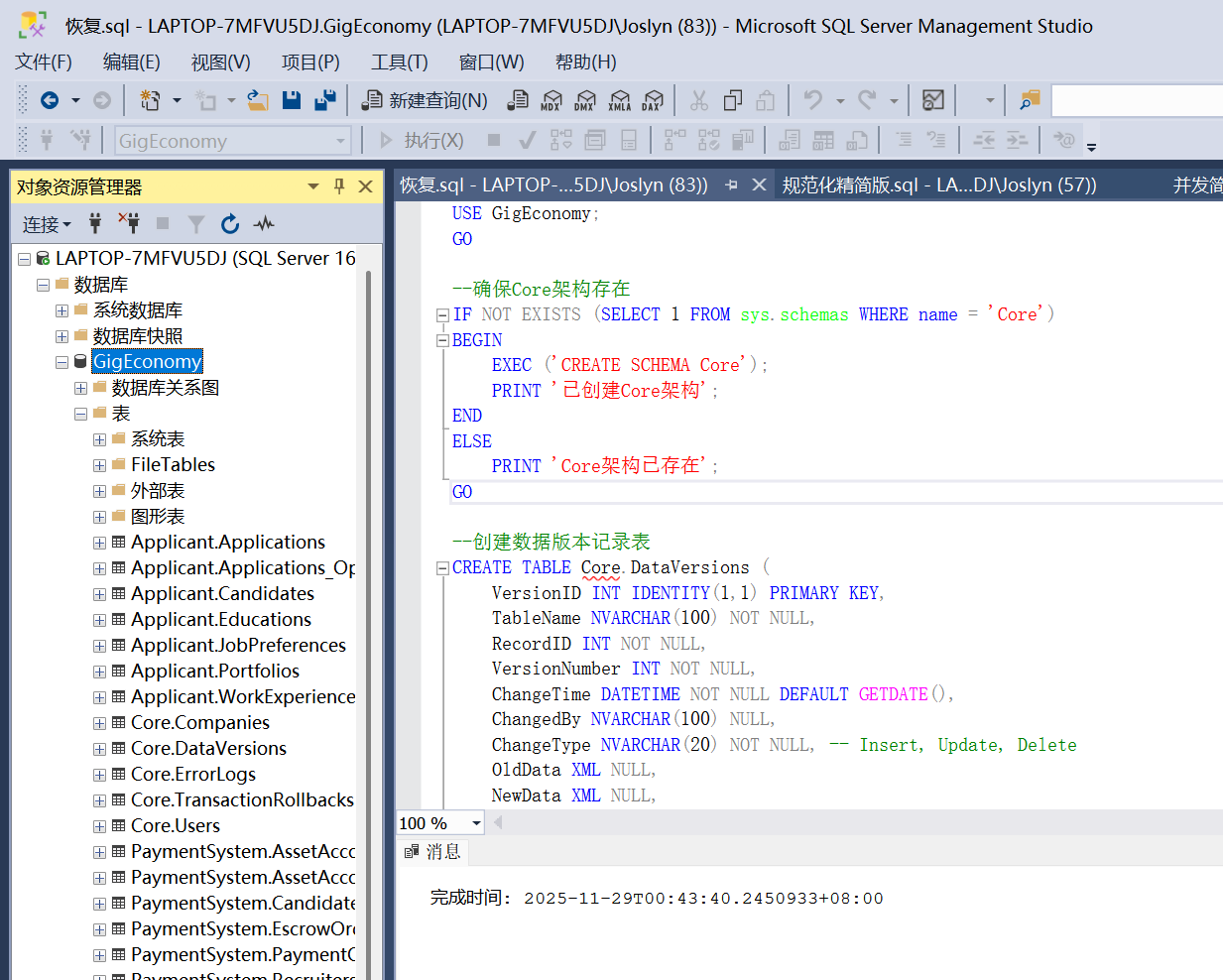
主要问题：

* 1. 数据冗余：多个模块中重复的用户表和候选者表
  2. 数据一致性风险：跨模块用户信息不同步
  3. 维护复杂度高：修改用户信息需要更新多个表

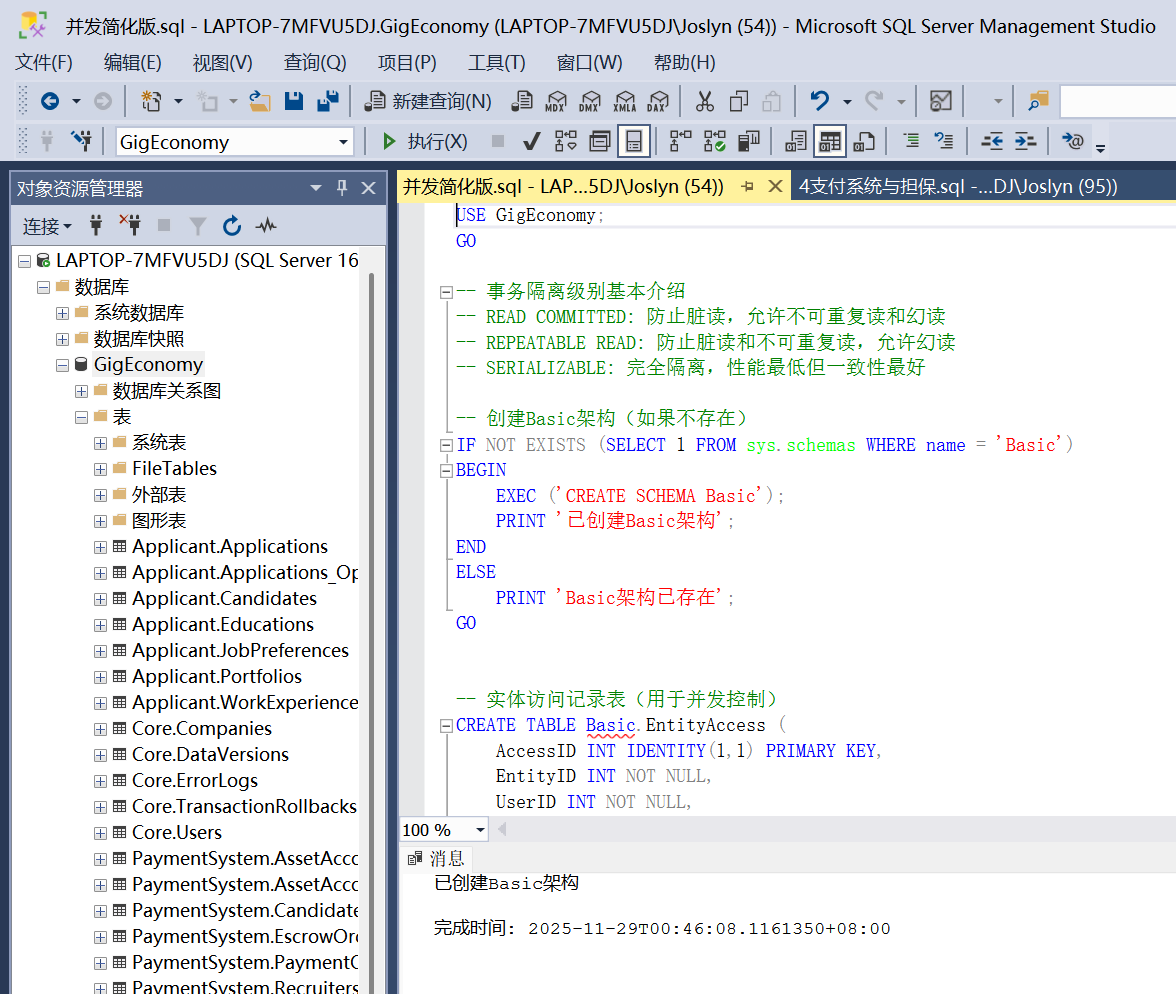
****

**后续报错已解决**

## 恢复技术

****

## 并发

****

**附录A 代码**

- 创建应聘者架构

CREATE SCHEMA Applicant;

GO

-- 创建应聘者表

CREATE TABLE Applicant.Candidates (

CandidateID INT PRIMARY KEY IDENTITY(1,1),--应聘者ID，自增

FullName NVARCHAR(100) NOT NULL,--姓名

Phone VARCHAR(20) NOT NULL,

Email NVARCHAR(100) NOT NULL

);

GO

-- 创建申请表

CREATE TABLE Applicant.Applications (

ApplicationID INT PRIMARY KEY IDENTITY(1,1),--申请ID，自增

CandidateID INT NOT NULL,--应聘者ID，引入应聘者表的应聘者ID做外键

Status NVARCHAR(20) NOT NULL,

ApplicationTime DATETIME NOT NULL DEFAULT GETDATE(),--申请时间

CompanyName NVARCHAR(100) NOT NULL,

CONSTRAINT FK\_Applications\_Candidates

FOREIGN KEY (CandidateID)

REFERENCES Applicant.Candidates(CandidateID)

);

GO

-- 创建求职意向表

CREATE TABLE Applicant.JobPreferences (

PreferenceID INT PRIMARY KEY IDENTITY(1,1),--求职意向ID，自增

CandidateID INT NOT NULL,

Position NVARCHAR(100) NOT NULL,

ExpectedSalary DECIMAL(10, 2) NOT NULL,

CONSTRAINT FK\_JobPreferences\_Candidates

FOREIGN KEY (CandidateID)

REFERENCES Applicant.Candidates(CandidateID)

);

GO

-- 创建工作经验表

CREATE TABLE Applicant.WorkExperiences (

ExperienceID INT PRIMARY KEY IDENTITY(1,1),--自增

CandidateID INT NOT NULL,

CompanyName NVARCHAR(100) NOT NULL,

Position NVARCHAR(100) NOT NULL,

CONSTRAINT FK\_WorkExperiences\_Candidates --引入应聘者表的应聘者ID做外键

FOREIGN KEY (CandidateID)

REFERENCES Applicant.Candidates(CandidateID)

);

GO

-- 创建教育经历表

CREATE TABLE Applicant.Educations (

EducationID INT PRIMARY KEY IDENTITY(1,1),--自增

CandidateID INT NOT NULL,

Degree NVARCHAR(50) NOT NULL,

School NVARCHAR(100) NOT NULL,

Major NVARCHAR(100) NOT NULL,

CONSTRAINT FK\_Educations\_Candidates --引入应聘者表的应聘者ID做外键

FOREIGN KEY (CandidateID)

REFERENCES Applicant.Candidates(CandidateID)

);

GO

-- 创建作品表

CREATE TABLE Applicant.Portfolios (

PortfolioID INT PRIMARY KEY IDENTITY(1,1),--自增

CandidateID INT NOT NULL,

PortfolioName NVARCHAR(100) NOT NULL,

PortfolioDescription NVARCHAR(MAX),--作品描述，数据容量需极大

CONSTRAINT FK\_Portfolios\_Candidates

FOREIGN KEY (CandidateID)

REFERENCES Applicant.Candidates(CandidateID)

);

GO

USE GigEconomy;

GO

-- 创建招聘者架构

CREATE SCHEMA Recruiter;

GO

-- 创建公司表（Companies），招聘者表需引用公司ID做外键，故先键此表

CREATE TABLE Recruiter.Companies (

CompanyID INT PRIMARY KEY IDENTITY(1,1),--主键，自增

CompanyName NVARCHAR(100) NOT NULL,

CompanyDescription NVARCHAR(MAX),--公司描述，容量需大

BusinessLicense NVARCHAR(200)--营业执照

);

GO

-- 创建招聘者表（Recruiters）

CREATE TABLE Recruiter.Recruiters (

RecruiterID INT PRIMARY KEY IDENTITY(1,1),--主键，自增

FullName NVARCHAR(100) NOT NULL,

Phone VARCHAR(20) NOT NULL,

Email NVARCHAR(100) NOT NULL,

CompanyID INT NOT NULL,

CONSTRAINT FK\_Recruiters\_Companies

FOREIGN KEY (CompanyID)

REFERENCES Recruiter.Companies(CompanyID)--引入公司表中的公司ID做外键，确保每个招聘者都属于一个有效的公司

);

GO

-- 创建招聘需求表（JobRequirements）

CREATE TABLE Recruiter.JobRequirements (

JobID INT PRIMARY KEY IDENTITY(1,1),--主键，自增

JobTitle NVARCHAR(100) NOT NULL,

JobDescription NVARCHAR(MAX),--工作描述，容量大

ContactPerson NVARCHAR(100),--负责人姓名

ContactEmail NVARCHAR(100),--负责人邮箱

CompanyID INT NOT NULL,

CONSTRAINT FK\_JobRequirements\_Companies

FOREIGN KEY (CompanyID)

REFERENCES Recruiter.Companies(CompanyID)--引入公司表中的公司ID做外键，确保每个招聘需求都属于一个有效的公司

);

GO

-- 创建候选人表（Candidates）

CREATE TABLE Recruiter.Candidates (

CandidateID INT PRIMARY KEY IDENTITY(1,1),--主键，自增

FullName NVARCHAR(100) NOT NULL,

Email NVARCHAR(100) NOT NULL,

Phone VARCHAR(20) NOT NULL,

TargetPosition NVARCHAR(100),

RecruiterID INT NOT NULL,

CONSTRAINT FK\_Candidates\_Recruiters

FOREIGN KEY (RecruiterID)

REFERENCES Recruiter.Recruiters(RecruiterID)--引入应聘者ID做外键，确保每个候选人都对应一个有效的招聘者

);

GO

-- 创建招聘者与招聘需求的多对多关联表，实现多对多关系，一个招聘者可以管理多个招聘者需求

CREATE TABLE Recruiter.RecruiterJobRelations (

RelationID INT PRIMARY KEY IDENTITY(1,1),

RecruiterID INT NOT NULL,

JobID INT NOT NULL,

CONSTRAINT FK\_RecruiterJobRelations\_Recruiters

FOREIGN KEY (RecruiterID)

REFERENCES Recruiter.Recruiters(RecruiterID),

CONSTRAINT FK\_RecruiterJobRelations\_JobRequirements

FOREIGN KEY (JobID)

REFERENCES Recruiter.JobRequirements(JobID),--实现多对多关系，一个招聘者可以管理多个招聘者需求

-- 添加唯一约束，确保一个招聘者不会重复关联到同一个职位

CONSTRAINT UQ\_Recruiter\_Job

UNIQUE (RecruiterID, JobID)

);

GO

USE GigEconomy;

GO

-- 创建工作引擎流程架构

CREATE SCHEMA WorkflowEngine;

GO

-- 创建招聘者表（Recruiters）

CREATE TABLE WorkflowEngine.Recruiters (

RecruiterID INT PRIMARY KEY IDENTITY(1,1),--主键，自增

Name NVARCHAR(100) NOT NULL,

Phone VARCHAR(20) NOT NULL,

Email NVARCHAR(100) NOT NULL

);

GO

-- 创建应聘者表（Candidates）

CREATE TABLE WorkflowEngine.Candidates (

CandidateID INT PRIMARY KEY IDENTITY(1,1),--主键，自增

Name NVARCHAR(100) NOT NULL,

Phone VARCHAR(20) NOT NULL,

Email NVARCHAR(100) NOT NULL

);

GO

-- 创建项目表（Projects）

CREATE TABLE WorkflowEngine.Projects (

ProjectID INT PRIMARY KEY IDENTITY(1,1),--主键，自增

Name NVARCHAR(100) NOT NULL,

Budget DECIMAL(18, 2) NOT NULL,--预算对精度要求严格故用decimal数据类型，总位数18，小数位数2

Description NVARCHAR(MAX),--项目描述，容量要求高

Status NVARCHAR(50) NOT NULL,--项目状态

RecruiterID INT NOT NULL,--引入招聘者ID做外键，确保每个项目都有招聘者负责

CONSTRAINT FK\_Projects\_Recruiters

FOREIGN KEY (RecruiterID)

REFERENCES WorkflowEngine.Recruiters(RecruiterID)

);

GO

-- 创建项目和应聘者的多对多关联表

CREATE TABLE WorkflowEngine.ProjectCandidates (--创建关联表，储存项目和应聘者的多对多关系

ProjectID INT NOT NULL,

CandidateID INT NOT NULL,

CONSTRAINT PK\_ProjectCandidates PRIMARY KEY (ProjectID, CandidateID),--复合主键，确保项目和应聘者组合的唯一性

CONSTRAINT FK\_ProjectCandidates\_Projects

FOREIGN KEY (ProjectID)

REFERENCES WorkflowEngine.Projects(ProjectID),

CONSTRAINT FK\_ProjectCandidates\_Candidates

FOREIGN KEY (CandidateID)

REFERENCES WorkflowEngine.Candidates(CandidateID)--两个外键，确保数据完整性

);

GO

-- 创建里程碑表（Milestones）

CREATE TABLE WorkflowEngine.Milestones (

MilestoneID INT PRIMARY KEY IDENTITY(1,1),--主键，自增

Name NVARCHAR(100) NOT NULL,

Date DATETIME NOT NULL,--时间

Status NVARCHAR(50) NOT NULL,--状态

ProjectID INT NOT NULL,

CONSTRAINT FK\_Milestones\_Projects --引入项目ID做外键，确保每个里程碑都对应有效的项目

FOREIGN KEY (ProjectID)

REFERENCES WorkflowEngine.Projects(ProjectID)

);

GO

-- 创建交付物表（Deliverables）

CREATE TABLE WorkflowEngine.Deliverables (

DeliverableID INT PRIMARY KEY IDENTITY(1,1),--主键，自增

Name NVARCHAR(100) NOT NULL,

Date DATETIME NOT NULL,--时间

Status NVARCHAR(50) NOT NULL,

PaymentAmount DECIMAL(18, 2) NOT NULL,--交付金额

MilestoneID INT NOT NULL UNIQUE, -- UNIQUE约束确保一对一关系，每个里程碑对应一个交付物

CandidateID INT NOT NULL,--每个交付物必须有一个对应的应聘者

CONSTRAINT FK\_Deliverables\_Milestones

FOREIGN KEY (MilestoneID)

REFERENCES WorkflowEngine.Milestones(MilestoneID),--确保引用的里程碑存在

CONSTRAINT FK\_Deliverables\_Candidates

FOREIGN KEY (CandidateID)

REFERENCES WorkflowEngine.Candidates(CandidateID)--确保引用的应聘者存在

);

GO

USE GigEconomy;

GO

-- 创建支付系统与担保架构

CREATE SCHEMA PaymentSystem;

GO

-- 创建招聘者表（Recruiters）

CREATE TABLE PaymentSystem.Recruiters (

RecruiterID INT PRIMARY KEY IDENTITY(1,1),

Name NVARCHAR(100) NOT NULL,

Phone VARCHAR(20) NOT NULL,

Email NVARCHAR(100) NOT NULL

);

GO

-- 创建应聘者表（Candidates）

CREATE TABLE PaymentSystem.Candidates (

CandidateID INT PRIMARY KEY IDENTITY(1,1),

Name NVARCHAR(100) NOT NULL,

Phone VARCHAR(20) NOT NULL,

Email NVARCHAR(100) NOT NULL,

BankAccountNumber VARCHAR(50) NOT NULL--银行卡号

);

GO

-- 创建资产账户表（AssetAccounts）

CREATE TABLE PaymentSystem.AssetAccounts (

AccountID INT PRIMARY KEY IDENTITY(1,1),

Balance DECIMAL(18, 2) NOT NULL DEFAULT 0.00,--账户余额

SubjectType NVARCHAR(20) NOT NULL, -- 账户类型，应聘者或招聘者

SubjectID INT NOT NULL,

CONSTRAINT CK\_AssetAccounts\_SubjectType

CHECK (SubjectType IN ('Recruiter', 'Candidate'))--确认账户类型

);

GO

-- 创建托管订单表（EscrowOrders）

CREATE TABLE PaymentSystem.EscrowOrders (

EscrowOrderID INT PRIMARY KEY IDENTITY(1,1),

Amount DECIMAL(18, 2) NOT NULL,--托管金额

Status NVARCHAR(50) NOT NULL,--状态

CreationTime DATETIME NOT NULL DEFAULT GETDATE(),--创建时间

ExpirationTime DATETIME NOT NULL,--到期时间

RecruiterID INT NOT NULL, --引入招聘者ID做外键

CONSTRAINT FK\_EscrowOrders\_Recruiters

FOREIGN KEY (RecruiterID)

REFERENCES PaymentSystem.Recruiters(RecruiterID)

);

GO

-- 创建支付订单表（PaymentOrders）

CREATE TABLE PaymentSystem.PaymentOrders (

PaymentOrderID INT PRIMARY KEY IDENTITY(1,1),

PaymentMethod NVARCHAR(50) NOT NULL,--支付方式

PaymentAmount DECIMAL(18, 2) NOT NULL,--支付金额

PaymentTime DATETIME NOT NULL DEFAULT GETDATE(),--支付时间

PaymentStatus NVARCHAR(50) NOT NULL,--支付状态

EscrowOrderID INT NOT NULL UNIQUE, -- UNIQUE确保一对一关系

CONSTRAINT FK\_PaymentOrders\_EscrowOrders

FOREIGN KEY (EscrowOrderID)

REFERENCES PaymentSystem.EscrowOrders(EscrowOrderID)--引入托管订单表的托管订单ID做外键

);

GO

-- 创建结算单表（SettlementOrders）

CREATE TABLE PaymentSystem.SettlementOrders (

SettlementOrderID INT PRIMARY KEY IDENTITY(1,1),

SettlementAmount DECIMAL(18, 2) NOT NULL,--结算金额

SettlementTime DATETIME NOT NULL DEFAULT GETDATE(),--结算时间

SettlementStatus NVARCHAR(50) NOT NULL,--结算状态

SettlementType NVARCHAR(50) NOT NULL,--结算类型

PaymentOrderID INT NOT NULL,--引入支付订单表的支付订单ID做外键

CandidateID INT NOT NULL,--引入应聘者表的应聘者ID做外键

CONSTRAINT FK\_SettlementOrders\_PaymentOrders --引入支付订单表的支付订单ID做外键

FOREIGN KEY (PaymentOrderID)

REFERENCES PaymentSystem.PaymentOrders(PaymentOrderID),

CONSTRAINT FK\_SettlementOrders\_Candidates --引入应聘者表的应聘者ID做外键

FOREIGN KEY (CandidateID)

REFERENCES PaymentSystem.Candidates(CandidateID)

);

GO

-- 添加资产账户与招聘者的唯一约束（确保一对一关系）

CREATE UNIQUE NONCLUSTERED INDEX IX\_AssetAccounts\_Recruiter

ON PaymentSystem.AssetAccounts (SubjectID)

WHERE SubjectType = 'Recruiter';--确认账户类型为招聘者

GO

-- 添加资产账户与应聘者的唯一约束（确保一对一关系）

CREATE UNIQUE NONCLUSTERED INDEX IX\_AssetAccounts\_Candidate

ON PaymentSystem.AssetAccounts (SubjectID)

WHERE SubjectType = 'Candidate';--确认账户类型为应聘者

GO

-- 创建支付流程存储过程，包含事务和回滚机制

CREATE PROCEDURE PaymentSystem.ProcessPaymentSettlement--处理支付结算

@EscrowOrderID INT,--托管订单ID

@CandidateID INT,--应聘者ID

@SettlementType NVARCHAR(50)--结算类型

AS

BEGIN

SET NOCOUNT ON;

SET XACT\_ABORT ON; -- 启用自动回滚

BEGIN TRANSACTION;

BEGIN TRY

-- 1. 获取托管订单信息

DECLARE @Amount DECIMAL(18, 2);--托管金额

DECLARE @RecruiterID INT;--招聘者ID

SELECT @Amount = Amount, @RecruiterID = RecruiterID--获取托管金额和招聘者ID

FROM PaymentSystem.EscrowOrders

WHERE EscrowOrderID = @EscrowOrderID;

-- 2. 获取招聘者和应聘者的账户信息

DECLARE @RecruiterAccountID INT;--招聘者账户ID

DECLARE @CandidateAccountID INT;--应聘者账户ID

SELECT @RecruiterAccountID = AccountID--获取招聘者账户ID

FROM PaymentSystem.AssetAccounts

WHERE SubjectType = 'Recruiter' AND SubjectID = @RecruiterID;

SELECT @CandidateAccountID = AccountID--获取应聘者账户ID

FROM PaymentSystem.AssetAccounts

WHERE SubjectType = 'Candidate' AND SubjectID = @CandidateID;

-- 3. 更新招聘者账户余额（减少）

UPDATE PaymentSystem.AssetAccounts--更新招聘者账户余额

SET Balance = Balance - @Amount

WHERE AccountID = @RecruiterAccountID;

-- 4. 更新应聘者账户余额（增加）

UPDATE PaymentSystem.AssetAccounts--更新应聘者账户余额

SET Balance = Balance + @Amount

WHERE AccountID = @CandidateAccountID;

-- 5. 创建结算单

INSERT INTO PaymentSystem.SettlementOrders--创建结算单

(SettlementAmount, SettlementStatus, SettlementType, PaymentOrderID, CandidateID)

VALUES--插入结算单信息

(@Amount, 'Completed', @SettlementType,

(SELECT PaymentOrderID FROM PaymentSystem.PaymentOrders WHERE EscrowOrderID = @EscrowOrderID),

@CandidateID);

-- 6. 更新托管订单状态

UPDATE PaymentSystem.EscrowOrders--更新托管订单状态

SET Status = 'Settled'

WHERE EscrowOrderID = @EscrowOrderID;

-- 7. 更新支付订单状态

UPDATE PaymentSystem.PaymentOrders--更新支付订单状态

SET PaymentStatus = 'Settled'

WHERE EscrowOrderID = @EscrowOrderID;

COMMIT TRANSACTION;

PRINT '支付结算流程成功完成！';

END TRY

BEGIN CATCH

ROLLBACK TRANSACTION;

PRINT '错误：支付结算流程失败，已回滚所有操作。';

PRINT '错误信息：' + ERROR\_MESSAGE();

END CATCH;

END;

GO

-- 创建托管订单创建存储过程

CREATE PROCEDURE PaymentSystem.CreateEscrowOrder--创建托管订单

@RecruiterID INT,--招聘者ID

@Amount DECIMAL(18, 2),--托管金额

@ExpirationTime DATETIME--托管过期时间

AS

BEGIN--创建托管订单

SET NOCOUNT ON;

SET XACT\_ABORT ON;

BEGIN TRANSACTION;

BEGIN TRY

-- 1. 创建托管订单

INSERT INTO PaymentSystem.EscrowOrders--创建托管订单

(Amount, Status, CreationTime, ExpirationTime, RecruiterID)

VALUES--插入托管订单信息

(@Amount, 'Created', GETDATE(), @ExpirationTime, @RecruiterID);

-- 2. 获取新创建的托管订单ID

DECLARE @EscrowOrderID INT;

SET @EscrowOrderID = SCOPE\_IDENTITY();--获取当前会话中最后插入的托管订单ID

-- 3. 创建对应的支付订单

INSERT INTO PaymentSystem.PaymentOrders--创建支付订单

(PaymentMethod, PaymentAmount, PaymentTime, PaymentStatus, EscrowOrderID)

VALUES

('System', @Amount, GETDATE(), 'Pending', @EscrowOrderID);

COMMIT TRANSACTION;

PRINT '托管订单和支付订单创建成功！';

RETURN @EscrowOrderID;

END TRY

BEGIN CATCH

ROLLBACK TRANSACTION;

PRINT '错误：托管订单创建失败，已回滚所有操作。';

PRINT '错误信息：' + ERROR\_MESSAGE();

RETURN -1;

END CATCH;

END;

GO

USE GigEconomy;

GO

-- 首先创建所有必要的架构

IF NOT EXISTS (SELECT 1 FROM sys.schemas WHERE name = 'Core')

BEGIN

EXEC ('CREATE SCHEMA Core');

PRINT '已创建Core架构';

END

GO

IF NOT EXISTS (SELECT 1 FROM sys.schemas WHERE name = 'Applicant')

BEGIN

EXEC ('CREATE SCHEMA Applicant');

PRINT '已创建Applicant架构';

END

GO

IF NOT EXISTS (SELECT 1 FROM sys.schemas WHERE name = 'Recruiter')

BEGIN

EXEC ('CREATE SCHEMA Recruiter');

PRINT '已创建Recruiter架构';

END

GO

IF NOT EXISTS (SELECT 1 FROM sys.schemas WHERE name = 'PaymentSystem')

BEGIN

EXEC ('CREATE SCHEMA PaymentSystem');

PRINT '已创建PaymentSystem架构';

END

GO

CREATE TABLE Core.Users (

UserID INT PRIMARY KEY IDENTITY(1,1),

UserRole NVARCHAR(20) NOT NULL CHECK (UserRole IN ('Candidate', 'Recruiter')),

FullName NVARCHAR(100) NOT NULL CHECK (LEN(FullName) > 0),

Phone VARCHAR(20) NOT NULL CHECK (Phone NOT LIKE '%[^0-9]%' OR Phone = ''),

Email NVARCHAR(100) NOT NULL UNIQUE CHECK (Email LIKE '%\_@\_\_%.\_\_%'),

BankAccountNumber VARCHAR(50) NULL,

CompanyID INT NULL,

CreatedAt DATETIME NOT NULL DEFAULT GETDATE(),

UpdatedAt DATETIME NOT NULL DEFAULT GETDATE()

);

GO

-- 创建统一公司表（添加检查约束）

CREATE TABLE Core.Companies (

CompanyID INT PRIMARY KEY IDENTITY(1,1),

CompanyName NVARCHAR(100) NOT NULL UNIQUE CHECK (LEN(CompanyName) > 0),

CompanyDescription NVARCHAR(MAX),

BusinessLicense NVARCHAR(200) UNIQUE,

CreatedAt DATETIME NOT NULL DEFAULT GETDATE(),

UpdatedAt DATETIME NOT NULL DEFAULT GETDATE(),

CHECK (BusinessLicense IS NOT NULL OR CompanyName NOT LIKE '%有限公司%')

);

GO

-- 添加外键关系，允许用户无公司关联

IF NOT EXISTS (SELECT 1 FROM sys.foreign\_keys WHERE name = 'FK\_Users\_Companies')

BEGIN

ALTER TABLE Core.Users

ADD CONSTRAINT FK\_Users\_Companies

FOREIGN KEY (CompanyID)

REFERENCES Core.Companies(CompanyID) ON DELETE SET NULL;

PRINT '已添加FK\_Users\_Companies外键约束';

END

ELSE

PRINT 'FK\_Users\_Companies外键约束已存在';

GO

-- 4. 优化后的应聘者模块表

CREATE TABLE Applicant.Applications\_Optimized (

ApplicationID INT PRIMARY KEY IDENTITY(1,1),

UserID INT NOT NULL,

JobID INT NOT NULL,

Status NVARCHAR(20) NOT NULL DEFAULT 'Applied',

ApplicationTime DATETIME NOT NULL DEFAULT GETDATE(),

CoverLetter NVARCHAR(MAX),

UpdatedAt DATETIME NOT NULL DEFAULT GETDATE(),

CONSTRAINT UQ\_Applications\_User\_Job UNIQUE (UserID, JobID)

);

GO

-- 添加CHECK约束

ALTER TABLE Applicant.Applications\_Optimized

ADD CONSTRAINT CK\_Applications\_Status

CHECK (Status IN ('Applied', 'Reviewed', 'Interviewed', 'Rejected', 'Accepted'));

GO

-- 添加用户外键约束

IF NOT EXISTS (SELECT 1 FROM sys.foreign\_keys WHERE name = 'FK\_Applications\_Users')

BEGIN

ALTER TABLE Applicant.Applications\_Optimized

ADD CONSTRAINT FK\_Applications\_Users

FOREIGN KEY (UserID)

REFERENCES Core.Users(UserID) ON DELETE CASCADE;

PRINT '已添加FK\_Applications\_Users外键约束';

END

ELSE

PRINT 'FK\_Applications\_Users外键约束已存在';

GO

-- 添加职位外键约束（确保JobRequirements\_Optimized表已存在）

IF OBJECT\_ID('Recruiter.JobRequirements\_Optimized', 'U') IS NOT NULL AND NOT EXISTS (SELECT 1 FROM sys.foreign\_keys WHERE name = 'FK\_Applications\_JobRequirements')

BEGIN

ALTER TABLE Applicant.Applications\_Optimized

ADD CONSTRAINT FK\_Applications\_JobRequirements

FOREIGN KEY (JobID)

REFERENCES Recruiter.JobRequirements\_Optimized(JobID) ON DELETE CASCADE;

PRINT '已添加FK\_Applications\_JobRequirements外键约束';

END

ELSE IF OBJECT\_ID('Recruiter.JobRequirements\_Optimized', 'U') IS NULL

PRINT '警告: Recruiter.JobRequirements\_Optimized表不存在，跳过FK\_Applications\_JobRequirements外键约束';

ELSE

PRINT 'FK\_Applications\_JobRequirements外键约束已存在';

GO

-- 5. 优化后的招聘者模块表

CREATE TABLE Recruiter.JobRequirements\_Optimized (

JobID INT PRIMARY KEY IDENTITY(1,1),

CompanyID INT NOT NULL,

JobTitle NVARCHAR(100) NOT NULL,

JobDescription NVARCHAR(MAX),

JobType NVARCHAR(50) NOT NULL,

SalaryRange NVARCHAR(50),

Location NVARCHAR(100),

ContactPerson NVARCHAR(100) NOT NULL,

ContactEmail NVARCHAR(100) NOT NULL,

IsActive BIT NOT NULL DEFAULT 1,

CreatedAt DATETIME NOT NULL DEFAULT GETDATE(),

UpdatedAt DATETIME NOT NULL DEFAULT GETDATE()

);

GO

-- 添加CHECK约束

ALTER TABLE Recruiter.JobRequirements\_Optimized

ADD CONSTRAINT CK\_JobRequirements\_JobTitle

CHECK (LEN(JobTitle) > 0);

GO

ALTER TABLE Recruiter.JobRequirements\_Optimized

ADD CONSTRAINT CK\_JobRequirements\_JobType

CHECK (JobType IN ('Full-time', 'Part-time', 'Contract', 'Temporary'));

GO

ALTER TABLE Recruiter.JobRequirements\_Optimized

ADD CONSTRAINT CK\_JobRequirements\_ContactEmail

CHECK (ContactEmail LIKE '%\_@\_\_%.\_\_%');

GO

-- 添加公司外键约束

IF NOT EXISTS (SELECT 1 FROM sys.foreign\_keys WHERE name = 'FK\_JobRequirements\_Companies')

BEGIN

ALTER TABLE Recruiter.JobRequirements\_Optimized

ADD CONSTRAINT FK\_JobRequirements\_Companies

FOREIGN KEY (CompanyID)

REFERENCES Core.Companies(CompanyID) ON DELETE CASCADE;

PRINT '已添加FK\_JobRequirements\_Companies外键约束';

END

ELSE

PRINT 'FK\_JobRequirements\_Companies外键约束已存在';

GO

-- 添加外键约束，检查是否已存在

IF NOT EXISTS (SELECT 1 FROM sys.foreign\_keys WHERE name = 'FK\_JobRequirements\_Companies')

BEGIN

ALTER TABLE Recruiter.JobRequirements\_Optimized

ADD CONSTRAINT FK\_JobRequirements\_Companies

FOREIGN KEY (CompanyID)

REFERENCES Core.Companies(CompanyID) ON DELETE CASCADE;

PRINT '已添加FK\_JobRequirements\_Companies外键约束';

END

ELSE

PRINT 'FK\_JobRequirements\_Companies外键约束已存在';

GO

-- 已在前面的代码块中添加此外键约束，避免重复

PRINT '外键约束FK\_Applications\_JobRequirements将通过前面的代码块添加';

GO

-- 6. 优化后的支付系统模块表

CREATE TABLE PaymentSystem.AssetAccounts\_Optimized (

AccountID INT PRIMARY KEY IDENTITY(1,1),

UserID INT NOT NULL,

Balance DECIMAL(18, 2) NOT NULL DEFAULT 0.00,

AccountType NVARCHAR(20) NOT NULL DEFAULT 'Standard',

IsFrozen BIT NOT NULL DEFAULT 0,

CreatedAt DATETIME NOT NULL DEFAULT GETDATE(),

UpdatedAt DATETIME NOT NULL DEFAULT GETDATE(),

CONSTRAINT UQ\_AssetAccounts\_User UNIQUE (UserID)

);

GO

-- 添加CHECK约束

ALTER TABLE PaymentSystem.AssetAccounts\_Optimized

ADD CONSTRAINT CK\_AssetAccounts\_Balance

CHECK (Balance >= 0);

GO

ALTER TABLE PaymentSystem.AssetAccounts\_Optimized

ADD CONSTRAINT CK\_AssetAccounts\_AccountType

CHECK (AccountType IN ('Standard', 'Premium', 'Escrow'));

GO

-- 添加用户外键约束

IF NOT EXISTS (SELECT 1 FROM sys.foreign\_keys WHERE name = 'FK\_AssetAccounts\_Users')

BEGIN

ALTER TABLE PaymentSystem.AssetAccounts\_Optimized

ADD CONSTRAINT FK\_AssetAccounts\_Users

FOREIGN KEY (UserID)

REFERENCES Core.Users(UserID) ON DELETE CASCADE;

PRINT '已添加FK\_AssetAccounts\_Users外键约束';

END

ELSE

PRINT 'FK\_AssetAccounts\_Users外键约束已存在';

GO

-- 7. 索引优化策略

-- 核心用户表索引

IF OBJECT\_ID('Core.Users', 'U') IS NOT NULL

BEGIN

IF NOT EXISTS (SELECT 1 FROM sys.indexes WHERE name = 'IX\_Users\_UserRole' AND object\_id = OBJECT\_ID('Core.Users'))

CREATE INDEX IX\_Users\_UserRole ON Core.Users(UserRole);

IF NOT EXISTS (SELECT 1 FROM sys.indexes WHERE name = 'IX\_Users\_Email' AND object\_id = OBJECT\_ID('Core.Users'))

CREATE INDEX IX\_Users\_Email ON Core.Users(Email);

IF NOT EXISTS (SELECT 1 FROM sys.indexes WHERE name = 'IX\_Users\_CompanyID' AND object\_id = OBJECT\_ID('Core.Users'))

CREATE INDEX IX\_Users\_CompanyID ON Core.Users(CompanyID);

IF NOT EXISTS (SELECT 1 FROM sys.indexes WHERE name = 'IX\_Users\_UserRole\_CompanyID' AND object\_id = OBJECT\_ID('Core.Users'))

CREATE INDEX IX\_Users\_UserRole\_CompanyID ON Core.Users(UserRole, CompanyID);

PRINT '核心用户表索引创建完成';

END

ELSE

PRINT '警告: Core.Users表不存在，跳过索引创建';

GO

-- 应聘者模块索引

IF OBJECT\_ID('Applicant.Applications\_Optimized', 'U') IS NOT NULL

BEGIN

IF NOT EXISTS (SELECT 1 FROM sys.indexes WHERE name = 'IX\_Applications\_UserID' AND object\_id = OBJECT\_ID('Applicant.Applications\_Optimized'))

CREATE INDEX IX\_Applications\_UserID ON Applicant.Applications\_Optimized(UserID);

IF NOT EXISTS (SELECT 1 FROM sys.indexes WHERE name = 'IX\_Applications\_JobID' AND object\_id = OBJECT\_ID('Applicant.Applications\_Optimized'))

CREATE INDEX IX\_Applications\_JobID ON Applicant.Applications\_Optimized(JobID);

IF NOT EXISTS (SELECT 1 FROM sys.indexes WHERE name = 'IX\_Applications\_Status' AND object\_id = OBJECT\_ID('Applicant.Applications\_Optimized'))

CREATE INDEX IX\_Applications\_Status ON Applicant.Applications\_Optimized(Status);

IF NOT EXISTS (SELECT 1 FROM sys.indexes WHERE name = 'IX\_Applications\_ApplicationTime' AND object\_id = OBJECT\_ID('Applicant.Applications\_Optimized'))

CREATE INDEX IX\_Applications\_ApplicationTime ON Applicant.Applications\_Optimized(ApplicationTime);

PRINT '应聘者模块索引创建完成';

END

ELSE

PRINT '警告: Applicant.Applications\_Optimized表不存在，跳过索引创建';

GO

-- 招聘者模块索引

IF OBJECT\_ID('Recruiter.JobRequirements\_Optimized', 'U') IS NOT NULL

BEGIN

IF NOT EXISTS (SELECT 1 FROM sys.indexes WHERE name = 'IX\_JobRequirements\_CompanyID' AND object\_id = OBJECT\_ID('Recruiter.JobRequirements\_Optimized'))

CREATE INDEX IX\_JobRequirements\_CompanyID ON Recruiter.JobRequirements\_Optimized(CompanyID);

IF NOT EXISTS (SELECT 1 FROM sys.indexes WHERE name = 'IX\_JobRequirements\_IsActive' AND object\_id = OBJECT\_ID('Recruiter.JobRequirements\_Optimized'))

CREATE INDEX IX\_JobRequirements\_IsActive ON Recruiter.JobRequirements\_Optimized(IsActive);

IF NOT EXISTS (SELECT 1 FROM sys.indexes WHERE name = 'IX\_JobRequirements\_JobType' AND object\_id = OBJECT\_ID('Recruiter.JobRequirements\_Optimized'))

CREATE INDEX IX\_JobRequirements\_JobType ON Recruiter.JobRequirements\_Optimized(JobType);

IF NOT EXISTS (SELECT 1 FROM sys.indexes WHERE name = 'IX\_JobRequirements\_Location' AND object\_id = OBJECT\_ID('Recruiter.JobRequirements\_Optimized'))

CREATE INDEX IX\_JobRequirements\_Location ON Recruiter.JobRequirements\_Optimized(Location);

PRINT '招聘者模块索引创建完成';

END

ELSE

PRINT '警告: Recruiter.JobRequirements\_Optimized表不存在，跳过索引创建';

GO

-- 支付系统模块索引

IF OBJECT\_ID('PaymentSystem.AssetAccounts\_Optimized', 'U') IS NOT NULL

BEGIN

IF NOT EXISTS (SELECT 1 FROM sys.indexes WHERE name = 'IX\_AssetAccounts\_UserID' AND object\_id = OBJECT\_ID('PaymentSystem.AssetAccounts\_Optimized'))

CREATE INDEX IX\_AssetAccounts\_UserID ON PaymentSystem.AssetAccounts\_Optimized(UserID);

IF NOT EXISTS (SELECT 1 FROM sys.indexes WHERE name = 'IX\_AssetAccounts\_Balance' AND object\_id = OBJECT\_ID('PaymentSystem.AssetAccounts\_Optimized'))

CREATE INDEX IX\_AssetAccounts\_Balance ON PaymentSystem.AssetAccounts\_Optimized(Balance);

IF NOT EXISTS (SELECT 1 FROM sys.indexes WHERE name = 'IX\_AssetAccounts\_AccountType' AND object\_id = OBJECT\_ID('PaymentSystem.AssetAccounts\_Optimized'))

CREATE INDEX IX\_AssetAccounts\_AccountType ON PaymentSystem.AssetAccounts\_Optimized(AccountType);

IF NOT EXISTS (SELECT 1 FROM sys.indexes WHERE name = 'IX\_AssetAccounts\_IsFrozen' AND object\_id = OBJECT\_ID('PaymentSystem.AssetAccounts\_Optimized'))

CREATE INDEX IX\_AssetAccounts\_IsFrozen ON PaymentSystem.AssetAccounts\_Optimized(IsFrozen);

PRINT '支付系统模块索引创建完成';

END

ELSE

PRINT '警告: PaymentSystem.AssetAccounts\_Optimized表不存在，跳过索引创建';

GO

PRINT '规范化优化脚本执行完成';

GO

USE GigEconomy;

GO

--确保Core架构存在

IF NOT EXISTS (SELECT 1 FROM sys.schemas WHERE name = 'Core')

BEGIN

EXEC ('CREATE SCHEMA Core');

PRINT '已创建Core架构';

END

ELSE

PRINT 'Core架构已存在';

GO

--创建数据版本记录表

CREATE TABLE Core.DataVersions (

VersionID INT IDENTITY(1,1) PRIMARY KEY,

TableName NVARCHAR(100) NOT NULL,

RecordID INT NOT NULL,

VersionNumber INT NOT NULL,

ChangeTime DATETIME NOT NULL DEFAULT GETDATE(),

ChangedBy NVARCHAR(100) NULL,

ChangeType NVARCHAR(20) NOT NULL, -- Insert, Update, Delete

OldData XML NULL,

NewData XML NULL,

-- 将复合键改为唯一索引而不是重复主键

CONSTRAINT UQ\_DataVersions\_Table\_Record\_Version

UNIQUE NONCLUSTERED (TableName, RecordID, VersionNumber)

);

GO

--创建错误日志表

CREATE TABLE Core.ErrorLogs (

ErrorID INT IDENTITY(1,1) PRIMARY KEY,

ErrorTime DATETIME NOT NULL DEFAULT GETDATE(),

ErrorMessage NVARCHAR(MAX) NOT NULL,

ErrorSeverity INT NOT NULL,

ErrorState INT NOT NULL,

ProcedureName NVARCHAR(100) NULL,

LineNumber INT NULL,

IsResolved BIT NOT NULL DEFAULT 0

);

GO

--创建事务回滚记录表

CREATE TABLE Core.TransactionRollbacks (

RollbackID INT IDENTITY(1,1) PRIMARY KEY,

RollbackTime DATETIME NOT NULL DEFAULT GETDATE(),

TransactionID NVARCHAR(50) NOT NULL,

ProcedureName NVARCHAR(100) NOT NULL,

ErrorMessage NVARCHAR(MAX) NOT NULL,

RollbackDetails NVARCHAR(MAX) NULL

);

GO

--创建数据恢复存储过程

CREATE PROCEDURE Core.RecoverDataVersion

@TableName NVARCHAR(100),

@RecordID INT,

@VersionNumber INT

AS

BEGIN

SET NOCOUNT ON;

BEGIN TRY

-- 检查版本记录是否存在

IF NOT EXISTS (

SELECT 1

FROM Core.DataVersions

WHERE TableName = @TableName

AND RecordID = @RecordID

AND VersionNumber = @VersionNumber

)

BEGIN

SELECT 'ERROR' AS Status, '找不到指定的版本记录' AS Message;

RETURN;

END

-- 这里仅返回恢复信息，实际恢复操作需要根据具体表结构实现

SELECT

'SUCCESS' AS Status,

'数据恢复准备完成' AS Message,

@TableName AS TableName,

@RecordID AS RecordID,

@VersionNumber AS VersionNumber;

END TRY

BEGIN CATCH

SELECT

'ERROR' AS Status,

'恢复失败: ' + ERROR\_MESSAGE() AS Message;

END CATCH;

END;

GO

--数据恢复测试

CREATE PROCEDURE Core.TestDataRecovery

AS

BEGIN

SET NOCOUNT ON;

PRINT '测试数据恢复机制';

-- 获取最新的数据版本记录

DECLARE @TableName NVARCHAR(100) = 'PaymentSystem.AssetAccounts';

DECLARE @RecordID INT = 1; -- 示例ID

DECLARE @Version INT = 1; -- 示例版本

PRINT '测试数据恢复：' + @TableName + ' 记录ID: ' + CAST(@RecordID AS VARCHAR) + ' 版本: ' + CAST(@Version AS VARCHAR);

-- 执行数据恢复测试

EXEC Core.RecoverDataVersion @TableName, @RecordID, @Version;

END;

GO

USE GigEconomy;

GO

-- 事务隔离级别基本介绍

-- READ COMMITTED: 防止脏读，允许不可重复读和幻读

-- REPEATABLE READ: 防止脏读和不可重复读，允许幻读

-- SERIALIZABLE: 完全隔离，性能最低但一致性最好

-- 创建Basic架构（如果不存在）

IF NOT EXISTS (SELECT 1 FROM sys.schemas WHERE name = 'Basic')

BEGIN

EXEC ('CREATE SCHEMA Basic');

PRINT '已创建Basic架构';

END

ELSE

PRINT 'Basic架构已存在';

GO

-- 实体访问记录表（用于并发控制）

CREATE TABLE Basic.EntityAccess (

AccessID INT IDENTITY(1,1) PRIMARY KEY,

EntityID INT NOT NULL,

UserID INT NOT NULL,

AccessTime DATETIME NOT NULL,

UNIQUE (EntityID, UserID) -- 防止重复访问

);

GO

-- 为表添加初始约束

ALTER TABLE Basic.EntityAccess

ADD CONSTRAINT DF\_EntityAccess\_AccessTime DEFAULT GETDATE() FOR AccessTime;

GO

-- 实体表（用于并发更新）

CREATE TABLE Basic.Entities (

EntityID INT PRIMARY KEY,

Value NVARCHAR(100) NOT NULL,

LastUpdated DATETIME NOT NULL

);

GO

-- 为表添加初始约束

ALTER TABLE Basic.Entities

ADD CONSTRAINT DF\_Entities\_LastUpdated DEFAULT GETDATE() FOR LastUpdated;

GO

CREATE PROCEDURE Basic.ConcurrentOperation-- 基础并发操作存储过程

@EntityID INT,

@UserID INT

AS

BEGIN

SET NOCOUNT ON;

-- 设置事务隔离级别

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

BEGIN TRANSACTION;

BEGIN TRY

-- 检查记录是否已存在（防重复）

IF EXISTS (

SELECT 1

FROM Basic.EntityAccess

WITH (UPDLOCK, HOLDLOCK) -- 加锁直到事务结束

WHERE EntityID = @EntityID AND UserID = @UserID

)

BEGIN

ROLLBACK TRANSACTION;

SELECT 'ERROR' AS Status, '该操作已执行过' AS Message;

RETURN;

END

-- 执行操作

INSERT INTO Basic.EntityAccess (EntityID, UserID, AccessTime)

VALUES (@EntityID, @UserID, GETDATE());

COMMIT TRANSACTION;

SELECT 'SUCCESS' AS Status, '操作执行成功' AS Message;

END TRY

BEGIN CATCH

-- 错误处理

IF @@TRANCOUNT > 0

ROLLBACK TRANSACTION;

SELECT 'ERROR' AS Status, '操作失败: ' + ERROR\_MESSAGE() AS Message;

END CATCH;

END;

GO

-- 基础并发更新存储过程

CREATE PROCEDURE Basic.ConcurrentUpdate

@EntityID INT,

@NewValue NVARCHAR(100)

AS

BEGIN

SET NOCOUNT ON;

-- 设置事务隔离级别

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

BEGIN TRANSACTION;

BEGIN TRY

-- 使用行级锁保护更新操作

UPDATE Basic.Entities

WITH (UPDLOCK, HOLDLOCK)

SET Value = @NewValue, LastUpdated = GETDATE()

WHERE EntityID = @EntityID;

-- 验证更新是否成功

IF @@ROWCOUNT = 0

BEGIN

ROLLBACK TRANSACTION;

SELECT 'ERROR' AS Status, '找不到要更新的记录' AS Message;

RETURN;

END

COMMIT TRANSACTION;

SELECT 'SUCCESS' AS Status, '更新成功' AS Message;

END TRY

BEGIN CATCH

-- 错误处理

IF @@TRANCOUNT > 0

ROLLBACK TRANSACTION;

SELECT 'ERROR' AS Status, '更新失败: ' + ERROR\_MESSAGE() AS Message;

END CATCH;

END;

GO

**附录B PPT**

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |