火车下料自动化装置上位机
数据库技术文档

2025 年 07 月 03 日

目录

1 概述 4

1.1 开发背景 4
1.2 编写目的 5
1.3 预期效果 5
1.4 相关术语 6

2 开发平台及语言简介 6

2.1 基于 B/S 结构的开发 6
2.2 SQL SERVER 2014 7
2.3 UML 建模语言 7
2.4 GUI 墨刀 8

3 需求分析 9

3.2 业务需求 9
3.2.1 轨道管理 10
3.2.2 下煤机管理 10
3.3 用例分析 11
3.3.1 涉众或角色 11
3.3.2 用例图 12
3.3.3 用例说明 21

4 总体设计 22

4.1 系统架构 22
4.2 系统部署 23

5 数据库设计 24

5.1 数据库逻辑设计 24
5.1.1 逻辑设计 24
5.1.2 SQL 代码 26
5.2 数据库物理设计 31
5.2.1 RAID 独立存盘冗余阵列 31
5.2.2 RAID5 31
5.3 数据库触发器设计 32
5.3.1 功能设计 32
5.3.2 SQL 代码 33
5.4 辅助存储过程设计 44
5.4.1 功能设计与 SQL 代码 44
5.5 数据库权限与角色设计 46
5.5.1 功能设计与 SQL 代码 46

6 其他 49

6.1 系统安全性设计 49
6.2 系统稳定性设计 50

1 概述

1.1 开发背景

随着我国铁路货运规模的持续扩大，特别是煤炭、矿石等大宗散装货物运输需求的快速增长，火车装煤下料作业的智能化升级已成为提升整体物流效率的关键环节。然而，当前火车装煤下料系统在设备管理、数据整合和过程控制等方面面临着诸多亟待解决的系统性挑战，这些问题的存在严重制约了装车效率的提升和运营成本的优化。在设备层面，装煤系统普遍存在严重的异构性问题。不同铁路站点配备的下料机、传送带、称重传感器等关键设备往往来自不同生产厂商，设备型号繁杂，控制协议和数据接口标准不统一。这种设备间的互操作性问题导致系统集成困难，难以实现集中化管控。更值得注意的是，部分设备因服役年限较长，普遍存在老化现象，加之维护投入不足，经常出现计量偏差、机械故障等问题，直接影响装煤精度和系统稳定性。由于缺乏统一的设备管理平台，维修记录、技术参数等重要资料多以纸质档案或分散的电子文档形式存储，不仅查询困难，更难以进行有效的设备状态分析和预测性维护。在数据管理方面，当前系统存在严重的信息孤岛现象。装煤作业过程中产生的关键数据，如车厢载重、下料量、火车运行速度等，要么依赖人工记录，要么存储在各个独立的子系统中。这些数据缺乏统一的格式标准和共享机制，无法实现实时传输和集中分析。决策层难以及时获取准确的装车进度、设备状态等动态信息，导致运力调配和异常响应严重滞后。更突出的是，由于历史数据归档不规范，故障分析和工艺优化往往只能依赖经验判断，缺乏数据支撑，严重影响决策的科学性。在过程控制方面，传统的装煤作业模式存在明显的效率瓶颈。火车以约 1m/s 的速度匀速通过装煤区时，两个固定位置的下料机需要协同完成装车作业。但由于缺乏智能化的控制策略，经常出现装煤不均匀、车厢未装满或煤炭溢出的情况。现有的控制系统难以根据实时工况动态调整下料参数，也无法对装车质量进行有效监控和反馈。这种粗放式的作业模式不仅造成煤炭浪费，还可能导致超载或欠载，带来安全隐患和运输罚款。在系统协同方面，火车装煤作业涉及多个子系统的配合，包括列车定位、速度控制、下料机调节、称重计量等。然而，这些子系统之间缺乏有效的信息交互机制，各自为

政的运行模式导致整体作业流程存在诸多不协调之处。例如，列车速度波动时，下料系统不能及时响应调整；称重数据与下料控制脱节；异常情况难以及时报警和处理等。这种协同性的缺失严重制约了装车效率的提升。

面对这些挑战，亟需通过智能化的软件建模方法，构建一个集成设备控制、数据融合、过程优化和安全管控于一体的火车智能装煤下料系统。该系统需要实现设备间的互联互通，建立统一的数据平台，开发智能控制算法，并构建完善的安全预警机制，从而全面提升装煤作业的效率、精度和安全性，为铁路货运的数字化转型提供有力支撑。这不仅有助于降低运营成本，提高运输效率，更能为后续的智能调度、 predictive maintenance 等高级应用奠定基础，推动铁路货运向智能化、绿色化方向发展。

1.2 编写目的

本系统基于 IBM 的成熟技术架构和行业数据，遵循标准化、智能化、可靠性和可扩展性的设计原则，采用物联网、大数据分析和智能控制算法等技术路线，构建包含设备层、控制层、数据层和应用层的四层系统架构，通过统一数据库实现装煤作业全流程数据的实时采集、存储与分析，打造智能化火车装煤下料管理平台，实现设备互联、精准下料、动态优化和智能预警，有效解决当前设备异构性、数据孤岛和过程控制等难题，最终达到提升装煤效率 30%、降低运营成本 20%、确保运输安全的目标，为铁路货运数字化转型提供有力支撑。

1.3 预期效果

本系统上位机控制预期实现以下效果：通过可视化人机交互界面，操作人员可实时监控装煤作业全流程状态，包括火车定位信息、下料机工作参数、煤量计量数据等关键指标；系统将基于智能算法自动生成最优控制策略，实现两台下料机的精准协同作业，确保每节车厢装煤均匀且满载率达到 98%以上；同时具备异常工况自动报警与处理建议功能，当检测到设备故障或装煤偏差时立即触发声光报警并提供处置方案；所有作业数据将自动归档并生成标准化报

表，支持历史记录追溯与工艺优化分析，最终实现装煤作业效率提升 40%、人力成本降低 50%、煤炭损耗减少 35%的显著效益。

1.4 相关术语

IS ： Information System 信息系统

UML ： Unified Modeling Language 统一建模语言

B/S ： Browser-Server 浏览器-服务器

GUI ： Generate User Interface 用户界面

2 开发平台及语言简介

2.1 基于 B/S 结构的开发

由于煤炭企业管理系统分布的特性，本系统采用 B/S 结构，即 Browser-Server（浏览器-服务器）架构，B/S 结构是目前最流行的数据库应用模式，它解决了各种分布式应用，扩展了业务范围；在 B/S 结构下，整个系统的管理、资源分配、数据库操作、业务逻辑部件的管理等工作集中用服务器，容易部署和管理。整个系统使用 B/S 架构，则在客户端使用标准的 Web 页面浏览器(如 Internet Explorer 等)，不需安装特殊的应用程序，减少了升级和维护的难度，所有的业务数据都保存在服务器（Server）端，确保了业务的安全；在通讯方面，由于使用的是标准的 Http 协议，使得系统可以轻松的实现移动管理和分布式管理。

2.2 SQL Server 2014

SQL Server 作为世界上部署最广泛的数据库管理软件，承袭「Cloud-First」的精神，SQL Server 2014 藉由突破性的效能与内建 In-Memory 技术，带来实时的性能改进，能够大幅提升资料处理与运算 10 倍的速度，该技术能够飞速处理数以百万条的记录，甚至通过 SQL Server 分析服务，轻松扩展至数以几十亿计的分析能力。有效帮助客户分析更庞大与多样的结构与非结构资料，进而投入商务创新研发，更进一步帮助企业建构出环境智慧平台，帮助企业每一位员工都获得自主分析的能力，创造出企业的资料文化。

2.3 UML 建模语言

UML 作为一种统一的软件建模语言具有广泛的建模能力。UML 是在消化、吸收、提炼至今存在的所有软件建模语言的基础上提出的，集百家之所长，它是软件建模语言的集大成者。UML 还突破了软件的限制，广泛吸收了其他领域的建模方法，并根据建模的一般原理，结合了软件的特点，因此具有坚实的理论基础和广泛性。UML 不仅可以用于软件建模，还可以用于其他领域的建模工作。UML 立足于对事物的实体、性质、关系、结构、状态和动态变化过程的全程描述和反映。UML 可以从不同角度描述人们所观察到的软件视图，也可以描述在不同开发阶段中的软件的形态。UML 可以建立需求模型、逻辑模型、设计模型和实现模型等，但 UML 在建立领域模型方面存在不足，需要进行补充。

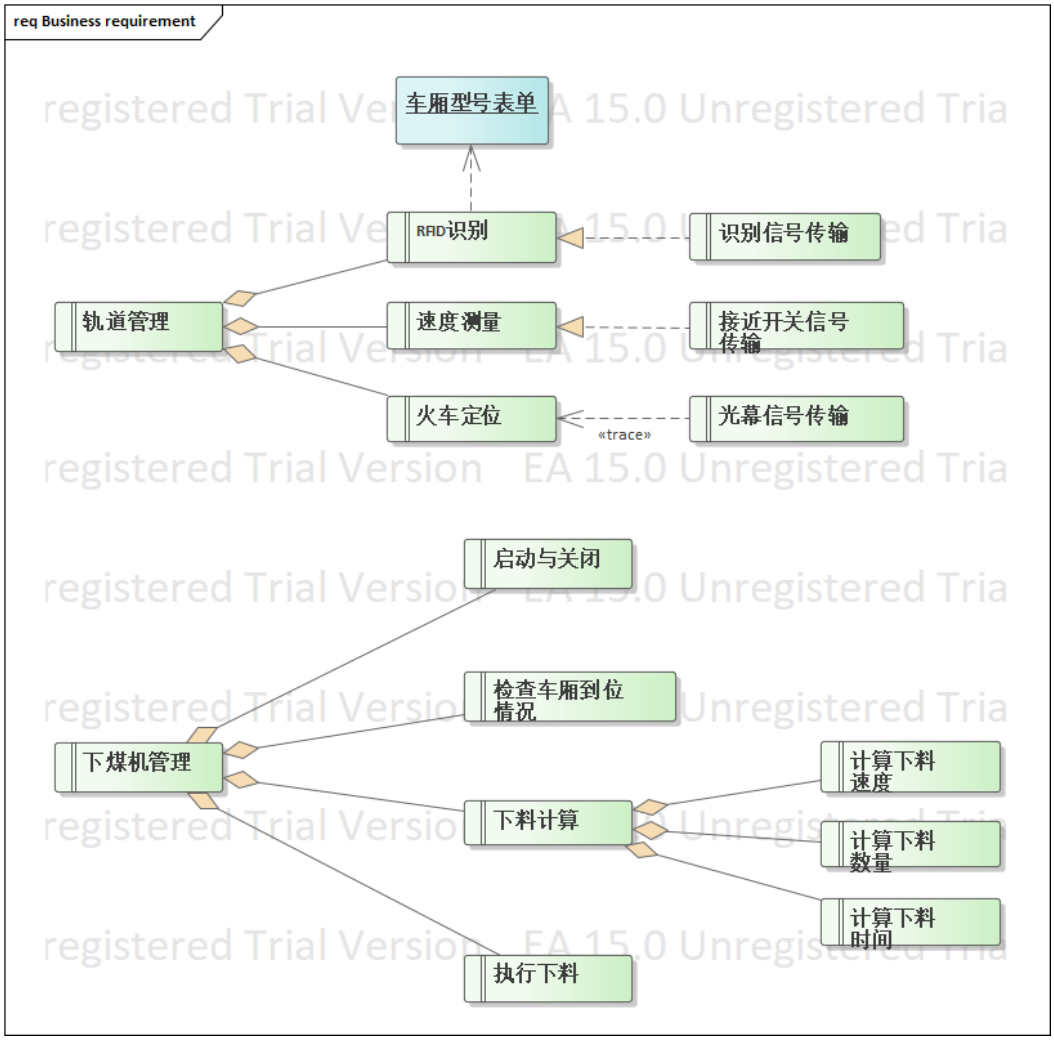
作为一种建模语言，UML 有严格的语法和语义规范。UML 建立在元模型理论基础上，包括 4 层元模型结构，分别是基元模型、元模型、模型和用户对象。4 层结构层层抽象，下一层是上一层的实例。UML 中的所有概念和要素均有严格的语义规范。UML 采用一组图形符号来描述软件模型，这些图形符号具有简单、直观和规范的特点，开发人员学习和掌握起来比较简单。所描述的软件模型，可以直观地理解和阅读，由于具有规范性，所以能够保证模型的准确、一致。

2.4 GUI 墨刀

墨刀是一款在线原型设计与协同工具，借助墨刀，产品经理、设计师、开发、销售、运营及创业者等用户群体，能够搭建为产品原型，演示项目效果。墨刀同时也是协作平台，项目成员可以协作编辑、审阅，不管是产品想法展示，还是向客户收集产品反馈，向投资人进行 Demo 展示，或是在团队内部进行协作沟通、项目管理 。

3 需求分析

3.2　业务需求



业务需求图

系统主要提供以下模块、功能：

（1） 轨道管理模块（2） 下煤机管理模块

（3） RFID 识别（4） 火车速度测量

（5） 火车定位（6） 下煤机启动关闭 （7） 就位控制

（8） 下料计算 （9）执行下料

3.2.1 轨道管理

R001 RFID 识别：系统管理员构建企业内部所含火车车型的 RFID 编码，录入数据库。当火车进入轨道后由 RFID 识别编码匹配数据库中数据，为后台提供车厢的长、宽、高、型号等信息，为智能化控制提供数据依据。

R002 速度测量：下游接近开关通过上升沿信号时间差值计算火车平均速度，实施录入至上位机数据库系统中。通过后端逻辑运行计算合理的煤炭机开机时间。

R003 火车定位：上位机系统接收来自下游光幕定位器的信号，监控火车是否已经到达下料标准位置。若已到位，执行下煤机管理模块，实施下料。若未到位，则锁定系统，等待信号指令。

3.2.2 下煤机管理

R004 启动与关闭：上位机通讯 PLC S7-300，执行下煤机开启与关闭程序。启动关闭操作录入数据库系统，形成下煤机开关机操作日志。

R005 检查车厢到位情况：财务人员等角色需要查询支票报销、资金往来等业务，员工需要根据要求报销费用。

R006 下料计算：具体工程项目由于一些原因，需要进行调整，并对相应的记录进行更改

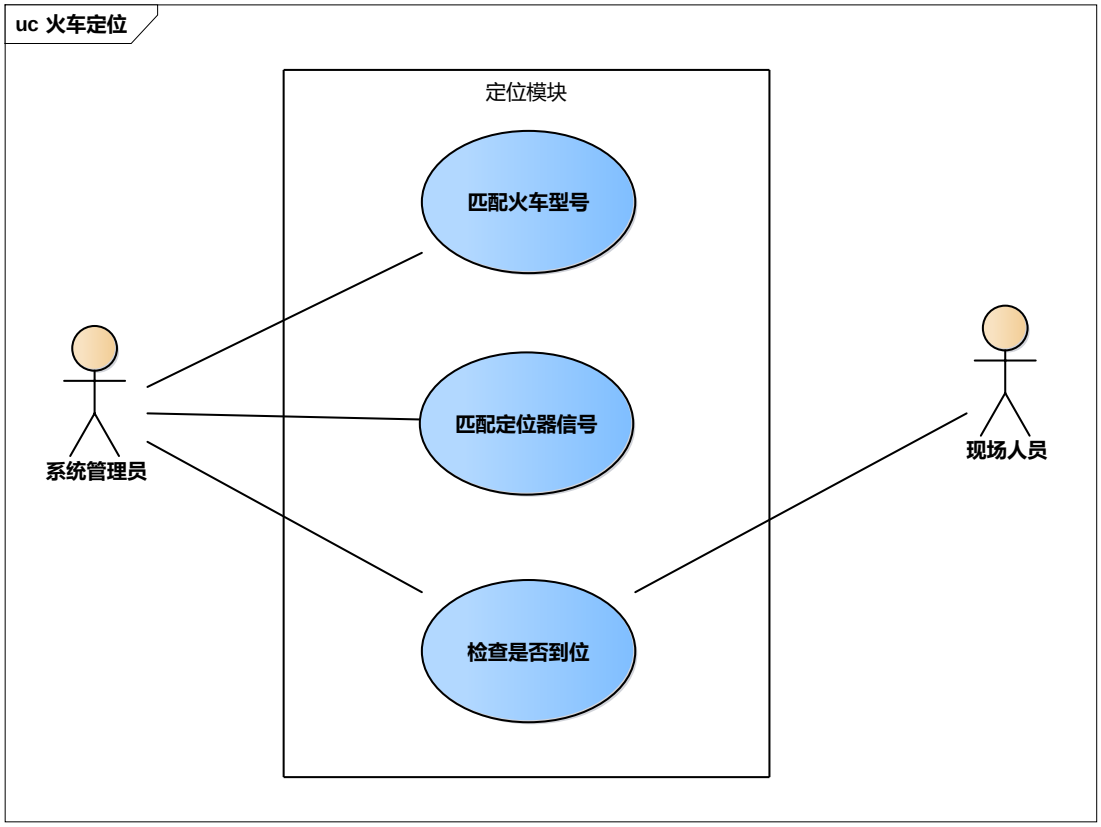
R007 执行下料：具体工程项目由于一些原因，需要进行调整，并对相应的记录进行更改

3.3 用例分析

3.3.1 涉众或角色

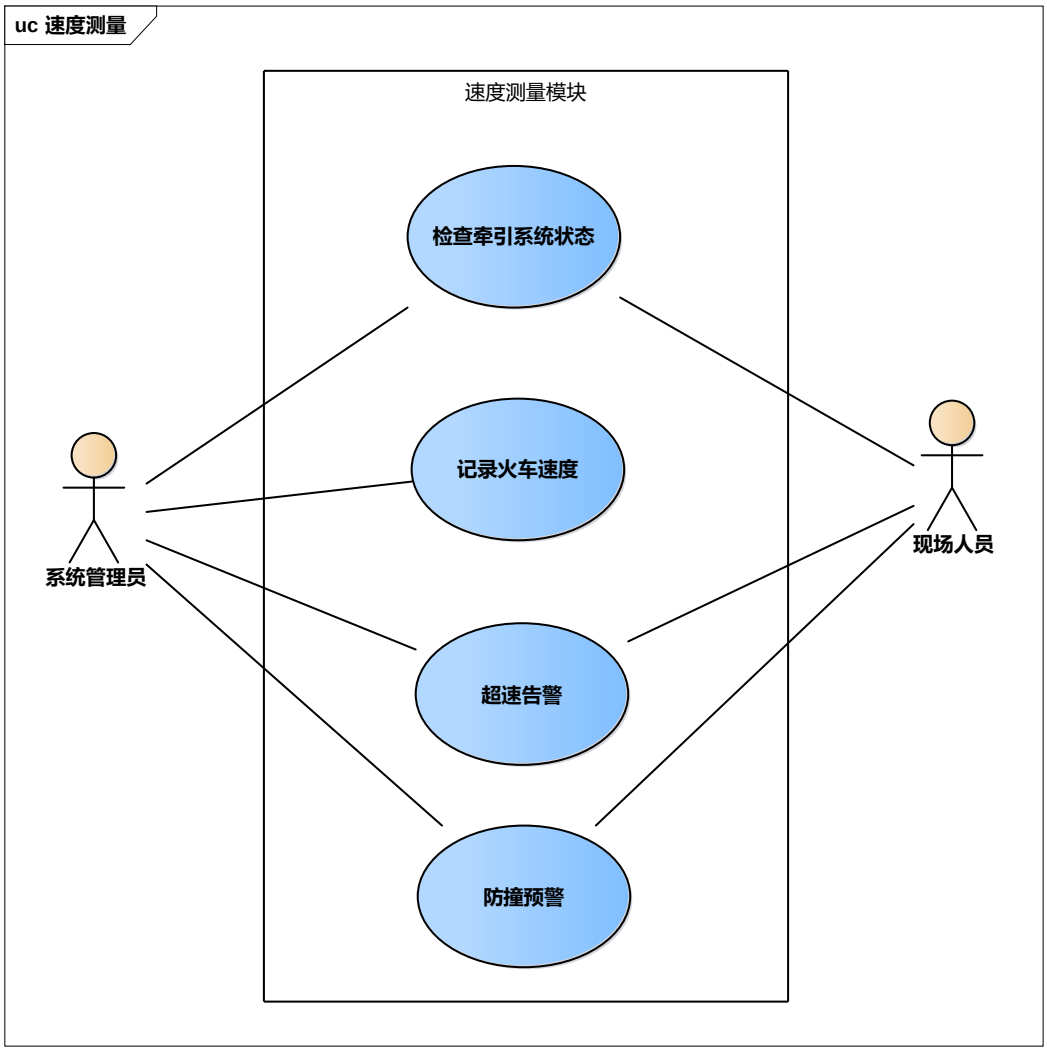
角色	工作内容
系统管理员	在火车出入库、定位、速度测量系统中拥有最高权限，拥有增删改查的能力，但操作记录会被系统历史记录。能够访问企业内部所有车厢的信息、调度记录。能够维护数据库系统，保证数据安全性，修改数据库配置与安全授权。
系统控制员	在下煤机管理模块中负责下煤机的开机、关机、执行下料的自动化指令操作，负责后台监控下煤机的下料过程。
仓库管理员	作为火车进入轨道后，现场控制人员，能够详细访问火车不同车厢型号进入轨道的时间、编号信息，负责审核火车是否进入预定轨道进行下煤操作。
现场人员	在下煤机管理模块中负责执行下料操作，监控现场控制下煤量。在速度测量模块中，负责防撞告警、超速告警部分，及时关闭牵引车，防止出现安全事故。在火车定位模块中，负责监控火车是否到达下料位置，结合工控机下达下料指令。
部门主管	拥有各个模块功能、表单的查询功能，并能将最高级的查询功能授予给其他类型的用户。

3.3.2 用例图



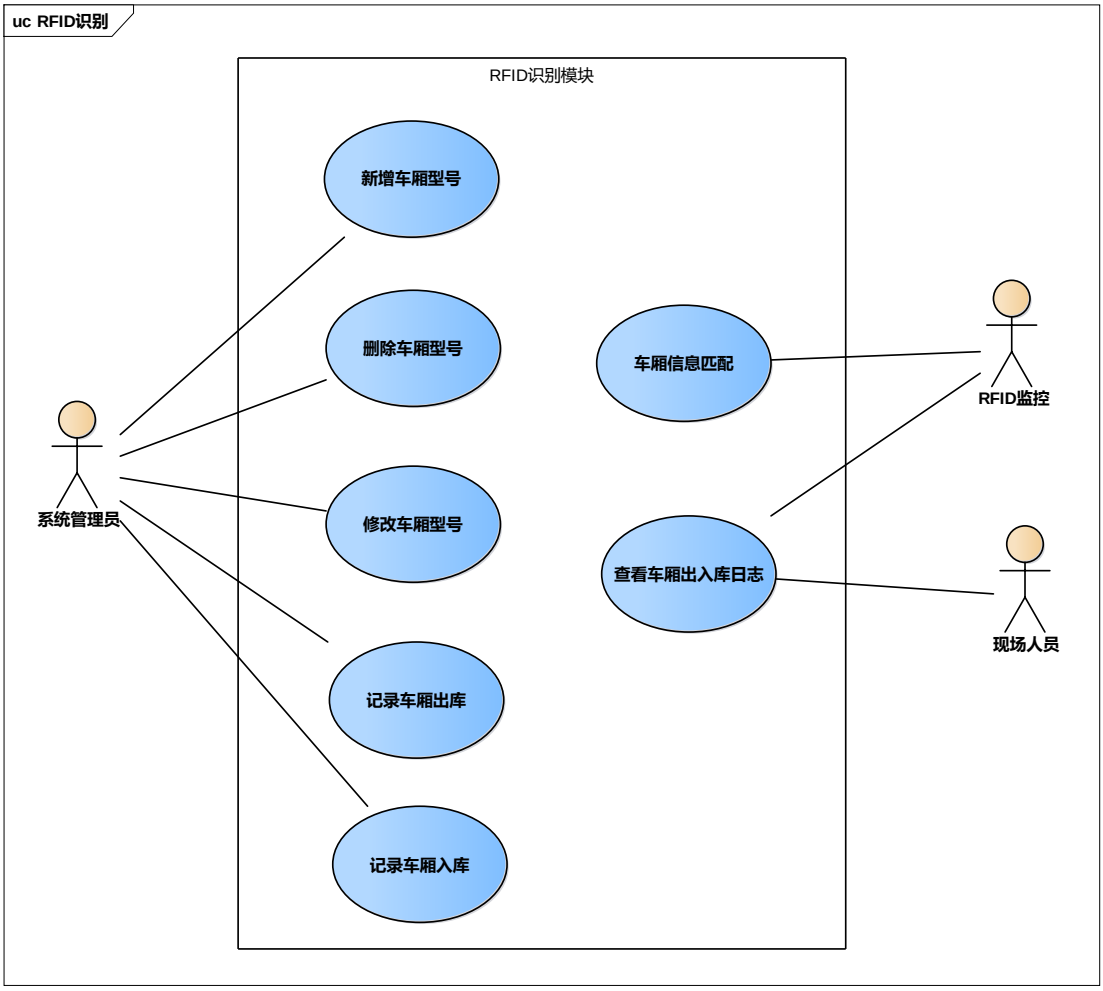
用例编号：001 002 003
用例名称：火车定位
概述：匹配火车车厢信息，并确认车厢是否到达下料位置
参与者：系统管理员、现场人员
前置条件：系统正常运行、权限通过
主序列： 1. 验证权限 2. 选择服务类型 3. 匹配火车型号：若有权限，生成火车车号、火车型号、到位时间、

到位仓库、仓库负责人，录入数据库历史表单。 4. 匹配定位器信号：若有权限，记录火车车号、火车型号、是否到位，录入数据库历史表单。 5. 检查是否到位：若未到位，启动预警，告知现场人员。若到位，返回指令，启动连接下煤机管理模块。
可替换序列： 第一步：如果客户没有权限，则无法进入系统，并存储后台访问记录 第三步：如果客户没有权限，则无法匹配车型，并存储后台访问记录 第五步：如果客户没有权限，则无法启动定位，并存储后台访问记录
后置条件：系统正常运行



用例编号：004 005 006 007
用例名称：速度测量
概述：对火车牵引系统、火车行进间进行监控告警
参与者：系统管理员、现场人员
前置条件：系统正常运行、权限通过
主序列： 1. 验证权限 2. 系统管理员检查牵引车状态：若正常，开启执行速度测量模块，启

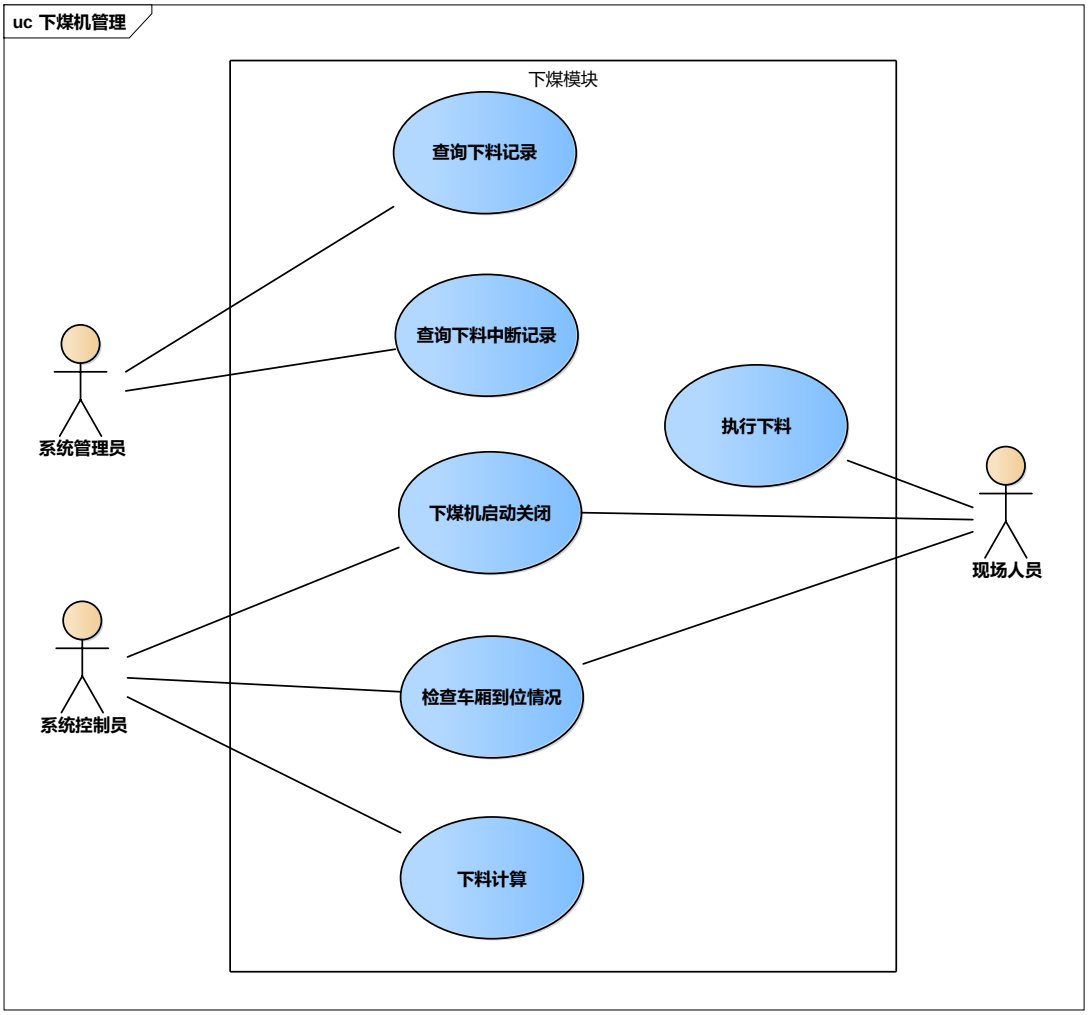
动牵引车。若异常，返回工控机界面启动异常。 3. 记录火车速度：接收下游传感器信号，计算出火车速度。每隔 2 秒记录一次火车速度，并以时间戳形式录入到数据库历史记录中。界面上着重显示出当前速度。 4. 防撞告警：当速度超过阈值，启动防撞告警，通知现场人员，并自动降低火车速度。
可替换序列： 第一步：如果客户没有权限，则无法启动模块，并存储后台访问记录
后置条件：系统正常运行



用例编号：008 009 0010
用例名称：RFID 识别模块——车厢型号管理
概述：利用 RFID 信号，管理车厢类型库。
参与者：系统管理员
前置条件：系统正常运行、权限通过
主序列： 1. 验证权限 2. 选择所需要的服务 3. 新增车厢型号：系统管理员手动输入车厢型号以及尺寸大小 4. 删除车厢型号：系统管理员删除数据库中的车厢型号信息，删除操作历史自动保存至操作日志。 5. 查询车厢型号：系统管理员根据车厢描述，查询对应的车厢型号信息。改功能以接口方式，为下煤机管理模块提供下煤计算量支持。
可替换序列： 第一步：如果客户没有权限，则无法领料，并存储后台访问记录 第四步：若系统管理员想删除操作日志，返回失败。
后置条件：系统正常运行

用例编号：0011 0012 0013 0014
用例名称：RFID 识别模块——车厢出入库管理
概述：利用 RFID 信号，记录不同车厢出入库历史。
参与者：系统管理员、现场人员、RFID 监控

前置条件：系统正常运行、权限通过
主序列： 1. 验证权限 2. 车辆信息匹配：RFID 监控到车辆入库，扫描访问数据库得到型号数据，并将型号数据上传给下煤机管理模块。 3. 记录车厢入库：自动录入车号、车厢类型、入库时间、仓库号 4. 记录车厢出库：自动录入车号、车厢类型、出库时间、仓库号 5. 查询车厢出入库日志：现场人员可以申请查询车厢的出入库记录表单。
可替换序列： 第一步：如果客户没有权限，则无法启动 RFID，并存储后台访问记录 第五步：如果部门主管未同意查询申请，则拒绝查询请求，并返回失败原因
后置条件：系统正常运行



用例编号：0015 0016
用例名称：下煤机管理模块—查询业务
概述：系统管理员查询下料记录、中断记录
参与者：系统管理员怎么样才算是中断时间
前置条件：系统正常运行、权限通过
主序列： 1. 验证权限 2. 查询下料记录：根据检索条件，查询下料机、仓库号、下料时间 3. 查询下料中断记录：根据检索条件，查看中断机器、中断原因、中

断时间 4. 保存查询日志
可替换序列： 第一步：如果客户没有权限，则无法进行查询访问，并存储后台访问 第三步：系统管理员权限必须为部门主管，如果角色权限不够，则无法进行查询。
后置条件：系统正常运行

用例编号：0017 0018 0019 0020
用例名称：下煤机管理模块—执行业务
概述：相关人员控制、执行、监控下料机的开关与执行操作
参与者：系统控制员、现场人员
前置条件：系统正常运行、权限通过
主序列： 1. 验证权限 2. 下煤机开启与关闭：打开或关闭下煤机控制系统 3. 检查车厢到位情况：接收来自定位模块接口的信号，若收到定位完毕信号，则返回车厢已到位，可以执行下料 4. 下料计算：接收来自 RFID 模块接口的信号，提取车厢的相关信息，根据下料类型、车厢尺寸，智能调整下料计算算法，计算出下料量 5. 执行下料：现场人员监控下料情况，若有溢出，调低下料量阈值，执行下料操作。

可替换序列： 第一步：如果客户没有权限，则无法启动下煤机模块，并存储后台访问 第三步：如果定位模块没有返回到位信号，则无法继续执行下煤机模块，并显示在后台。 第四步：如果没有收到 RFID 模块所查询到的车厢信息，则无法继续执行下煤机模块，并显示在后台
后置条件：系统正常运行

3.3.3 用例说明

用例编号	用例名称	说明
U001	匹配火车型号	系统匹配火车型号信息，实现车型识别
U002	匹配定位器信号	系统匹配定位器信号，实现精确定位
U003	检查车厢是否到位	系统检查车厢是否到达指定位置，确保装卸作业准备就绪
U004	检查牵引系统状态	系统检查火车牵引系统运行状态，确保牵引功能正常
U005	记录火车速度	系统实时记录火车运行速度，提供速度监控数据
U006	超速告警	系统检测到火车超速时发出告警，确保行车安全
U007	防撞告警	系统检测到碰撞风险时发出告警，预防事故发生
U008	新增车厢型号	系统添加新的车厢型号信息，扩展车型数据库
U009	删除车厢型号	系统移除不再使用的车厢型号信息，维护车型数据库
U010	修改车厢型号	系统更新现有车厢型号信息，保持车型数据准确性
U011	记录车厢出库	系统记录车厢出库信息，实现出库流程管理
U012	记录车厢入库	系统记录车厢入库信息，实现入库流程管理
U013	车厢信息匹配	系统匹配车厢信息与数据库记录，确保信息一致性
U014	查看车厢出入库日志	系统提供车厢出入库历史记录查询功能
U015	查看下料记录	系统提供下料作业历史记录查询功能
U016	查看下料中断记录	系统提供下料作业中断异常记录查询功能
U017	下煤机启动与关闭	系统控制下煤机的启动和关闭操作，实现装卸自动化
U018	检查车厢定位情况	系统检查车厢的实时定位信息，确保装卸位置准确
U019	下料计算	系统计算下料重量和速率，提供装卸作业参数
U020	执行下料	系统执行自动化下料操作，完成装卸作业流程

4 总体设计

4.1 系统架构

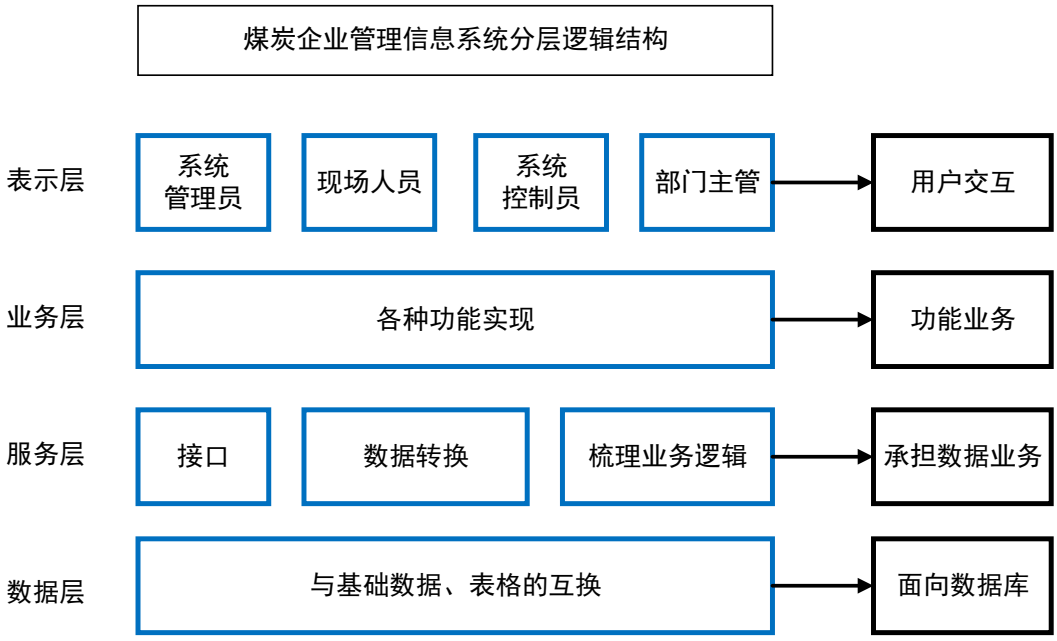


图 1. 煤炭企业管理信息系统架构图

煤炭企业管理信息系统架构主要分为四层：

- **数据层** 主要实现与物理表、基础数据的交互，面向数据库进行物理存储与查询；建立视图利用接口，加速查询速度。
- **服务层** 主要实现数据层和业务层之间的过渡，完成数据转换、逻辑梳理等任务，确保接口的正常工作。
- **业务层** 主要完成各种功能的实现，在本系统中，主要包括 4 大功能模块，一共 20 个子功能模块，涵盖**车厢、测速、定位、告警、控制、查询**等日常业务。
- **表示层** 主要实现与用户的交互，即面向不同的角色提供不同的权限与功能实现。

4.2 系统部署

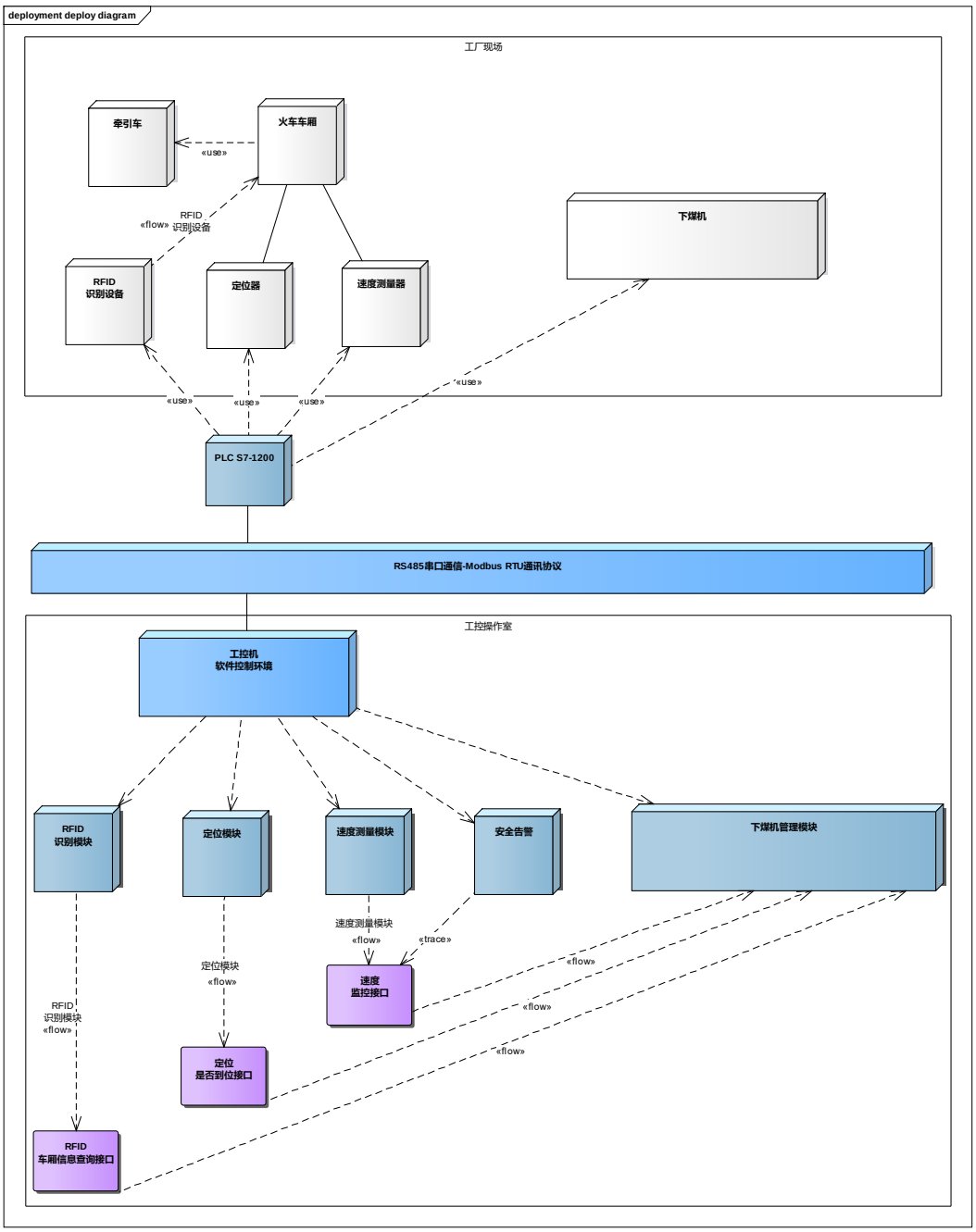


图 2. 煤炭企业管理信息系统软硬件系统部署图

5 数据库设计

5.1 数据库逻辑设计

5.1.1 逻辑设计

Train Model Table (火车型号表)——主键：TrainTypeID

字段名	字段代码	数据类型	数据类型大小	可否为空
火车型号	TrainTypeID	int	20	N
装料类型	HoldType	varchar	50	N
外部长度	ExLength	float	20	N
外部宽度	ExWidth	float	20	N
外部高度	ExHeight	float	20	N
内部长度	InLength	float	20	N
内部宽度	InWidth	float	20	N
内部高度	InHeight	float	20	N
容积	Volume	float	20	N
焦炭密度	Density	float	20	
装车重量	LoadWeight	float	20	
全长	FullLength	float	20	
底板面高	BottomHeight	float	20	
车钩型号	HookType	varchar	50	

Train Arrival Table (火车到位表)——主键：ID

字段名	字段代码	数据类型	数据类型大小	可否为空
车号	TrainID	int	20	N
型号	TrainTypeID	int	20	N
仓库号	WarehouseID	varchar	50	N
仓库名称	WarehouseName	varchar	50	N
是否到位	OnPosition	varchar	20	N
时间	Time	int	20	N

Transaction Status Table (牵引系统状态表)——主键：ID

字段名	字段代码	数据类型	数据类型大小	可否为空
仓库号	WarehouseID	int	20	N
仓库名称	WarehouseName	int	20	N
牵引车状态	Transaction Status	int	20	N
时间	Time	varchar	50	N

Train Speed Table（火车速度表）——主键：ID

字段名	字段代码	数据类型	数据类型大小	可否为空
车号	TrainID	char	20	N
型号	TrainTypeID	char	50	N
仓库号	WarehouseID	char	20	N
速度	Speed	time	50	N
是否超速	Overspeed	char	20	N
时间	Time			

Warning Table（防撞告警表）——主键：ID

字段名	字段代码	数据类型	数据类型大小	可否为空
车号	TrainID	char	20	N
型号	TrainTypeID	char	50	N
仓库号	WarehouseID	char	20	N
是否告警	Warning	time	50	N
是否超速	Overspeed	char	20	N
时间	Time			

Inventory Table（车厢入库表）——主键：ID

字段名	字段代码	数据类型	数据类型大小	可否为空
车号	TrainID	char	20	N
型号	TrainTypeID	char	50	N
仓库号	WarehouseID	char	20	N
审核人	Reporter	time	50	N
入库时间	InTime	char	20	N

Delivery Table（车厢出库表）——主键：ID

字段名	字段代码	数据类型	数据类型大小	可否为空
车号	TrainID	char	20	N
型号	TrainTypeID	char	50	N
仓库号	WarehouseID	char	20	N
审核人	Reporter	time	50	N
出库时间	OutTime	char	20	N

Layoff History Table（下料记录表）——主键 ContractID

字段名	字段代码	数据类型	数据类型大小	可否为空
仓库号	WarehouseID	char	20	N
煤机号	MachineID	char	80	N
下料时间	Time	char	20	N

Layoff Break Table（下料中断表）——主键：AccountID

字段名	字段代码	数据类型	数据类型大小	可否为空
仓库号	WarehouseID	char	20	N
煤机号	MachineID	char	20	N

中断原因	Reason	char	50	N
中断时间	Time	char	20	N

Machine Table（下煤机表）——主键：MaterialID

字段名	字段代码	数据类型	数据类型大小	可否为空
煤机号	MaterialID	char	20	N
仓库号	WarehouseID	char	50	N
机容量	Volume	char	50	N
现场负责人	Reporter	char	20	N
开闭状态	OpenStatus	char	20	N
运行状态	RunStatus	char	20	N

5.1.2 SQL 代码

```
-- 创建火车型号表
CREATE TABLE [Train Model Table] (
    [TrainTypeID] INT NOT NULL,
    [HoldType] VARCHAR(50) NOT NULL,
    [ExLength] FLOAT NOT NULL,
    [ExWidth] FLOAT NOT NULL,
    [ExHeight] FLOAT NOT NULL,
    [InLength] FLOAT NOT NULL,
    [InWidth] FLOAT NOT NULL,
    [InHeight] FLOAT NOT NULL,
    [Volume] FLOAT NOT NULL,
    [Density] FLOAT NULL,
    [LoadWeight] FLOAT NULL,
    [FullLength] FLOAT NULL,
    [BottomHeight] FLOAT NULL,
    [HookType] VARCHAR(50) NULL,
    CONSTRAINT [PK_TrainModel] PRIMARY KEY CLUSTERED ([TrainTypeID] ASC)
);

-- 创建火车到位表
CREATE TABLE [Train Arrival Table] (
```

```
    [ID] INT IDENTITY(1,1) NOT NULL,
    [TrainID] INT NOT NULL,
    [TrainTypeID] INT NOT NULL,
    [WarehouseID] VARCHAR(50) NOT NULL,
    [WarehouseName] VARCHAR(50) NOT NULL,
    [OnPosition] VARCHAR(20) NOT NULL,
    [Time] INT NOT NULL,
    CONSTRAINT [PK_TrainArrival] PRIMARY KEY CLUSTERED ([ID] ASC),
    CONSTRAINT [FK_TrainArrival_TrainModel] FOREIGN KEY ([TrainTypeID])
        REFERENCES [Train Model Table] ([TrainTypeID])
);

-- 创建牵引系统状态表
CREATE TABLE [Transaction Status Table] (
    [ID] INT IDENTITY(1,1) NOT NULL,
    [WarehouseID] INT NOT NULL,
    [WarehouseName] INT NOT NULL,
    [Transaction Status] INT NOT NULL,
    [Time] VARCHAR(50) NOT NULL,
    CONSTRAINT [PK_TransactionStatus] PRIMARY KEY CLUSTERED ([ID] ASC)
);

-- 创建火车速度表
CREATE TABLE [Train Speed Table] (
    [ID] INT IDENTITY(1,1) NOT NULL,
    [TrainID] CHAR(20) NOT NULL,
    [TrainTypeID] CHAR(50) NOT NULL,
    [WarehouseID] CHAR(20) NOT NULL,
    [Speed] TIME NOT NULL,
    [Overspeed] CHAR(20) NOT NULL,
    [Time] DATETIME NULL,
    CONSTRAINT [PK_TrainSpeed] PRIMARY KEY CLUSTERED ([ID] ASC)
```

```
);

-- 创建防撞告警表
CREATE TABLE [Warning Table] (
    [ID] INT IDENTITY(1,1) NOT NULL,
    [TrainID] CHAR(20) NOT NULL,
    [TrainTypeID] CHAR(50) NOT NULL,
    [WarehouseID] CHAR(20) NOT NULL,
    [Warning] TIME NOT NULL,
    [Overspeed] CHAR(20) NOT NULL,
    [Time] DATETIME NULL,
    CONSTRAINT [PK_Warning] PRIMARY KEY CLUSTERED ([ID] ASC)
);

-- 创建车厢入库表
CREATE TABLE [Inventory Table] (
    [ID] INT IDENTITY(1,1) NOT NULL,
    [TrainID] CHAR(20) NOT NULL,
    [TrainTypeID] CHAR(50) NOT NULL,
    [WarehouseID] CHAR(20) NOT NULL,
    [Reporter] TIME NOT NULL,
    [InTime] CHAR(20) NOT NULL,
    CONSTRAINT [PK_Inventory] PRIMARY KEY CLUSTERED ([ID] ASC)
);

-- 创建车厢出库表
CREATE TABLE [Delivery Table] (
    [ID] INT IDENTITY(1,1) NOT NULL,
    [TrainID] CHAR(20) NOT NULL,
    [TrainTypeID] CHAR(50) NOT NULL,
    [WarehouseID] CHAR(20) NOT NULL,
    [Reporter] TIME NOT NULL,
```

```
    [OutTime] CHAR(20) NOT NULL,
    CONSTRAINT [PK_Delivery] PRIMARY KEY CLUSTERED ([ID] ASC)
);

-- 创建下料记录表
CREATE TABLE [Layoff History Table] (
    [ContractID] INT IDENTITY(1,1) NOT NULL,
    [WarehouseID] CHAR(20) NOT NULL,
    [MachineID] CHAR(80) NOT NULL,
    [Time] CHAR(20) NOT NULL,
    CONSTRAINT [PK_LayoffHistory] PRIMARY KEY CLUSTERED ([ContractID] ASC)
);

-- 创建下料中断表
CREATE TABLE [Layoff Break Table] (
    [AccountID] INT IDENTITY(1,1) NOT NULL,
    [WarehouseID] CHAR(20) NOT NULL,
    [MachineID] CHAR(20) NOT NULL,
    [Reason] CHAR(50) NOT NULL,
    [Time] CHAR(20) NOT NULL,
    CONSTRAINT [PK_LayoffBreak] PRIMARY KEY CLUSTERED ([AccountID] ASC)
);

-- 创建下煤机表
CREATE TABLE [Machine Table] (
    [MaterialID] CHAR(20) NOT NULL,
    [WarehouseID] CHAR(50) NOT NULL,
    [Volume] CHAR(50) NOT NULL,
    [Reporter] CHAR(20) NOT NULL,
    [OpenStatus] CHAR(20) NOT NULL,
    [RunStatus] CHAR(20) NOT NULL,
    CONSTRAINT [PK_Machine] PRIMARY KEY CLUSTERED ([MaterialID] ASC)
```

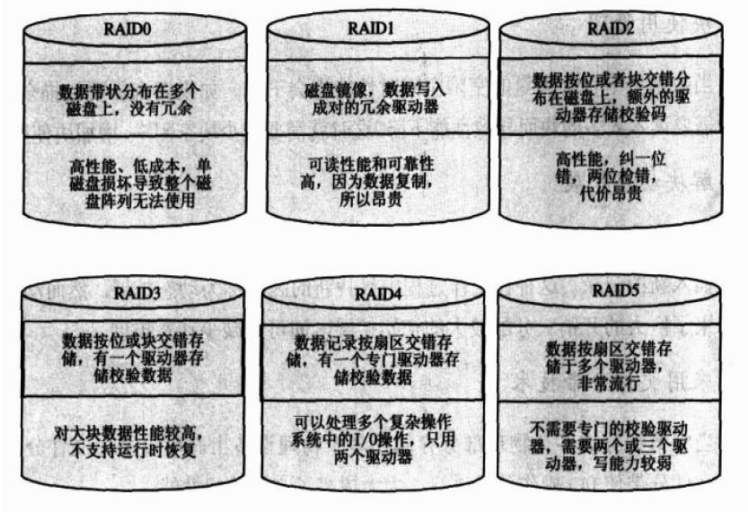
```
);

-- 添加外键约束（火车速度表引用火车型号表）
ALTER TABLE [Train Speed Table]
ADD CONSTRAINT [FK_TrainSpeed_TrainModel]
FOREIGN KEY ([TrainTypeID]) REFERENCES [Train Model Table] ([TrainTypeID]);

-- 添加外键约束（防撞告警表引用火车型号表）
ALTER TABLE [Warning Table]
ADD CONSTRAINT [FK_Warning_TrainModel]
FOREIGN KEY ([TrainTypeID]) REFERENCES [Train Model Table] ([TrainTypeID]);

-- 添加外键约束（车厢入库表引用火车型号表）
ALTER TABLE [Inventory Table]
ADD CONSTRAINT [FK_Inventory_TrainModel]
FOREIGN KEY ([TrainTypeID]) REFERENCES [Train Model Table] ([TrainTypeID]);

-- 添加外键约束（车厢出库表引用火车型号表）
ALTER TABLE [Delivery Table]
ADD CONSTRAINT [FK_Delivery_TrainModel]
FOREIGN KEY ([TrainTypeID]) REFERENCES [Train Model Table] ([TrainTypeID]);
```

5.2 数据库物理设计

5.2.1 RAID 独立存盘冗余阵列

本系统采用 RAID5 存储技术，中文是“独立磁盘冗余阵列”，简称磁盘阵列。简单的说，RAID 是一种把多块独立的硬盘（物理硬盘）按不同的方式组合起来形成一个硬盘组（逻辑硬盘），从而提供比单个硬盘更高的存储性能和提供数据备份技术。

组成磁盘阵列的不同方式称为 RAID 级别（RAID Levels），现在已拥有了从 RAID 0 到 6 七种基本的 RAID 级别。另外，还有一些基本 RAID 级别的组合形式，如 RAID 10（RAID 0 与 RAID 1 的组合）等等



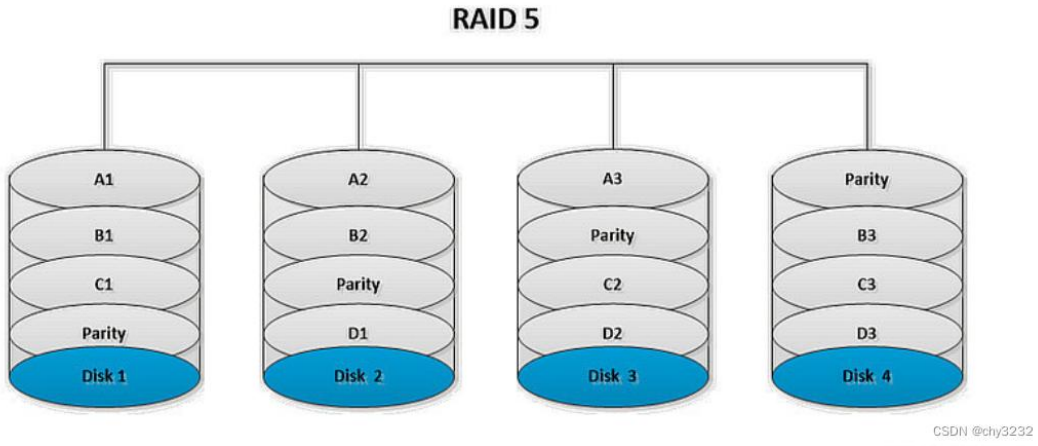
选择 RAID 技术进行物理存储的依据是基于 RAID 的两大特点：一是速度、二是安全。

5.2.2 RAID5

RAID 5 技术把多块硬盘设备（至少三块）的数据奇偶校验信息保

存到其他硬盘设备中。RAID 5 磁盘阵列组中数据的奇偶校验信息并不是单独保存到某一块硬盘设备中，而是存储到除自身以外的其他每一块硬盘设备上，这样的好处是其中任何一设备损坏后不至于出现致命缺陷；

RAID5 不对存储的数据进行备份，而是把数据和相对应的奇偶校验信息存储到组成 RAID5 的各个磁盘上，并且奇偶校验信息和相对应的数据分别存储于不同的磁盘上。当 RAID5 的一个磁盘数据发生损坏后，利用剩下的数据和相应的奇偶校验信息去恢复被损坏的数据。



- 优点：兼顾空间利用率与安全性。
- 缺点：需要额外的运算资源，仅能忍受 1 个硬盘损毁。
- 硬盘数量：至少 3 个。

5.3 数据库触发器设计

5.3.1 功能设计

功能需求	具体描述
数据完整性保障	通过验证触发器确保火车型号、车厢状态等关键数据的一致性
业务规则执行	强制执行仓库容量限制、出入库顺序等业务规则

安全监控	实时检测超速、碰撞风险并触发应急响应
状态追踪	自动记录设备状态变更和操作历史
异常处理	对下料中断等异常情况自动创建维护工单
审计追踪	记录所有关键数据的变更历史

5.3.2 SQL 代码

1）火车型号管理相关触发器

a）火车型号修改审计触发器

```
CREATE TRIGGER tr_TrainModel_AuditUpdate
ON [Train Model Table]
AFTER UPDATE
AS
BEGIN
    SET NOCOUNT ON;

    INSERT INTO TrainModelAuditLog (
        TrainTypeID, OperationType, OperationTime,
        Old_HoldType, New_HoldType,
        Old_ExLength, New_ExLength,
        Old_Volume, New_Volume,
        Operator
    )
    SELECT
        i.TrainTypeID, 'UPDATE', GETDATE(),
        d.HoldType, i.HoldType,
        d.ExLength, i.ExLength,
        d.Volume, i.Volume,
        SYSTEM_USER
    FROM inserted i
    JOIN deleted d ON i.TrainTypeID = d.TrainTypeID;
END;
```

b）火车型号删除限制触发器

```
CREATE TRIGGER tr_TrainModel_PreventDeleteIfInUse
ON [Train Model Table]
INSTEAD OF DELETE
AS
BEGIN
    SET NOCOUNT ON;

    IF EXISTS (
        SELECT 1 FROM deleted d
        JOIN [Train Arrival Table] t ON d.TrainTypeID = t.TrainTypeID
    )
    BEGIN
        RAISERROR('Cannot delete train model that is currently in use by active trains.', 16, 1);
        ROLLBACK TRANSACTION;
        RETURN;
    END;

    -- If not in use, proceed with deletion
    DELETE FROM [Train Model Table]
    WHERE TrainTypeID IN (SELECT TrainTypeID FROM deleted);
END;
```

2）火车到位与牵引系统触发器

a）火车状态更新触发器

```
CREATE TRIGGER tr_TrainArrival_StatusUpdate
ON [Train Arrival Table]
AFTER UPDATE
AS
BEGIN
    SET NOCOUNT ON;
```

```
    -- If train just arrived (status changed to '到位')
    IF UPDATE(OnPosition) AND EXISTS (
        SELECT 1 FROM inserted i
        JOIN deleted d ON i.ID = d.ID
        WHERE d.OnPosition = '未到位' AND i.OnPosition = '到位'
    )
    BEGIN
        -- Log the arrival event
        INSERT INTO TrainArrivalEvents (TrainID, TrainTypeID, WarehouseID, ArrivalTime)
        SELECT TrainID, TrainTypeID, WarehouseID, GETDATE()
        FROM inserted
        WHERE OnPosition = '到位';

        -- Update warehouse status
        UPDATE w
        SET w.CurrentTrains = w.CurrentTrains + 1
        FROM WarehouseStatus w
        JOIN inserted i ON w.WarehouseID = i.WarehouseID;
    END;
END;
```

b）牵引系统状态异常触发器

```
CREATE TRIGGER tr_TransactionStatus_Alert
ON [Transaction Status Table]
AFTER INSERT, UPDATE
AS
BEGIN
    SET NOCOUNT ON;

    -- Check for abnormal traction system status
    IF EXISTS (
        SELECT 1 FROM inserted
```

```
        WHERE [Transaction Status] NOT IN (0, 1) -- Assuming 0=normal, 1=warning
    )
    BEGIN
        -- Insert into alert table
        INSERT INTO SystemAlerts (AlertType, WarehouseID, AlertTime, Status)
        SELECT '牵引系统异常', WarehouseID, GETDATE(), [Transaction Status]
        FROM inserted
        WHERE [Transaction Status] NOT IN (0, 1);

        -- Notify maintenance team
        EXEC sp_SendAlertNotification '牵引系统异常', '请立即检查牵引系统状态';
    END;
END;
```

3）速度与安全监控触发器

a）超速检测与告警触发器

```
CREATE TRIGGER tr_TrainSpeed_OverspeedCheck
ON [Train Speed Table]
AFTER INSERT, UPDATE
AS
BEGIN
    SET NOCOUNT ON;

    -- Get speed limit for each train type (assuming this exists in Train Model Table)
    -- Mark as overspeed if exceeds limit
    UPDATE ts
    SET ts.Overspeed = CASE WHEN i.Speed > tm.MaxSpeed THEN 'Y' ELSE 'N' END
    FROM [Train Speed Table] ts
    JOIN inserted i ON ts.ID = i.ID
    JOIN [Train Model Table] tm ON i.TrainTypeID = tm.TrainTypeID;
```

```
    -- Insert alert for overspeed situations
    INSERT INTO SpeedAlerts (TrainID, TrainTypeID, Speed, SpeedLimit, AlertTime)
    SELECT i.TrainID, i.TrainTypeID, i.Speed, tm.MaxSpeed, GETDATE()
    FROM inserted i
    JOIN [Train Model Table] tm ON i.TrainTypeID = tm.TrainTypeID
    WHERE i.Speed > tm.MaxSpeed;

    -- If overspeed, trigger emergency protocol
    IF EXISTS (SELECT 1 FROM inserted i WHERE i.Overspeed = 'Y')
    BEGIN
        EXEC sp_TriggerEmergencyProtocol '超速告警';
    END;
END;
```

b）防撞告警触发器

```
CREATE TRIGGER tr_Warning_CollisionAlert
ON [Warning Table]
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;

    -- For new collision warnings
    IF EXISTS (SELECT 1 FROM inserted WHERE Warning = 'Y')
    BEGIN
        -- Log the collision warning
        INSERT INTO CollisionEvents (TrainID, TrainTypeID, WarehouseID, EventTime, Severity)
        SELECT TrainID, TrainTypeID, WarehouseID, GETDATE(), 'HIGH'
        FROM inserted
        WHERE Warning = 'Y';

        -- Trigger emergency stop for affected trains
```

```
        UPDATE [Train Control Table]
        SET EmergencyStop = 'Y'
        WHERE TrainID IN (SELECT TrainID FROM inserted WHERE Warning = 'Y');

        -- Notify control center
        EXEC sp_SendEmergencyAlert '防撞告警', '检测到潜在碰撞风险，已启动紧急制动';
    END;
END;
```

4）车厢出入库管理触发器

a）车厢验证入库触发器

```
CREATE TRIGGER tr_Inventory_ValidateInbound
ON [Inventory Table]
INSTEAD OF INSERT
AS
BEGIN
    SET NOCOUNT ON;

    -- Check if train exists and is marked as arrived
    IF EXISTS (
        SELECT 1 FROM inserted i
        LEFT JOIN [Train Arrival Table] ta ON i.TrainID = ta.TrainID
        WHERE ta.ID IS NULL OR ta.OnPosition <> '到位'
    )
    BEGIN
        RAISERROR('Cannot record inventory for train that has not arrived or does not exist.', 16, 1);
        ROLLBACK TRANSACTION;
        RETURN;
    END;

    -- Check for duplicate inventory records
    IF EXISTS (
```

```
        SELECT 1 FROM inserted i
        JOIN [Inventory Table] it ON i.TrainID = it.TrainID
        WHERE it.InTime > DATEADD(HOUR, -24, GETDATE())
    )
    BEGIN
        RAISERROR('This train already has a recent inventory record.', 16, 1);
        ROLLBACK TRANSACTION;
        RETURN;
    END;

    -- If all checks pass, proceed with insertion
    INSERT INTO [Inventory Table] (
        TrainID, TrainTypeID, WarehouseID, Reporter, InTime
    )
    SELECT
        TrainID, TrainTypeID, WarehouseID, Reporter, GETDATE()
    FROM inserted;

    -- Update warehouse inventory count
    UPDATE w
    SET w.InventoryCount = w.InventoryCount + 1
    FROM WarehouseStatus w
    JOIN inserted i ON w.WarehouseID = i.WarehouseID;
END;
```

b）车厢入库与出库关联触发器

```
CREATE TRIGGER tr_Delivery_LinkToInventory
ON [Delivery Table]
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;
```

```
        -- Mark corresponding inventory record as dispatched
        UPDATE i
        SET i.Status = '已出库',
            i.OutboundRef = d.ID
        FROM [Inventory Table] i
        JOIN inserted d ON i.TrainID = d.TrainID
        WHERE i.Status = '在库'
        AND i.WarehouseID = d.WarehouseID;

        -- Update warehouse inventory count
        UPDATE w
        SET w.InventoryCount = w.InventoryCount - 1
        FROM WarehouseStatus w
        JOIN inserted i ON w.WarehouseID = i.WarehouseID;

        -- Log the complete inventory cycle
        INSERT INTO InventoryCycleLog (
            TrainID, TrainTypeID, WarehouseID,
            InTime, OutTime, InReporter, OutReporter
        )
        SELECT
            i.TrainID, i.TrainTypeID, i.WarehouseID,
            i.InTime, d.OutTime, i.Reporter, d.Reporter
        FROM inserted d
        JOIN [Inventory Table] i ON d.TrainID = i.TrainID
        WHERE i.WarehouseID = d.WarehouseID;
    END;
```

5）下料作业触发器

a）下煤机状态变更触发器

```
    CREATE TRIGGER tr_Machine_StatusChange
```

```
ON [Machine Table]
AFTER UPDATE
AS
BEGIN
    SET NOCOUNT ON;

    -- Detect state changes
    IF UPDATE(OpenStatus) OR UPDATE(RunStatus)
    BEGIN
        -- Log all state changes
        INSERT INTO MachineStateLog (
            MaterialID, WarehouseID,
            Old_OpenStatus, New_OpenStatus,
            Old_RunStatus, New_RunStatus,
            ChangeTime, Operator
        )
        SELECT
            i.MaterialID, i.WarehouseID,
            d.OpenStatus, i.OpenStatus,
            d.RunStatus, i.RunStatus,
            GETDATE(), SYSTEM_USER
        FROM inserted i
        JOIN deleted d ON i.MaterialID = d.MaterialID
        WHERE d.OpenStatus <> i.OpenStatus OR d.RunStatus <> i.RunStatus;

        -- If machine was just started, create a new layoff record
        IF EXISTS (
            SELECT 1 FROM inserted i
            JOIN deleted d ON i.MaterialID = d.MaterialID
            WHERE d.RunStatus = '停止' AND i.RunStatus = '运行'
        )
        BEGIN
```

```
            INSERT INTO [Layoff History Table] (
                WarehouseID, MachineID, Time
            )
            SELECT WarehouseID, MaterialID, GETDATE()
            FROM inserted
            WHERE RunStatus = '运行';
        END;
    END;
END;
```

b）下料中断触发器

```
CREATE TRIGGER tr_LayoffBreak_HandleInterruption
ON [Layoff Break Table]
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;

    -- For each new interruption, stop the corresponding machine
    UPDATE m
    SET m.RunStatus = '停止',
        m.LastError = i.Reason
    FROM [Machine Table] m
    JOIN inserted i ON m.MaterialID = i.MachineID AND m.WarehouseID = i.WarehouseID;

    -- Notify maintenance team
    INSERT INTO MaintenanceTickets (
        MachineID, WarehouseID, IssueDescription,
        ReportedTime, Priority
    )
    SELECT
        MachineID, WarehouseID,
```

```
        '下料中断: ' + Reason,
        GETDATE(), 'HIGH'
    FROM inserted;

    -- Calculate downtime impact (assuming we have a production table)
    UPDATE p
    SET p.DowntimeMinutes = p.DowntimeMinutes + DATEDIFF(MINUTE, i.Time, GETDATE()),
        p.LastInterruption = i.Time
    FROM ProductionStats p
    JOIN inserted i ON p.WarehouseID = i.WarehouseID;
END;
```

6）综合业务触发器

a）车厢型号与出入库一致性检查触发器

```
CREATE TRIGGER tr_Consistency_TrainTypeCheck
ON [Inventory Table]
AFTER INSERT, UPDATE
AS
BEGIN
    SET NOCOUNT ON;

    -- Verify that the train type matches between arrival and inventory
    IF EXISTS (
        SELECT 1 FROM inserted i
        JOIN [Train Arrival Table] ta ON i.TrainID = ta.TrainID
        WHERE i.TrainTypeID <> ta.TrainTypeID
    )
    BEGIN
        RAISERROR('Inventory train type does not match arrival record.', 16, 1);
        ROLLBACK TRANSACTION;
        RETURN;
```

```
        END;
END;
```

5.4 辅助存储过程设计

5.4.1 功能设计与 SQL 代码

1）速度告警存储过程

```
CREATE PROCEDURE sp_HandleSpeedAlert
    @TrainID INT,
    @AlertType VARCHAR(50)
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @MaxSpeed FLOAT, @CurrentSpeed FLOAT;

    -- Get speed data
    SELECT @CurrentSpeed = Speed
    FROM [Train Speed Table]
    WHERE TrainID = @TrainID
    AND Time = (SELECT MAX(Time) FROM [Train Speed Table] WHERE TrainID = @TrainID);

    -- Get speed limit
    SELECT @MaxSpeed = MaxSpeed
    FROM [Train Model Table] tm
    JOIN [Train Arrival Table] ta ON tm.TrainTypeID = ta.TrainTypeID
    WHERE ta.TrainID = @TrainID;

    -- If still overspeeding after 1 minute, escalate
    IF @CurrentSpeed > @MaxSpeed
```

```
        BEGIN
            -- Log escalated alert
            INSERT INTO EscalatedAlerts (TrainID, AlertType, Speed, SpeedLimit, EscalationTime)
            VALUES (@TrainID, @AlertType, @CurrentSpeed, @MaxSpeed, GETDATE());

            -- Trigger emergency protocol
            UPDATE [Train Control Table]
            SET EmergencyStop = 'Y'
            WHERE TrainID = @TrainID;

            -- Notify authorities
            EXEC sp_SendEmergencyNotification @TrainID, '持续超速告警';
        END;
    END;
```

2）下料计算存储过程

```
    CREATE PROCEDURE sp_CalculateLayoff
        @MachineID VARCHAR(20),
        @DurationMinutes INT
    AS
    BEGIN
        SET NOCOUNT ON;

        DECLARE @Volume DECIMAL(18,2);
        DECLARE @Rate DECIMAL(18,2);
        DECLARE @MaterialDensity DECIMAL(18,2);

        -- Get machine capacity and current rate
        SELECT @Volume = Volume, @Rate = CurrentRate
        FROM [Machine Table]
        WHERE MaterialID = @MachineID;
```

```
    -- Get material density (assuming coal)
    SET @MaterialDensity = 0.8; -- Default coal density in tons/m³

    -- Calculate estimated layoff
    DECLARE @EstimatedWeight DECIMAL(18,2);
    SET @EstimatedWeight = @Volume * @Rate * @DurationMinutes * @MaterialDensity;

    -- Return calculation results
    SELECT
        @MachineID AS MachineID,
        @Volume AS MachineVolume,
        @Rate AS CurrentRate,
        @DurationMinutes AS DurationMinutes,
        @EstimatedWeight AS EstimatedWeight;

    -- Update machine stats
    UPDATE MachineStats
    SET TotalLayoff = TotalLayoff + @EstimatedWeight,
        LastCalculation = GETDATE()
    WHERE MachineID = @MachineID;
END;
```

5.5 数据库权限与角色设计

5.5.1 功能设计与 SQL 代码

1）数据库角色创建与基本权限分配

```
-- 创建数据库角色
USE [RailwayCoalManagement];
GO
```

```
-- 系统管理员角色 - 最高权限
CREATE ROLE [SystemAdministrator];
GRANT CONTROL ON DATABASE::[RailwayCoalManagement] TO [SystemAdministrator];
GRANT SELECT, INSERT, UPDATE, DELETE ON SCHEMA::dbo TO [SystemAdministrator];
GRANT EXECUTE ON SCHEMA::dbo TO [SystemAdministrator];

-- 系统控制员角色 - 下煤机管理权限
CREATE ROLE [SystemController];
GRANT SELECT, INSERT, UPDATE ON [Machine Table] TO [SystemController];
GRANT SELECT, INSERT ON [Layoff History Table] TO [SystemController];
GRANT SELECT, INSERT ON [Layoff Break Table] TO [SystemController];
GRANT EXECUTE ON [sp_CalculateLayoff] TO [SystemController];
GRANT EXECUTE ON [sp_HandleMachineOperation] TO [SystemController];

-- 仓库管理员角色 - 火车进出库管理权限
CREATE ROLE [WarehouseManager];
GRANT SELECT, INSERT, UPDATE ON [Train Arrival Table] TO [WarehouseManager];
GRANT SELECT, INSERT, UPDATE ON [Inventory Table] TO [WarehouseManager];
GRANT SELECT, INSERT, UPDATE ON [Delivery Table] TO [WarehouseManager];
GRANT SELECT ON [Train Model Table] TO [WarehouseManager];
GRANT EXECUTE ON [sp_VerifyTrainPosition] TO [WarehouseManager];

-- 现场人员角色 - 操作监控权限
CREATE ROLE [FieldOperator];
GRANT SELECT ON [Train Arrival Table] TO [FieldOperator];
GRANT SELECT ON [Train Speed Table] TO [FieldOperator];
GRANT SELECT ON [Warning Table] TO [FieldOperator];
GRANT SELECT, UPDATE ON [Machine Table] TO [FieldOperator];
GRANT EXECUTE ON [sp_ReportEmergency] TO [FieldOperator];
GRANT EXECUTE ON [sp_CheckPositionStatus] TO [FieldOperator];

-- 部门主管角色 - 查询与授权权限
```

```
CREATE ROLE [DepartmentHead];
GRANT SELECT ON SCHEMA::dbo TO [DepartmentHead];
GRANT EXECUTE ON [sp_GrantQueryAccess] TO [DepartmentHead];
GRANT EXECUTE ON [sp_GenerateOperationalReport] TO [DepartmentHead];
```

2）行级安全设计（基于角色的数据过滤）

```
-- 创建安全策略实现行级数据过滤
CREATE SCHEMA [Security];
GO

-- 安全谓词函数 - 限制现场人员只能看到自己仓库的数据
CREATE FUNCTION [Security].[fn_WarehouseAccessPredicate](@WarehouseID varchar(50))
RETURNS TABLE
WITH SCHEMABINDING
AS
RETURN SELECT 1 AS [result]
FROM [dbo].[UserWarehouseMapping]
WHERE
    [UserName] = USER_NAME() AND
    [WarehouseID] = @WarehouseID;
GO

-- 对火车到位表应用安全策略
CREATE SECURITY POLICY [Security].[TrainArrivalFilter]
ADD FILTER PREDICATE [Security].[fn_WarehouseAccessPredicate]([WarehouseID])
ON [dbo].[Train Arrival Table];

-- 对下煤机表应用安全策略
CREATE SECURITY POLICY [Security].[MachineAccessFilter]
ADD FILTER PREDICATE [Security].[fn_WarehouseAccessPredicate]([WarehouseID])
ON [dbo].[Machine Table];
```

6 其他

6.1 系统安全性设计

煤炭企业管理信息系统中的物料信息及其他工作人员信息如果被恶意利用可能会造成企业的重大损失，所以需要建立完善的安全保障体系，既能防止外部的非法破坏，也能阻止来自内部的蓄意攻击。系统的安全性设计应当包括以下五点：

（1） **标识与确认。**任何用户访问系统资源，必须得到系统的身份认证以及身份标识，比如用户账号及密码，数据证书等。只有当用户信息与登录信息一致是才可以访问系统。

（2） **授权。**对系统资源，包括程序、数据文件、数据库等，根据其特性分别划分保护等级。对不同的用户，规定不同访问资源权限，系统将根据用户特性，授予相应级别的系统资源访问权限。

（3） **日志。**为保护数据资源安全，在系统中进行的任何涉及重要数据的操作都需要做相应的记录，形成日志存档。

（4） **加密。**为保护数据安全，在系统中对网络中传输的信息必须经过高强度的加密来保证数据的安全性。

（5） **数据备份和恢复。**为预防系统因各种原因崩溃导致的数据丢失情况，必须进行数据备份。备份方式可以采用完全备份与增量备份相结合方式进行备份。系统故障时，要及时利用备份文件使系统恢复至最近的完整状态，并通知用户及时补输期间丢失的数据，直至恢复到

系统故障前的状态。

6.2 系统稳定性设计

稳定性保障通常来讲是指保障系统在运行、运维过程中,即时面对各种极端情况或突发事件仍然能够提供持续的、可靠的服务能力。各种极端情况或突发事件包括且并不局限于机房级故障,城市级故障,线上故障,线上业务量瞬时爆发,持续快速增长,系统服务器故障,依赖数据库故障,环境数据改变,依赖系统故障等。 系统要求持续的,可靠的服务能力，能够保障基本的功能使用。

系统稳定性原则有：简单、冗余、标准化等。

（1） **简单：**要求系统职责清晰，单一。

（2） **冗余：**系统需要进行冗余设计,包括数据冗余,数据多份备份,出现问题及时切换；网络、存储及其他基础设施冗余。

（3） **标准化：**可以规范船舶信息系统系统中各子系统之间的交互方式，便于管理提高效率。使各个部门对信息理解无偏差，数据流动顺畅，对服务约定明确，能力清晰，对接效率高。