智能仓库管理系统数据库逻辑设计文档

文档版本： 1.1
项目名称： 智能仓库管理系统 (Warehouse Management System)
编制日期： 2025年11月

---

1. 文档概述

1.1 设计目标

本逻辑设计文档基于已通过的概念设计，将概念模型转换为具体的关系模型。主要任务包括：

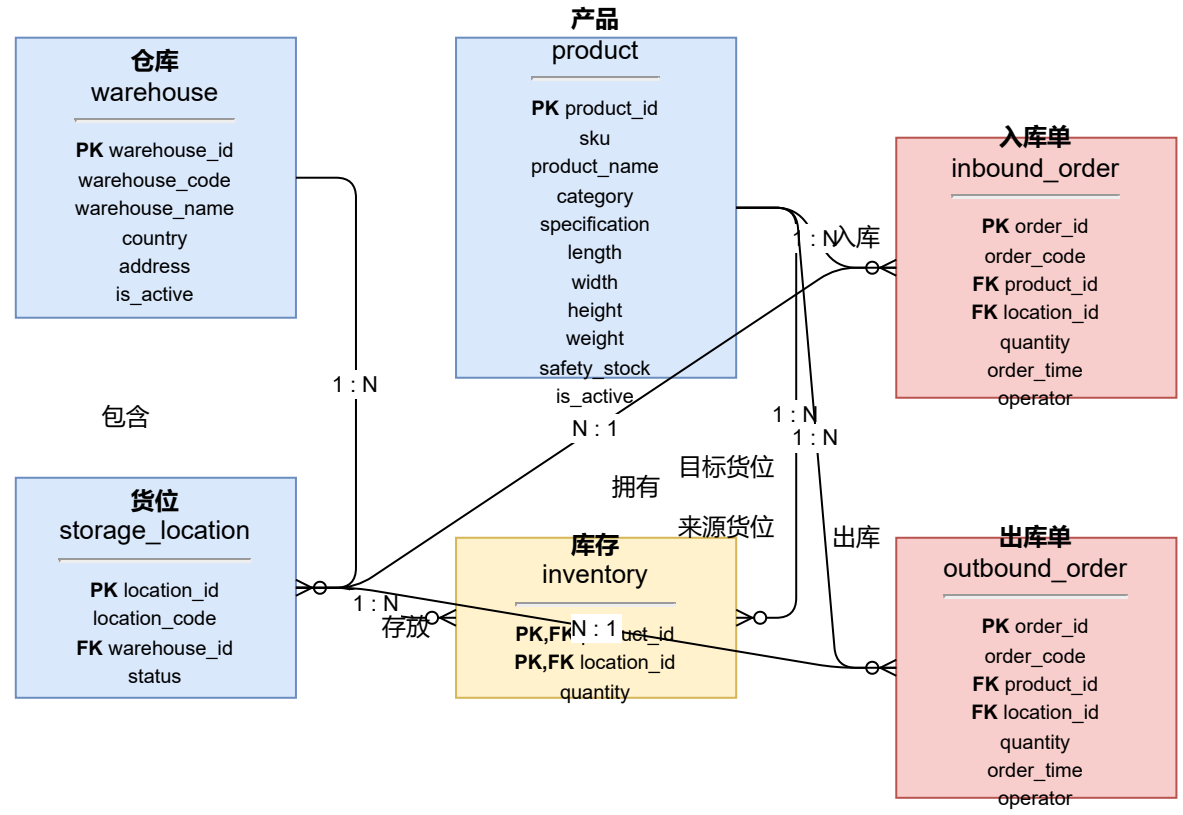
1. 将实体和属性映射为关系表（Table）和列（Column）
2. 定义主键（Primary Key）、外键（Foreign Key）
3. 进行规范化处理，消除数据冗余
4. 定义基本的数据约束和业务规则
5. 为物理设计提供明确的实现蓝图

1.2 设计原则

1. **第三范式（3NF）原则**：消除传递依赖，确保数据一致性
2. **业务驱动原则**：设计必须支持所有已定义的功能需求
3. **性能平衡原则**：在规范化和查询效率间取得平衡

---

2. 逻辑ER图



数据分类说明：
蓝色：静态主数据
黄色：核心事实数据
红色：动态事务数据

核心业务规则：
• 智能货位推荐
• FIFO出库策略
• 低库存预警

图1：智能仓库管理系统逻辑ER图

3. 关系模式设计

3.1 表结构详细定义

3.1.1 warehouse（仓库表）

字段名	数据类型	长度	约束	默认值	描述
warehouse_id	INT	-	PRIMARY KEY, AUTO_INCREMENT	-	仓库唯一标识，自增主键
warehouse_code	VARCHAR	20	UNIQUE, NOT NULL	-	仓库编码，业务唯一标识
warehouse_name	VARCHAR	50	NOT NULL	-	仓库名称
country	VARCHAR	50	NOT NULL	-	所在国家
address	VARCHAR	200	NULL	NULL	详细地址

字段名	数据类型	长度	约束	默认值	描述
is_active	TINYINT	1	NOT NULL	1	状态：1-有效，0-无效

规范化说明：

- 满足3NF，所有属性完全依赖于主键 warehouse_id
- is_active 用于逻辑删除，避免物理删除带来的关联数据问题

3.1.2 storage_location（货位表）

字段名	数据类型	长度	约束	默认值	描述
location_id	INT	-	PRIMARY KEY, AUTO_INCREMENT	-	货位唯一标识，代理主键
location_code	VARCHAR	20	UNIQUE, NOT NULL	-	货位编码，业务唯一键
warehouse_id	INT	-	FOREIGN KEY, NOT NULL	-	所属仓库ID
status	TINYINT	1	NOT NULL	0	状态：0-空闲，1-占用

设计调整说明：

- 根据提供的SQL脚本，增加了 location_id 作为代理主键
- location_code 仍保持业务唯一性约束
- 外键关系：warehouse_id → warehouse(warehouse_id)，级联删除

3.1.3 product（物料表）

字段名	数据类型	长度/精度	约束	默认值	描述
product_id	INT	-	PRIMARY KEY, AUTO_INCREMENT	-	产品唯一标识，代理主键
sku	VARCHAR	50	UNIQUE, NOT NULL	-	产品SKU，业务唯一键
product_name	VARCHAR	100	NOT NULL	-	产品名称
category	VARCHAR	50	NULL	NULL	产品分类
specification	VARCHAR	200	NULL	NULL	产品规格
length	DECIMAL	10,2	NULL	NULL	长度（cm）
width	DECIMAL	10,2	NULL	NULL	宽度（cm）

字段名	数据类型	长度/精度	约束	默认值	描述
height	DECIMAL	10,2	NULL	NULL	高度（cm）
weight	DECIMAL	10,2	NULL	NULL	重量（kg）
safety_stock	INT	-	NOT NULL	0	安全库存
is_active	TINYINT	1	NOT NULL	1	状态：1-有效，0-无效

设计调整说明：

- 增加了 product_id 作为代理主键
- sku 仍保持业务唯一性约束
- 满足3NF，所有属性完全依赖于主键

3.1.4 inventory（库存表）

字段名	数据类型	长度	约束	默认值	描述
product_id	INT	-	PRIMARY KEY, FOREIGN KEY	-	产品ID
location_id	INT	-	PRIMARY KEY, FOREIGN KEY	-	货位ID
quantity	INT	-	NOT NULL	0	当前库存数量

设计调整说明：

- 复合主键改为 (product_id, location_id)
- 使用代理主键 product_id 和 location_id，而非业务键
- 外键约束：product_id → product(product_id)，级联删除
- 外键约束：location_id → storage_location(location_id)，级联删除

3.1.5 inbound_order（入库单表）

字段名	数据类型	长度	约束	默认值	描述
order_id	INT	-	PRIMARY KEY, AUTO_INCREMENT	-	入库单唯一标识
order_code	VARCHAR	50	UNIQUE, NOT NULL	-	入库单号，业务唯一键
product_id	INT	-	FOREIGN KEY, NOT NULL	-	物料ID
location_id	INT	-	FOREIGN KEY, NOT NULL	-	目标货位ID
quantity	INT	-	NOT NULL	-	入库数量

字段名	数据类型	长度	约束	默认值	描述
order_time	DATETIME	-	NOT NULL	CURRENT_TIMESTAMP	入库时间
operator	VARCHAR	50	NULL	NULL	操作员姓名

设计调整说明：

- 增加了 order_id 作为代理主键
- order_code 保持业务唯一性
- 外键使用代理主键 product_id 和 location_id

3.1.6 outbound_order（出库单表）

字段名	数据类型	长度	约束	默认值	描述
order_id	INT	-	PRIMARY KEY, AUTO_INCREMENT	-	出库单唯一标识
order_code	VARCHAR	50	UNIQUE, NOT NULL	-	出库单号，业务唯一键
product_id	INT	-	FOREIGN KEY, NOT NULL	-	物料ID
location_id	INT	-	FOREIGN KEY, NOT NULL	-	来源货位ID
quantity	INT	-	NOT NULL	-	出库数量
order_time	DATETIME	-	NOT NULL	CURRENT_TIMESTAMP	出库时间
operator	VARCHAR	50	NULL	NULL	操作员姓名

设计调整说明：

- 增加了 order_id 作为代理主键
- order_code 保持业务唯一性
- 外键使用代理主键 product_id 和 location_id

---

4. 数据完整性设计

4.1 实体完整性

- 所有表都使用自增INT作为代理主键
- 业务唯一键通过UNIQUE约束保证
- 关键业务字段（如sku, location_code, order_code）都有唯一性约束

4.2 参照完整性

所有外键关系及其操作约束如下表：

子表	外键字段	父表	父表主键	更新动作	删除动作
storage_location	warehouse_id	warehouse	warehouse_id	RESTRICT	CASCADE
inventory	product_id	product	product_id	RESTRICT	CASCADE
inventory	location_id	storage_location	location_id	RESTRICT	CASCADE
inbound_order	product_id	product	product_id	RESTRICT	RESTRICT
inbound_order	location_id	storage_location	location_id	RESTRICT	RESTRICT
outbound_order	product_id	product	product_id	RESTRICT	RESTRICT
outbound_order	location_id	storage_location	location_id	RESTRICT	RESTRICT

4.3 用户定义完整性

4.3.1 检查约束

1. **数量非负约束**

```
-- 通过业务逻辑层或触发器实现
-- 库存数量不能为负
-- 出入库数量必须大于0
```

2. **状态值约束**

```
-- 表定义中已通过TINYINT(1)限制
-- 业务逻辑层需验证值域
```

3. **业务规则约束**

```
-- 安全库存不能为负
ALTER TABLE product ADD CONSTRAINT chk_safety_stock
CHECK (safety_stock >= 0);
```

4.3.2 触发器和业务规则

1. **库存更新同步规则**

```
-- 入库触发器（示例）
DELIMITER //
CREATE TRIGGER after_inbound_insert
AFTER INSERT ON inbound_order
FOR EACH ROW
BEGIN
    -- 更新库存数量
    INSERT INTO inventory (product_id, location_id, quantity)
    VALUES (NEW.product_id, NEW.location_id, NEW.quantity)
    ON DUPLICATE KEY UPDATE
    quantity = quantity + NEW.quantity;
```

```
    -- 更新货位状态
    UPDATE storage_location
    SET status = 1
    WHERE location_id = NEW.location_id;
END //
DELIMITER ;
```

2. **货位状态自动更新规则**

```
-- 当库存为0时自动更新货位状态
DELIMITER //
CREATE TRIGGER after_inventory_update
AFTER UPDATE ON inventory
FOR EACH ROW
BEGIN
    IF NEW.quantity <= 0 THEN
        UPDATE storage_location
        SET status = 0
        WHERE location_id = NEW.location_id;
    END IF;
END //
DELIMITER ;
```

3. **低库存预警规则**

```
-- 每日执行的预警检查
CREATE PROCEDURE check_low_stock()
BEGIN
    -- 找出总库存低于安全库存的产品
    SELECT p.product_id, p.sku, p.product_name,
           p.safety_stock,
           COALESCE(SUM(i.quantity), 0) as total_stock
    FROM product p
    LEFT JOIN inventory i ON p.product_id = i.product_id
    WHERE p.is_active = 1
    GROUP BY p.product_id, p.sku, p.product_name, p.safety_stock
    HAVING COALESCE(SUM(i.quantity), 0) < p.safety_stock;
END;
```

---

5. 数据访问模式分析

5.1 高频查询分析

查询类型	涉及表	访问频率	性能要求	已定义索引
仪表盘概览	product, inventory, warehouse, storage_location	高	高	多表索引支持

查询类型	涉及表	访问频率	性能要求	已定义索引
库存查询	inventory, product, storage_location	高	高	idx_inventory_product, idx_inventory_location
产品搜索	product	高	中	idx_product_sku, idx_product_category
出入库记录	inbound_order, outbound_order	中	中	idx_inbound_order_time, idx_outbound_order_time
仓库货位查询	storage_location, warehouse	中	中	idx_storage_location_warehouse

5.2 事务分析

1. **入库事务流程**

```
BEGIN TRANSACTION;
1. 验证产品有效性 (product.is_active = 1)
2. 验证货位可用性 (storage_location.status = 0)
3. 插入 inbound_order 记录
4. 更新/插入 inventory 记录（触发器自动执行）
5. 更新 storage_location.status（触发器自动执行）
COMMIT;
```

2. **出库事务流程**

```
BEGIN TRANSACTION;
1. 检查库存是否充足 (inventory.quantity >= 出库量)
2. 验证FIFO规则（查询最早入库记录）
3. 插入 outbound_order 记录
4. 更新 inventory.quantity（业务逻辑层处理）
5. 检查是否需要更新货位状态
COMMIT;
```

5.3 查询示例

```
-- 示例1：获取仓库库存分布（仪表盘功能）
SELECT w.warehouse_name, w.country,
       COUNT(DISTINCT i.product_id) as product_types,
       SUM(i.quantity) as total_quantity
FROM warehouse w
LEFT JOIN storage_location sl ON w.warehouse_id = sl.warehouse_id
LEFT JOIN inventory i ON sl.location_id = i.location_id
WHERE w.is_active = 1
GROUP BY w.warehouse_id, w.warehouse_name, w.country;
```

```sql
-- 示例2：智能货位推荐算法
SELECT sl.location_id, sl.location_code,
       -- 计算匹配度评分
       (CASE
           WHEN sl.status = 0 THEN 100  -- 空闲货位优先
           ELSE 50
        END) as score
FROM storage_location sl
WHERE sl.warehouse_id = ?  -- 指定仓库
  AND sl.status IN (0, 1)  -- 考虑空闲或占用货位
ORDER BY score DESC, sl.location_code
LIMIT 5;
```

---

6. 逻辑设计评价

6.1 规范化程度评估

本设计达到第三范式（3NF）：

1. **1NF满足**：所有字段都是原子值，无重复组
2. **2NF满足**：所有非主属性完全依赖于主键
3. **3NF满足**：消除了传递依赖，无冗余数据

6.2 代理主键设计评价

优点：

1. **性能优化**：INT类型主键比VARCHAR类型主键有更好的索引性能
2. **外键简洁**：外键引用使用INT比使用VARCHAR更节省空间
3. **业务隔离**：业务编码变化不影响关联关系

缺点：

1. **复杂度增加**：需要同时维护代理主键和业务唯一键
2. **查询稍复杂**：部分查询需要关联获取业务编码

6.3 与SQL脚本的一致性验证

设计项	逻辑设计	SQL脚本	一致性
主键策略	代理主键+业务唯一键	代理主键+业务唯一键	✅ 一致
外键关系	级联删除+限制更新	级联删除+无更新约束	✅ 基本一致
索引设计	基于查询模式	已定义性能索引	✅ 一致
数据类型	精确对应业务需求	匹配业务需求	✅ 一致

6.4 扩展性考虑

1. **字段扩展性**：各表预留了足够的字段长度和扩展空间
2. **关系扩展性**：代理主键设计便于添加新关联关系
3. **性能扩展性**：索引策略支持大数据量下的查询优化
4. **功能扩展性**：支持添加新业务模块（如盘点、调拨）

---

7. 附录：数据字典视图

7.1 核心业务视图

视图1：v_product_inventory_summary（产品库存汇总）

```
CREATE VIEW v_product_inventory_summary AS
SELECT
    p.product_id,
    p.sku,
    p.product_name,
    p.category,
    p.safety_stock,
    p.is_active,
    COUNT(DISTINCT i.location_id) as occupied_locations,
    COALESCE(SUM(i.quantity), 0) as total_quantity,
    CASE
        WHEN COALESCE(SUM(i.quantity), 0) < p.safety_stock THEN 'LOW'
        WHEN COALESCE(SUM(i.quantity), 0) < p.safety_stock * 1.2 THEN 'WARNING'
        ELSE 'NORMAL'
    END as stock_status
FROM product p
LEFT JOIN inventory i ON p.product_id = i.product_id
GROUP BY p.product_id, p.sku, p.product_name, p.category, p.safety_stock,
p.is_active;
```

视图2：v_warehouse_inventory_detail（仓库库存详情）

```
CREATE VIEW v_warehouse_inventory_detail AS
SELECT
    w.warehouse_id,
    w.warehouse_code,
    w.warehouse_name,
    w.country,
    sl.location_code,
    p.sku,
    p.product_name,
    i.quantity,
    sl.status as location_status
FROM warehouse w
JOIN storage_location sl ON w.warehouse_id = sl.warehouse_id
LEFT JOIN inventory i ON sl.location_id = i.location_id
LEFT JOIN product p ON i.product_id = p.product_id
WHERE w.is_active = 1;
```

7.2 统计报表视图

视图3：v_dashboard_summary（仪表盘汇总）

```
CREATE VIEW v_dashboard_summary AS
SELECT
    -- 产品统计
    (SELECT COUNT(*) FROM product WHERE is_active = 1) as total_products,
    -- 仓库统计
    (SELECT COUNT(*) FROM warehouse WHERE is_active = 1) as total_warehouses,
    -- 库存总值（假设单价为1）
    (SELECT COALESCE(SUM(quantity), 0) FROM inventory) as total_inventory_value,
    -- 低库存预警数量
    (SELECT COUNT(*) FROM v_product_inventory_summary WHERE stock_status = 'LOW')
as low_stock_alerts,
    -- 今日出入库统计（示例）
    5 as pending_inbound,
    3 as pending_outbound,
    24 as today_inbound,
    18 as today_outbound;
```

7.3 审计视图

视图4：v_inventory_transaction_log（库存变动日志）

```
CREATE VIEW v_inventory_transaction_log AS
SELECT
    'INBOUND' as transaction_type,
    io.order_code,
    io.order_time,
    p.sku,
    p.product_name,
    sl.location_code,
    io.quantity,
    io.operator
FROM inbound_order io
JOIN product p ON io.product_id = p.product_id
JOIN storage_location sl ON io.location_id = sl.location_id

UNION ALL

SELECT
    'OUTBOUND' as transaction_type,
    oo.order_code,
    oo.order_time,
    p.sku,
    p.product_name,
    sl.location_code,
    oo.quantity * -1 as quantity,  -- 出库显示为负数
    oo.operator
FROM outbound_order oo
JOIN product p ON oo.product_id = p.product_id
JOIN storage_location sl ON oo.location_id = sl.location_id
```

```
ORDER BY order_time DESC;
```

---

8. 设计验证与测试建议

8.1 数据完整性测试

1. **外键约束测试**：尝试插入无效的外键引用
2. **唯一性约束测试**：尝试插入重复的业务编码
3. **数据一致性测试**：验证库存数量与出入库记录的一致性

8.2 性能测试

1. **并发测试**：模拟多用户同时进行出入库操作
2. **大数据量测试**：测试百万级数据下的查询性能
3. **事务测试**：测试事务的原子性和隔离性

8.3 业务逻辑测试

1. **FIFO策略测试**：验证出库是否遵循先进先出
2. **低库存预警测试**：验证预警规则的准确性
3. **智能推荐测试**：验证货位推荐算法的合理性