附件 2

《数据库系统原理》大作业

系统实现报告

题目名称：　寝室出物管理系统

学号及姓名：__23371341 黎欣妍_____

_23371251 吴思齐_____

2025 年　12　月 30　日

一、数据库基本表的定义

room

字段名	数据类型	约束/默认	说明
room_id	INT	PK, AUTO_INCREMENT	寝室主键
building	VARCHAR(50)	NOT NULL	楼栋
floor	INT	NOT NULL	楼层
room_no	VARCHAR(20)	NOT NULL	寝室号
created_at	DATETIME	NOT NULL, DEFAULT CURRENT_TIMESTAMP	创建时间
updated_at	DATETIME	NOT NULL, DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP	更新时间

约束/索引	定义
UNIQUE	uniq_room(building, floor, room_no)
INDEX	idx_room(building, floor, room_no)

user

字段名	数据类型	约束/默认	说明
user_id	INT	PK, AUTO_INCREMENT	用户主键
username	VARCHAR(150)	NOT NULL	用户名
password	VARCHAR(128)	NOT NULL	密码（哈希/存储值）
email	VARCHAR(150)	NULL	邮箱
wechat	VARCHAR(30)	UNIQUE, NULL	微信号
student_id	VARCHAR(20)	UNIQUE, NULL	学号
role	TINYINT	NOT NULL, DEFAULT 0, CHECK(role IN (0,1))	角色字段（项目内约束）
user_role	TINYINT	NOT NULL, DEFAULT 1, CHECK(user_role IN (1,2,3))	用户身份（如普通/卖家/管理员等口径）
status	ENUM('正常','封禁')	NOT NULL, DEFAULT '正常'	账号状态
room_id	INT	NOT NULL, FK → room(room_id)	所属寝室
is_staff	BOOLEAN	DEFAULT FALSE	Django 兼容字段
is_superuser	BOOLEAN	DEFAULT FALSE	Django 兼容字段
created_at	DATETIME	NOT NULL, DEFAULT CURRENT_TIMESTAMP	创建时间
updated_at	DATETIME	NOT NULL, DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP	更新时间

约束/索引	定义
FK	room_id → room(room_id)
INDEX	idx_wechat(wechat)
INDEX	idx_room(room_id)
INDEX	idx_status(status)

tag

字段名	数据类型	约束/默认	说明
tag_id	INT	PK, AUTO_INCREMENT	标签主键
tag_name	VARCHAR(50)	NOT NULL, UNIQUE	标签名
ref_count	INT	NOT NULL, DEFAULT 0	引用次数/热度
created_at	DATETIME	NOT NULL, DEFAULT CURRENT_TIMESTAMP	创建时间
updated_at	DATETIME	NOT NULL, DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP	更新时间

posting

字段名	数据类型	约束/默认	说明
posting_id	INT	PK, AUTO_INCREMENT	帖子主键
title	VARCHAR(100)	NOT NULL	标题
content	TEXT	NULL	内容描述
price	DECIMAL(10,2) UNSIGNED	NOT NULL	价格
quantity	INT UNSIGNED	NOT NULL, DEFAULT 0	库存/数量
brand	VARCHAR(50)	NULL	品牌
image_url	VARCHAR(255)	NULL, COMMENT '封面图片URL(演示/展示用)'	封面图路径/URL
condition	ENUM('全新','几乎全新','轻微使用痕迹','空')	NULL	成色/新旧程度
tag_id	INT	NULL, FK → tag(tag_id)	标签
status	ENUM('上架','下架','已约满')	NOT NULL, DEFAULT '上架'	上下架与售罄状态
scope	ENUM('寝室','楼层','楼栋','全楼')	NOT NULL, DEFAULT '全楼'	可见范围
owner_id	INT	NOT NULL, FK → user(user_id)	发布者
created_at	DATETIME	NOT NULL, DEFAULT CURRENT_TIMESTAMP	创建时间
updated_at	DATETIME	NOT NULL, DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP	更新时间

约束/索引	定义
FK	tag_id → tag(tag_id)
FK	owner_id → user(user_id)
INDEX	idx_owner(owner_id)
INDEX	idx_tag(tag_id)
INDEX	idx_status_scope(status, scope)

favorite

字段名	数据类型	约束/默认	说明
f_id	INT	PK, AUTO_INCREMENT	收藏记录主键
user_id	INT	NOT NULL, FK → user(user_id)	收藏者
posting_id	INT	NOT NULL, FK → posting(posting_id)	被收藏帖子

字段名	数据类型	约束/默认	说明
created_at	DATETIME	NOT NULL, DEFAULT CURRENT_TIMESTAMP	收藏时间

约束/索引	定义
UNIQUE	uniq_favorite(user_id, posting_id)
FK	user_id → user(user_id)
FK	posting_id → posting(posting_id)
INDEX	idx_posting(posting_id)

order

字段名	数据类型	约束/默认	说明
order_id	INT	PK, AUTO_INCREMENT	订单主键
posting_id	INT	NOT NULL, FK → posting(posting_id)	对应帖子
buyer_id	INT	NOT NULL, FK → user(user_id)	买家
seller_id	INT	NOT NULL, FK → user(user_id)	卖家
num	INT	NOT NULL, CHECK(num > 0)	购买数量
status	ENUM('待交接','已交接','完成','取消')	NOT NULL, DEFAULT '待交接'	订单状态
created_at	DATETIME	NOT NULL, DEFAULT CURRENT_TIMESTAMP	创建时间
confirmed_at	DATETIME	NULL	确认交接时间
cancel_reason	VARCHAR(200)	NULL	取消原因
updated_at	DATETIME	NOT NULL, DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP	更新时间

约束/索引	定义
FK	posting_id → posting(posting_id)
FK	buyer_id → user(user_id)
FK	seller_id → user(user_id)
INDEX	idx_posting(posting_id)
INDEX	idx_buyer(buyer_id)
INDEX	idx_seller(seller_id)

notice

字段名	数据类型	约束/默认	说明
notice_id	INT	PK, AUTO_INCREMENT	通知主键
type	ENUM('系统','交接提醒','公告')	NOT NULL	通知类型
content	TEXT	NOT NULL	内容
receiver_id	INT	NOT NULL, FK → user(user_id)	接收者
related_order_id	INT	NULL, FK → order(order_id)	关联订单（可空）

字段名	数据类型	约束/默认	说明
status	ENUM('未读','已读')	NOT NULL, DEFAULT '未读'	阅读状态
created_at	DATETIME	NOT NULL, DEFAULT CURRENT_TIMESTAMP	创建时间
updated_at	DATETIME	NOT NULL, DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP	更新时间

约束/索引	定义
FK	receiver_id → user(user_id)
FK	related_order_id → order(order_id)
INDEX	idx_receiver(receiver_id)
INDEX	idx_related_order(related_order_id)

complaint

字段名	数据类型	约束/默认	说明
complaint_id	INT	PK, AUTO_INCREMENT	投诉主键
order_id	INT	NOT NULL, FK → order(order_id)	关联订单
complainant_id	INT	NOT NULL, FK → user(user_id)	投诉人
accused_id	INT	NOT NULL, FK → user(user_id)	被投诉人
content	TEXT	NOT NULL	投诉内容
status	ENUM('待处理','已处理','驳回','处理中')	NOT NULL, DEFAULT '待处理'	处理状态
result	VARCHAR(200)	NULL	处理结论
created_at	DATETIME	NOT NULL, DEFAULT CURRENT_TIMESTAMP	创建时间
handled_at	DATETIME	NULL	处理时间
updated_at	DATETIME	NOT NULL, DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP	更新时间

约束/索引	定义
FK	order_id → order(order_id)
FK	complainant_id → user(user_id)
FK	accused_id → user(user_id)
INDEX	idx_order(order_id)
INDEX	idx_users(complainant_id, accused_id)

image

字段名	数据类型	约束/默认	说明
image_id	INT	PK, AUTO_INCREMENT	图片主键
posting_id	INT	NULL, FK → posting(posting_id)	关联帖子（可空）
uploader_id	INT	NOT NULL, FK → user(user_id)	上传者
path	VARCHAR(255)	NOT NULL	图片路径

字段名	数据类型	约束/默认	说明
category	VARCHAR(50)	NULL	图片类别（如封面/详情）
created_at	DATETIME	NOT NULL, DEFAULT CURRENT_TIMESTAMP	上传时间

约束/索引	定义
FK	posting_id → posting(posting_id)
FK	uploader_id → user(user_id)

二、触发器与存储过程的设计与实现说明

2.1 触发器

本项目为寝室二手交易平台，核心业务围绕 **posting（帖子/库存）** 与 **order（订单/状态流转）** 展开。系统存在两类必须强一致的业务约束：

1. **订单卖家身份一致性约束**
   订单表 `order.seller_id` 必须等于对应帖子 `posting.owner_id` 。该约束属于**跨表业务规则**，单纯依靠外键无法表达。
2. **订单确认交接后库存扣减与帖状态联动**
   当订单状态从“待交接”推进为“已交接”时，必须对帖子库存 `posting.quantity` 扣减，并在库存耗尽时将帖子状态变更为“已约满”，以保证前端列表仅展示可交易帖子（ `posting_dao.get_posting_list()` 仅取 `status='上架'` ）。

因此，触发器被用于实现**数据库端的业务规则兜底**：把“不可表达的跨表规则”固化在触发器；把“状态流转带来的派生数据更新”自动化。

2.2.1 触发器总体策略

触发器名称	触发时机	作用对象	核心职责
`trg_order_check_seller`	`BEFORE INSERT ON order`	`order` 插入	校验 `seller_id` 与 `posting.owner_id` 一致；若 posting 不存在则显式报错
`trg_order_confirm`	`AFTER UPDATE ON order`	`order` 更新	仅在订单状态首次变为“已交接”时：扣减库存、库存归零则置帖为“已约满”

2.2.2 触发器一： `trg_order_check_seller` （下单时卖家校验）

- **触发事件**：对 `order` 表执行 `INSERT`
- **触发时机**： `BEFORE INSERT` （在落库前阻断不一致数据）
- **粒度**： `FOR EACH ROW` （逐行校验）

部分代码（来自 `triggers.sql` ）：

```
CREATE TRIGGER trg_order_check_seller
BEFORE INSERT ON `order`
FOR EACH ROW
BEGIN
    DECLARE real_seller INT;

    SELECT owner_id
    INTO real_seller
    FROM posting
    WHERE posting_id = NEW.posting_id;

    IF real_seller IS NULL THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'posting 不存在';
    END IF;

    IF NEW.seller_id <> real_seller THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = '订单 seller_id 与 posting.owner_id 不一致';
    END IF;
END
```

逻辑解释：

1. `DECLARE real_seller INT;`
   定义局部变量，用于承接被下单帖子真正的卖家（即帖子发布者）。
2. `SELECT owner_id INTO real_seller ...`
   依据 `NEW.posting_id` 查询 `posting.owner_id` 。
3. 若 `real_seller IS NULL`
   表示 posting 不存在，触发器用 `SIGNAL SQLSTATE '45000'` 主动抛出业务异常，并给出明确错误文本。
   注：虽然 `order.posting_id` 有外键，但触发器的意义在于提供更清晰的错误信息，同时在约束检查链路上更早失败。
4. 若 `NEW.seller_id <> real_seller`
   抛出“卖家不一致”异常，阻止插入。

执行效果：

- 存储过程正确实现时：触发器校验通过，不产生额外副作用；
- 存储过程/应用层未来变更或存在 bug 时：触发器阻断脏数据进入 `order` 。

典型测试：

1. **正常下单**： `seller_id = posting.owner_id` → 插入成功
2. **伪造卖家**： `seller_id != posting.owner_id` → 报错： `订单 seller_id 与 posting.owner_id 不一致`
3. **posting 不存在**： `posting_id` 无记录 → 报错： `posting 不存在`

2.2.3 触发器二： `trg_order_confirm` （交接后扣减库存与状态联动）

- **触发事件**：对 `order` 表执行 `UPDATE`
- **触发时机**： `AFTER UPDATE`
- **触发条件**：仅当状态**首次**变为 `已交接` ：

部分代码：

```
CREATE TRIGGER trg_order_confirm
AFTER UPDATE ON `order`
FOR EACH ROW
BEGIN
    IF NEW.status = '已交接'
       AND OLD.status <> '已交接' THEN

        UPDATE posting
        SET quantity = quantity - NEW.num
        WHERE posting_id = NEW.posting_id;

        UPDATE posting
        SET status = '已约满'
        WHERE posting_id = NEW.posting_id
          AND quantity <= 0;
    END IF;
END
```

逻辑解释：

1. **幂等控制（避免重复扣减）**
   通过 `NEW.status='已交接' AND OLD.status<>'已交接'` 限定“只在第一次进入已交接”时执行。
   若后续对该订单做其他字段更新（如 `updated_at` 自动更新），不会再次扣库存。
2. **扣减库存**
   `quantity = quantity - NEW.num`
   `NEW.num` 来源于订单数量字段（ `schema.sql` 中 `num INT NOT NULL CHECK (num > 0)` ）。
3. **库存耗尽自动改为已约满**
   第二条 `UPDATE posting ... AND quantity <= 0`
   在扣减后检查数量，若归零/负数则置帖为 `已约满` ，配合 `posting_dao.get_posting_list()` （仅查 `status='上架'` ）实现前端自动隐藏。

执行效果：

- 仅当订单“已交接”时才视为交易发生，此时扣减库存；
- 扣减后若库存归零（或小于等于 0），帖子状态置为 已约满，从“上架列表”中消失；
- 该触发器保证了“订单已交接 → 帖子库存减少”的联动一致性；

典型测试：

1. `待交接 → 已交接`：库存扣减，若归零则帖状态变 `已约满`
2. `已交接 → 完成`：不再扣库存（幂等条件阻止）
3. `待交接 → 取消`：不扣库存
4. 对已交接订单重复执行 `UPDATE`（例如修改 cancel_reason/无关字段）：不重复扣库存

2.2 存储过程

过程名	入参	结果	代码调用位置
`create_order_proc`	`(p_posting_id, p_buyer_id, p_quantity)`	创建订单、扣库存、通知	`create_order_by_proc()` / 下单视图
`confirm_order_proc`	`(p_order_id)`	卖家确认交接（待交接→已交接）、通知	`confirm_order_by_proc()` / 确认交接视图
`complete_order_proc`	`(p_order_id, p_user_id)`	买家完成订单（已交接→完成）、通知	`complete_order_by_proc()` / 完成视图
`cancel_order_proc`	`(p_order_id, p_reason, p_user_id)`	取消订单、回补库存、通知	`cancel_order_by_proc()` / 取消视图

三、系统各项功能数据库端操作主要代码与结果说明

3.1 DAO 总体架构与统一执行入口（`common/db.py`）

项目采用“**Raw SQL + DAO 封装**”模式，所有数据库访问统一经由 `common/db.py`：

- `query_one(sql, params, as_dict)`：执行 SELECT，返回**单行**；
- `query_all(sql, params, as_dict)`：执行 SELECT，返回**多行**；
- `execute(sql, params)`：执行写操作（INSERT/UPDATE/DELETE），无返回集，语义上“成功即 True”；
- `call_proc(proc_name, params, as_dict)`：调用存储过程 `CALL proc_name(...)`，如过程内部有 SELECT 会返回结果集，否则返回 None。

该设计的工程意义：DAO 层聚焦业务 SQL；上层 views 不直接拼 SQL；过程/触发器等数据库对象可在 DAO 层以“接口”形式被调用与复用。

3.2 功能一览图

模块	功能点	数据表/对象	DAO 入口（示例）	主要数据库操作
用户	注册	`user`	`user_dao.create_user()`	`INSERT INTO user(...) VALUES(...)`
用户	注册排重	`user`	`user_dao.check_user_exists()`	`SELECT COUNT(*) FROM user WHERE email/st`
用户	登录取用户	`user`	`user_dao.get_user_by_student_id()`	`SELECT user_id, username, password, stat`
用户	查看个人资料	`user`，`room`	`user_dao.get_profile()`	`SELECT u.*, r.floor, r.building FROM use`
用户	修改资料	`user`	`user_dao.update_profile()`	`UPDATE user SET email/wechat/room_id...`
用户	修改密码	`user`	`user_dao.update_password()`	`UPDATE user SET password=%s WHERE user_i`
管理员-用户	用户列表	`user`，`room`	`user_dao.list_users()`	`SELECT u..., r.floor, r.building FROM us`
管理员-用户	封禁/解封	`user`	`user_dao.ban_user()` / `unban_user()`	`UPDATE user SET status='封禁/正常' WHERE`

模块	功能点	数据表/对象	DAO 入口（示例）	主要数据库操作
帖子	发布帖子	posting	posting_dao.create_posting()	INSERT INTO posting(...) VALUES(..., sta
帖子	帖子列表（首页）	posting	posting_dao.get_posting_list()	SELECT ... FROM posting WHERE status='上
帖子	我的发布	posting	posting_dao.get_my_postings()	SELECT ... FROM posting WHERE owner_id=%
帖子	帖子详情	posting，user，tag	posting_dao.get_posting_detail()	SELECT p.*, u.username, t.tag_name FROM
帖子	修改帖子	posting	posting_dao.update_posting()	UPDATE posting SET ... WHERE posting_id=
帖子	下架/删除（软删）	posting	posting_dao.delete_posting()	UPDATE posting SET status='下架' WHERE p
帖子-图片	上传图片记录	image	posting_dao.add_image()	INSERT INTO image(posting_id,uploader_id
帖子-图片	查询图片	image	posting_dao.get_images()	SELECT image_id, path, category FROM ima
帖子-图片	删除图片	image	posting_dao.delete_image()	DELETE FROM image WHERE image_id=%s AND
收藏	添加收藏（幂等）	favorite	posting_dao.add_favorite()	INSERT ... ON DUPLICATE KEY UPDATE creat
收藏	取消收藏	favorite	posting_dao.remove_favorite()	DELETE FROM favorite WHERE user_id=%s AN
收藏	收藏列表	favorite，posting	posting_dao.get_favorites()	SELECT p... FROM posting p JOIN favorite
标签	标签列表/选择	tag	tag_dao.list_tags()	SELECT tag_id, tag_name FROM tag ...
搜索	关键词搜索	posting，tag	search_dao.search_postings()	WHERE p.status='上架' AND (title/content
搜索	标签筛选	posting	search_dao.search_postings()	AND p.tag_id=%s
搜索	可见性 scope 过滤	posting，user，room	search_dao._scope_filter_sql()	p.scope='全楼' OR (楼栋/楼层/寝室匹配)
搜索	用户主动范围过滤	posting，room	search_dao.search_postings()	在 scope 规则外叠加“只看同楼层/同寝室”
订单	下单（过程）	order + 过程	order_dao.create_order_by_proc()	CALL create_order_proc(posting_id,buyer_
订单	卖家订单列表	order，posting，user	order_dao.get_seller_orders()	SELECT o..., p.title, buyer.username ..
订单	买家订单列表	order，posting，user	order_dao.get_buyer_orders()	SELECT o..., p.title, seller.username .
订单	订单详情	order，posting，user	order_dao.get_order_detail()	多表 JOIN 返回订单+帖子+双方用户名
订单	卖家确认交接（过程）	order + 过程	order_dao.confirm_order_by_proc()	CALL confirm_order_proc(order_id, seller
订单	买家完成订单（过程）	order + 过程	order_dao.complete_order_by_proc()	CALL complete_order_proc(order_id,buyer_
订单	取消订单（过程）	order + 过程	order_dao.cancel_order_by_proc()	CALL cancel_order_proc(order_id,reason,u
投诉	发起投诉	complaint，order	complaint_dao.create_complaint()	SELECT seller_id FROM order... → INSER
投诉	防重复投诉	complaint	complaint_dao.check_duplicate()	SELECT COUNT(*) ... WHERE order_id AND c
投诉	我的投诉	complaint，order，posting，user	complaint_dao.get_my_complaints()	JOIN 查询帖子标题、被投诉用户名等

模块	功能点	数据表/对象	DAO 入口（示例）	主要数据库操作
管理员-投诉	投诉列表/筛选	complaint，user，posting	complaint_dao.admin_list()	JOIN + WHERE status=...
管理员-投诉	标记处理中	complaint	complaint_dao.mark_processing()	UPDATE complaint SET status='处理中' WHE
管理员-投诉	处理投诉/写结果	complaint	complaint_dao.handle()	UPDATE complaint SET status,result,hand
消息/通知	发送站内通知	notice	notice_dao.create_user_notice()	INSERT INTO notice(type,content,receive
消息/通知	未读数/未读列表	notice	notice_dao.get_unread()	SELECT ... WHERE receiver_id=%s AND sta
消息/通知	标记已读	notice	notice_dao.mark_read()	UPDATE notice SET status='已读' WHERE nc
公告	公告列表/详情	notice(type='公告')	notice_dao.list_announcements() / detail()	SELECT ... WHERE type='公告' ORDER BY cr
管理员-公告	发布/删除公告	notice	notice_dao.create_announcement() / delete()	INSERT type='公告'... / DELETE ...
统计	管理员总览 KPI	聚合查询	stats_dao.get_admin_overview()	多个 SELECT COUNT(*) + 比率计算
统计	用户个人画像	聚合查询	stats_dao.get_user_stats(user_id)	统计发布数、成交数、被投诉数、投诉成立
统计	空间维度统计	user，room + 聚合	stats_dao.get_room_stats() / get_floor_stats()	JOIN room 后按楼栋/楼层/寝室聚合
统计	趋势统计（按天）	order	stats_dao.get_daily_trend()	GROUP BY DATE(created_at)（近 N 天）

由于功能较多，以下就阐述2-3个比较能体现设计思想或者比较重点的功能模块。

3.3 订单状态流转：以数据库为中心的“状态机 + 原子事务闭环”

订单模块是一个明确的**有限状态机（FSM）**，核心状态通常为：

- 待交接 （下单后初态）
- 已交接 （卖家确认交接，视为成交事件）
- 完成 （买家确认完成，视为交易闭环完成）
- 取消 （任一方取消，终止流程）

在代码落实中实现“允许的状态迁移 + 必要校验 + 写入”：将复杂状态规则集中在数据库端，避免多处散落导致分支不一致；与事务天然绑定，能做到“状态改变与伴随更新要么都成功、要么都回滚”。

3.3.1 实现细节

①触发器实现成交事件的派生更新闭环（订单 → 库存 → 帖子可见性）

AFTER UPDATE ON order 触发器在 **订单首次进入** 已交接 时自动：

- posting.quantity -= order.num
- 若扣减后 quantity <= 0 ，则 posting.status = '已约满'

把**成交事件**在数据层物化：订单状态推进即触发库存扣减与资源可见性变化，帖子列表（只展示 status='上架' ）自然与库存一致，形成“成交—库存—展示”的强一致闭环。

②跨表业务约束（seller_id 与 posting.owner_id 一致）在数据库端硬约束化

BEFORE INSERT ON order 触发器校验 seller_id 必须等于对应 posting.owner_id ，下沉到数据库后可以防止：

- 订单归属错乱导致的权限/投诉/统计口径污染；

- 应用层遗漏校验造成的长期脏数据。

③幂等性设计
触发器通过比较 `OLD.status` 与 `NEW.status`，保证“只在首次进入已交接时扣库存”，避免重复 UPDATE 造成的二次扣减。

3.4 投诉功能：围绕“可追溯、可处理、可统计”的业务闭环设计

投诉模块是与订单强绑定的**治理链路**：

- 投诉对象可追溯：投诉必须关联 `order_id`，并能通过订单确定被投诉方；
- 投诉流程可推进：从提交到处理中到处理结论，形成可管理队列；
- 投诉数据可统计：为个人被投诉率、系统投诉率、处理率等指标提供数据源。

①投诉归责基于订单关系推导，而非前端随意传参
投诉表保存 `order_id`，并从订单记录推导 `accused_id`（被投诉者）。
投诉对象不由用户任意指定，而由**交易关系**确定，降低数据污染。

②防重复投诉与状态队列化
数据库端通过计数/状态约束（如同一 complainant 对同一 order 若仍处于待处理/处理中则禁止重复提交）实现“去噪”，使投诉数据具备真实的“事件”属性，而不是可刷量的“反馈”。

③管理员处理采用“状态推进 + 结果留痕”
处理不是简单改个字段，而是依旧使用状态机：

- `待处理 → 处理中`（通常带条件更新，防止并发多人同时接单）
- `处理中/待处理 → 已处理/驳回`，写入 `result`，并在存在字段时记录 `handled_by / handled_at`

3.5 统计功能：以统一口径的聚合查询驱动“指标卡 + 趋势图 + 风险画像”

统计模块把“交易、投诉、用户行为”转化为可解释指标，服务以下场景：

- 用户/卖家画像：成交数、被投诉率、信誉相关指标；
- 管理员总览：平台活跃度、成交趋势、投诉处理效率、风险集中区域等。

实现细节

①统计不依赖应用层缓存，直接基于业务真值表聚合
所有关键指标由数据库 `COUNT / GROUP BY / 条件过滤` 得出，避免“应用层维护冗余计数”带来的漂移风险。

②趋势页按天聚合，直接为图表提供数据序列

3.6 公告与消息模块：统一表设计 + 未读状态机 + 业务对象关联

统一的数据模型承载两类信息分发：

- 公告：面向全体的公共信息（首页最新公告/公告详情）
- 站内消息：面向具体用户的通知（消息中心、未读徽标）

实现细节

①统一表 + type 区分，避免重复建模
公告与通知共享一张表（或同一 notice 体系），通过 `type` 区分“公告/系统通知/订单通知”等。可以实现统一查询、统一展示、统一状态管理；未来新增消息类型无需新增表结构，只需扩展 type 与模板。

②通知与业务对象通过 `related_order_id` 关联，增强可追溯性
订单通知（取消、交接、完成等）不只是文本，而是能定位到具体订单。这为“从消息跳转到业务详情”提供数据库级支撑，也让消息成为可审计事件，而非孤立提醒。

③管理员广播能力（面向角色集合）
通过查询管理员用户集合（例如 `user_role=3`）并批量插入通知，实现“系统事件 → 管理员队列”的治理能力，为投诉、异常订单等后台运营场景预留扩展位。

四、组员大作业总结

同学1：黎欣妍，参与整体系统的设计，并主要承担数据端架构+应用端辅助补充。

- **任务内容概述**：主要负责数据库端核心业务链路的设计与落地实现，并编写存储过程与触发器，确保订单状态流转与库存扣减、帖子状态联动在数据库端形成强一致闭环。同时配合后端 DAO 层完成对存储过程的调用接口封装，保证应用层以统一方式访问数据库能力。
- **任务难点及解决方法**：我在实现的过程中遇到的难点一是订单状态流转带来的跨表一致性维护，例如“确认交接”后库存扣减与帖状态变更、以及避免重复更新导致的重复扣减。后面通过在数据库端引入基于 OLD/NEW 状态比较的触发器逻辑，实现仅在状态首次进入“已交接”时触发扣减，从而保证幂等性与可恢复性进行解决；同时还将“库存耗尽置约满”的规则固化在数据库端，保证前端列表展示与库存真值一致。
  另外一个一开始比较困惑的难点是外键无法表达的跨表业务约束（订单 seller 与 posting owner 的一致性）。后面我通过询问ai，使用 BEFORE INSERT 触发器进行硬约束校验，并通过 SIGNAL 提供可诊断的错误信息，避免脏数据落库。
- **任务完成结果**：完成了平台核心表结构与约束设计，建立了订单相关存储过程与触发器联动机制=；同时在 DAO 层提供可复用接口，支撑订单、帖子、投诉、通知等功能模块的稳定运行。
- **收获与体会**：通过本次大作业进一步理解了“将关键一致性规则下沉到数据库层”的工程意义，尤其体会到**触发器/存储过程在保障事务原子性、幂等性与数据可信度方面的优势**。

同学2：吴思齐，参与整体系统的设计，并主要承担应用端架构+数据端辅助补充。

- **任务内容概述**：主要负责应用层与功能模块的实现集成与展示闭环，围绕用户可见功能完成了后端视图层（views）与模板页面的开发与联调，包括帖子发布/管理、投诉提交与管理员处理页面、公告发布与消息中心展示（含未读徽标）、以及个人/管理员统计页面与趋势图展示等。同时负责将数据库端能力通过 DAO 层方法组织为可调用接口，保证前后端数据字段口径统一，并完成整体页面交互与权限控制（买家/卖家/管理员的操作差异）。
- **任务难点及解决方法**：我遇到的其中一个难点是多角色与多状态带来的页面逻辑复杂度，例如订单按钮的显示条件（卖家确认交接、买家完成订单、取消时双方提示）、投诉入口与处理状态流转等。解决方式是以订单状态机为主线梳理页面流程，明确每个状态下各角色允许的操作，并在 view 层集中计算可执行动作与展示文案，避免模板中出现大量分支。
  搜索模块的可见性控制（scope）与用户筛选条件叠加，既要保证“数据库硬规则不泄露”，又要提供“用户主动过滤”的体验也是一个比较大的难点。解决方式是将可见性规则固定在 DAO 的 SQL 过滤中作为硬约束，再在此基础上叠加范围筛选，从而保证安全性与一致性。
  难点三是统计数据口径一致性，尤其是“成交量”“投诉率”等指标需要与订单终态、投诉处理结果严格对应。这个需要两个人进行充分的交流协商，并且保证所有组员都对整个架构与实现的代码非常熟悉。
- **任务完成结果**：完成了项目主要功能页面与交互闭环，如用户端可实现注册登录、发布与浏览帖子、按范围搜索、下单并跟踪订单状态、系统总览统计与趋势展示等。整体实现保证数据展示字段与数据库口径一致，演示流程完整可复现。
- **收获与体会**：通过本次开发强化了对“以业务流程驱动系统设计”的理解，尤其是在多角色、多状态场景下需要先建立清晰的状态机与权限边界，再落到界面与接口实现。