v6_step_01_02_字段详解_代码详解

代码详细解释

文件 `v6_step_01.sql` 解析

1. **环境清理（步骤1）**
   - 删除所有相关表和视图（`CASCADE` 确保级联删除依赖对象），避免旧数据干扰新模拟运行。
2. **基础表结构（步骤2）**
   - `process` **表**：核心工序数据，字段包括工序名称（主键）、工种、初始生产率（单位工作量/人/天）。
   - `dependency` **表**：记录工序依赖关系（如“地板管道”需在“碎石基层”完成后才能开始），复合主键为前置和后续工序。
   - `task` **表**：存储任务量，包含扰动后的初始任务量（`initial_quantity`）和原始任务量（`og_initial_quantity`）。
   - `space` **表**：将每个工序分配到 1-5 层，每层工作量等于任务初始量。
3. **基础数据插入（步骤3）**
   - **工序数据**：插入 10 个工序，如 `Gravel base layer`（碎石基层）、`Floor tiling`（地板铺砖）等。
   - **任务数据**：基于原始任务量 ±10% 随机扰动生成 `initial_quantity`，并保留原始值 `og_initial_quantity`。
   - **依赖关系**：定义工序执行顺序（如地板管道需在碎石基层完成后开始）。
   - **空间数据**：通过交叉连接为每个工序生成 5 层数据，每层工作量等于任务初始量。
4. **资源与进度跟踪（步骤4）**
   - `trade_resource` **表**：记录工种可用工人数和状态（`available` 或 `busy`），默认每个工种 1 人。
   - `task_progress` **表**：跟踪各工序在各楼层的进度，包括剩余工作量、状态（`pending`/`in_progress`/`completed`）、开始/最后更新日等。
   - **初始化数据**：为每个工序的每个楼层创建初始进度记录，剩余量等于总工作量。
5. **任务更新函数（步骤5）**
   - `update_task_progress` **函数**：
     - 检查依赖是否完成（同一楼层的前置工序）。

- 检查低楼层是否完成（同一工序需按楼层顺序执行）。
- 分配可用工种资源，更新任务状态为 `in_progress`，并标记工种为 `busy`。

6. **每日记录表（步骤6）**
   - `daily_work_history`：记录每日各任务的计划剩余量、实际完成量、是否有效日。
   - `daily_task_record`：详细记录每日任务的扰动、返工、效率、完成日等信息。
7. **模拟执行逻辑（步骤8）**
   - 匿名代码块循环处理每一天的工作：
     - 插入当日状态到 `daily_work_history`。
     - 调用 `process_work_day` 函数处理当日任务：
       - 计算实际完成量（10% 概率为无效日，完成量为 0）。
       - 更新剩余工作量，释放已完成的工种资源。
       - 记录详细数据到 `daily_task_record`。
     - 持续运行直到所有任务完成或达到 365 天。
8. **结果查询（步骤9）**
   - **视图** `project_progress_details`：汇总项目进度，包括累计完成量、效率、任务状态等。
   - 最终查询 `task_progress` 和 `daily_task_record` 表。

文件 `v6_step_02.sql` 解析

1. **视图** `daily_task_details`
   - 通过 CTE `calculated_efficiency` 整合多个子查询，计算以下字段：
     - **任务状态**（`task_status`）：根据任务的开始日和完成日判断为 `pending`、`in_progress` 或 `completed`。
     - **实际效率**（`actual_efficiency`）：当日完成量（若超过计划剩余量则取剩余量）。
     - **剩余工作量**（`remain_work`）：计划剩余量减去累计完成量（确保不小于 0）。
     - **原始任务量**（`unchanged_quantity_of_work`）：从 `task` 表获取原始值。
   - **窗口函数**：计算总工作量（`total_work`）和当日工作量（`today_workload`）。
   - **返工标记**（`is_rework`）：当日完成量为 0 且任务已启动时标记为返工。
   - **无效日工作量**（`invalid_workload_today`）：若为无效日，记录当日无效工作量。

代码作用总结

1. `v6_step_01.sql`
   - 构建完整的项目管理模拟系统，涵盖工序定义、依赖关系、资源分配、任务跟踪和每日进度模拟。
   - 通过随机扰动模拟实际生产中的不确定性（如无效日、效率波动）。
   - 使用函数和匿名代码块实现自动化流程控制。
2. `v6_step_02.sql`
   - 创建高级视图 `daily_task_details`，提供每日任务的详细分析视图，包括状态、效率、返工等指标。
   - 支持后续数据分析和报告生成，便于监控项目进度和资源使用情况。

关键设计点

- **扰动与随机性**：任务量和每日效率均加入 ±10% 随机扰动，模拟实际生产中的不确定性。
- **楼层顺序执行**：同一工序需按楼层顺序执行（低楼层完成后才能处理高楼层）。
- **资源竞争**：同一工种同一时间只能处理一个任务，避免资源冲突。
- **无效日处理**：10% 概率无效日，当日工作量为 0，模拟意外停工。
- **数据追溯**：通过 `og_initial_quantity` 和 `daily_task_record` 表记录原始数据和每日详情，支持复盘分析。

SQL代码教学文档

概述

这段SQL代码是一个复杂的项目进度管理模拟系统，主要用于建筑施工项目的任务调度和进度跟踪。它通过创建多个表、插入基础数据、定义函数和执行模拟，来实现对施工任务的精细化管理。

代码结构与功能详解

步骤1：清理环境

在每次重新运行代码之前，需要清理之前创建的表和视图，以确保环境的干净。使用`DROP TABLE`和`DROP VIEW`语句，结合`IF EXISTS`条件，避免因对象不存在而导致的错误。

```SQL
DROP TABLE IF EXISTS
    daily_task_record, daily_work_history, task_progress,
trade_resource, space, task,
    dependency, process CASCADE;

DROP VIEW IF EXISTS current_project_status,
project_progress_details CASCADE;
```

步骤2：创建基础表结构

定义了多个表来存储项目相关的基础数据，包括工序信息、依赖关系、任务量、空间分配等。每个表都有明确的字段定义和约束，确保数据的完整性和一致性。

process表

存储工序的基本信息，包括工序名称、所属工种和初始生产率。

```SQL
CREATE TABLE process (
    process TEXT PRIMARY KEY,
    trade TEXT NOT NULL,
    initial_production_rate INT NOT NULL CHECK
(initial_production_rate > 0)
);
```

dependency表

记录工序间的依赖关系，确保施工顺序的正确性。

```SQL
CREATE TABLE dependency (
    predecessor_process TEXT REFERENCES process(process),
    successor_process TEXT REFERENCES process(process),
    PRIMARY KEY (predecessor_process, successor_process)
);
```

task表

存储每个工序的初始任务量，包括原始任务量和带±10%随机扰动后的任务量。

```SQL
CREATE TABLE task (
    process TEXT PRIMARY KEY REFERENCES process(process),
    initial_quantity INT NOT NULL CHECK (initial_quantity > 0),
    og_initial_quantity INT NOT NULL CHECK (og_initial_quantity >
0)
);
```

space表

将每个工序的任务量分配到不同的楼层，每个楼层的工作量等于任务的初始量。

```SQL
CREATE TABLE space (
    process TEXT REFERENCES task(process),
    floor INT CHECK (floor BETWEEN 1 AND 5),
    quantity INT NOT NULL,
    PRIMARY KEY (process, floor)
);
```

步骤3：插入基础数据

向各个表中插入基础数据，包括工序信息、任务量、依赖关系和空间分配等。这些数据是模拟项目进度的基础。

插入工序数据

```SQL
INSERT INTO process VALUES
('Gravel base layer', 'Gravel', 34),
('Pipes in the floor', 'Plumbing', 69),
('Electric conduits in the floor', 'Electricity', 51),
('Floor tiling', 'Tiling', 62),
('Partition phase 1', 'Partition', 55),
('Pipes in the wall', 'Plumbing', 37),
('Partition phase 2', 'Partition', 51),
('Electric conduits in the wall', 'Electricity', 41),
('Partition phase 3', 'Partition', 32),
('Wall tiling', 'Tiling', 27);
```

插入任务数据

```SQL
INSERT INTO task (process, initial_quantity, og_initial_quantity)
VALUES
('Gravel base layer', ROUND(170 * (0.9 + 0.2 * random()))::INT,
170),
('Pipes in the floor', ROUND(100 * (0.9 + 0.2 * random()))::INT,
100),
('Electric conduits in the floor', ROUND(80 * (0.9 + 0.2 *
random()))::INT, 80),
('Floor tiling', ROUND(720 * (0.9 + 0.2 * random()))::INT, 720),
('Partition phase 1', ROUND(750 * (0.9 + 0.2 * random()))::INT,
750),
('Pipes in the wall', ROUND(190 * (0.9 + 0.2 * random()))::INT,
190),
('Partition phase 2', ROUND(20 * (0.9 + 0.2 * random()))::INT, 20),
('Electric conduits in the wall', ROUND(180 * (0.9 + 0.2 *
random()))::INT, 180),
('Partition phase 3', ROUND(200 * (0.9 + 0.2 * random()))::INT,
200),
('Wall tiling', ROUND(290 * (0.9 + 0.2 * random()))::INT, 290);
```

插入依赖关系

```SQL
INSERT INTO dependency VALUES
('Gravel base layer', 'Pipes in the floor'),
('Gravel base layer', 'Electric conduits in the floor'),
('Pipes in the floor', 'Floor tiling'),
('Electric conduits in the floor', 'Floor tiling'),
('Partition phase 1', 'Pipes in the wall'),
('Pipes in the wall', 'Partition phase 2'),
('Partition phase 2', 'Electric conduits in the wall'),
('Electric conduits in the wall', 'Partition phase 3'),
('Partition phase 3', 'Wall tiling');
```

插入空间数据

```SQL
INSERT INTO space
SELECT
    t.process,
    f.floor,
    t.initial_quantity
FROM task t
CROSS JOIN generate_series(1, 5) AS f(floor);
```

步骤4：创建资源与任务进度跟踪表

定义了工种资源表和任务进度表，用于跟踪各工种的可用资源和每个工序在各楼层的进度。

trade_resource表

存储各工种的可用工人数和状态。

```SQL
CREATE TABLE trade_resource (
    trade TEXT PRIMARY KEY,
    available_workers INT DEFAULT 1 CHECK (available_workers >= 1),
    status TEXT DEFAULT 'available' CHECK (status IN
('available','busy'))
);
```

task_progress表

跟踪每个工序在各楼层的进度，包括剩余工作量、状态、开始时间等信息。

```SQL
CREATE TABLE task_progress (
    process TEXT REFERENCES process(process),
    floor INT,
    remaining_quantity INT CHECK (remaining_quantity >= 0),
    status TEXT DEFAULT 'pending' CHECK (status IN
('pending','in_progress','completed')),
    start_day INT,
    last_update_day INT,
    assigned_trade TEXT REFERENCES trade_resource(trade),
    planned_remaining INT,
    daily_productivity INT,
    PRIMARY KEY (process, floor)
);
```

步骤5：创建更新任务进度的函数

定义了一个函数`update_task_progress`，用于启动符合条件的新任务。它会检查依赖关系、低楼层的完成情况和工种的可用性，确保任务按正确的顺序和条件启动。

SQL

```sql
CREATE OR REPLACE FUNCTION update_task_progress(current_day INT)
RETURNS void AS $$
DECLARE
    task_record RECORD;
    assigned_trades TEXT[] := ARRAY[]::TEXT[];
BEGIN
    FOR task_record IN (
        SELECT
            tp.process,
            tp.floor,
            tp.assigned_trade
        FROM task_progress tp
        WHERE tp.status = 'pending'
          AND NOT EXISTS (
              SELECT 1
              FROM dependency d
              JOIN task_progress tp2 ON d.predecessor_process =
tp2.process
              WHERE d.successor_process = tp.process
                AND tp2.floor = tp.floor
                AND tp2.status != 'completed'
          )
          AND NOT EXISTS (
              SELECT 1
              FROM task_progress lower_tp
              WHERE lower_tp.process = tp.process
                AND lower_tp.floor < tp.floor
                AND lower_tp.status != 'completed'
          )
          AND EXISTS (
              SELECT 1
              FROM trade_resource tr
              WHERE tr.trade = tp.assigned_trade
                AND tr.status = 'available'
          )
        ORDER BY tp.floor, tp.process
    )
    LOOP
        IF NOT task_record.assigned_trade = ANY(assigned_trades)
THEN
            UPDATE task_progress
```

```
            SET
                status = 'in_progress',
                start_day = current_day,
                last_update_day = current_day
            WHERE process = task_record.process
              AND floor = task_record.floor;

            UPDATE trade_resource
            SET status = 'busy'
            WHERE trade = task_record.assigned_trade;

            assigned_trades := array_append(assigned_trades,
task_record.assigned_trade);
        END IF;
    END LOOP;
END;
$$ LANGUAGE plpgsql;
```

步骤6：创建每日记录表

定义了两个表用于记录每日的工作状态和详细信息，包括完成的工作量、效率、任务状态等。

daily_work_history表

记录每日工作状态的原始数据。

SQL

```
CREATE TABLE daily_work_history (
    day_number INT,
    floor INT,
    process TEXT,
    trade TEXT,
    planned_remaining INT,
    actual_done INT,
    is_valid BOOLEAN,
    recorded_at TIMESTAMPTZ DEFAULT NOW(),
    PRIMARY KEY (day_number, floor, process)
);
```

daily_task_record表

记录每日任务的详细信息，包括扰动、返工、效率等。

```SQL
CREATE TABLE daily_task_record (
    day_number INT,
    process TEXT,
    floor INT,
    trade TEXT,
    is_invalid BOOLEAN,
    is_rework BOOLEAN,
    daily_work_done INT,
    efficiency TEXT,
    start_day INT,
    complete_day INT,
    recorded_at TIMESTAMPTZ DEFAULT NOW(),
    PRIMARY KEY (day_number, process, floor)
);
```

步骤7：扩展处理每日工作的函数

定义了一个函数`process_work_day`，用于处理每日的工作，包括计算实际完成量、更新任务进度、记录详细信息等。

SQL

```sql
CREATE OR REPLACE FUNCTION process_work_day(current_day INT)
RETURNS void AS $$
DECLARE
    task_rec RECORD;
    work_done INT;
    is_invalid BOOLEAN;
    new_remaining INT;
    eff NUMERIC(10,2);
    is_rework BOOLEAN;
BEGIN
    FOR task_rec IN (
        SELECT
            tp.*,
            p.initial_production_rate,
            tr.available_workers
        FROM task_progress tp
        JOIN process p ON tp.process = p.process
        JOIN trade_resource tr ON tp.assigned_trade = tr.trade
        WHERE tp.status = 'in_progress'
    )
    LOOP
        is_invalid := (random() < 0.1);

        work_done := CASE
            WHEN is_invalid THEN 0
            ELSE GREATEST(1, ROUND(task_rec.initial_production_rate
* (0.9 + 0.2 * random())) * task_rec.available_workers)
        END;

        new_remaining := CASE
            WHEN task_rec.remaining_quantity <= work_done THEN 0
            ELSE task_rec.remaining_quantity - work_done
        END;

        UPDATE task_progress
        SET
            remaining_quantity = new_remaining,
            last_update_day = current_day,
            daily_productivity = work_done,
            status = CASE
                        WHEN new_remaining = 0 THEN 'completed'
```

```
                        ELSE 'in_progress'
                    END
        WHERE process = task_rec.process AND floor =
task_rec.floor;

        IF new_remaining = 0 THEN
            UPDATE trade_resource
            SET status = 'available'
            WHERE trade = task_rec.assigned_trade;
        END IF;

        is_rework := (task_rec.start_day IS NOT NULL AND
task_rec.start_day < current_day);

        IF (task_rec.initial_production_rate *
task_rec.available_workers) > 0 THEN
            eff := ROUND((work_done::NUMERIC /
(task_rec.initial_production_rate * task_rec.available_workers)) *
100, 2);
        ELSE
            eff := 0;
        END IF;

        INSERT INTO daily_task_record (
            day_number, process, floor, trade,
            is_invalid, is_rework, daily_work_done, efficiency,
            start_day, complete_day
        ) VALUES (
            current_day,
            task_rec.process,
            task_rec.floor,
            task_rec.assigned_trade,
            is_invalid,
            is_rework,
            work_done,
            eff::TEXT || '%',
            task_rec.start_day,
            CASE WHEN new_remaining = 0 THEN current_day ELSE NULL
END
        );
    END LOOP;
```

```
    PERFORM update_task_progress(current_day);
END;
$$ LANGUAGE plpgsql;
```

步骤8：执行模拟

使用匿名代码块来模拟每日的工作，直到所有任务完成或达到最大模拟天数（365天）。在模拟过程中，会记录每日的工作状态和详细信息。

```SQL
DO $$
DECLARE
    current_day INTEGER := 0;
    max_days INTEGER := 365;
BEGIN
    TRUNCATE TABLE daily_work_history;
    TRUNCATE TABLE daily_task_record;

    WHILE current_day < max_days LOOP
        INSERT INTO daily_work_history (
            day_number, floor, process, trade,
            planned_remaining, actual_done, is_valid
        )
        SELECT
            current_day,
            tp.floor,
            tp.process,
            tp.assigned_trade,
            tp.planned_remaining,
            COALESCE(tp.daily_productivity, 0),
            (COALESCE(tp.daily_productivity,0) > 0)
        FROM task_progress tp;

        PERFORM process_work_day(current_day);

        EXIT WHEN NOT EXISTS (
            SELECT 1 FROM task_progress WHERE status != 'completed'
        );
        current_day := current_day + 1;
    END LOOP;

    RAISE NOTICE 'Simulation completed in % days', current_day;
END $$;
```

步骤9：查看结果

创建了一个视图`project_progress_details`，用于汇总项目的进度详情，包括工种状态、累计完成量、效率等信息，方便用户查看和分析项目进度。

```SQL
CREATE OR REPLACE VIEW project_progress_details AS
SELECT
    dwh.day_number AS "Day",
    dwh.floor AS "Floor",
    dwh.process AS "Process",
    p.trade AS "Process Trade",
    r.status AS "Trade Status",
    t.initial_quantity AS "Total Work",
    dwh.planned_remaining AS "Plan Remaining",
    dwh.actual_done AS "Daily Done",
    (t.initial_quantity - dwh.planned_remaining) AS "Cumulative
Done",
    CASE WHEN dwh.is_valid THEN 'Valid' ELSE 'Invalid' END AS
"Validity",
    ROUND(dwh.actual_done::NUMERIC / p.initial_production_rate *
100) || '%' AS "Efficiency",
    tp.status AS "Task Status"
FROM daily_work_history dwh
JOIN task t USING (process)
JOIN process p USING (process)
JOIN task_progress tp ON dwh.process = tp.process AND dwh.floor =
tp.floor
JOIN trade_resource r ON tp.assigned_trade = r.trade;
```

最后，提供了查询语句来查看任务进度和每日详细记录。

```SQL
SELECT * FROM task_progress;

SELECT * FROM daily_task_record ORDER BY day_number, process,
floor;
```

使用说明

1. 将代码复制到SQL编辑器中，如pgAdmin或DBeaver。
2. 确保已连接到目标数据库。
3. 执行代码，开始模拟项目进度。
4. 模拟完成后，使用提供的查询语句查看结果。

注意事项

- 在实际使用前，建议备份数据库，以防数据丢失。
- 可根据实际需求调整模拟参数，如最大模拟天数、工种资源等。
- 代码中的随机扰动和无效日概率等设置，可根据具体情况进行修改。

SQL代码教学文档 - v6_step_02.sql

概述

这段SQL代码主要用于创建一个视图`daily_task_details`，该视图整合了每日任务的详细信息，包括任务状态、实际效率、总工作量、剩余工作量等。它通过复杂的查询和计算，提供了对项目进度的深入洞察。

代码结构与功能详解

1. 删除现有视图

首先，代码删除了已有的`daily_task_details`视图（如果存在），确保在创建新视图时不会出现冲突。

```SQL
DROP VIEW IF EXISTS daily_task_details;
```

2. 创建视图`daily_task_details`

2.1 使用公共表表达式（CTE）计算效率和其他指标

代码通过一个公共表表达式（CTE）`calculated_efficiency`来计算每个任务在每一天的效率、状态和其他相关指标。

SQL

```sql
WITH calculated_efficiency AS (
    SELECT
        dwh.day_number,
        dwh.floor,
        dwh.process,
        dwh.trade AS assigned_trade,
        dwh.planned_remaining,
        dwh.actual_done,
        dwh.is_valid,
        CASE
          WHEN (
                SELECT MIN(start_day)
                FROM daily_task_record dtr
                WHERE dtr.process = dwh.process
                  AND dtr.floor = dwh.floor
                  AND dtr.day_number <= dwh.day_number
               ) IS NULL
          THEN 'pending'
          WHEN (
                SELECT MIN(complete_day)
                FROM daily_task_record dtr
                WHERE dtr.process = dwh.process
                  AND dtr.floor = dwh.floor
                  AND dtr.complete_day IS NOT NULL
                  AND dtr.day_number <= dwh.day_number
               ) IS NULL
          THEN 'in_progress'
          ELSE 'completed'
        END AS task_status,
        dwh.recorded_at,
        (SELECT initial_production_rate FROM process WHERE process
= dwh.process) AS initial_production_rate,
        CASE
            WHEN (SELECT SUM(daily_work_done)
                  FROM daily_task_record dtr
                  WHERE dtr.process = dwh.process
                    AND dtr.floor = dwh.floor
                    AND dtr.day_number = dwh.day_number) >
dwh.planned_remaining
            THEN dwh.planned_remaining
            ELSE (SELECT SUM(daily_work_done)
```

```
                    FROM daily_task_record dtr
                    WHERE dtr.process = dwh.process
                      AND dtr.floor = dwh.floor
                      AND dtr.day_number = dwh.day_number)
        END AS actual_efficiency,
        CASE
            WHEN dwh.planned_remaining < (SELECT
initial_production_rate FROM process WHERE process = dwh.process)
            THEN (SELECT SUM(daily_work_done)
                  FROM daily_task_record dtr
                  WHERE dtr.process = dwh.process
                    AND dtr.floor = dwh.floor
                    AND dtr.day_number = dwh.day_number)
            ELSE (SELECT SUM(daily_work_done)
                  FROM daily_task_record dtr
                  WHERE dtr.process = dwh.process
                    AND dtr.floor = dwh.floor)
        END AS total_work_done,
        CASE
            WHEN dwh.planned_remaining -
                 COALESCE((SELECT SUM(daily_work_done)
                           FROM daily_task_record dtr
                           WHERE dtr.process = dwh.process
                             AND dtr.floor = dwh.floor
                             AND dtr.day_number <= dwh.day_number),
0) < 0
            THEN 0
            ELSE dwh.planned_remaining -
                 COALESCE((SELECT SUM(daily_work_done)
                           FROM daily_task_record dtr
                           WHERE dtr.process = dwh.process
                             AND dtr.floor = dwh.floor
                             AND dtr.day_number <= dwh.day_number),
0)
        END AS remain_work,
        (SELECT og_initial_quantity FROM task WHERE process =
dwh.process) AS unchanged_quantity_of_work
    FROM daily_work_history dwh
)
```

2.2 主查询：整合和扩展CTE中的数据

在主查询中，代码进一步处理CTE中的数据，计算总工作量、是否返工、无效工作量等，并最终选择所有列以形成视图。

SQL

```sql
SELECT
    calculated_efficiency.*,
    SUM(CASE WHEN actual_efficiency IS NOT NULL THEN
actual_efficiency ELSE 0 END) OVER (PARTITION BY floor, process) AS
total_work,
    CASE
        WHEN actual_efficiency = 0 THEN TRUE
        WHEN actual_efficiency IS NULL THEN NULL
        ELSE FALSE
    END AS is_rework,
    CASE
        WHEN actual_efficiency = 0 THEN
            (SELECT daily_productivity
             FROM task_progress tp
             WHERE tp.process = calculated_efficiency.process
               AND tp.floor = calculated_efficiency.floor
               AND tp.start_day <= calculated_efficiency.day_number
               AND tp.last_update_day >=
calculated_efficiency.day_number)
        ELSE 0
    END AS invalid_workload_today,
    CASE
        WHEN (calculated_efficiency.planned_remaining -
COALESCE((SELECT SUM(daily_work_done)
                                                FROM
daily_task_record dtr
                                                WHERE dtr.process
= calculated_efficiency.process
                                                  AND dtr.floor =
calculated_efficiency.floor
                                                  AND
dtr.day_number <= calculated_efficiency.day_number), 0)) < 0
        THEN GREATEST(0, calculated_efficiency.actual_efficiency)
        ELSE calculated_efficiency.actual_efficiency
    END AS updated_actual_efficiency,
    LAG(remain_work) OVER (PARTITION BY floor, process ORDER BY
day_number) - calculated_efficiency.remain_work AS today_workload

FROM calculated_efficiency
ORDER BY day_number, process, floor;
```

3. 查询视图数据

最后，代码通过`SELECT * FROM daily_task_details;`语句，将视图中的数据展示出来，方便用户查看和分析。

关键点解释

- **任务状态（task_status）**：通过查询`daily_task_record`表中的数据，确定每个任务在每一天的状态（pending、in_progress、completed）。
- **实际效率（actual_efficiency）**：计算每天实际完成的工作量，考虑了计划剩余工作量和已完成工作量。
- **总工作量（total_work）**：使用窗口函数`SUM()`，按楼层和工序分组，累加实际效率，得到总工作量。
- **是否返工（is_rework）**：判断当天是否有返工情况，即实际效率为0时标记为返工。
- **无效工作量（invalid_workload_today）**：当实际效率为0时，从`task_progress`表中获取当天的计划生产力，作为无效工作量。
- **更新后的实际效率（updated_actual_efficiency）**：确保实际效率不小于0，通过`GREATEST()`函数进行调整。
- **当天工作量（today_workload）**：使用`LAG()`函数计算前一天的剩余工作量与当天剩余工作量的差值，得到当天的工作量。

使用说明

1. 将代码复制到SQL编辑器中，如pgAdmin或DBeaver。
2. 确保已连接到目标数据库，并且数据库中已存在相关的表和数据（如`daily_work_history`、`daily_task_record`、`process`、`task`等）。
3. 执行代码，创建视图并查看结果。

注意事项

- 在执行代码前，建议备份数据库，以防数据丢失。
- 确保相关表中的数据完整且准确，否则可能影响视图的计算结果。
- 根据实际需求，可以对视图中的计算逻辑进行调整和优化。

SQL代码教学文档 - v6_step_02.sql

详细注释

1. 删除现有视图

```sql
-- 如果存在名为 daily_task_details 的视图，则将其删除
-- 这是为了避免在创建新视图时出现冲突
DROP VIEW IF EXISTS daily_task_details;
```

2. 创建视图`daily_task_details`

2.1 使用公共表表达式（CTE）计算效率和其他指标

SQL

```sql
-- 定义一个公共表表达式（CTE），用于计算每个任务在每一天的效率、状态和其他相关指标
WITH calculated_efficiency AS (
    -- 从 daily_work_history 表中选择数据
    SELECT
        dwh.day_number,          -- 模拟天数
        dwh.floor,               -- 楼层
        dwh.process,             -- 工序
        dwh.trade AS assigned_trade,  -- 分配的工种
        dwh.planned_remaining,   -- 计划剩余工作量
        dwh.actual_done,         -- 实际完成工作量
        dwh.is_valid,            -- 是否有效日
        -- 确定任务状态（pending、in_progress、completed）
        CASE
          -- 如果没有找到 start_day，则任务状态为 pending
          WHEN (
                SELECT MIN(start_day)
                FROM daily_task_record dtr
                WHERE dtr.process = dwh.process
                  AND dtr.floor = dwh.floor
                  AND dtr.day_number <= dwh.day_number
              ) IS NULL
          THEN 'pending'
          -- 如果没有找到 complete_day，则任务状态为 in_progress
          WHEN (
                SELECT MIN(complete_day)
                FROM daily_task_record dtr
                WHERE dtr.process = dwh.process
                  AND dtr.floor = dwh.floor
                  AND dtr.complete_day IS NOT NULL
                  AND dtr.day_number <= dwh.day_number
              ) IS NULL
          THEN 'in_progress'
          -- 否则任务状态为 completed
          ELSE 'completed'
        END AS task_status,
        dwh.recorded_at,         -- 记录时间
        -- 获取工序的初始生产率
```

```
        (SELECT initial_production_rate FROM process WHERE process
= dwh.process) AS initial_production_rate,
        -- 计算实际效率
        CASE
            -- 如果当天完成的工作量超过计划剩余工作量，则实际效率为计划
剩余工作量
            WHEN (SELECT SUM(daily_work_done)
                  FROM daily_task_record dtr
                  WHERE dtr.process = dwh.process
                    AND dtr.floor = dwh.floor
                    AND dtr.day_number = dwh.day_number) >
dwh.planned_remaining
            THEN dwh.planned_remaining
            -- 否则实际效率为当天完成的工作量
            ELSE (SELECT SUM(daily_work_done)
                  FROM daily_task_record dtr
                  WHERE dtr.process = dwh.process
                    AND dtr.floor = dwh.floor
                    AND dtr.day_number = dwh.day_number)
        END AS actual_efficiency,
        -- 计算总工作量
        CASE
            -- 如果计划剩余工作量小于初始生产率，则总工作量为当天完成的
工作量
            WHEN dwh.planned_remaining < (SELECT
initial_production_rate FROM process WHERE process = dwh.process)
            THEN (SELECT SUM(daily_work_done)
                  FROM daily_task_record dtr
                  WHERE dtr.process = dwh.process
                    AND dtr.floor = dwh.floor
                    AND dtr.day_number = dwh.day_number)
            -- 否则总工作量为该工序在该楼层的累计完成工作量
            ELSE (SELECT SUM(daily_work_done)
                  FROM daily_task_record dtr
                  WHERE dtr.process = dwh.process
                    AND dtr.floor = dwh.floor)
        END AS total_work_done,
        -- 计算剩余工作量
        CASE
            -- 如果计划剩余工作量减去累计完成工作量小于0，则剩余工作量为
0
            WHEN dwh.planned_remaining -
```

```
                COALESCE((SELECT SUM(daily_work_done)
                          FROM daily_task_record dtr
                          WHERE dtr.process = dwh.process
                            AND dtr.floor = dwh.floor
                            AND dtr.day_number <= dwh.day_number),
0) < 0
            THEN 0
            -- 否则剩余工作量为计划剩余工作量减去累计完成工作量
            ELSE dwh.planned_remaining -
                COALESCE((SELECT SUM(daily_work_done)
                          FROM daily_task_record dtr
                          WHERE dtr.process = dwh.process
                            AND dtr.floor = dwh.floor
                            AND dtr.day_number <= dwh.day_number),
0)
        END AS remain_work,
        -- 获取未改变的工作量（原始任务量）
        (SELECT og_initial_quantity FROM task WHERE process =
dwh.process) AS unchanged_quantity_of_work
    -- 从 daily_work_history 表中选择数据
    FROM daily_work_history dwh
)
```

2.2 主查询：整合和扩展CTE中的数据

SQL

```
-- 从 calculated_efficiency CTE 中选择数据
SELECT
    calculated_efficiency.*,
    -- 计算总工作量，按楼层和工序分组累加实际效率
    SUM(CASE WHEN actual_efficiency IS NOT NULL THEN
actual_efficiency ELSE 0 END) OVER (PARTITION BY floor, process) AS
total_work,
    -- 判断是否为返工
    CASE
        -- 如果实际效率为0，则标记为返工
        WHEN actual_efficiency = 0 THEN TRUE
        -- 如果实际效率为空，则标记为 NULL
        WHEN actual_efficiency IS NULL THEN NULL
        -- 否则标记为不返工
        ELSE FALSE
    END AS is_rework,
    -- 计算当天的无效工作量
    CASE
        -- 如果实际效率为0，则从 task_progress 表中获取当天的计划生产力
作为无效工作量
        WHEN actual_efficiency = 0 THEN
            (SELECT daily_productivity
             FROM task_progress tp
             WHERE tp.process = calculated_efficiency.process
               AND tp.floor = calculated_efficiency.floor
               AND tp.start_day <= calculated_efficiency.day_number
               AND tp.last_update_day >=
calculated_efficiency.day_number)
        -- 否则无效工作量为0
        ELSE 0
    END AS invalid_workload_today,
    -- 更新实际效率，确保不小于0
    CASE
        -- 如果计划剩余工作量减去累计完成工作量小于0，则实际效率为0
        WHEN (calculated_efficiency.planned_remaining -
COALESCE((SELECT SUM(daily_work_done)
                                                FROM
daily_task_record dtr
                                                WHERE dtr.process
```

```
= calculated_efficiency.process
                                                    AND dtr.floor =
calculated_efficiency.floor
                                                    AND
dtr.day_number <= calculated_efficiency.day_number), 0)) < 0
        THEN GREATEST(0, calculated_efficiency.actual_efficiency)
-- 确保不小于0
        -- 否则实际效率保持不变
        ELSE calculated_efficiency.actual_efficiency
    END AS updated_actual_efficiency,
    -- 计算当天的工作量
    -- 使用 LAG() 函数获取前一天的剩余工作量，与当天的剩余工作量相减得到
当天的工作量
    LAG(remain_work) OVER (PARTITION BY floor, process ORDER BY
day_number) - calculated_efficiency.remain_work AS today_workload

-- 从 calculated_efficiency CTE 中选择数据
FROM calculated_efficiency
-- 按天数、工序、楼层排序
ORDER BY day_number, process, floor;
```

3. 查询视图数据

SQL

```
-- 查询 daily_task_details 视图中的所有数据
SELECT * FROM daily_task_details;
```

总结

这段代码创建了一个视图`daily_task_details`，用于整合和展示每日任务的详细信息，包括任务状态、实际效率、总工作量、剩余工作量等。通过复杂的查询和计算，提供了对项目进度的全面洞察。

使用说明

1. 将代码复制到SQL编辑器中，如pgAdmin或DBeaver。
2. 确保已连接到目标数据库，并且数据库中已存在相关的表和数据（如`daily_work_history`、`daily_task_record`、`process`、`task`等）。
3. 执行代码，创建视图并查看结果。

注意事项

- 在执行代码前，建议备份数据库，以防数据丢失。

- 确保相关表中的数据完整且准确，否则可能影响视图的计算结果。
- 根据实际需求，可以对视图中的计算逻辑进行调整和优化。

数据解释_字段详解

`v6_数据解释.sql` 字段详解

---

1. `day_number`

- **含义**：模拟的日期编号，起始日为 `0`，表示模拟开始前的初始状态。实际有效数据从 `day_number = 1` 开始记录。
- **用途**：标记模拟的每一天，用于跟踪任务进度随时间的变化。

---

2. `floor`

- **含义**：任务所在的楼层编号，取值范围为 `1-5`。
- **用途**：区分同一工序在不同楼层的任务实例，确保按楼层顺序执行任务（低楼层优先）。

---

3. `process`

- **含义**：工序名称，对应 `process` 表中的工序（如 `Gravel base layer`、`Floor tiling` 等）。
- **用途**：标识当前记录关联的具体工序任务。

---

4. `assigned_trade`

- **含义**：分配给该任务的工种（如 `Plumbing`、`Electricity`）。
- **用途**：记录资源分配情况，确保同一工种不同时处理多个任务。

---

5. `planned_remaining`

- **含义**：当前任务的剩余计划工作量（经过 ±10% 随机扰动后的初始任务量）。
- **计算逻辑**：基于原始任务量（`unchanged_quantity_of_work`）生成，公式为：

```
planned_remaining = ROUND(og_initial_quantity * (0.9 + 0.2 *
random()))::INT
```

- **用途**：动态反映任务量因扰动产生的实际规划值。

---

6. `task_status`

- **含义**：任务当前状态，分为三种：
  - `pending`：未开始（依赖未完成或资源未分配）。
  - `in_progress`：进行中（资源已分配且正在执行）。
  - `completed`：已完成（剩余工作量为 0）。
- **状态流转**：
  `pending` → `in_progress`（依赖完成且资源可用） → `completed`（剩余量清零）。

---

7. `initial_production_rate`

- **含义**：工种的初始生产效率（单位：工作量/人/天），来自 `process` 表。
- **示例**：`Gravel` 工种的初始生产率为 `34`，表示每人每天可完成 34 单位工作量。
- **用途**：计算理论最大日产能（与扰动后的实际效率对比）。

---

8. `remain_work`

- **含义**：截至当天结束时，任务的剩余工作量。
- **计算逻辑**：

```
remain_work = GREATEST(0, planned_remaining - SUM(当日及之前完成量))
```

- **用途**：动态反映任务剩余量，用于判断任务是否完成。

---

9. `unchanged_quantity_of_work`

- **含义**：任务的原始工作量（未经 ±10% 扰动的初始值），直接来自 `task` 表的 `og_initial_quantity`。
- **用途**：追溯原始规划数据，对比扰动后的实际执行情况。

---

10. `is_rework`

- **含义**：标记当日是否发生返工。
  - `TRUE`：当日因无效日或任务重启导致需要返工。
  - `FALSE`：正常执行。
- **触发条件**：任务已启动（`start_day` 存在）但当日完成量为 0。

---

11. `invalid_workload_today`

- **含义**：当日因无效日（10% 概率）导致的工作量损失。
- **计算逻辑**：
  - 若为无效日（`is_invalid = TRUE`），则 `invalid_workload_today = 当日计划完成量`。
  - 否则为 `0`。
- **用途**：量化无效日对进度的负面影响。

---

12. `today_workload`

- **含义**：当日实际完成的有效工作量。

- **计算逻辑**：

```
today_workload = 前一日剩余量 - 当日剩余量
```

- **示例**：若 `remain_work` 从 100 变为 80，则 `today_workload = 20`。
- **用途**：直观展示每日进展。

字段关系与业务意义

- **扰动对比**：
  `unchanged_quantity_of_work`（原始量）→ `planned_remaining`（扰动后量），体现规划灵活性。
- **效率分析**：
  `initial_production_rate`（理论值）与 `today_workload`（实际值）对比，计算效率波动。
- **返工管理**：
  `is_rework` 和 `invalid_workload_today` 帮助识别资源浪费，优化调度策略。

总结

此文件定义了模拟系统中关键数据字段的含义和逻辑，涵盖任务规划、执行、监控全流程。通过分析这些字段，可评估项目进度、资源利用率及扰动影响，为项目管理提供数据支持。