Superset 在项企中的应用

AUTHOR: 彭玲 TIME: 2022/11/18

Superset 在项企中的应用

- 背景
- 数据库连接
- 数据集
  - 流程节点数统计
  - 流程处理时长统计
  - 部门流程处理时长统计
  - 用户流程处理时长统计
- 图表
  1. 流程节点数统计
  2. 流程处理时长（累计值）
  3. 流程处理时长（平均值）
  4. 部门-流程处理时长（累计值）
  5. 部门-流程处理时长（平均值）
  6. 用户-流程处理时长（累计值）
  7. 用户-流程处理时长（平均值）
- 看板
- 流程节点数统计数据表维护

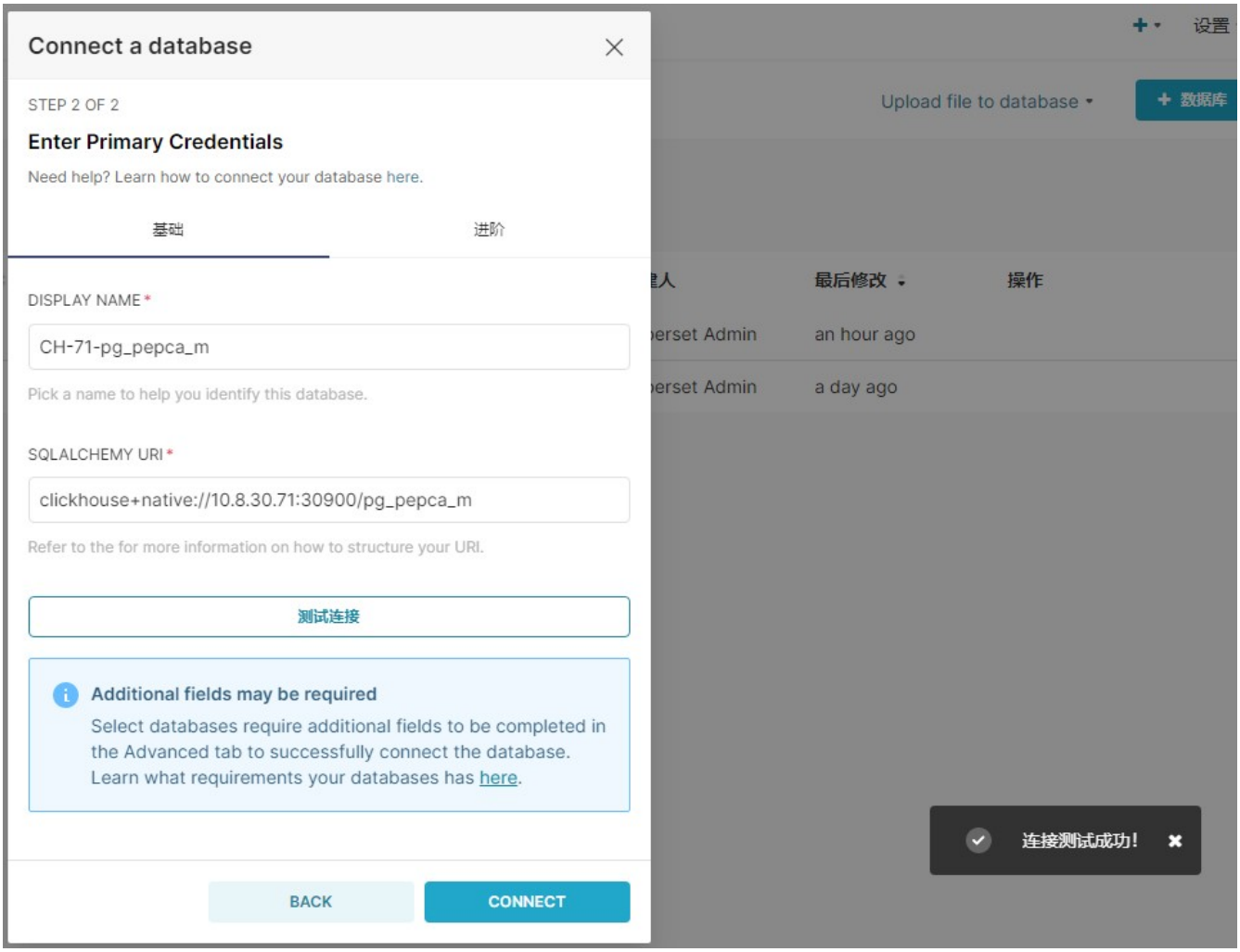
背景

预对项企流程的效率进行统计分析，比如，哪些流程特别长，哪些流程节点多，耗时多，哪些部门审核慢。并利用 Superset 工具进行可视化。

数据库连接

登录 Superset 网站，通过 `设置` 菜单进入`数据库连接`（Database Connections）页面，点击 `+ 数据库` 新增 ClickHouse 数据库：

1. 在 `SUPPORTED DATABASES` 下拉列表中选择 `ClickHouse`；
2. 给数据库起一个名字，并在 `SQLALCHEMY URI` 中输入 `clickhouse+native://10.8.30.71:30900/pg_pepca_m`；
3. 点击 `测试连接`，如果页面提示 `连接测试成功!`，说明配置 OK，点击 `CONNECT` 完成数据库连接配置。



数据集

数据集的创建有 2 种方式：

- 选择数据库表作为数据集
- 将 SQL 语句保存为数据集

根据实际需求，选择数据集的创建方式。其中，从 SQL 语句保存为数据集时，需要同步数据列，如下图所示。

项企流程效率统计中，将做如下几方面的统计：流程节点数统计、流程处理时长统计、部门流程处理时长统计、用户流程处理时长统计。

流程节点数统计

进入 `数据集` 菜单页，点击 `+ 数据集`，选择 `stats_process_nodes` 数据表即可。

流程处理时长统计

进入 SQL -> SQL 工具箱，创建 SQL 语句并保存为 dataset（点击保存右侧箭头 -> `Save dataset`）：

```sql
select wp.name as process_name,
       wpa.procinst_id as procinst_id,
       wpa.total_minutes as duration_in_minutes
from workflow_process_achievements as wpa
inner join workflow_process_history as wph on wpa.procinst_id=wph.procinst_id
inner join workflow_process_version as wpv on wph.version_id=wpv.id
inner join workflow_process as wp on wpv.process_id=wp.id;
```

部门流程处理时长统计

创建 SQL 语句并保存为 dataset：

```sql
select distinct d.id as department_id,
```

```
        d.name as department_name,
        u.id as user_id,
        u.name as user_name,
        wpa.procinst_id as procinst_id,
        wpa.task_id as taskinst_id,
        wpa.task_name as task_name,
        wpa.start_time as start_time,
        wpa.end_time as end_time,
        wpa.total_minutes as duration_in_minutes
from workflow_process_achievements as wpa
inner join user as u on wpa.deal_user_id=u.id
inner join department_user as du on u.id=du."user"
inner join department as d on du.department=d.id
where d.delete=0 and u.state=1 and u.delete=0;
```

用户流程处理时长统计

创建 SQL 语句并保存为 dataset：

```
select wpa.deal_user_id as user_id,
       u.name as user_name,
       wpa.procinst_id as procinst_id,
       wpa.total_minutes as duration_in_minutes
from workflow_process_achievements as wpa
inner join user u on wpa.deal_user_id = u.id
where u.state=1 and u.delete=0;
```

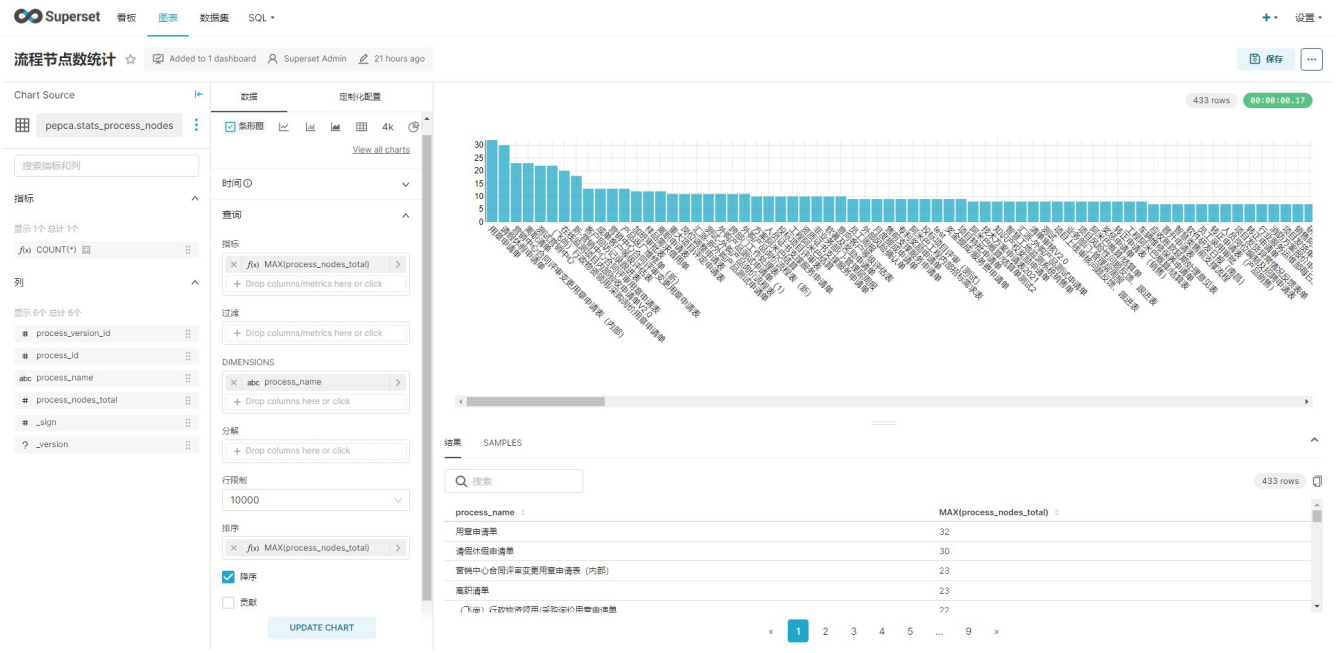
图表

准备好数据集后，即可使用数据集创建需要的图表。

1. 流程节点数统计

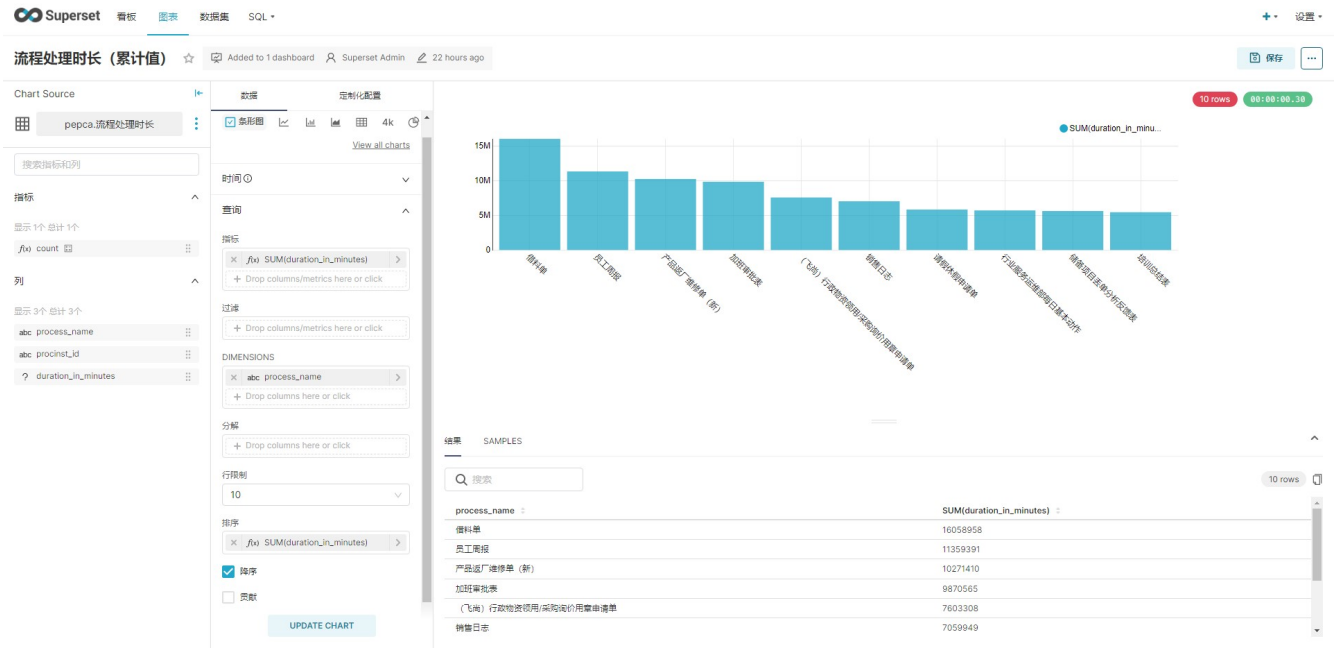
流程节点数统计：统计各流程配置中 “用户任务” 的节点总数。

流程节点数统计，数据源选择 `stats_process_nodes` 数据集，图表类型为 `条形图（Bar）`，具体如下图所示：



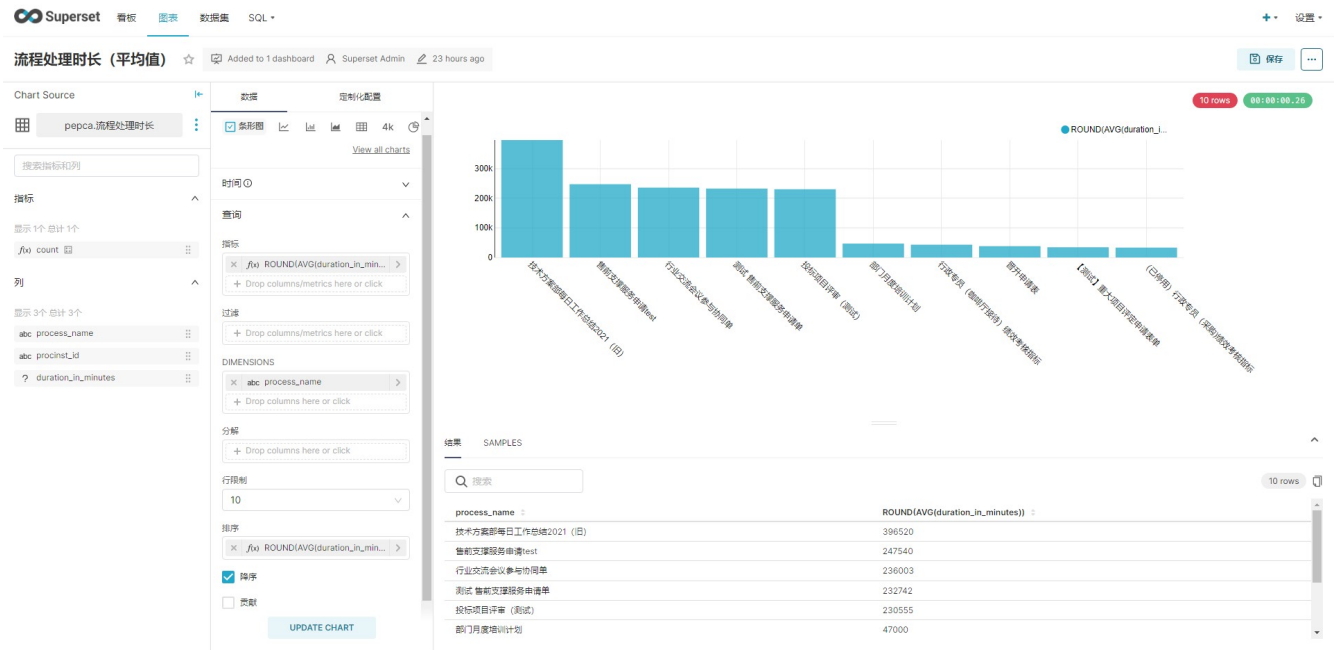
2. 流程处理时长（累计值）

流程处理时长（累计值）：按照流程名称，统计流程处理的总共用时（单位：分钟），默认展示最耗时的 10 条记录，如，所有“员工周报”的处理用时总共 11192017 分钟。



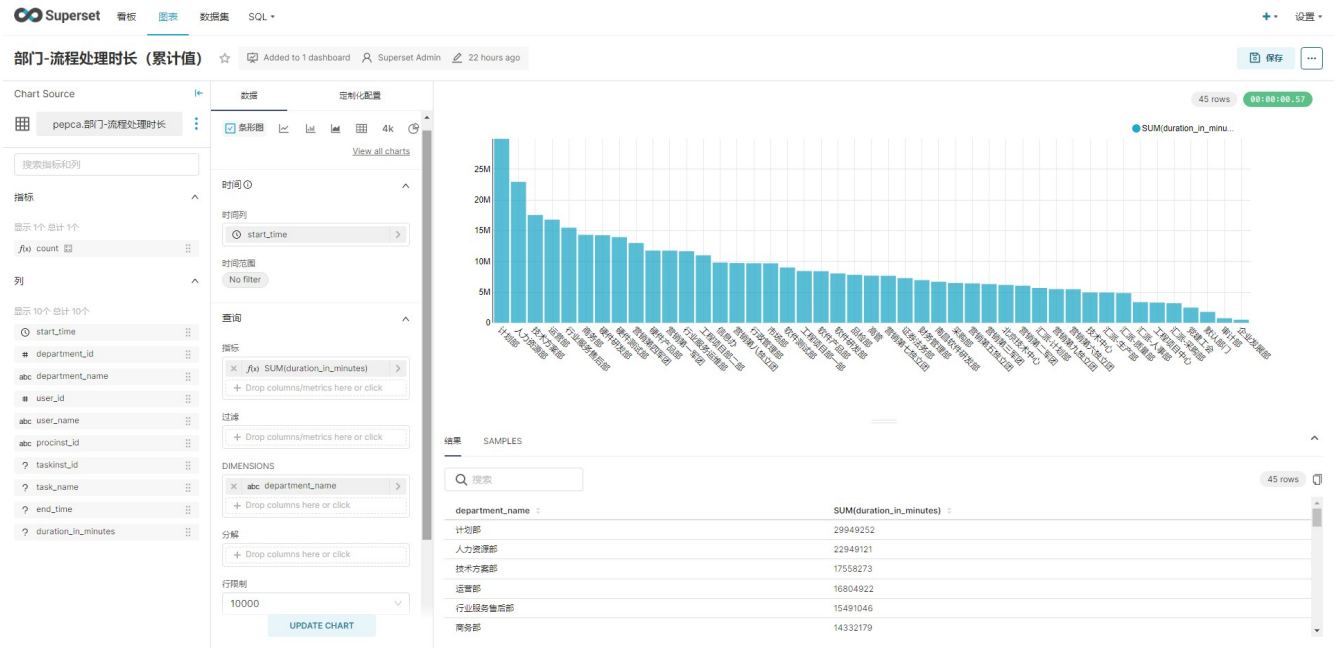
3. 流程处理时长（平均值）

流程处理时长（平均值）：按照流程名称，统计流程任务处理的平均用时（单位：分钟），默认展示最耗时的 10 条记录，如，“技术方案部每日工作总结2021（旧）”各节点平均处理用时 396520 分钟。



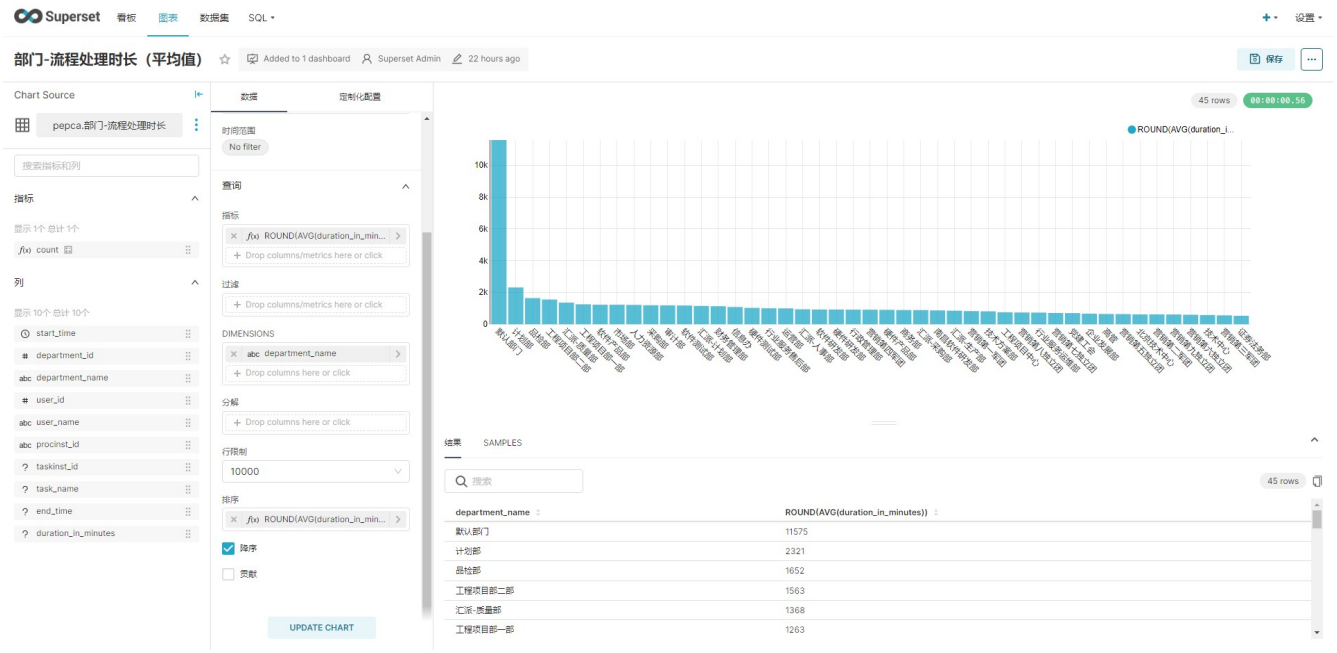
4. 部门-流程处理时长（累计值）

部门-流程处理时长（累计值）：按照部门，统计流程处理的总共用时（单位：分钟），如，“计划部” 处理流程上，总共用了23554603 分钟。



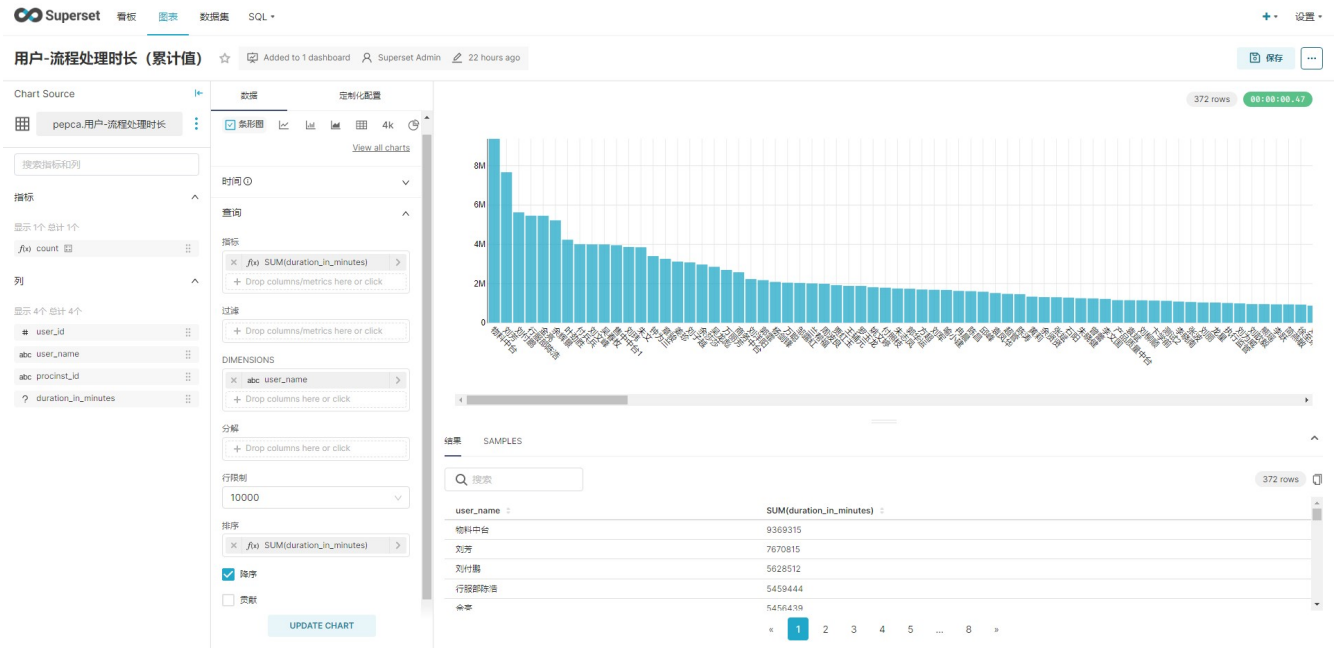
5. 部门-流程处理时长（平均值）

部门-流程处理时长（平均值）：按照部门，统计流程任务处理的平均用时（单位：分钟），如，“计划部” 处理流程上，各节点平均用时 1840 分钟。



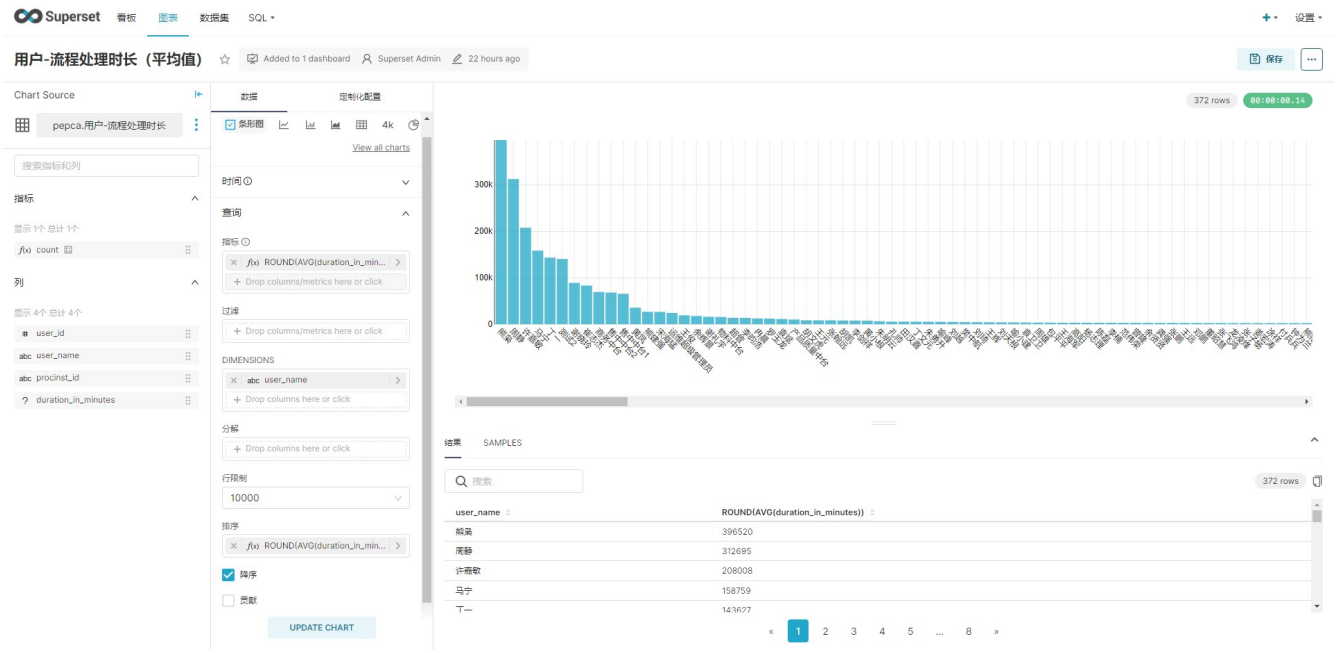
6. 用户-流程处理时长（累计值）

用户-流程处理时长（累计值）：按用户统计，流程处理总用时（单位：分钟），如，“刘芳“ 处理流程总共用了 7670815 分钟。



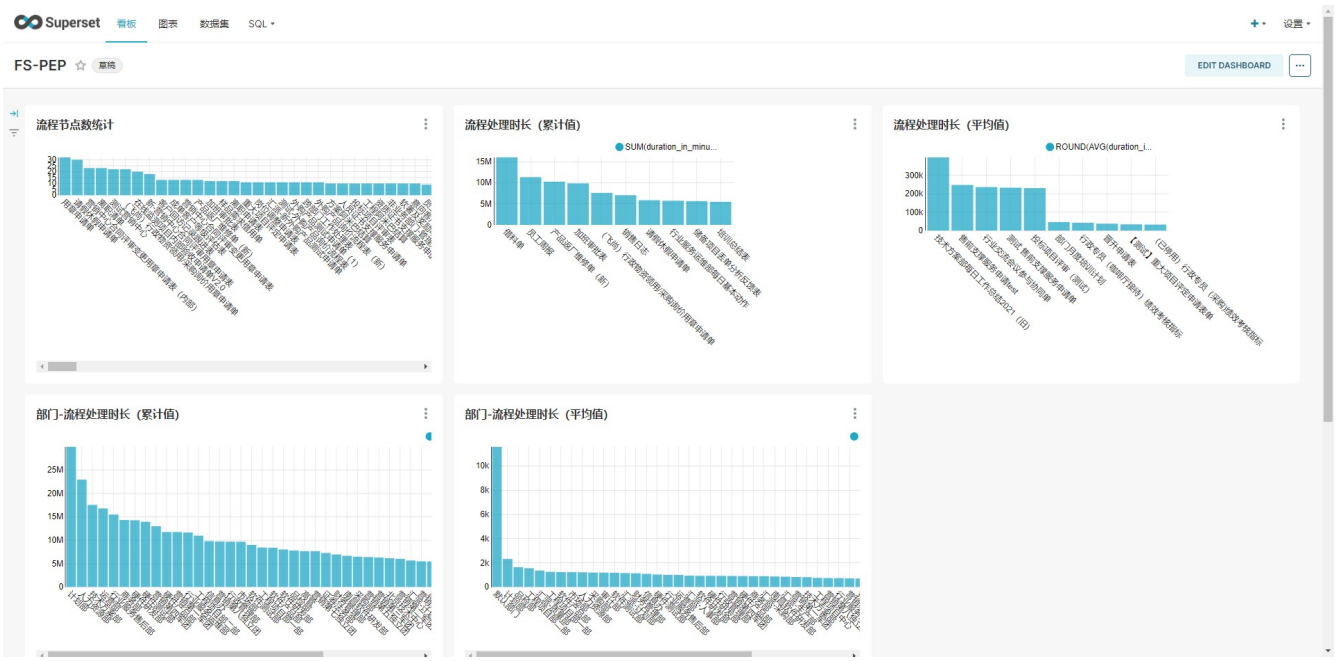
7. 用户-流程处理时长（平均值）

用户-流程处理时长（平均值）：按用户统计，流程任务处理平均用时（单位：分钟），如，“周静“ 各节点处理平均用时 312695 分钟。



看板

为了便于对图表进行组织管理和快速查看，将以上图表添加到同一个看板中。创建看板 `FS-PEP`，图表可以在保存的时候选择该看板，也可以在看板中将图表拖拽到看板区域。



流程节点数统计数据表维护

流程节点数统计数据表 `stats_process_nodes` 通过以下代码维护（关键代码）：

```
/**
 * Created by Julin on 2022/11/07.
```

```
 */
'use strict';

module.exports = async function () {
    try {
        // 1. delete all rows in the table
        await clearStatsProcessNodes();
        // 2. insert data into the table
        let processes = await process.clickhouse['pg_pepca_m'].query(`
            select v.id as version_id,
                v.process_id as process_id,
                p.name as process_name,
                v.bpmn_json
            from workflow_process_version as v
            inner join workflow_process as p on v.process_id=p.id
            where v.current=true and p.is_enable=true and p.deleted=false
            order by v.id desc
        `).toPromise();
        let dataToDB = processes.reduce((p, c) => {
            let nodes = JSON.parse(c.bpmn_json);
            let taskNodesCount = 0;
            for (let key in nodes) {
                if (nodes[key].type == 'bpmn:UserTask') taskNodesCount++;
            }
            p.push({
                processVersionId: c.version_id,
                processId: c.process_id,
                processName: c.process_name,
                processNodesTotal: taskNodesCount
            });
            return p;
        }, []);
        await storageStatsProcessNodes(dataToDB);
    } catch (err) {
        process.logger.error('Something error in function [statProcessNodes]:', err);
    }
};

async function clearStatsProcessNodes() {
    const transaction = await process.postgres.orm.transaction();
    const models = process.postgres.models;
    const { Op } = process.postgres.ORM;
    try {
        await models.StatsProcessNodes.destroy({
            where: { processVersionId: { [Op.gt]: 0 } },
            transaction
        });
        await transaction.commit();
    } catch (err) {
        await transaction.rollback();
        process.logger.error('Destroy data from Postgres DB [stats_process_nodes]
error:', err);
    }
```

```
};

async function storageStatsProcessNodes(data) {
    const transaction = await process.postgres.orm.transaction();
    const models = process.postgres.models;
    try {
        await models.StatsProcessNodes.bulkCreate(data, { transaction });
        await transaction.commit();
        process.logger.info('Sync data to Postgres DB [stats_process_nodes]');
    } catch (err) {
        await transaction.rollback();
        process.logger.error('Storage data to Postgres DB [stats_process_nodes]
error:', err);
    }
};
```