E-Transport Analytics Documentation

Introduction

This document provides a comprehensive perspective on the role of Superset in E-Transport Analytics. It covers various aspects, including the project's scope, the rationale behind dashboard creation, specifics of the queries utilized, visualization formats adopted, the metrics and filters employed, essential insights derived, and the administration of access via Superset's roles and permissions.

Project Overview

E-Transport Analytics is a strategic initiative aimed at visualizing and analyzing transportation-related issues recorded in the Redmine platform. The project leverages Apache Superset to create intuitive dashboards and charts, facilitating insightful decision-making based on real-time data. This document serves as a comprehensive guide to the charts and queries created in Superset for this project.

List of tools and technologies

Redmine, Superset, Keycloak, PostgreSQL:

The integrated toolset consists of Redmine, Superset, Keycloak, and PostgreSQL. Each tool serves a specific purpose within the system, collectively enhancing functionality and user experience. It's noteworthy that both Redmine and Superset are individually linked to the database, and a unified Single Sign-On (SSO) mechanism has been established through Keycloak for seamless access.

Definition of tools

- **Redmine:** Redmine serves as the issue management platform, enabling efficient tracking and handling of transportation-related issue-tickets. It centralizes issue reporting, assignment, and resolution.
- **Superset:** Superset functions as a powerful data visualization tool, providing interactive and insightful dashboards and charts. It aids in transforming raw data into comprehensible visual representations for informed decision-making.
- **Keycloak:** Keycloak plays a pivotal role in identity and access management. It facilitates a single sign-on (SSO) solution, ensuring that users can access both Redmine and Superset with a single set of credentials, enhancing user convenience and security.
- **PostgreSQL:** PostgreSQL serves as the underlying database management system, storing and organizing the data used by Redmine and Superset. It provides a reliable foundation for data storage and retrieval.

By seamlessly integrating Redmine, Superset, Keycloak, and PostgreSQL, the toolset offers an efficient and comprehensive solution for managing transportation-related issues, visualizing data, streamlining user access, and maintaining a robust database environment.

Why did we set up the Keycloak

Problem:

In the absence of a Single Sign-On (SSO) solution, users accessing the Redmine issue management tool and the Apache Superset data visualization tool were required to log in separately. This dual authentication process was time-consuming and potentially burdensome, especially for users who frequently accessed both tools.

Solution:

Keycloak: Simplifying User Access with Single Sign-On (SSO)
To alleviate the inconvenience of separate logins and streamline the user experience, a Single Sign-On solution called Keycloak was introduced. Keycloak acts as an intermediary between Redmine and Superset, offering users a unified authentication process.

Benefits:

- Unified User Authentication: With Keycloak, users need to authenticate only once. Once logged in, they gain seamless access to both the Redmine and Superset platforms, eliminating the need for redundant logins.

- Enhanced User Experience: The convenience of SSO significantly enhances user experience, reducing friction in the authentication process and improving user satisfaction.
- Efficiency and Productivity: Users can seamlessly transition between Redmine and Superset without interruption. This efficiency boost leads to enhanced productivity and smoother workflow.

Problem: Creating Personalized Dashboards with Restricted Access

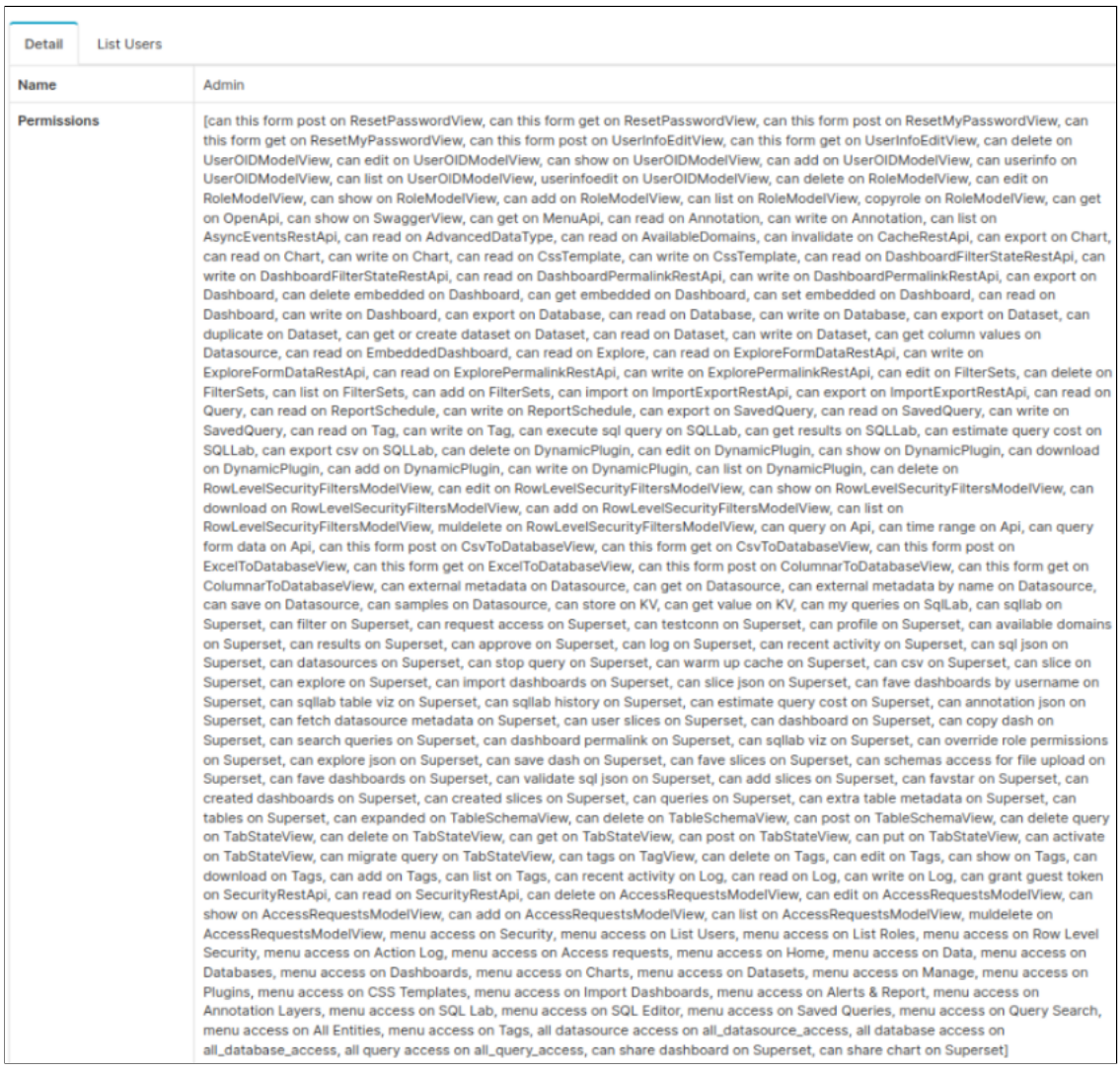
Description:

Our client's requirement is to develop personalized dashboards that provide users with access only to specific tickets, projects, and dashboards assigned to them. The client also wants to ensure that sensitive master data remains hidden from unauthorized users. We possess a comprehensive user dataset containing crucial information related to permissions, roles, and data access permissions. This data is organized within a database table. Each user is associated with one or multiple role groups, each having varying permissions and different levels of access to rows within the data table.

Roles in Superset:

Superset has predefined roles that define user privileges:

- **Admin:** Admins possess full rights, including the ability to manage other users' permissions and modify their dashboards. They can also alter objects owned by others and manage data sources.

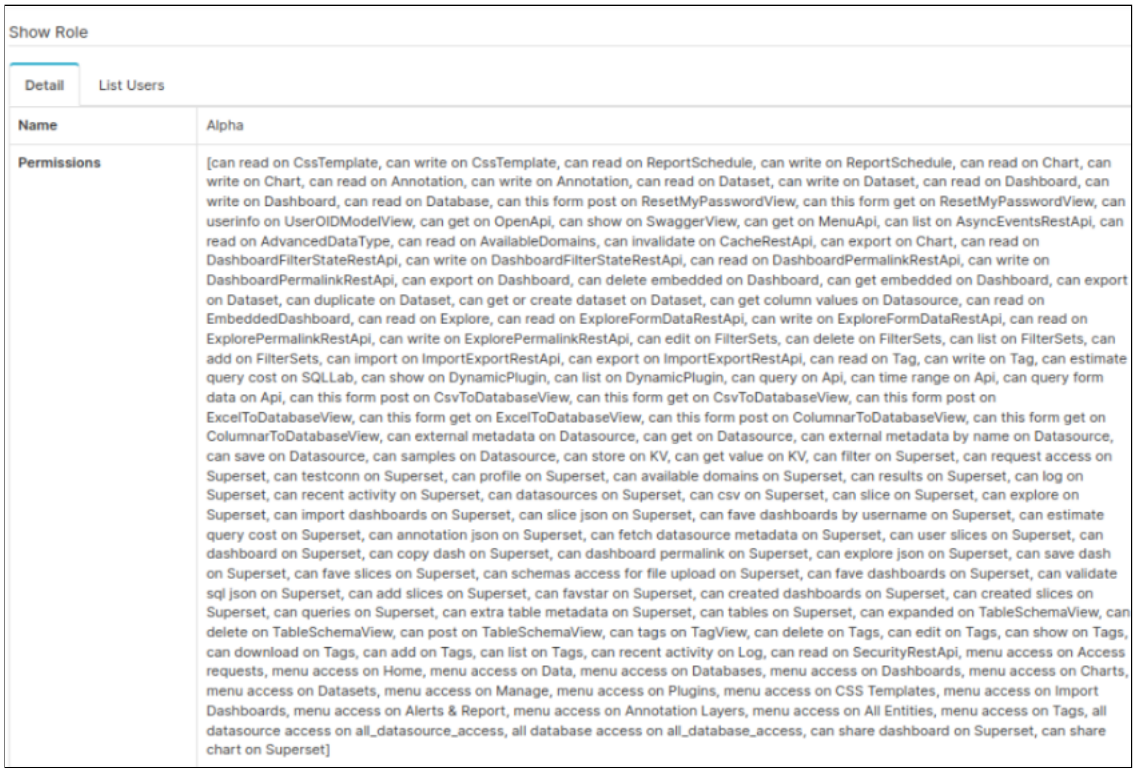


Detail | List Users

Name	Admin
Permissions	[can this form post on ResetPasswordView, can this form get on ResetPasswordView, can this form post on ResetMyPasswordView, can this form get on ResetMyPasswordView, can this form post on UserInfoEditView, can this form get on UserInfoEditView, can delete on UserOIDModelView, can edit on UserOIDModelView, can show on UserOIDModelView, can add on UserOIDModelView, can userinfo on UserOIDModelView, can list on UserOIDModelView, userinfoedit on UserOIDModelView, can delete on RoleModelView, can edit on RoleModelView, can show on RoleModelView, can add on RoleModelView, can list on RoleModelView, copyrole on RoleModelView, can get on OpenApi, can show on SwaggerView, can get on MenuApi, can read on Annotation, can write on Annotation, can list on AsyncEventsRestApi, can read on AdvancedDataType, can read on AvailableDomains, can invalidate on CacheRestApi, can export on Chart, can read on Chart, can write on Chart, can read on CssTemplate, can write on CssTemplate, can read on DashboardFilterStateRestApi, can write on DashboardFilterStateRestApi, can read on DashboardPermalinkRestApi, can write on DashboardPermalinkRestApi, can export on Dashboard, can delete embedded on Dashboard, can get embedded on Dashboard, can set embedded on Dashboard, can read on Dashboard, can write on Dashboard, can export on Database, can read on Database, can write on Database, can export on Dataset, can duplicate on Dataset, can get or create dataset on Dataset, can read on Dataset, can write on Dataset, can get column values on Datasource, can read on EmbeddedDashboard, can read on Explore, can read on ExploreFormDataRestApi, can write on ExploreFormDataRestApi, can read on ExplorePermalinkRestApi, can write on ExplorePermalinkRestApi, can edit on FilterSets, can delete on FilterSets, can list on FilterSets, can add on FilterSets, can import on ImportExportRestApi, can export on ImportExportRestApi, can read on Query, can read on ReportSchedule, can write on ReportSchedule, can export on SavedQuery, can read on SavedQuery, can write on SavedQuery, can read on Tag, can write on Tag, can execute sql query on SQLLab, can get results on SQLLab, can estimate query cost on SQLLab, can export csv on SQLLab, can delete on DynamicPlugin, can edit on DynamicPlugin, can show on DynamicPlugin, can download on DynamicPlugin, can add on DynamicPlugin, can write on DynamicPlugin, can list on DynamicPlugin, can delete on RowLevelSecurityFiltersModelView, can edit on RowLevelSecurityFiltersModelView, can show on RowLevelSecurityFiltersModelView, can download on RowLevelSecurityFiltersModelView, can add on RowLevelSecurityFiltersModelView, can list on RowLevelSecurityFiltersModelView, muldelete on RowLevelSecurityFiltersModelView, can query on Api, can time range on Api, can query form data on Api, can this form post on CsvToDatabaseView, can this form get on CsvToDatabaseView, can this form post on ExcelToDatabaseView, can this form get on ExcelToDatabaseView, can this form post on ColumnarToDatabaseView, can this form get on ColumnarToDatabaseView, can external metadata on Datasource, can get on Datasource, can external metadata by name on Datasource, can save on Datasource, can samples on Datasource, can store on KV, can get value on KV, can my queries on SqlLab, can sqllab on Superset, can filter on Superset, can request access on Superset, can testconn on Superset, can profile on Superset, can available domains on Superset, can results on Superset, can approve on Superset, can log on Superset, can recent activity on Superset, can sql json on Superset, can datasources on Superset, can stop query on Superset, can warm up cache on Superset, can csv on Superset, can slice on Superset, can explore on Superset, can import dashboards on Superset, can slice json on Superset, can fave dashboards by username on Superset, can sqllab table viz on Superset, can sqllab history on Superset, can estimate query cost on Superset, can annotation json on Superset, can fetch datasource metadata on Superset, can user slices on Superset, can dashboard on Superset, can copy dash on Superset, can search queries on Superset, can dashboard permalink on Superset, can sqllab viz on Superset, can override role permissions on Superset, can explore json on Superset, can save dash on Superset, can fave slices on Superset, can schemas access for file upload on Superset, can fave dashboards on Superset, can validate sql json on Superset, can add slices on Superset, can favstar on Superset, can created dashboards on Superset, can created slices on Superset, can queries on Superset, can extra table metadata on Superset, can tables on Superset, can expanded on TableSchemaView, can delete on TableSchemaView, can post on TableSchemaView, can delete query on TabStateView, can delete on TabStateView, can get on TabStateView, can post on TabStateView, can put on TabStateView, can activate on TabStateView, can migrate query on TabStateView, can tags on TagView, can delete on Tags, can edit on Tags, can show on Tags, can download on Tags, can add on Tags, can list on Tags, can recent activity on Log, can read on Log, can write on Log, can grant guest token on SecurityRestApi, can read on SecurityRestApi, can delete on AccessRequestsModelView, can edit on AccessRequestsModelView, can show on AccessRequestsModelView, can add on AccessRequestsModelView, can list on AccessRequestsModelView, muldelete on AccessRequestsModelView, menu access on Security, menu access on List Users, menu access on List Roles, menu access on Row Level Security, menu access on Action Log, menu access on Access requests, menu access on Home, menu access on Data, menu access on Databases, menu access on Dashboards, menu access on Charts, menu access on Datasets, menu access on Manage, menu access on Plugins, menu access on CSS Templates, menu access on Import Dashboards, menu access on Alerts & Report, menu access on Annotation Layers, menu access on SQL Lab, menu access on SQL Editor, menu access on Saved Queries, menu access on Query Search, menu access on All Entities, menu access on Tags, all datasource access on all_datasource_access, all database access on all_database_access, all query access on all_query_access, can share dashboard on Superset, can share chart on Superset]

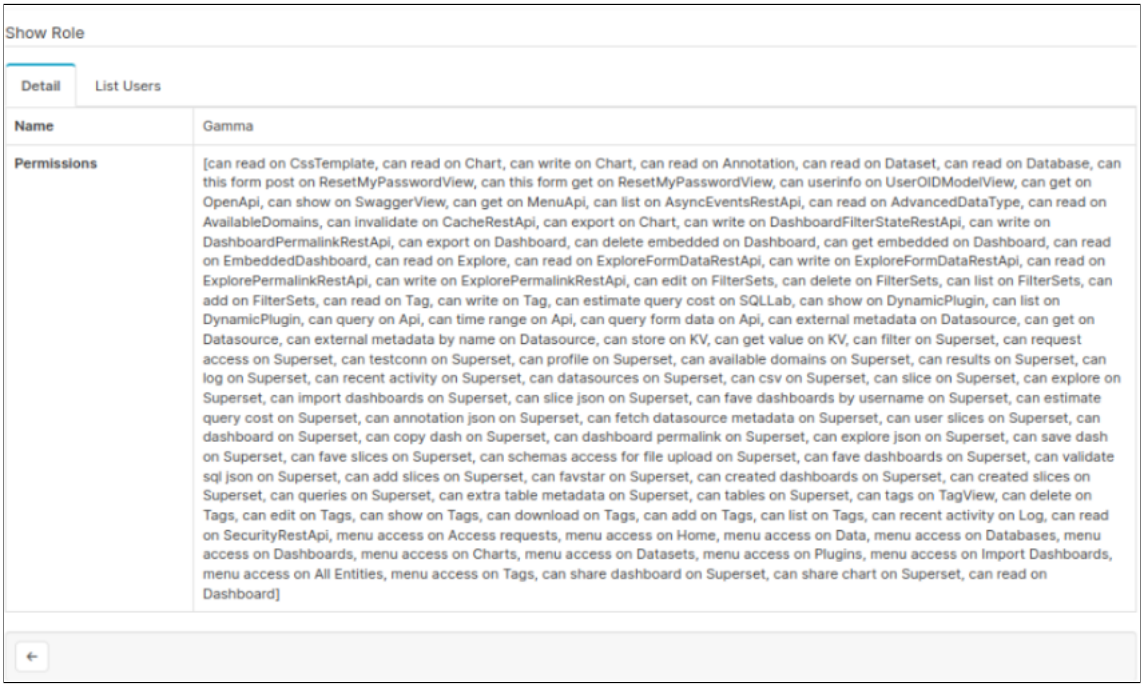
Admin Role and its permissions

Alpha: Alpha users can access all data sources but cannot manage other users' permissions. They are limited to altering objects they own, and they have permissions to add and modify data sources.



Alpha role and its permissions

Gamma: Gamma users have restricted access and can only consume data from assigned data sources. They can view slices and dashboards created from these data sources.

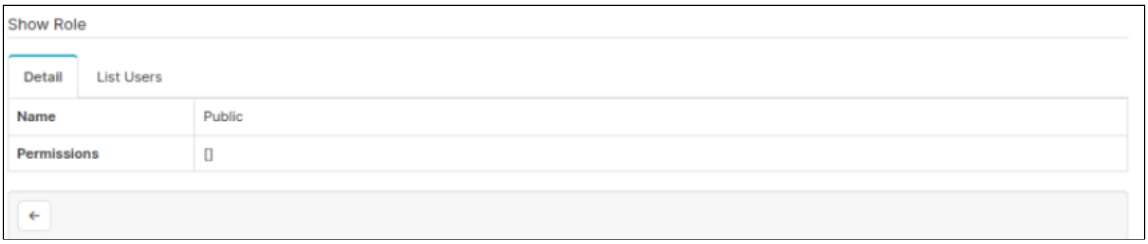


Gamma role and its permissions

sql_lab: This role grants access to SQL Lab. Admins have default access to all databases, while Alpha and Gamma users need access granted per database.

Sql_lab role and its permissions

Public: This role is used to allow logged-out users some access to Superset features, following the PUBLIC_ROLE_LIKE config setting.



Public role and its permissions

Solution:

To address the challenge of personalized dashboards, we duplicated the Gamma role permissions, which aligned best with the client's needs. We created a new role named "Gamma copy 1" and replicated all permissions from the Gamma role. However, we encountered issues with user access to charts and datasets associated with dashboards.

In the new "Gamma copy 1" role, we strategically modified permissions. We removed the permissions to view charts and queries, and instead granted access to a specific dataset (e.g., dataset_123 alias). Superset's role permissions allow us to manage all aspects related to dashboards, database access, SQL queries, and more.

This approach ensures that users can only view dashboards assigned to them and access datasets or databases for which they have authorization. To address data visibility concerns, we undertook a data integration process. We joined tables containing user information, role details, group information, member details, and project information. Within the queries, we utilized the {{current_user}} function as an input value. By setting users.type='User' and user.login={{current_user}}, we e!ffectively limit results to projects and tickets relevant to the logged-in user.

This comprehensive approach successfully delivers personalized dashboards while strictly controlling data access, ensuring that users can only interact with the data relevant to their role and assigned tasks.

Here is the Gamma copy Permissions 1 role and its permissions.

Dashboard Overview

The E-Transport Analytics dashboard in Superset contains a collection of charts and visualizations that provide insights into various aspects of transportation data.

Dashboard Title: Master Data_eTransport_Pending Tickets

Chart Name: Master Data_eTransport_Pending Tickets

Description:
The query provides insights into pending issue tickets related to the eTransport project. It retrieves information such as issue IDs, priorities, subjects, creation dates, authors, pending durations, and more. The query aims to facilitate monitoring and analysis of pending tickets within the specified project.

SQL Query:

```
SELECT row_number() over (partition by 'Subject'
                          order by issue_id asc) AS "S.No.",
                         "Project Name" AS "Project Name",
                         concat('<a href="https://helpdesk.parivahan.gov.in/issues/', "issue_id", '" >', issue_id, '</a>') AS issue_id,
                         "Priority" AS "Priority",
                         "Subject" AS "Subject",
                         "Created on Date" AS "Created on Date",
                         "Author Name" AS "Author Name",
                         "Author_Role" AS "Author_Role",
                         "Pending Since Date" AS "Pending Since Date",
                         "Pending with Name" AS "Pending with Name",
                         "Pending with Role" AS "Pending with Role"
FROM
  (With names as
     (select id,
             firstname,
             lastname
      from Users), projectnames as
     (select id,
             name
      from projects),
                  projectmember as
     (select id,
             user_id,
             project_id
      from members),
                  roledetails as
     (select id,
             name
      from roles),
                  prioritydetails as
     (select id,
             name
      from enumerations),
                  rolenames as
     (select projectmember.user_id as user_id,
             string_agg(DISTINCT (roledetails.name),',') as role_list
      from member_roles,
           projectmember,
           roledetails
      where member_roles.member_id in (projectmember.id)
        and member_roles.role_id = roledetails.id
      group by projectmember.user_id) select projectnames.name as "Project Name",
                                             issues.project_id as "Project ID",
                                             issues.id Issue_ID,
                                             issues.priority_id as "Priority ID",
                                             prioritydetails.name as "Priority",
                                             issues.subject as "Subject",
                                             issues.created_on as "Created on Date",
                                             (authorname.firstname || ' ' || authorname.lastname) as "Author Name",
                                             authors.role_list as "Author_Role",
                                             issues.author_id,
                                             issues.updated_on "Pending Since Date",
                                             (assigned_to_name.firstname || ' ' || assigned_to_name.lastname) as "Pending with Name",
                                             issues.assigned_to_id,
                                             assignees.role_list as "Pending with Role"
   from issues,
        projectnames,
        names authorname,
              names assigned_to_name,
                    rolenames assignees,
                    rolenames authors,
                    prioritydetails
   where issues.status_id not in (3,
                                  5,
                                  6,
                                  10)
     and issues.project_id =projectnames.id
     and issues.author_id = authorname.id
     and issues.assigned_to_id = assigned_to_name.id
     and issues.assigned_to_id <> 362
     and assignees.user_id = issues.assigned_to_id
     and authors.user_id = issues.author_id
     and issues.priority_id = prioritydetails.id) AS virtual_table
LIMIT 50000;
```

Filters Used:

The query incorporates the following filters:

- User ID: `<> 362` (excludes a specific user)
- Issue Status Filter: Excludes issues with status IDs 3, 5, 6, and 10 from the report.

Metrics Used:

The query employs the following metric:

- `row_number()` : Generates a unique serial number for each issue.
- `concat()` : Creates a hyperlink for each issue ID to the same ticket in Redmine.
  Example: `concat('<a href="https://helpdesk.parivahan.gov.in/issues/', "issue_id", '" >', issue_id, '</a>') AS issue_id`

Various fields are utilized in the query such as Project Name, Priority, Subject, Created on Date, etc.

Screenshots:

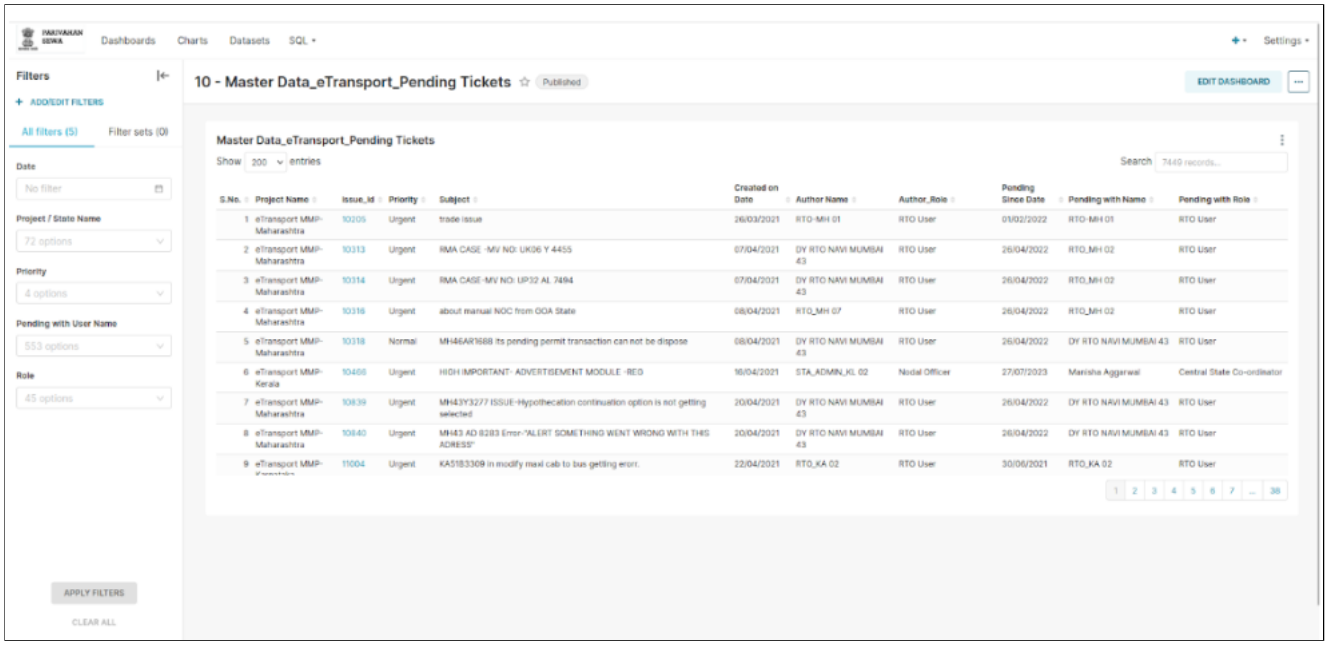


Figure_1:Dashboard Preview of Pending Tickets

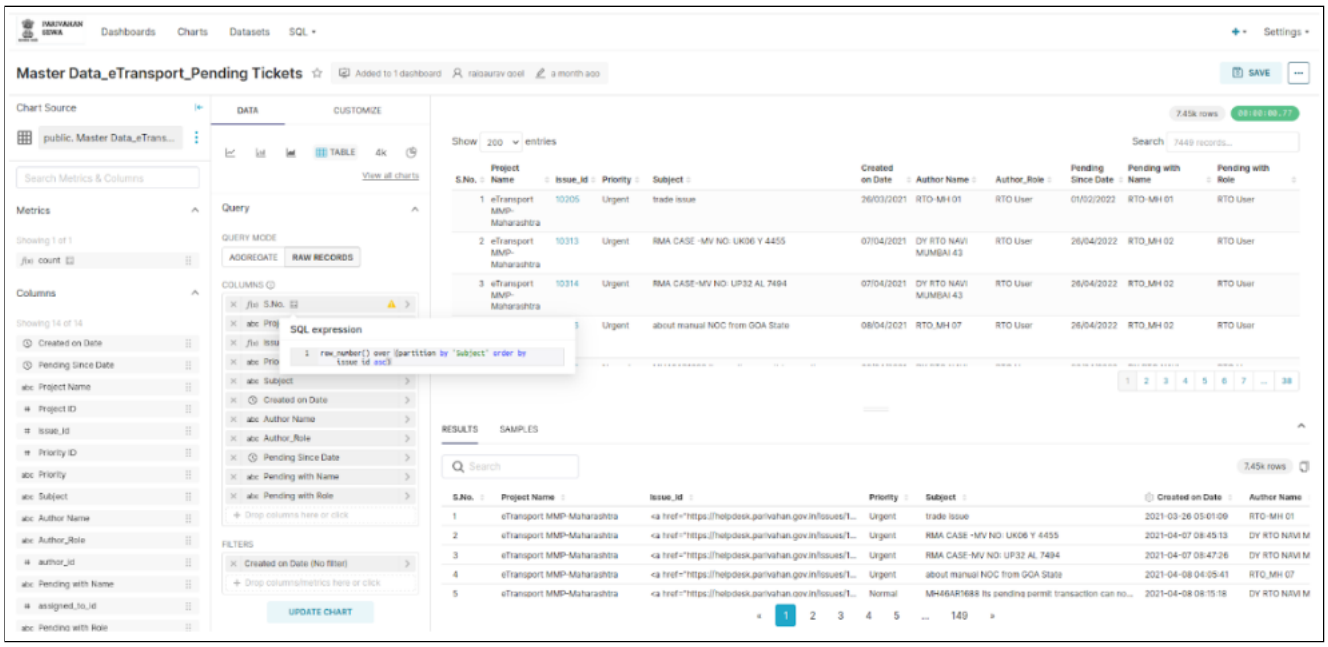


Figure_1.1:Chart Preview with metric for Row number

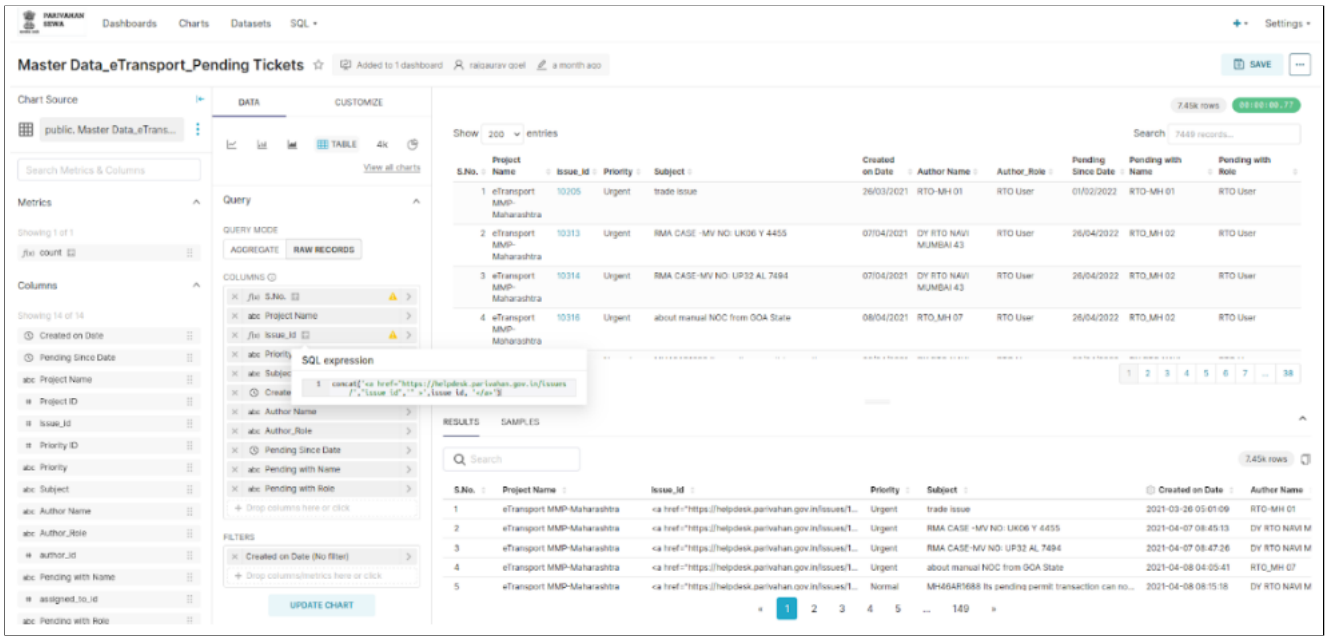


Figure_1.1(a):Chart Preview with metric for concatenating hyperlink to issue_id column

Visualization Type

The query output is presented in table form, displaying detailed issue attributes.

Insights and Observations:

- The "Pending Since Date" metric helps in understanding the duration of pending issues.
- Patterns in issue priorities, roles, and subjects can be identified for better resource allocation.
- The "Pending Since Date" metric helps in understanding the duration of pending issues.
- Patterns in issue priorities, roles, and subjects can be identified for better resource allocation.

Dashboard Title: Count_Chart

Chart Name: Count_Chart

Description:

This query provides insights into issue metrics for a specific user, filtered using the `{{current_user}}` placeholder. The query focuses on issue statistics based on different projects. It calculates the total number of issues, open issues, and the percentage of open issues for each project.

SQL Query:

```
SELECT projectname AS projectname,
       count(distinct(issueid)) AS count,
       (FORMAT('<a href="https://analyticsnew.parivahan.gov.in/superset/dashboard/531">%s</a>', COUNT(DISTINCT CASE
                                                                                                    WHEN issuestatusid NOT IN (3, 5, 6, 10) TH
                                                                                                    ELSE NULL
                                                                                                END))) AS open,
       (count(Distinct case
                           when issuestatusid not in (3, 5, 6, 10) then issueid
                           else null
                       end)/coalesce(nullif(count(distinct(issueid)), 0), 1)::float)*100 AS "Open_Percent"
FROM
  (select users.id as userid,
          users.login as userlogin,
          users.firstname as userfirstname,
          users.lastname as userlastname,
          members.id as memberid,
          members.project_id as projectid,
          member_roles.role_id as roleid,
          roles.name as rolename,
          projects.name as projectname,
          issues.id as issueid,
          issues.subject as issuesubject,
          issues.due_date as issueduedate,
          issues.status_id as issuestatusid,
          issue_statuses.name as issuestatusname,
          issues.priority_id as issueprirorityid,
          enumerations.name as issueprirorityname,
          issues.assigned_to_id as issueassignedtoid,
          assigneeusers.type as assigneetype,
          assigneeusers.login as assigneelogin,
          assigneeusers.firstname as assigneefirstname,
          assigneeusers.lastname as assigneelastname,
          issues.author_id as issueauthorid,
          authorusers.login as authorlogin,
          authorusers.firstname as authorfirstname,
          authorusers.lastname as authorlastname,
          issues.created_on as issuecreatetime,
          issues.updated_on as issueupdatetime,
          issues.start_date as issuestartdate,
          issues.parent_id as issueparentid,
          issues.root_id as issuerootid,
          issues.closed_on as issueclosetime
    from members
    left join users on users.id=members.user_id
    left join member_roles on member_roles.member_id = members.id
    left join roles on member_roles.role_id=roles.id
    left join projects on members.project_id=projects.id
    left join issues on members.project_id = issues.project_id
    left join users assigneeusers on assigneeusers.id = issues.assigned_to_id
    left join users authorusers on authorusers.id = issues.author_id
    left join enumerations on issues.priority_id = enumerations.id
    left join issue_statuses on issues.status_id=issue_statuses.id
    where users.type='User'
      and users.login = '{{current_user}}') AS virtual_table
GROUP BY projectname
ORDER BY count DESC
LIMIT 50000;
```

Usage of {{current_user}} in Query:

The parameter {{current_user}} is expected to be replaced with the actual login name of the user for whom the report is being generated. It filters the results to include only issues associated with the specified user.

Filters Used: The query incorporates the following filters:

- **User Filter:** The query is filtered to fetch data for users of type 'User' and with the login name 'raj'.
- **Issue Status Filter:**
  `Excludes issues with status IDs 3, 5, 6, and 10 from the report because issues with these status are Resolved, Closed, Rejected, De`

Metrics Used: The query employs the following metric:

- **count:** Total count of distinct issue IDs.
  `count(distinct(issueid))`
- **open:** Count of open issues with a hyperlink to an ‘ `Open Ticket` ’ dashboard to view details of open tickets.

```
(FORMAT('<a href="https://analyticsnew.parivahan.gov.in/superset/dashboard/531">%s</a>', COUNT(DISTINCT CASE WHEN issuestatusid NOT IN
```

- **Open_Percent:** Percentage of open issues compared to the total count.

(count(Distinct case when issuestatusid not in (3,5,6,10) then issueid else null end)/coalesce(nullif(count(distinct(issueid)),0),1)::float)*100

Screenshots:

Figure_2.0:Dashboard when login with user ‘Raj’

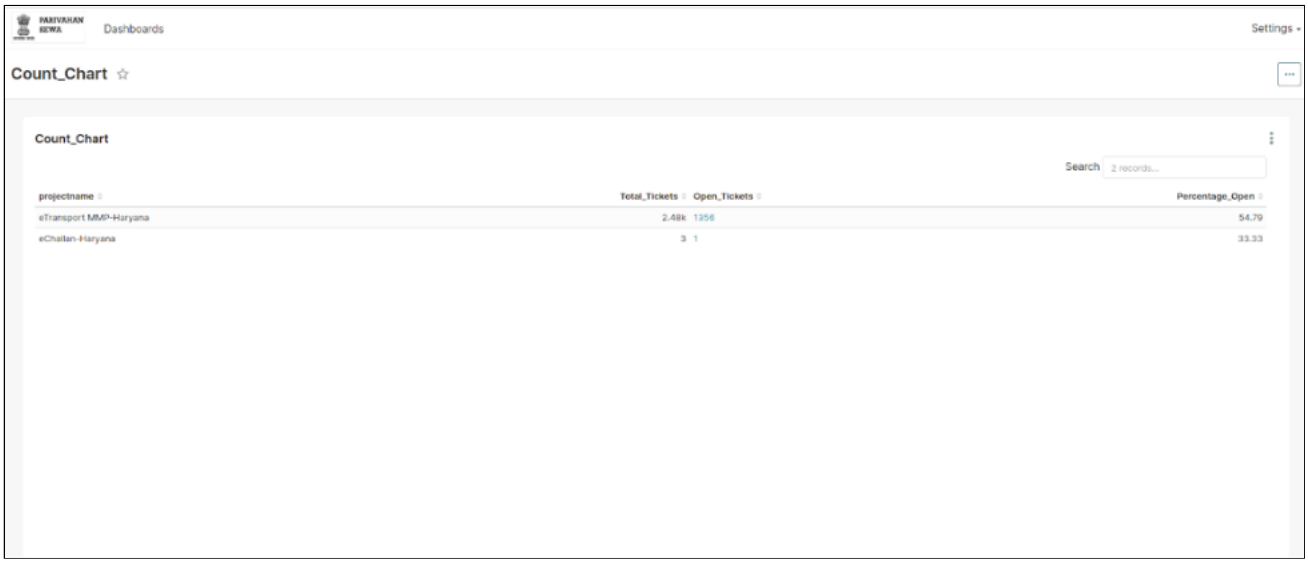


Figure_2.0(a):Dashboard when login with user ‘support_hr’

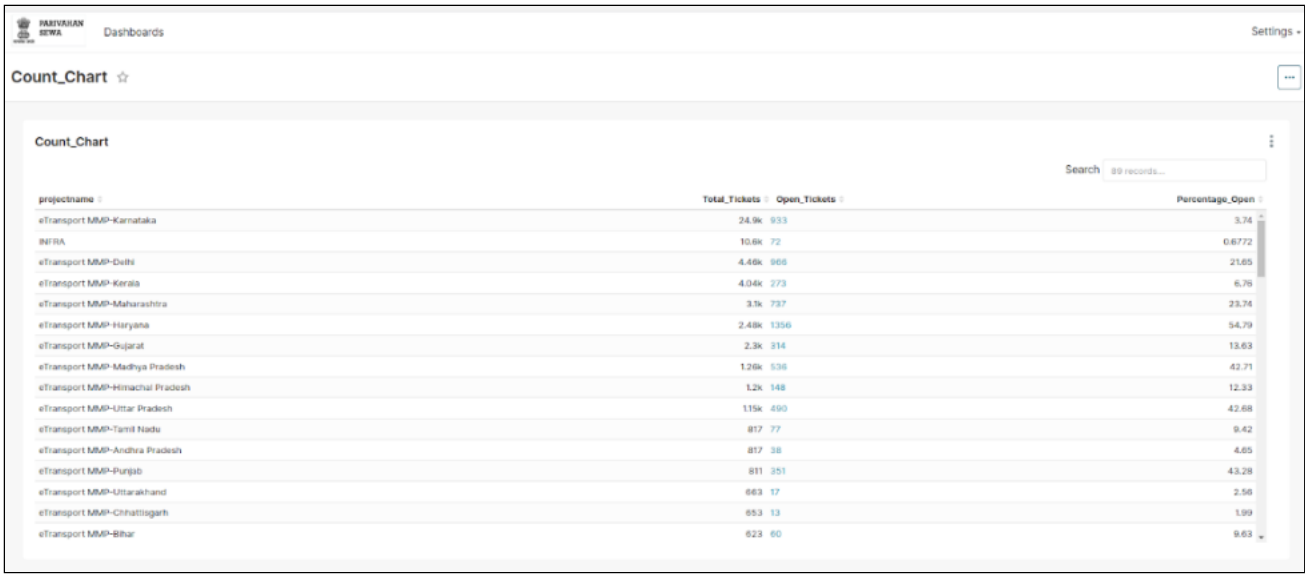


Figure_2.0(b):Dashboard when login with user ‘Avnesh’

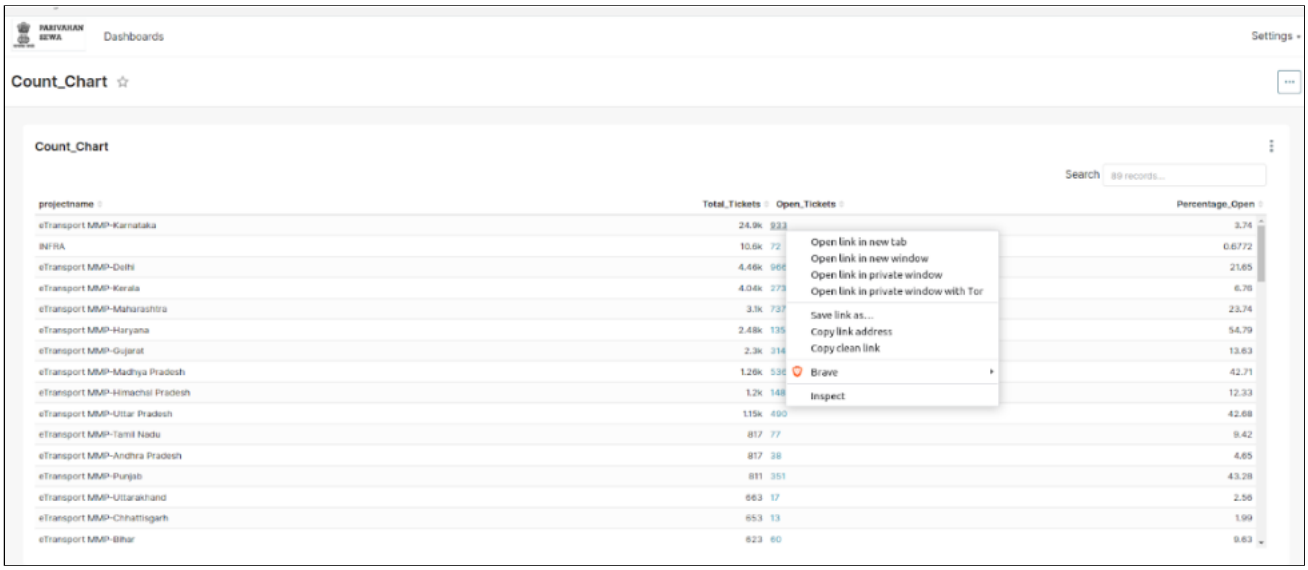


Figure_2.1:Hyperlink to an ‘Open Ticket’ dashboard for detail of all the open tickets

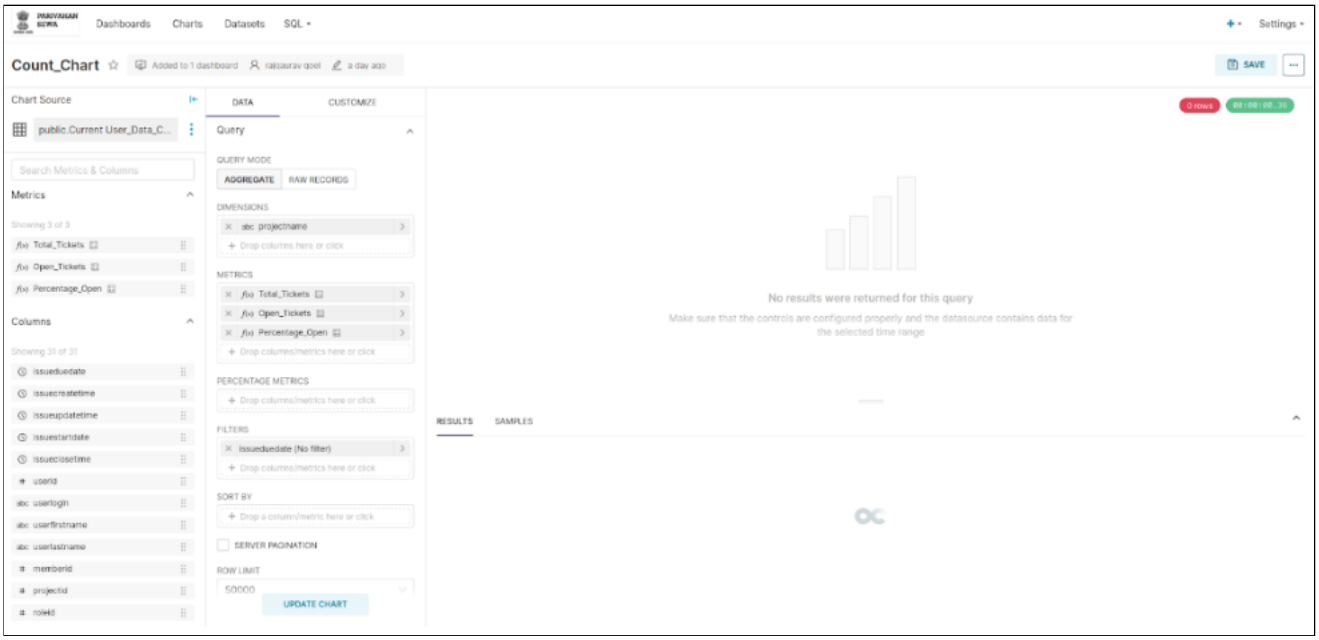


Figure_2.2:Chart Preview with all metrics

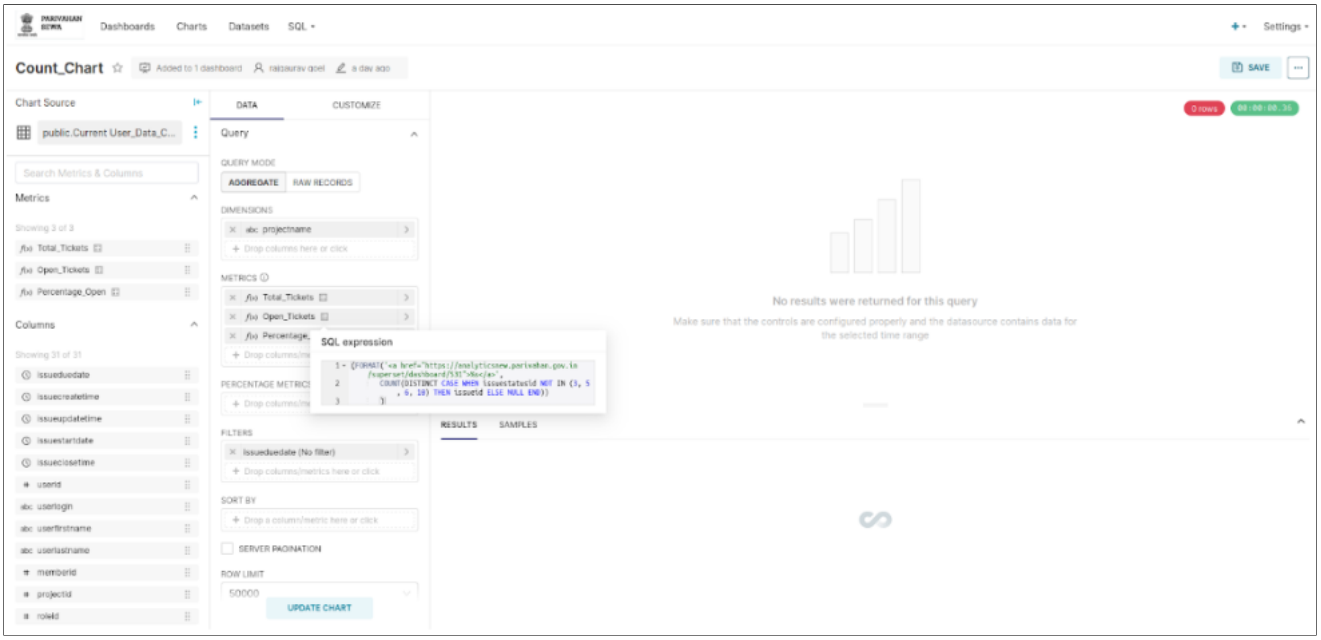


Figure_2.3:Chart Preview with metric for concatenating hyperlink to issue_id column

Visualization Type:

The output of this query is visualized as a table to provide a clear and organized representation of user-specific issue details.

Insights and Observations:

- Users can assess their involvement in different projects by analyzing the issue counts.
- The "Open_Percent" metric provides insights into the user's impact on open issues within each project.
- Projects with higher issue counts might require more attention from the user.
- The percentage of open issues highlights the user's effectiveness in resolving issues promptly.

Dashboard Title: Open Tickets

Chart Name: Current User Open tickets

Description:

This query retrieves detailed Open issues information and metrics for a specific user, identified by the {{current_user}} placeholder. It presents a comprehensive overview of issues associated with the user, including issue IDs, statuses, priorities, assignments, and more. The query is designed to facilitate open issue tracking and analysis for the user.

SQL Query:

```
SELECT row_number() over (partition by 'issueid'
                          order by issueid asc) AS "S.No.",
                         userlogin AS userlogin,
                         userfirstname AS userfirstname,
                         projectname AS projectname,
                         concat('<a href="https://helpdesk.parivahan.gov.in/issues/', "issueid", '" >', issueid, '</a>') AS issueid,
                         issuecreatetime AS issuecreatetime,
                         issueduedate AS issueduedate,
                         issuesubject AS issuesubject,
                         rolename AS rolename,
                         authorfirstname AS authorfirstname,
                         issuestatusname AS issuestatusname,
                         authorlastname AS authorlastname,
                         issuepriorityname AS issuepriorityname,
                         assigneefirstname AS assigneefirstname,
                         issueupdatetime AS issueupdatetime,
                         assigneelastname AS assigneelastname,
                         issueclosetime AS issueclosetime
FROM
  (with mydata as
     (select users.id as userid,
             users.login as userlogin,
             users.firstname as userfirstname,
             users.lastname as userlastname,
             members.id as memberid,
             members.project_id as projectid,
             member_roles.role_id as roleid,
             roles.name as rolename,
             projects.name as projectname,
             issues.id as issueid,
             issues.subject as issuesubject,
             issues.due_date as issueduedate,
             issues.status_id as issuestatusid,
             issue_statuses.name as issuestatusname,
             issues.priority_id as issueprorityid,
             enumerations.name as issuepriorityname,
             issues.assigned_to_id as issueassignedtoid,
             assigneeusers.type as assigneetype,
             assigneeusers.login as assigneelogin,
             assigneeusers.firstname as assigneefirstname,
             assigneeusers.lastname as assigneelastname,
             issues.author_id as issueauthorid,
             authorusers.login as authorlogin,
             authorusers.firstname as authorfirstname,
             authorusers.lastname as authorlastname,
             issues.created_on as issuecreatetime,
             issues.updated_on as issueupdatetime,
             issues.start_date as issuestartdate,
             issues.parent_id as issueparentid,
             issues.root_id as issuerootid,
             issues.closed_on as issueclosetime
      from members
      left join users on users.id=members.user_id
      left join member_roles on member_roles.member_id = members.id
      left join roles on member_roles.role_id=roles.id
      left join projects on members.project_id=projects.id
      left join issues on members.project_id = issues.project_id
      left join users assigneeusers on assigneeusers.id = issues.assigned_to_id
      left join users authorusers on authorusers.id = issues.author_id
      left join enumerations on issues.priority_id = enumerations.id
      left join issue_statuses on issues.status_id=issue_statuses.id
      where users.type='User'
        and users.login = 'raj') SELECT DISTINCT ON (issueid) *
   FROM mydata
   WHERE issuestatusid NOT IN
        (3,
                              5,
                              6,
                              10)) AS virtual_table
LIMIT 50000;
```

Filters Used: The query incorporates the following filters:

- User Filter: The query is filtered to fetch data for users of type 'User' and with the login name 'raj' because I am logged with the user Raj.
- Issue Status Filter: Excludes issues with status IDs 3, 5, 6, and 10 from the report.

Metrics Used: The query employs the following metric:

- row_number(): Provides a serial number for each issue.
- concat(): Generates a hyperlink for each issue ID.

```
concat('<a href="https://helpdesk.parivahan.gov.in/issues/', "issueid", '" >', issueid, '</a>') AS issueid
```

- Various fields such as userlogin, projectname, issuecreatetime, issueduedate, etc.

Screenshots:

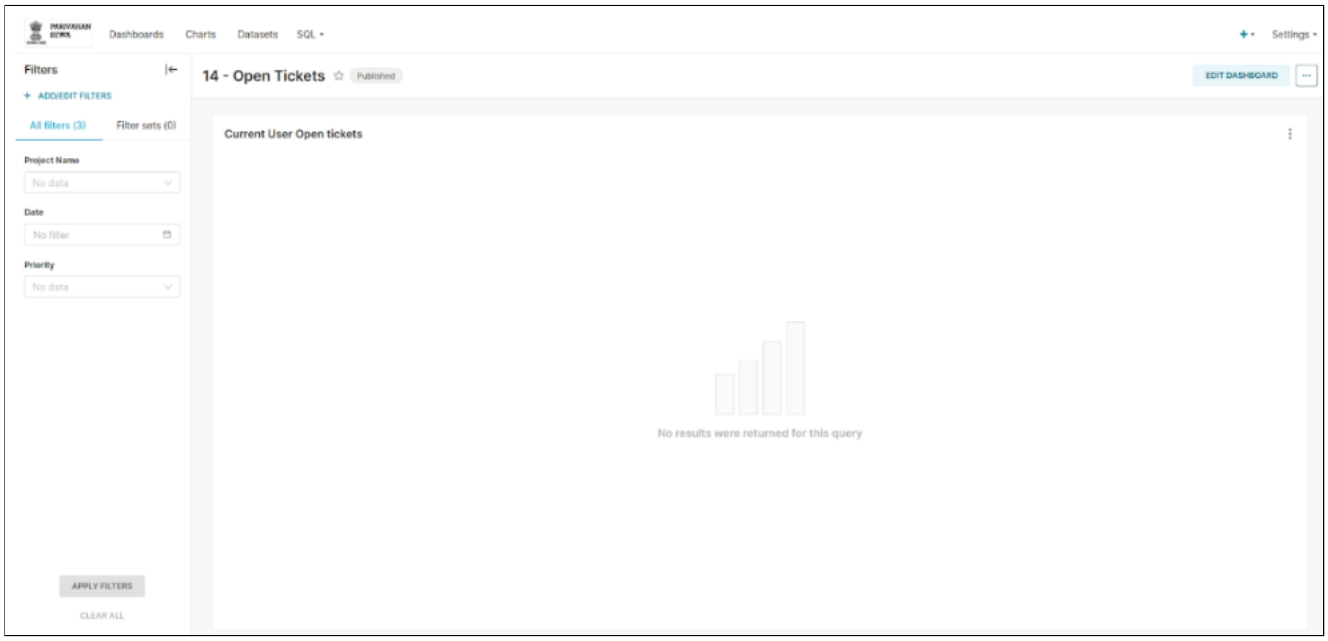


Figure_3.0:Dashboard when login with user ‘Raj’

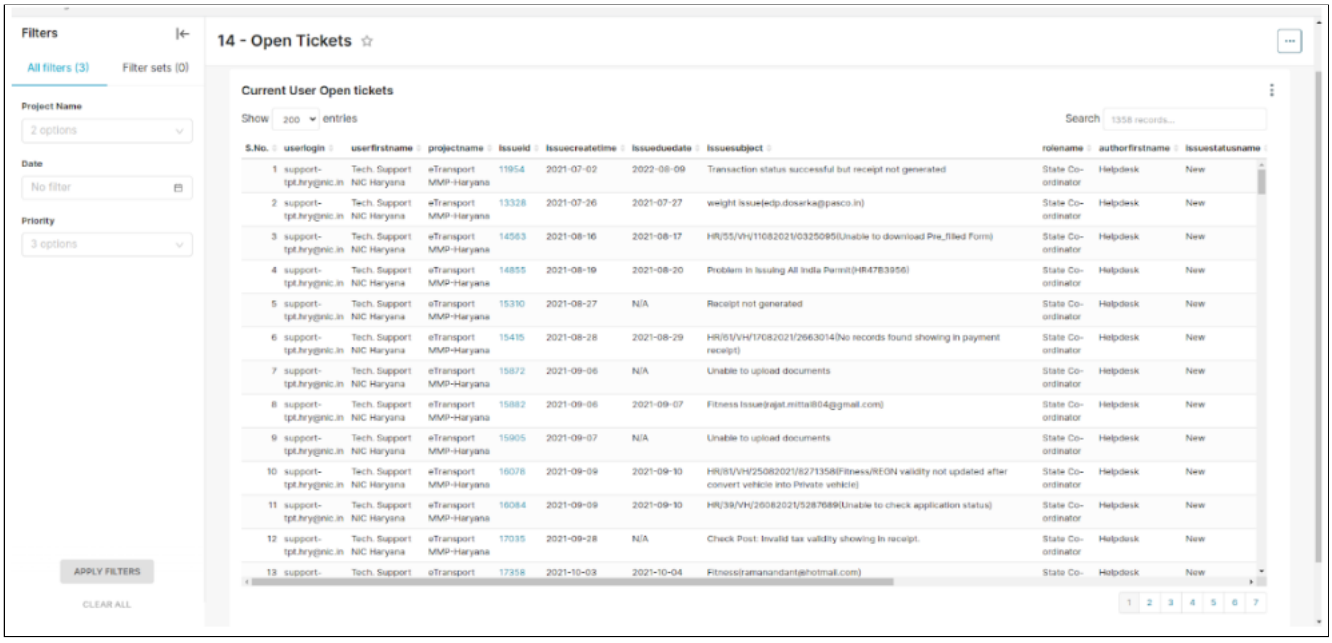


Figure_3.0(a):Dashboard with only haryana open tickets when login with user ‘support_hr’

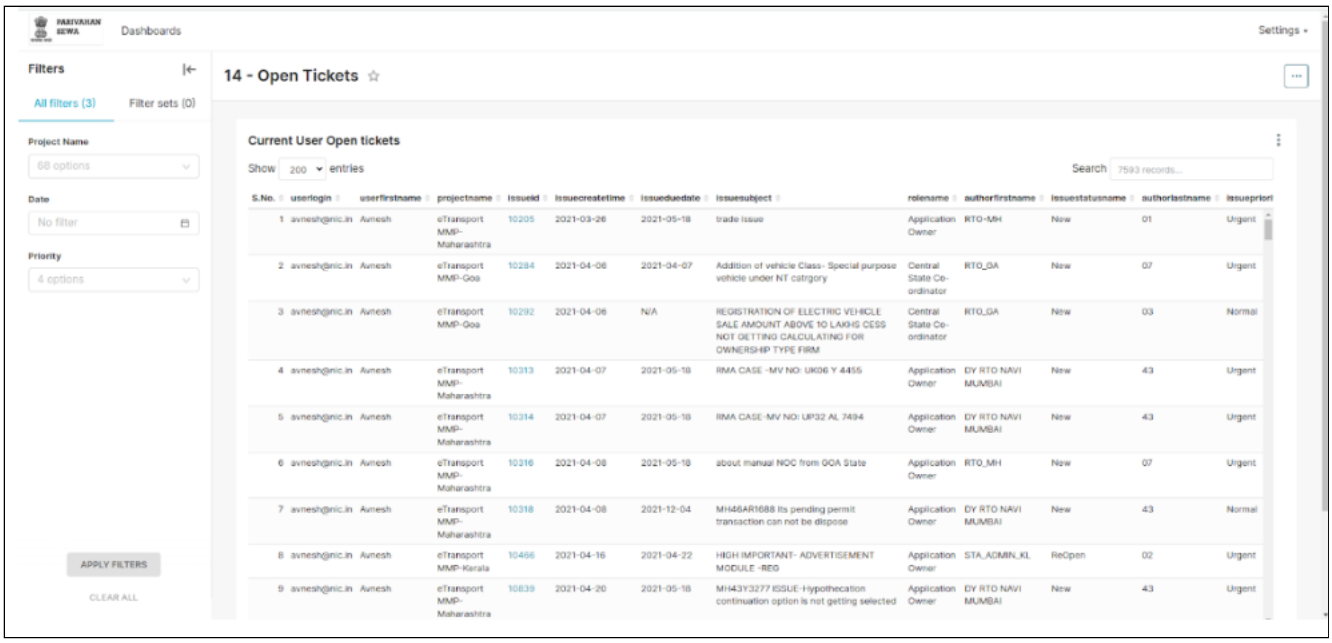


Figure_3.0(b):Dashboard with open tickets of multiple projects when login with user 'Avnesh' as the user have access to multiple projects

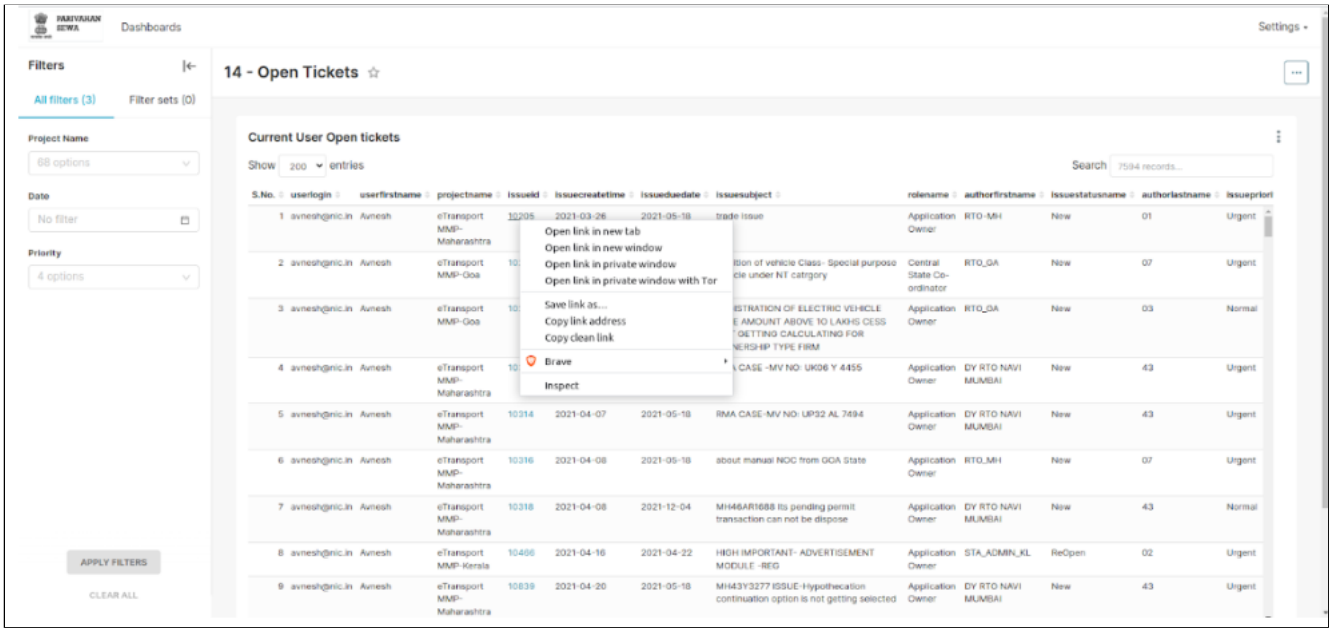


Figure_3.1:Hyperlink embedded in 'issueid' to view the ticket in Redmine

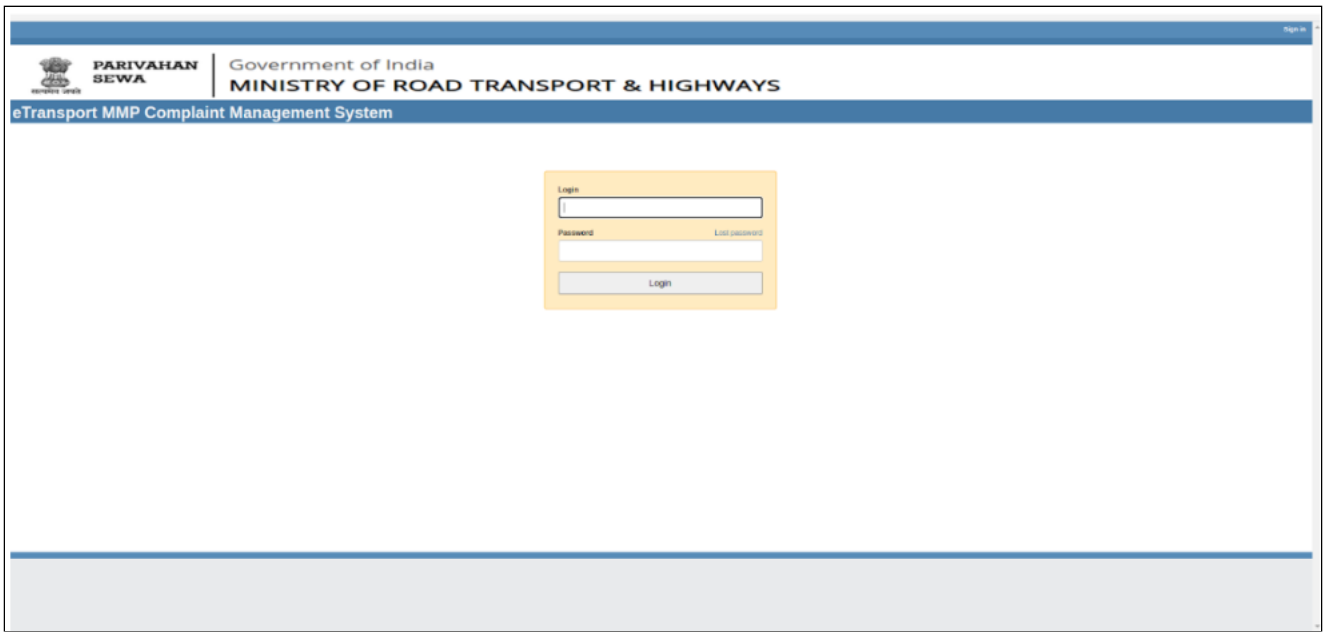


Figure_3.2:Hyperlink takes to the login page of Redmine

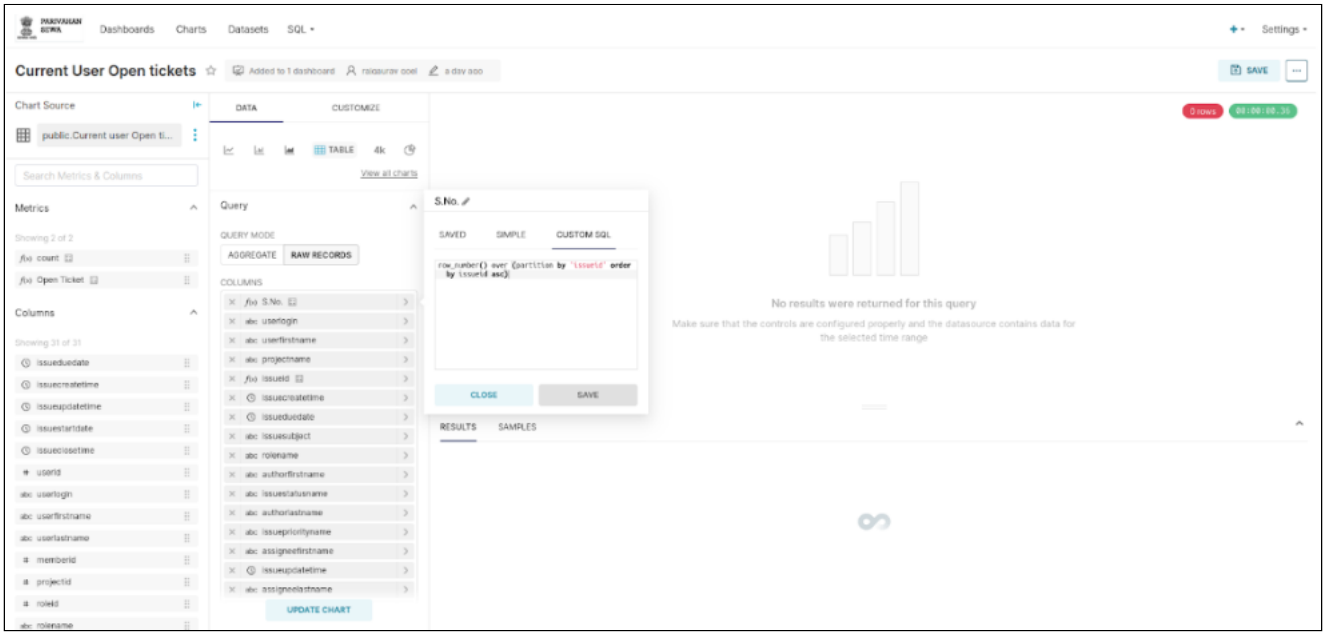


Figure_3.3:Chart Preview with row_number metrics

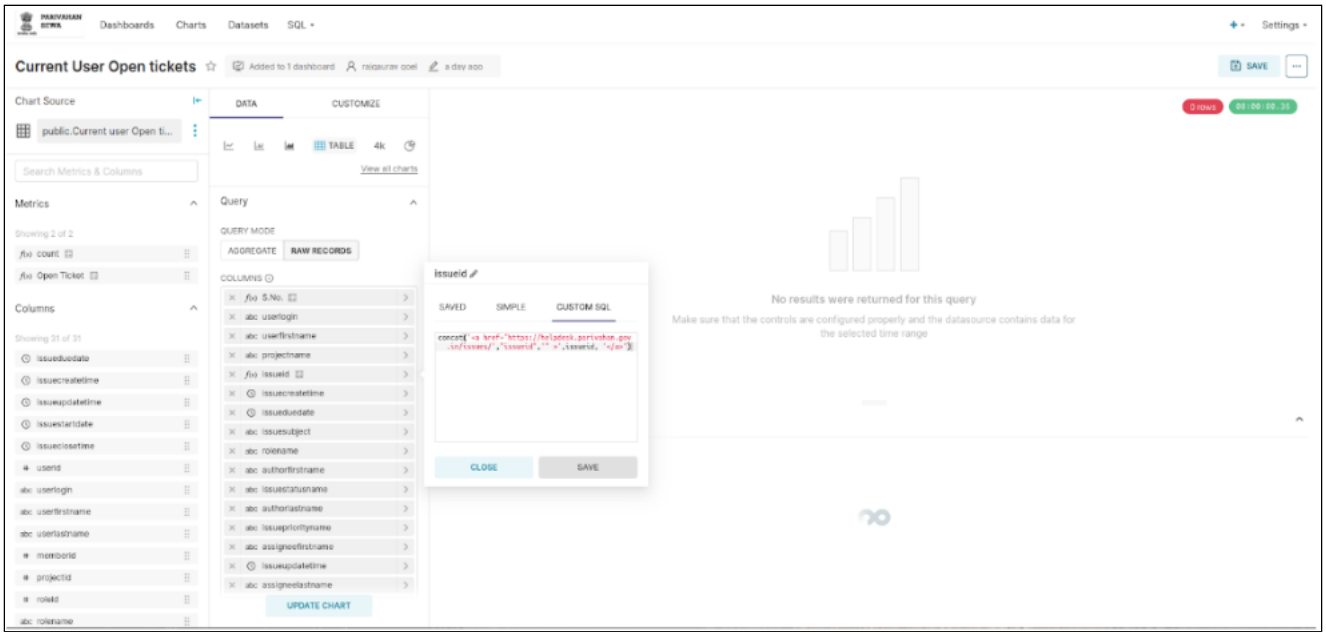


Figure_3.3(a):Chart Preview with metric for concatenating hyperlink of redmine to issue_id column

Visualization Type: The output of this query is visualized as a table to provide a clear and organized representation of user-specific issue details.

Insights and Observations:

- Users can examine their issue involvement across projects and track key details.
- The query highlights issue IDs with hyperlinks, facilitating direct access to issue details.
- Users can identify their contribution to various issues and their role in resolving them.
- Trends in issue creation, updates, and closures can be assessed for informed decision-making.

Dashboard Title: Master_Ticket Data

Chart Name: Master_Ticket Data

Description:

The query retrieves essential information about transportation-related issue tickets from the E-Transport Analytics project. The query aims to provide insights into ticket details, such as project information, issue status, priority, assignees, authors, and key dates. This data contributes to informed decision-making and efficient issue management.

SQL Query:

```
SELECT row_number() over (partition by 'Subject'
                          order by issueid asc) AS "S.No.",
                         userfirstname AS userfirstname,
                         userlastname AS userlastname,
                         projectname AS projectname,
                         rolename AS rolename,
                         issueid AS issueid,
                         issueduedate AS issueduedate,
                         issuesubject AS issuesubject,
                         issuestatusname AS issuestatusname,
                         issueprioritыname AS issueprioritыname,
                         assigneetype AS assigneetype,
                         assigneefirstname AS assigneefirstname,
                         assigneelastname AS assigneelastname,
                         authorfirstname AS authorfirstname,
                         authorlastname AS authorlastname,
                         issuecreatetime AS issuecreatetime,
                         issueupdatetime AS issueupdatetime,
                         issuestartdate AS issuestartdate,
                         issueclosetime AS issueclosetime
FROM
  (select users.id as userid,
          users.login as userlogin,
          users.firstname as userfirstname,
          users.lastname as userlastname,
          members.id as memberid,
          members.project_id as projectid,
          member_roles.role_id as roleid,
          roles.name as rolename,
          projects.name as projectname,
          issues.id as issueid,
          issues.subject as issuesubject,
          issues.due_date as issueduedate,
          issues.status_id as issuestatusid,
          issue_statuses.name as issuestatusname,
          issues.priority_id as issueprioritыid,
          enumerations.name as issueprioritыname,
          issues.assigned_to_id as issueassignedtoid,
          assigneeusers.type as assigneetype,
          assigneeusers.login as assigneelogin,
          assigneeusers.firstname as assigneefirstname,
          assigneeusers.lastname as assigneelastname,
          issues.author_id as issueauthorid,
          authorusers.login as authorlogin,
          authorusers.firstname as authorfirstname,
          authorusers.lastname as authorlastname,
          issues.created_on as issuecreatetime,
          issues.updated_on as issueupdatetime,
          issues.start_date as issuestartdate,
          issues.parent_id as issueparentid,
          issues.root_id as issuerootid,
          issues.closed_on as issueclosetime
   from members
   left join users on users.id=members.user_id
   left join member_roles on member_roles.member_id = members.id
   left join roles on member_roles.role_id=roles.id
   left join projects on members.project_id=projects.id
   left join issues on members.project_id = issues.project_id
   left join users assigneeusers on assigneeusers.id = issues.assigned_to_id
   left join users authorusers on authorusers.id = issues.author_id
   left join enumerations on issues.priority_id = enumerations.id
   left join issue_statuses on issues.status_id=issue_statuses.id
   where users.type='User'
     and users.login = 'raj') AS virtual_table
LIMIT 50000;
```

Filters Used: The query incorporates the following filters:

- Filters by the current user using the {{current_user}} placeholder.

Metrics Used: The query employs the following metric:

- row_number(): Provides a serial number for each issue.

- Various fields such as "Project Name", "Priority", "Subject", "Created on Date", etc.

Screenshots:

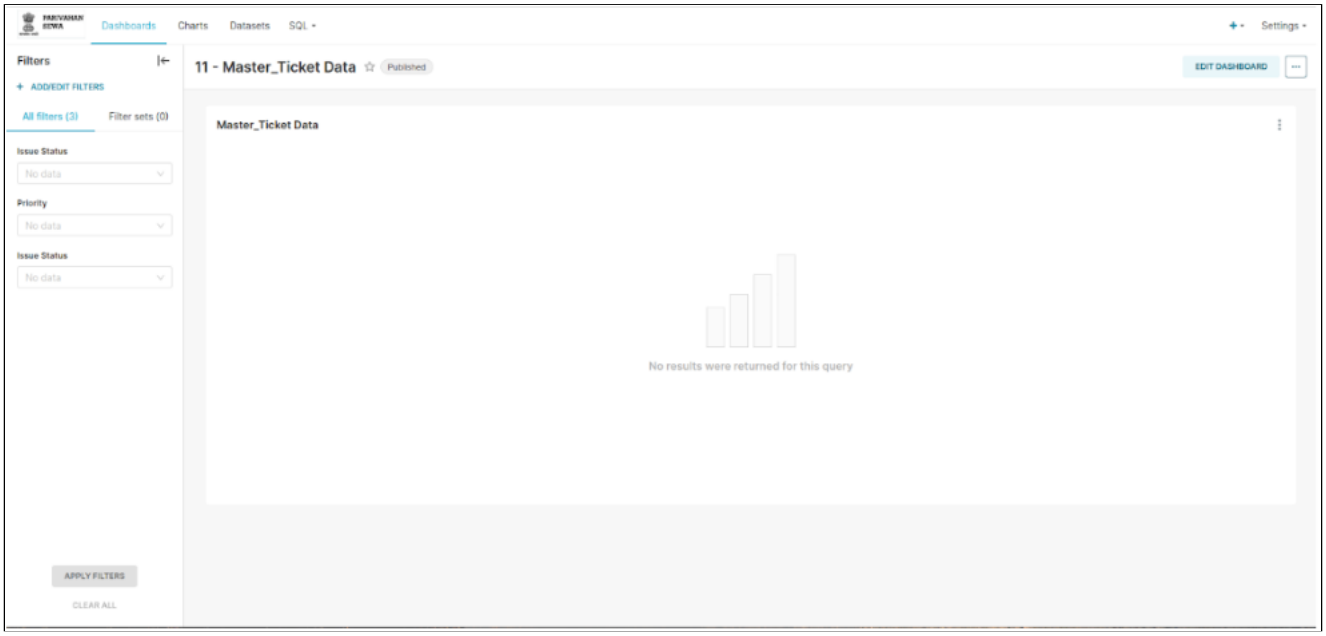


Figure_4.0:Dashboard when login with user ‘Raj’

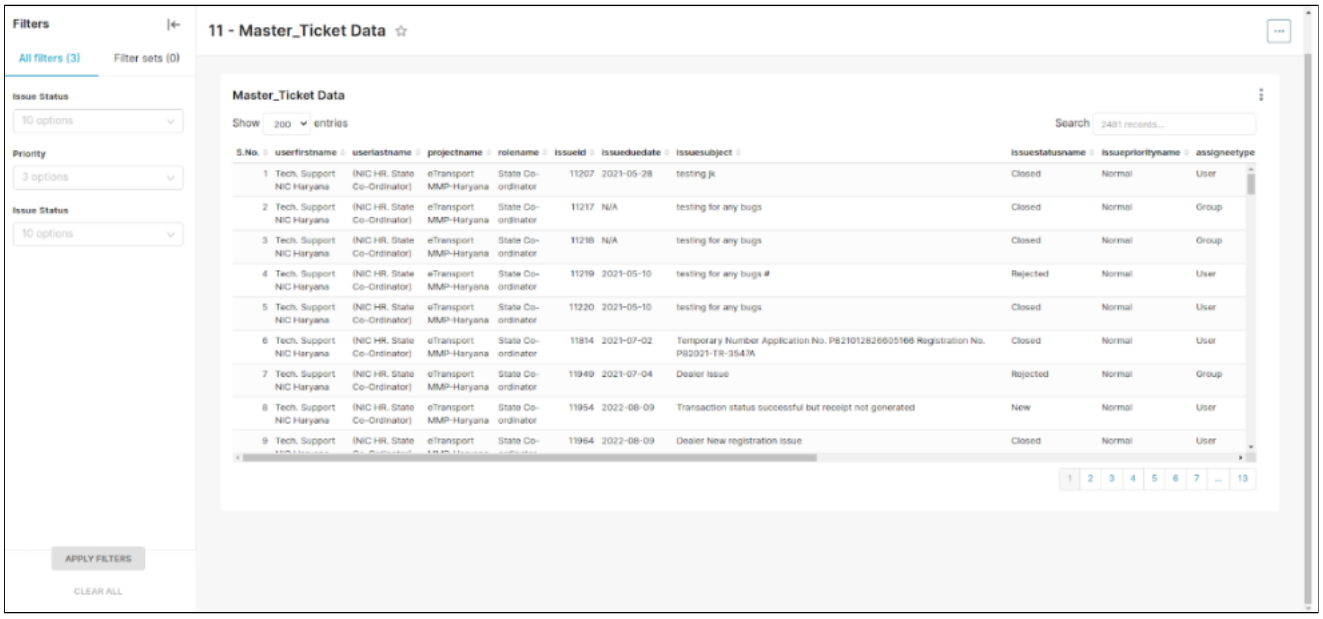


Figure_4.0(a):Dashboard with insights into logged in user ticket details, such as project information, issue status, priority, assignees, authors, and key dates.

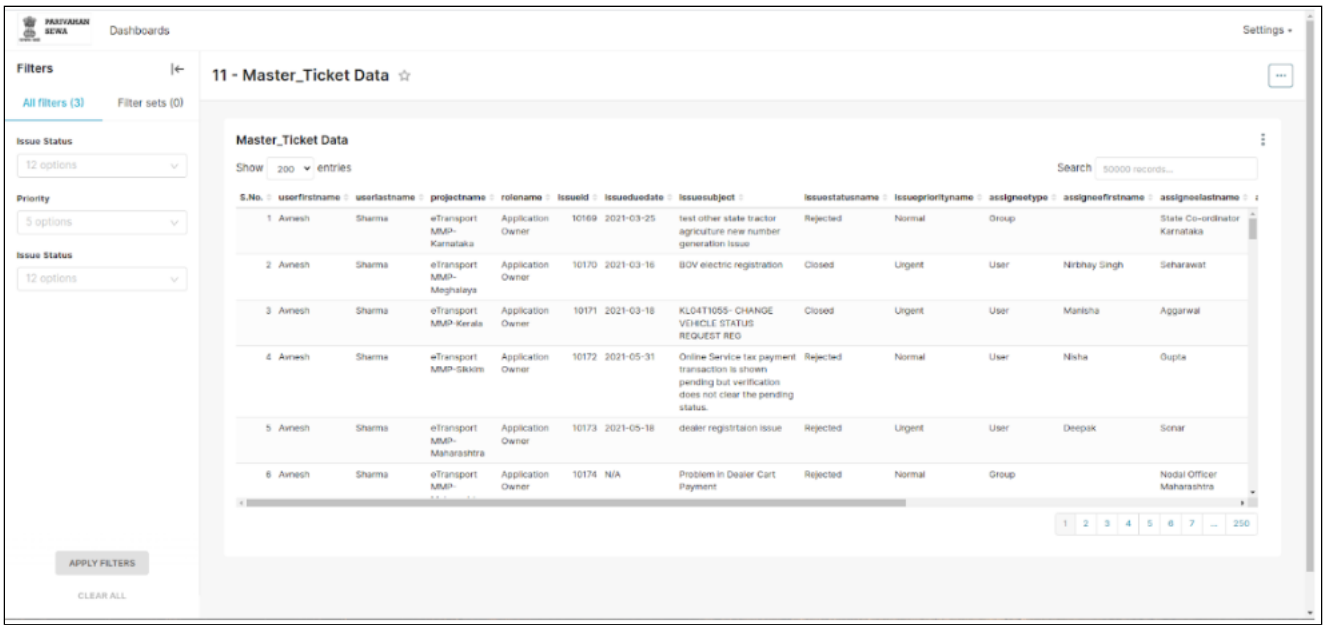


Figure_4.0(b):Dashboard with master data when logged in with user 'Avnesh' as the user have access to multiple projects

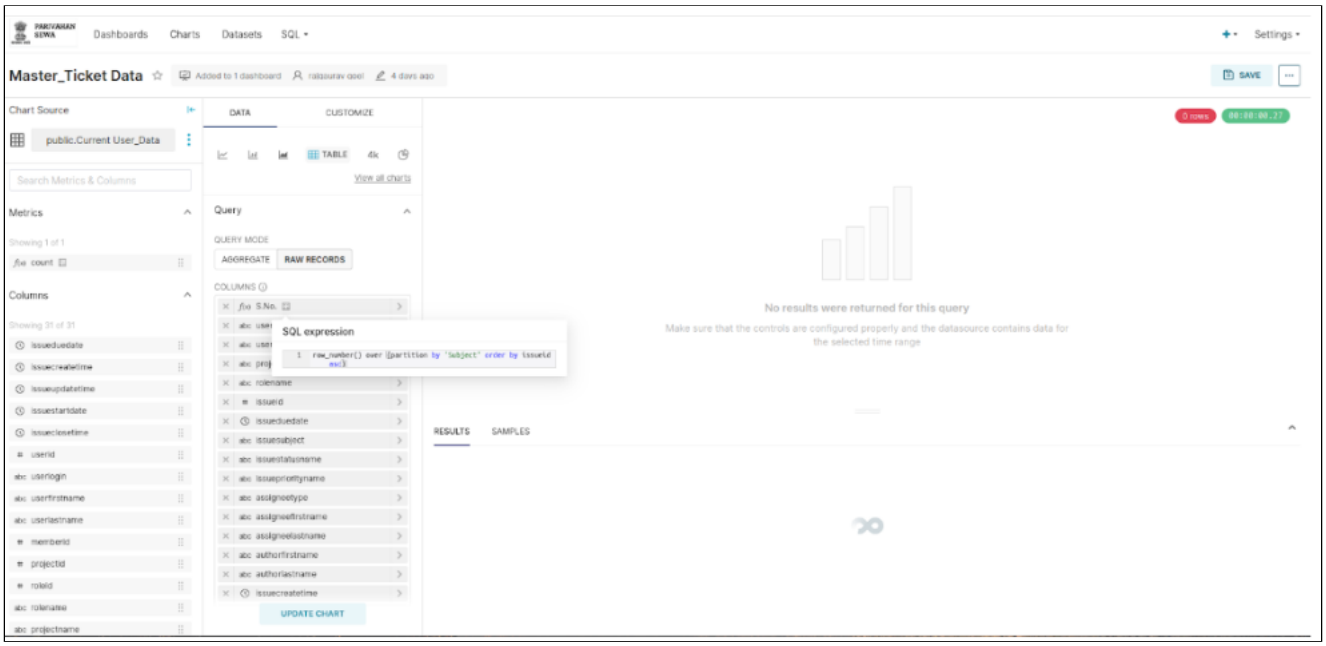


Figure_4.1:Chart Preview with row_number metrics

Visualization Type: The output of this query is visualized as a table to provide a clear and organized representation of user-specific issue details.

Insights and Observations:

- Users can examine their issue involvement across projects and track key details.
- The query highlights issue IDs with hyperlinks, facilitating direct access to issue details.
- Users can identify their contribution to various issues and their role in resolving them.
- Trends in issue creation, updates, and closures can be assessed for informed decision-making.

Dashboard Title: Monthly Issues Summary Report (State Projectwise)_1st of Month

Chart Name: 3 - Monthly Issues Summary Report (State Projectwise)_1st of Month

Description:

The query generates a summary report showcasing various statistics related to issue-tickets within state projects. It presents data for issues that were pending on the 1st of the month, issues opened in the current month, total issues in the month, resolved and closed issues in the month, and more. This

report offers insights into the status of issues and their resolution over a specific time frame.

SQL Query:

```
SELECT "Project Name" AS "Project Name",
       "Issue Pending on 1 st of Month" AS "Issue Pending on 1 st of Month",
       "Opened in the Month" AS "Opened in the Month",
       "Total Issues In the Month" AS "Total Issues In the Month",
       "Resolved in the Month" AS "Resolved in the Month",
       "Percentage Resolved in the month" AS "Percentage Resolved in the month",
       "Closed in the Month" AS "Closed in the Month",
       "Percentage Closure in the month" AS "Percentage Closure in the month",
       "Balance at the end of Month" AS "Balance at the end of Month"
FROM
  (select project_name "Project Name",
          last_mnt_pending "Issue Pending on 1 st of Month",
          opened "Opened in the Month",
          coalesce(last_mnt_pending+opened, 0)"Total Issues In the Month",
          total_resolved "Resolved in the Month",
          round((total_resolved * 100.0) / coalesce(last_mnt_pending+opened, 0), 1) "Percentage Resolved in the month",
          total_closed "Closed in the Month",
          round((total_closed * 100.0) / coalesce(last_mnt_pending+opened, 0), 1) "Percentage Closure in the month",
          last_mnt_pending+opened-(total_closed+total_resolved) "Balance at the end of Month"
   FROM
     (select project_name,
             coalesce(
                        (select count(issues.id) last_mnth_pending
                         from issues
                         left join projects on issues.project_id=projects.id
                         where (date_trunc('month', issues.created_on)<date_trunc('month', current_date - interval '1 month')
                                and date_trunc('month', issues.closed_on)>=date_trunc('month', current_date - interval '1 month'))
                           and projects.name=A.project_name ),0) last_mnt_pending,
             coalesce(opened, 0)opened,
             coalesce(
                        (select sum(case
                                        when issues.status_id in (5, 6, 10) then 1
                                        else 0
                                    end) closed
                         from issues
                         left join projects on issues.project_id=projects.id
                         where date_trunc('month', issues.closed_on)=date_trunc('month', current_date - interval '1 month')
                           and projects.name=A.project_name ),0)total_closed,
             coalesce(
                        (select sum(case
                                        when issues.status_id in (3) then 1
                                        else 0
                                    end) resolved
                         from issues
                         left join projects on issues.project_id=projects.id
                         where date_trunc('month', issues.updated_on)=date_trunc('month', current_date - interval '1 month')
                           and projects.name=A.project_name ),0)total_resolved
      from
        (select projects.name project_name,
                count(issues.id) opened
         from issues
         left join projects on issues.project_id=projects.id
         where date_trunc('month', issues.created_on)=date_trunc('month', current_date - interval '1 month')
         group by projects.name
         order by projects.name)A)B) AS virtual_table
LIMIT 1000;
```

Filters Used: The query incorporates the following filters:

- Filters data to include only the specific time frame of the previous month.

Metrics Used: The query employs the following metric:

- **count():** Calculates the number of issue-tickets based on various conditions.
- **sum():** Calculates the sum of resolved and closed issue-tickets.
- **round():** Computes the percentage of resolved and closed issues.

Screenshots:

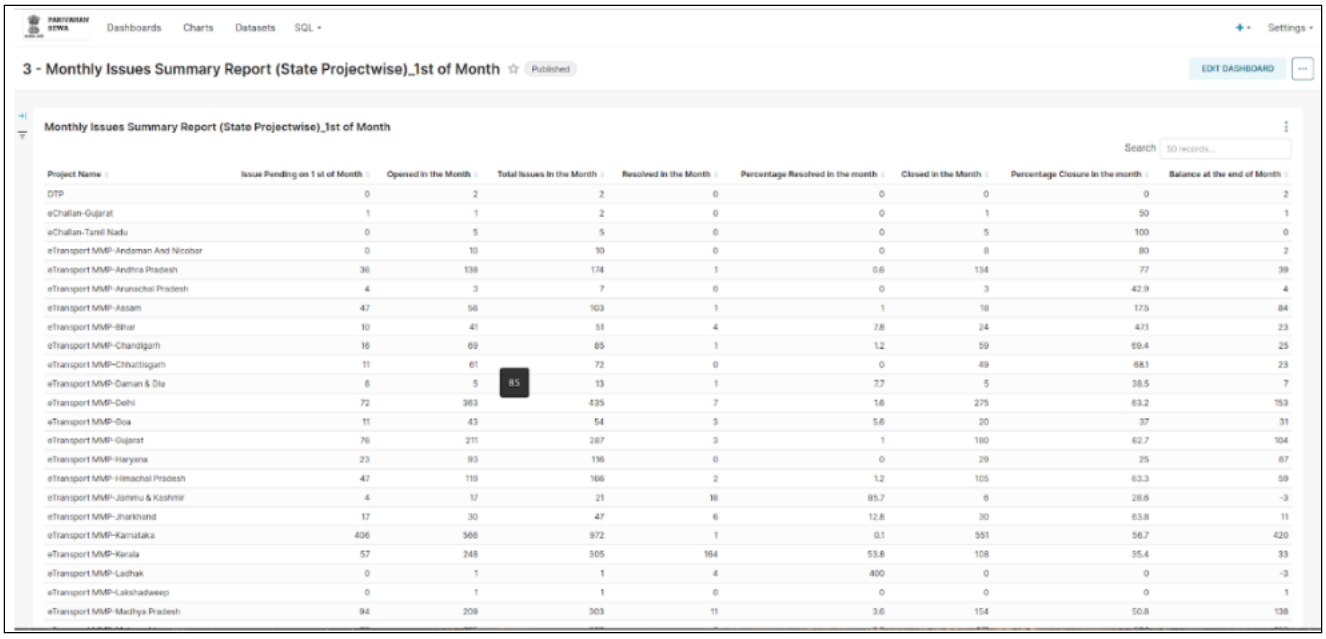


Figure_5.0:Dashboard for total no. of tickets pending on 1st of month, issues opened in the current month, total issues in the month, resolved and closed issues in the month

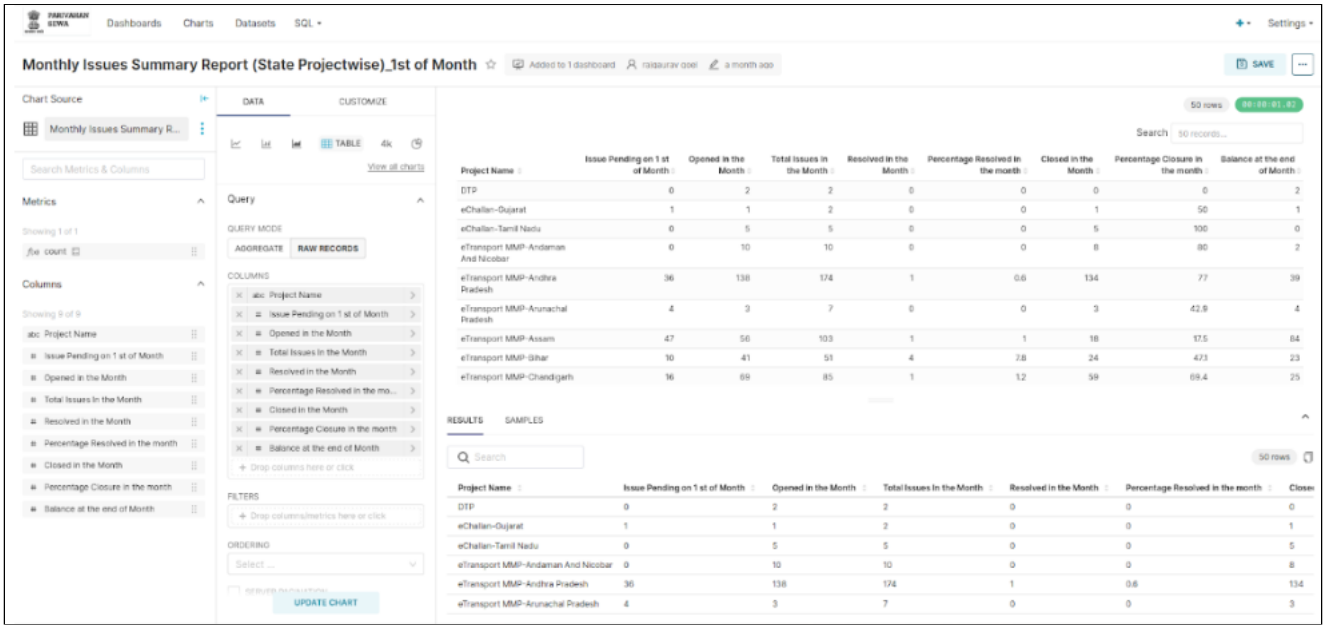


Figure_5.1:Chart of monthly analysis of ticket data

Visualization Type: The output of this query is visualized as a table to provide a clear and organized representation of user-specific issue details.

Insights and Observations:

- This report provides a snapshot of the state projects' issue management and resolution performance.
- The statistics offer insights into the progress of issue resolution and project-specific trends.
- The balance at the end of the month represents the remaining unresolved issues.
- By comparing the opened, resolved, and closed issues, project managers can evaluate their teams' performance.
- Trends in issue resolution and closure rates can guide resource allocation and improvement efforts.

Dashboard Title: Top 7 Projects Pending Tickets > 1 Week

Chart Name 1: Top 7 Projects Pending Tickets > 1 Week

Description:

The query retrieves information about the top 7 projects with pending issue-tickets that have been open for more than 1 week. It calculates the total count of such pending tickets for each project and presents the results in descending order of the pending ticket count.

SQL Query 1:

```
SELECT "Project Name" AS "Project Name",
       sum(count) AS "SUM(count)"
FROM
  (SELECT projects.name as "Project Name",
          count(issues.id)
   from issues
   left join projects on issues.project_id=projects.id
   where date(issues.created_on) < current_date - 7
     and issues.status_id not in(3,
                                 5,
                                 6,
                                 10)
   group by projects.name
   order by count DESC
   limit 7) AS virtual_table
GROUP BY "Project Name"
ORDER BY "SUM(count)" DESC
LIMIT 100;
```

Filters Used: The query incorporates the following filters:

- Filters data to include only issue-tickets that were created more than 1 week ago and have a status that is not closed or resolved (3,5,6,10).
  Metrics Used: The query employs the following metric:
- **count():** Counts the number of pending issue-tickets for each project that have been open for more than 1 week.
- **sum():** Summarizes the count of pending issue-tickets for each project.

Screenshots:

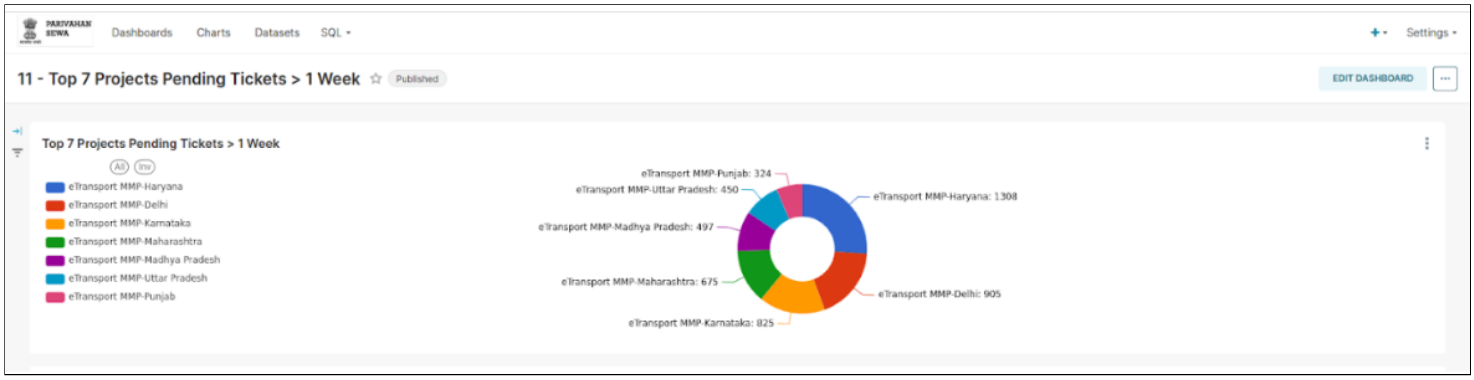


Figure_6.0:Dashboard of Top 7 projects having highest no. of tickets pending for more than 1 week.

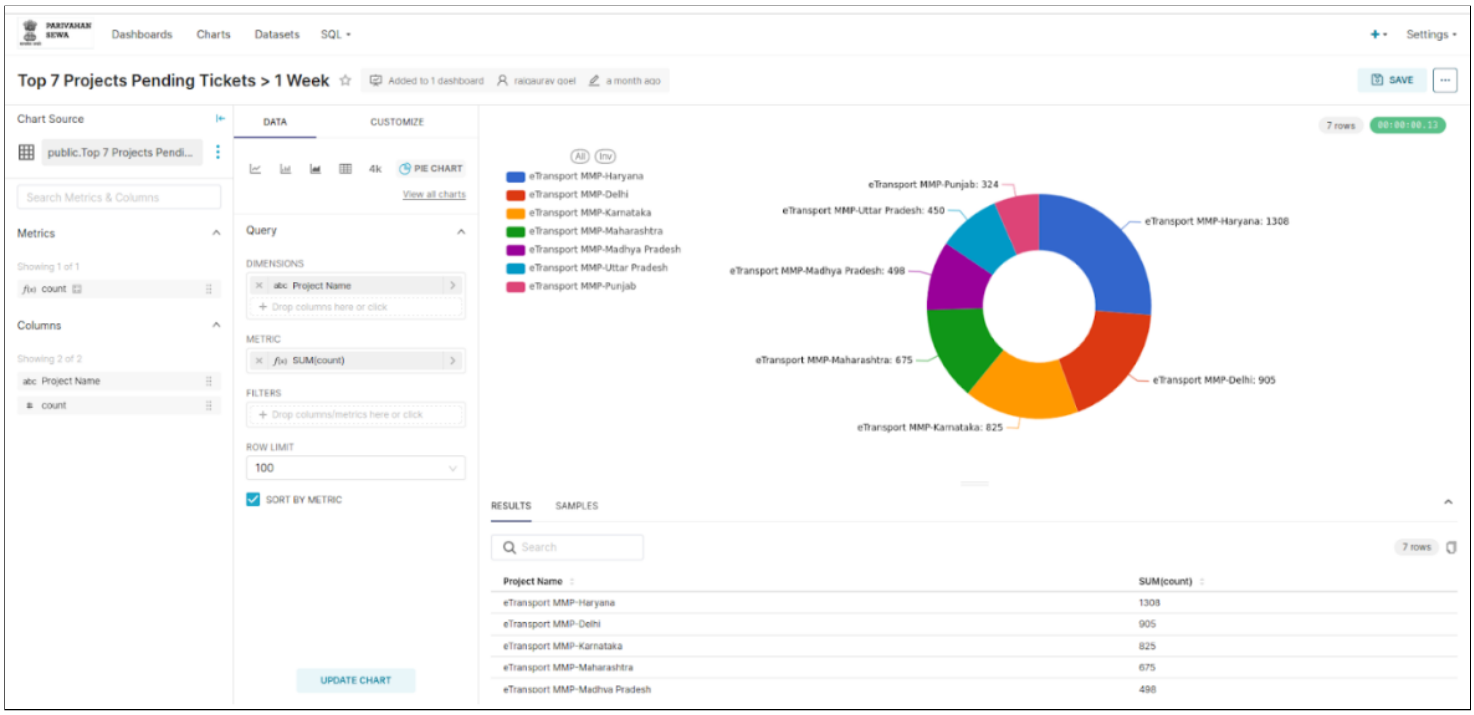


Figure_6.1:Column and metric for creating pie chart

Visualization Type: The output of this query is visualized as a Pie-chart to provide a clear and organized representation of project names having open tickets for more than a week.

Insights and Observations:

- This query highlights projects with a significant backlog of pending issue-tickets that have remained unresolved for over a week.
- Project managers and teams can quickly identify priority areas requiring immediate attention and intervention.
- The pie chart visualization can offer a clear comparison of pending tickets among the top projects.
- Projects with higher counts in the "SUM(count)" column are likely facing challenges in issue-ticket resolution.
- The pie chart can provide a visual cue to gauge the distribution of pending tickets across the top projects, aiding decision-making and resource allocation.

Chart Name 2: Tabular Representation - Top 7 Projects Pending Tickets > 1 Week

Description: The "Tabular Representation - Top 7 Projects Pending Tickets > 1 Week" query identifies the top 7 projects with the highest number of pending issue-tickets that have remained unresolved for over a week. It calculates the total count of these pending tickets for each project and presents the results in descending order of the ticket count. The query incorporates the use of the concat() function to generate clickable links for each project, leading to the dashboards having issues details which are pending for > 1 Week.

SQL Query 2:

```
SELECT "Project Name" AS "Project Name",
       concat('<a href="https://analyticsnew.parivahan.gov.in/superset/dashboard/older1week/', "Project Name", '" >', count, '</a>') AS count
FROM
  (SELECT projects.name as "Project Name",
          count(issues.id)
   from issues
   left join projects on issues.project_id=projects.id
   where date(issues.created_on) < current_date - 7
     and issues.status_id not in(3,
                                 5,
                                 6,
                                 10)
   group by projects.name
   order by count DESC
   limit 7) AS virtual_table
LIMIT 1000;
```

Filters Used: The query incorporates the following filters:

- The query filters issue-tickets created more than a week ago (greater than 7 days) to identify pending tickets.
- Issues with specific status IDs (3, 5, 6, 10) are excluded from the count.

Metrics Used: The query employs the following metric:

- **count():** Calculates the total number of pending issue-tickets for each project.
- **concat():** Combines total count of Pending Tickets > 1 Week to generate clickable links to individual Superset dashboards having details of issues.

Screenshots:

Figure_6.0.0:Dashboard having Top 7 projects having highest no. of tickets pending for more than 1 week

Figure_6.0.1:Dashboard having clickable link to individual-project dashboard having details of tickets pending for more than 1 week

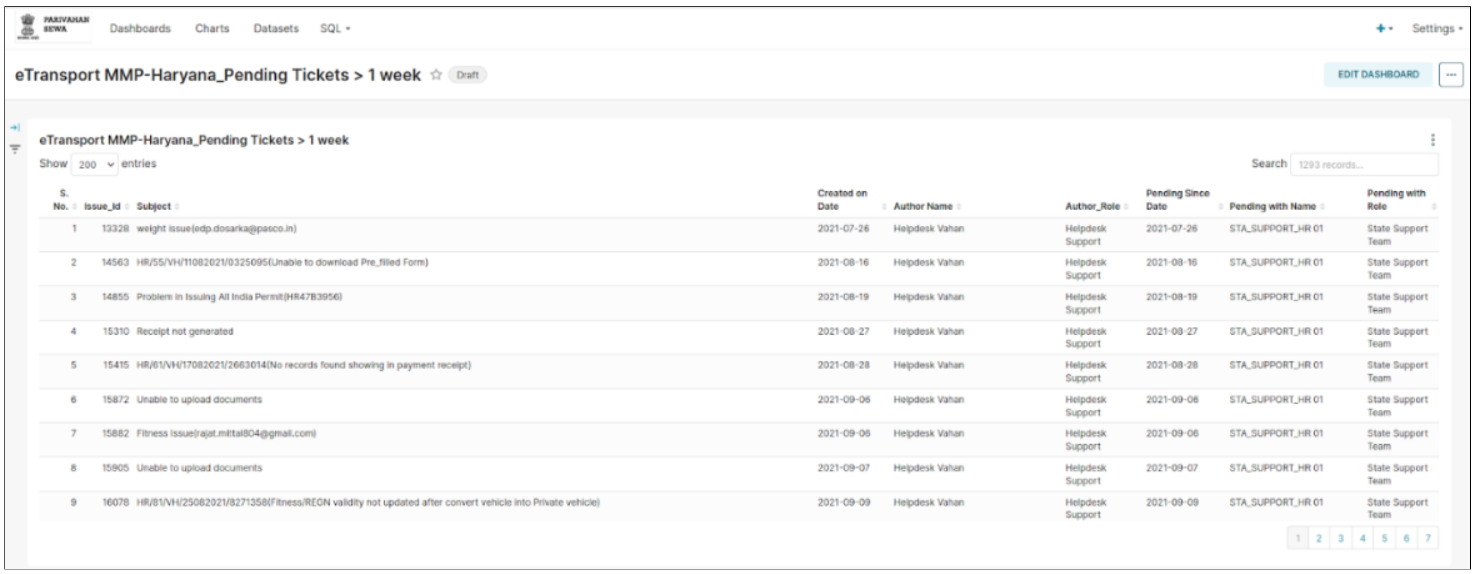


Figure_6.0.2:Project-wise Dashboard with details of tickets open for more than 1 week

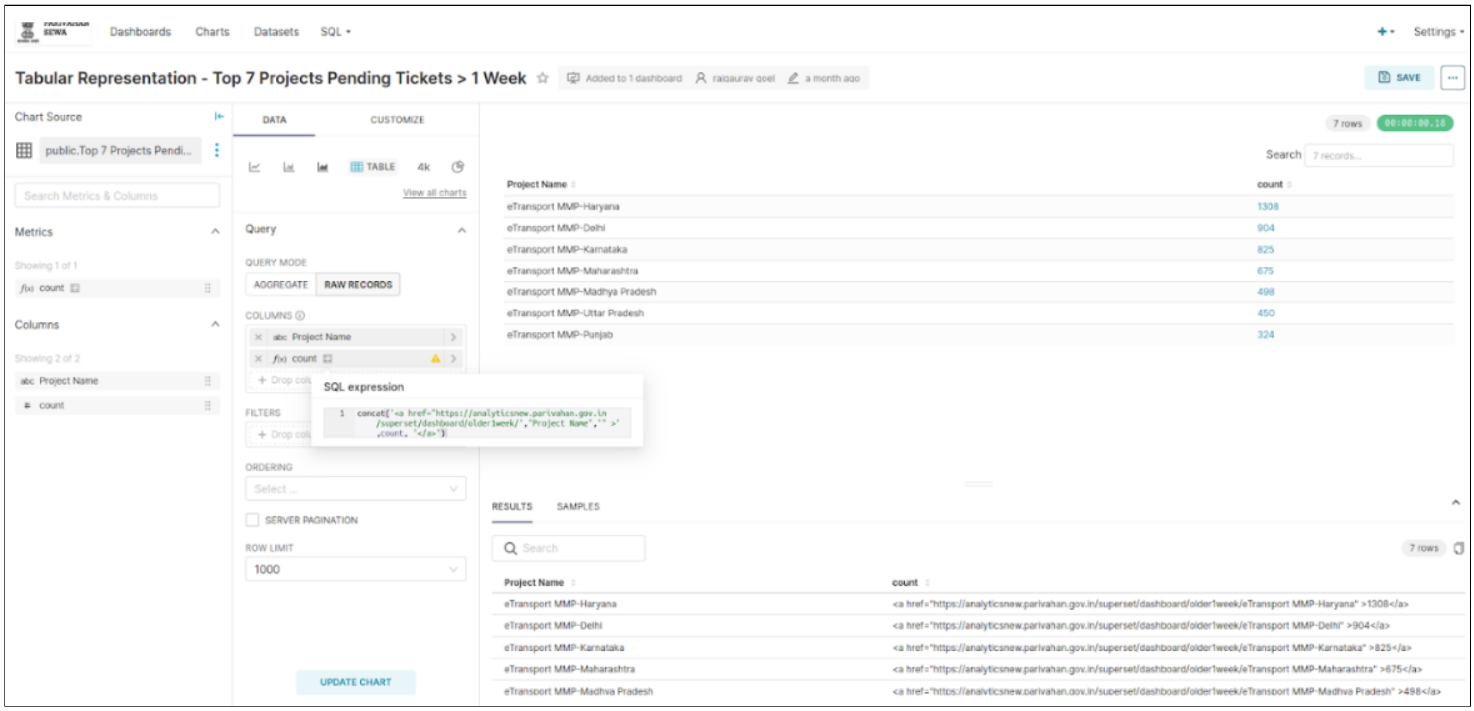


Figure_6.0.3:Chart column and metric

Insights and Observations:

- The table provides a direct view of the top projects with pending issue-tickets, allowing for quick assessment and prioritization.
- Users can access specific Superset dashboards for further analysis and resolution by clicking on the project names.
- Projects with higher counts in the "count" column have a larger number of unresolved pending tickets, requiring attention and action.
- The clickable links enhance user experience by providing direct access to relevant dashboards for deeper investigation.