Data Connector Integration Guide

Yeastar P-Series Appliance Edition

Version: 1.0
Date: 2026-05-25

+86-592-5503309 sales@yeastar.com www.yeastar.com

Contents

Data Connector Integration Overview

Data Connector on Yeastar P-Series PBX System allows you to export PBX data to a third-party SQL database (such as PostgreSQL, MySQL, or Microsoft SQL). The data synchronized to the database can then be further accessed by Business Intelligence (BI) tools, accounting systems, or other applications for data visualization and analytics.

Requirements

Server	Requirement
Yeastar PBX	Version 37.23.0.83 or later.
Third-party Database	One of the following database systems: • PostgreSQL • MySQL • Microsoft SQL

Supported data range for synchronization

Data	Description
CDR	Synchronize CDR list, call leg, and timeline data.
Recording Logs	Synchronize recording logs. **Note:** Recording files will not be synchronized.
System Metric	Synchronize the following system metrics displayed on the Dashboard. • **Active Calls** • **CPU Utilization** • **Memory Usage** • **Local Storage Usage** • **Registered Extensions** • **SIP Trunks Available** • **Linkus Client Logins** **Note:** The system synchronizes the data captured at the time of synchronization.

Data	Description
Call Notes	Synchronize call disposition and remark content.
Report Data	Synchronize call report data.

Tutorial

Data Connector integration with Yeastar P-Series PBX System can be implemented in just a few clicks by configuring the database connection information and synchronization policy.

For more information, see Connect Yeastar P-Series PBX System to an SQL Database Using Data Connector.

Note:
After the integration is complete, you can connect the third-party database with Business Intelligence (BI) tools, accounting systems, or other applications for data visualization and analytics. We provide detailed instructions for the database integration with Grafana and Power BI. For more information, see Import PBX Data from a Database into Grafana and Import PBX Data from a Database into Power BI.

Connect Yeastar P-Series PBX System to an SQL Database Using Data Connector

This topic describes how to configure data connector on Yeastar P-Series PBX System to export PBX data to a third-party SQL database.

Requirements

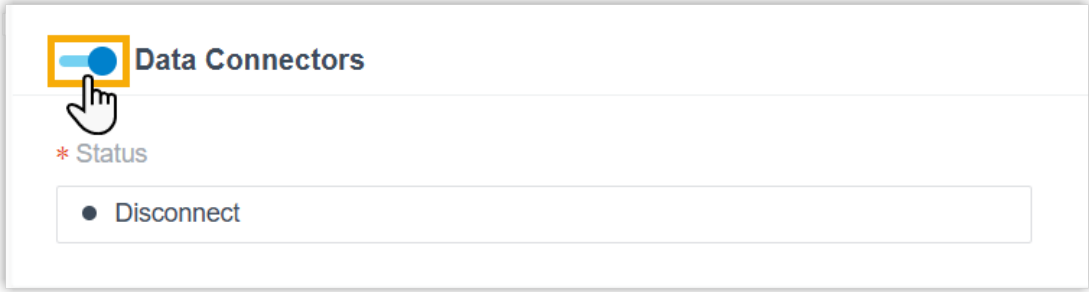
The firmware version of Yeastar P-Series PBX System is 37.23.0.83 or later.

Prerequisites

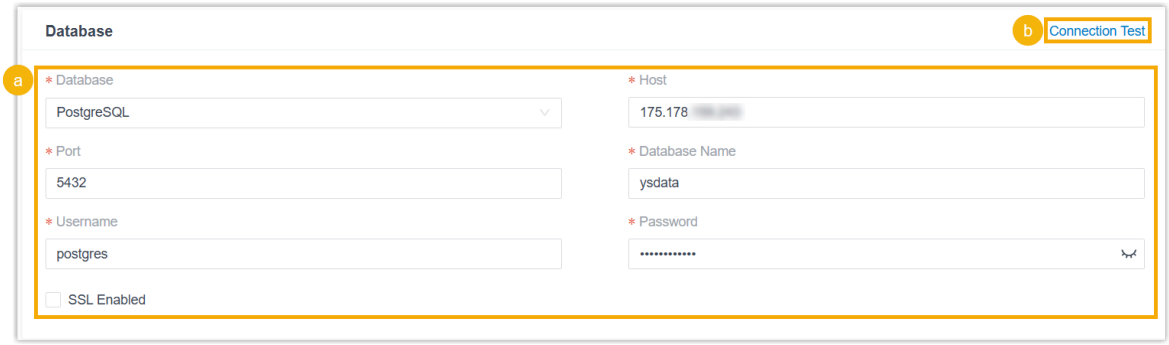
- You have set up one of the following database systems:
  - PostgreSQL
  - MySQL
  - Microsoft SQL
- You have created a database to store the PBX data to be synced.

Procedure

1. Log in to PBX web portal, go to **Integrations > Data Connectors**.
2. Turn on the switch of **Data Connectors**.



3. In the **Database** section, configure the connection to a third-party database.



a. Complete the following settings to establish the database connection.

Item	Description
Database	Select a database type. • **PostgreSQL** • **MySQL** • **Microsoft SQL**
Host	Enter the IP address or domain name of the host on which the database is installed.
Port	Enter the database port.
Database Name	Enter the name of the database.
Username	Enter the username used to connect to the database.
Password	Enter the password used to connect to the database.
SSL Enabled	Enable or disable SSL certificate verification. If enabled, click **Browse** to select and upload the SSL certificate.

b. **Optional:** Click **Connection Test** to verify that the PBX can connect to the database successfully.

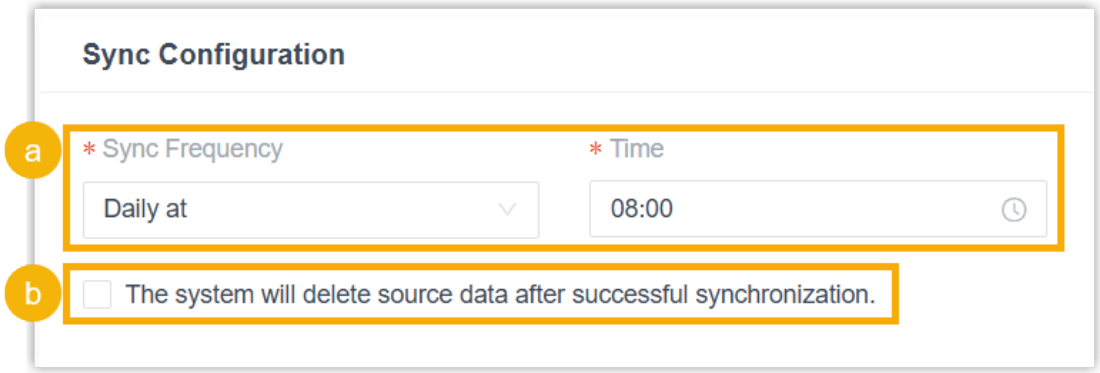
If "Connection test successful" is displayed, it indicates the connection to database is successful.

4. In the **Data Range** section, select the data to synchronize to the third-party database.

Data	Description
CDR	Synchronize CDR list, call leg, and timeline data.
Recording Logs	Synchronize recording logs. **Note:** Recording files will not be synchronized.

Data	Description
System Metric	Synchronize the following system metrics displayed on the Dashboard. • **Active Calls** • **CPU Utilization** • **Memory Usage** • **Local Storage Usage** • **Registered Extensions** • **SIP Trunks Available** • **Linkus Client Logins** **Note:** The system synchronizes the data captured at the time of synchronization.
Call Notes	Synchronize call disposition and remark content.
Report Data	Synchronize call report data.

5. In the **Sync Configuration** section, configure the synchronization frequency and data cleanup policy.



   a. In the **Sync Frequency** drop-down list, select how often data will be synchronized to the third-party database.

   Note: The system synchronizes data at the selected interval based on the default time zone.

   b. **Optional:** To delete the synced data from PBX once synchronization is complete, select the checkbox of **The system will delete source data after successful synchronization**.

> **Note:**
> If source CDR is deleted, you can NO LONGER access the associated call records from PBX.

6. Perform the initial data synchronization to the third-party database.
    a. Click **Sync Now**.
    b. In the pop-up window, click **OK**, then click **Confirm**.

    Wait for the data synchronization to complete.

Result

The integration status is displayed as **Connected**, indicating that the connection to the third-party database is established and one-way data synchronization to the database is working properly.

> **Note:**
> You can connect the third-party database with Business Intelligence (BI) tools, accounting systems, or other applications for data visualization and analytics. We provide detailed instructions for the database integration with Grafana and Power BI. For more information, see Import PBX Data from a Database into Grafana and Import PBX Data from a Database into Power BI.

Import PBX Data from a Database into BI Tools

Import PBX Data from a Database into Grafana

When PBX data is synchronized to a third-party database, you can add the database as a data source in Grafana, then use the Yeastar-provided dashboard template to import and visualize your data on Grafana.

Requirements

Platform	Requirement
Grafana	An account with **Organization administrator** role.
Third-party Database	• **Version**: The database version is supported by Grafana. For more information, see the following reference links: ◦ Supported Microsoft SQL Server versions ◦ Supported MySQL-compatible databases ◦ Supported PostgreSQL-compatible databases • **Network**: The network connectivity between database and Grafana is established. • **Account**: An account with read-only access (SELECT permissions) on the required schemas and tables. **Note:** Grafana does not validate the safety of queries, so users could run potentially harmful SQL. We recommend that you create a dedicated user with restricted permissions to limit risk.

Prerequisites

You have synchronized PBX data to a third-party database.

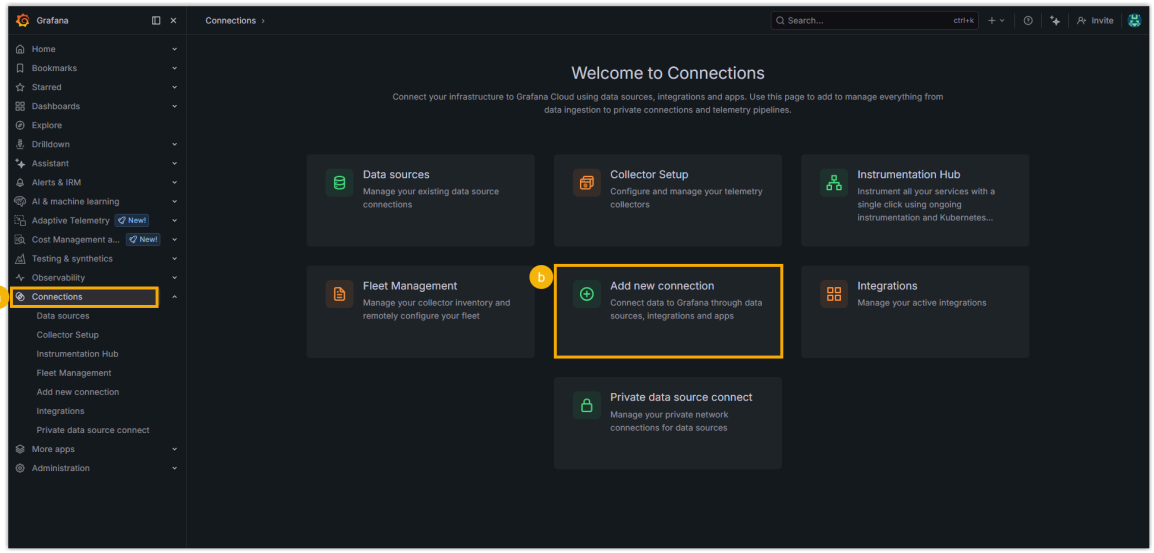
Procedure

Note:

> This topic takes **PostgreSQL** as an example to demonstrate how to import PBX data from a PostgreSQL-compatible database into Grafana. For **Microsoft SQL** and **MySQL**, the operations are basically the same.

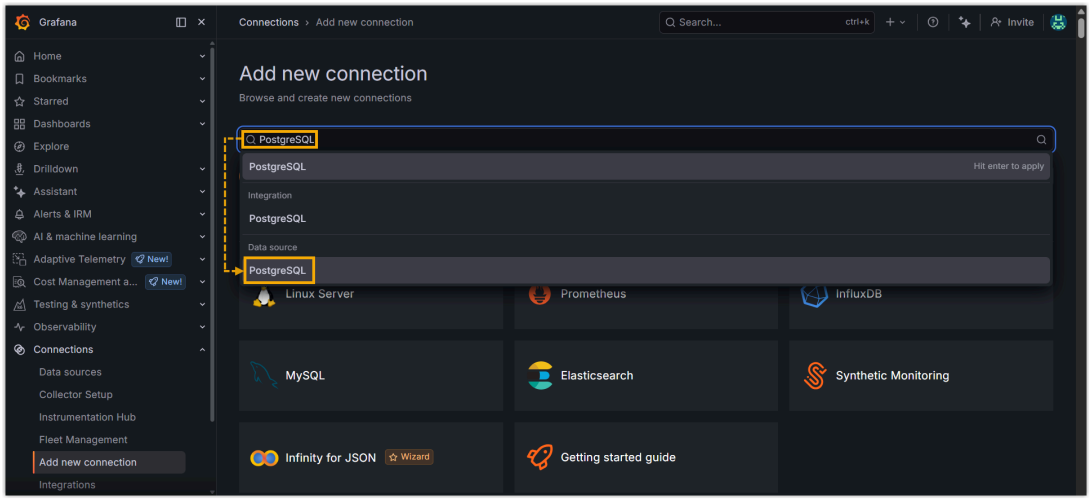
Step 1. Connect Grafana to database

1. Log in to Grafana portal, go to **Connections > Add new connection**.

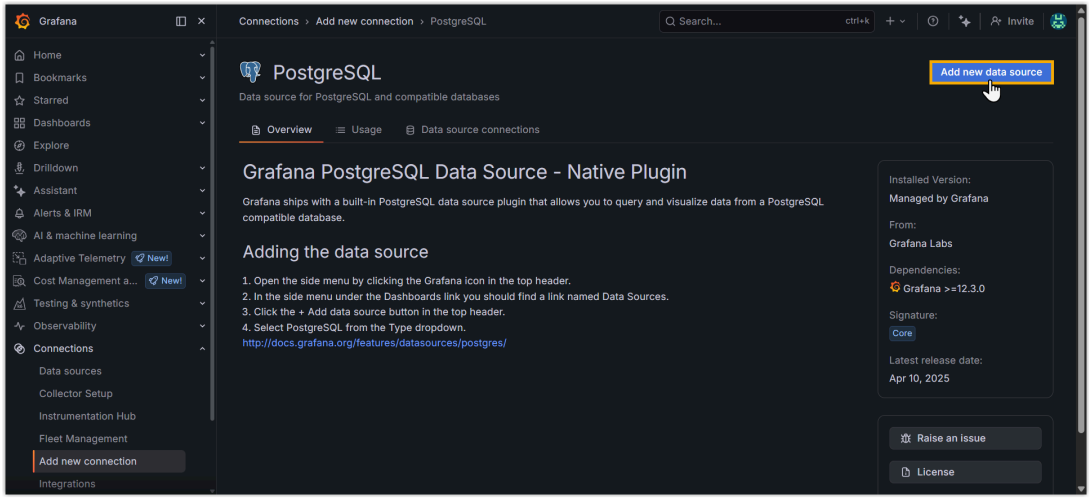


2. Add a data source.
    a. Search for the target database type and select it from the list of available data sources.

       In this example, select **PostgreSQL** data source.



b. At the top-right corner, click **Add new data source**.



3. Fill in the following information to connect to the database.
   a. In the **Connection** section, fill in the database connection information.

Item	Description
Host URL	Enter the IP address or domain name of the host on which the database is installed, as well as the database port.
Database name	Enter the name of the database.

   b. In the **Authentication** section, fill in the authentication information.

Item	Description
Username	Enter the username used to connect to the database.
Password	Enter the password used to connect to the database.

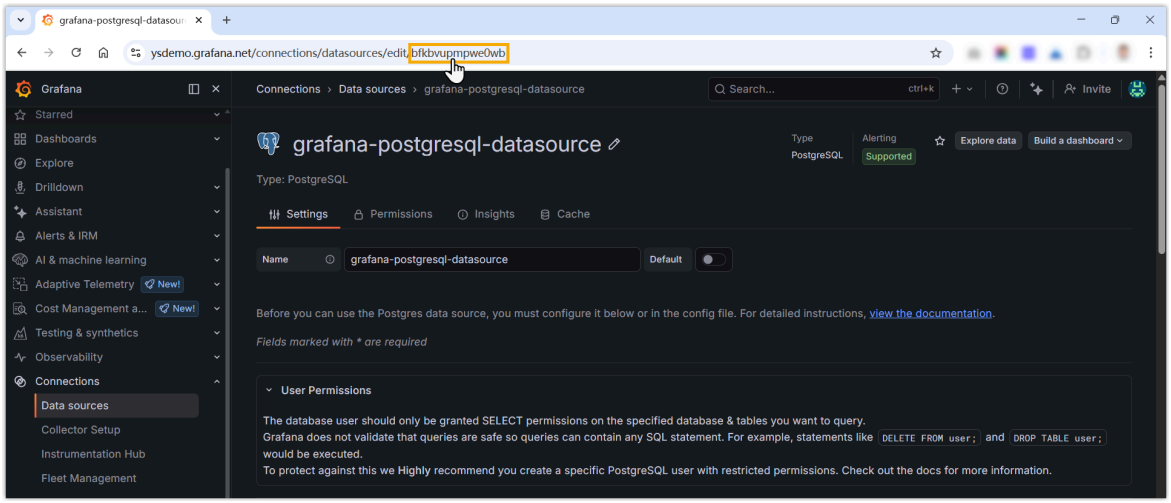
4. **Optional:** Configure additional settings as needed.

5. Scroll down to the bottom of the page, click **Save & test** to test and save the data source connection.

   If "Database Connection OK" is displayed, it indicates that the connection to the database is successful.

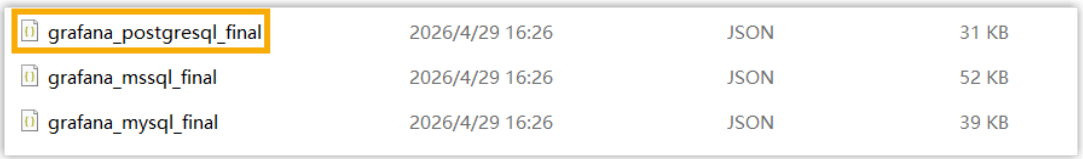
Step 2. Import a dashboard from the database using a template

1. In the browser's address bar, copy the data source UID (the last segment of the URL) for template configuration.

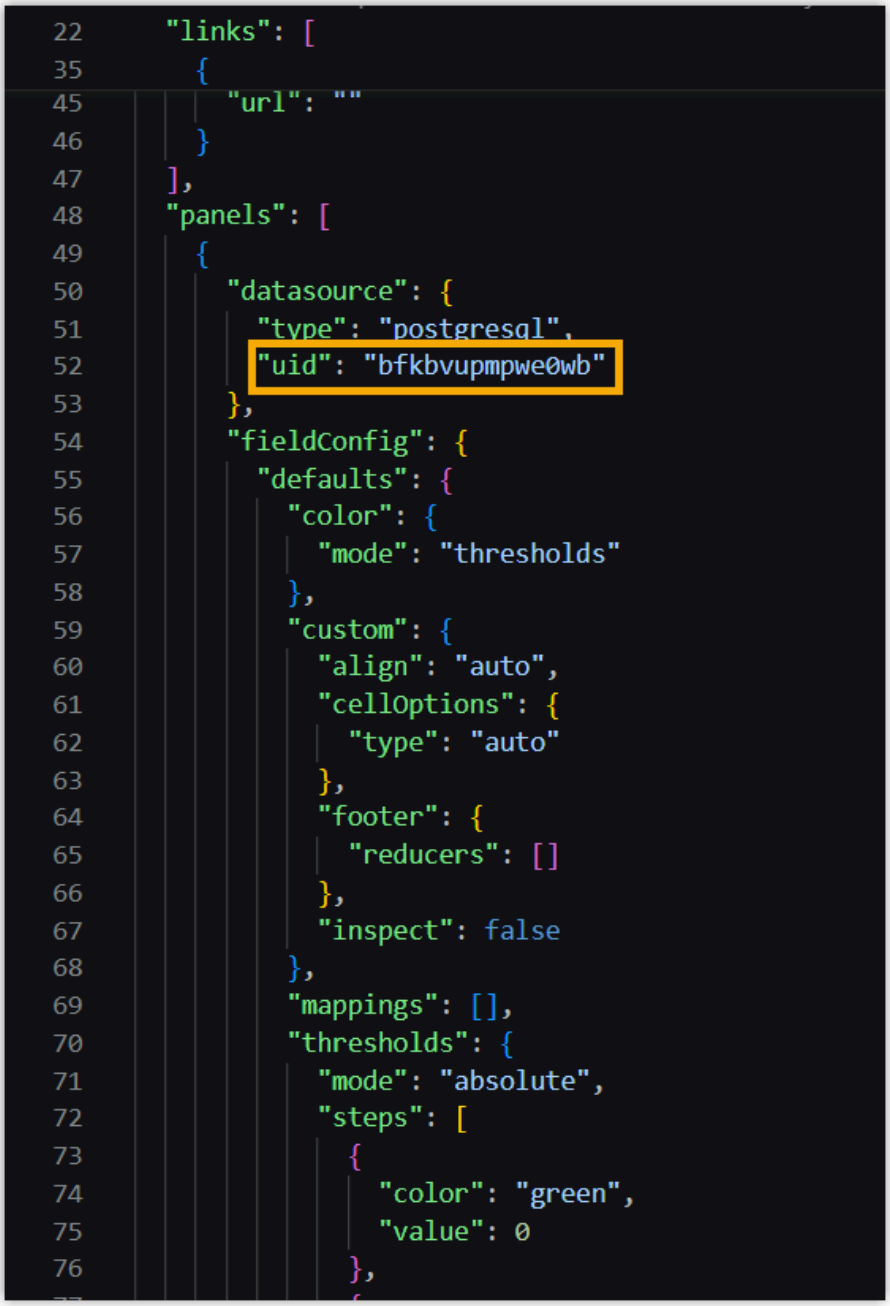


   In this example, the value we retrieve is `bfkbvupmpwe0wb`.
2. Download and update the dashboard JSON template.
   a. Download and unzip the dashboard template package, which includes templates for different databases, and select the appropriate one based on your database.

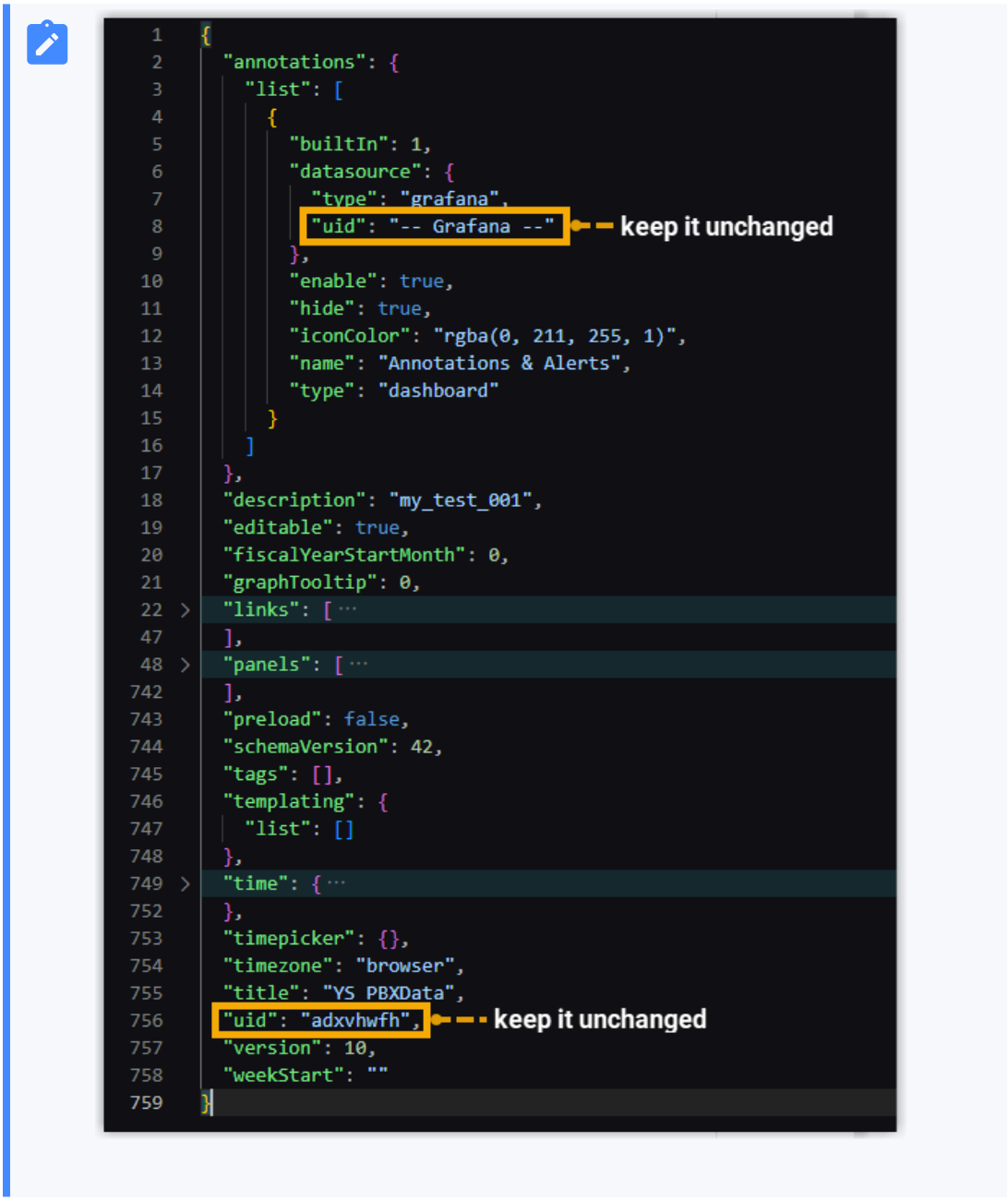
      In this example, we will use the template for PostgreSQL.



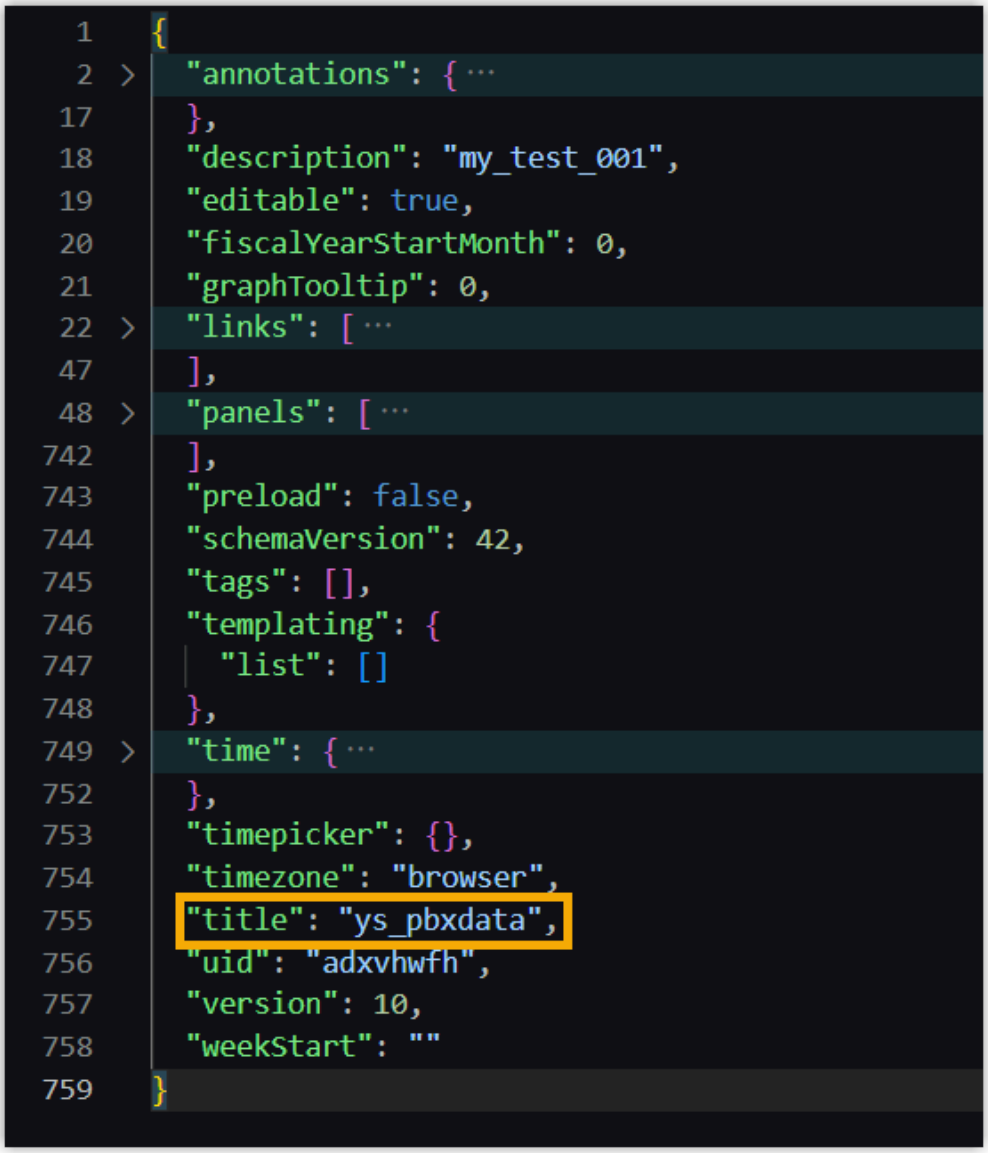
   b. Replace all `uid` values under `datasource` in each panel with the retrieved values.



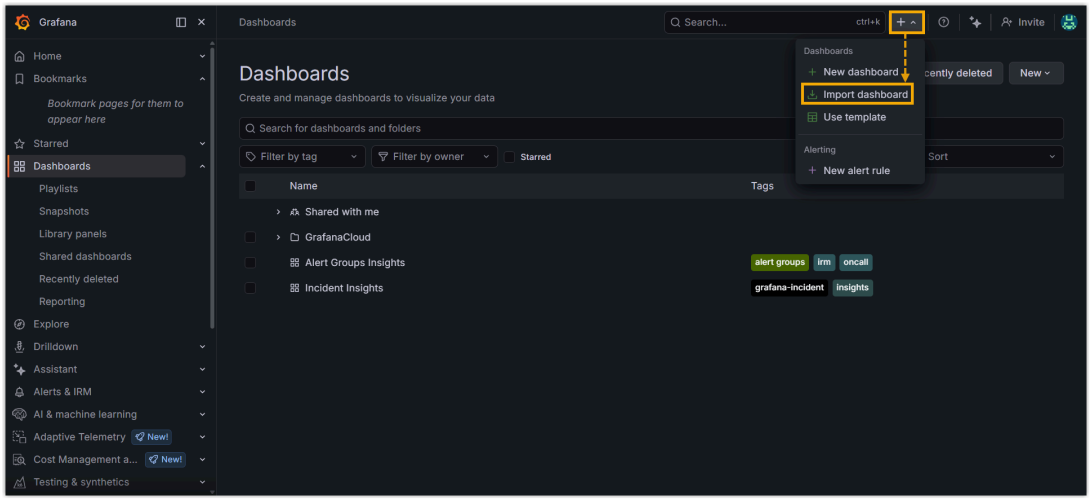
Note:
Leave the `uid` values in annotations and dashboard settings unchanged.



c. **Optional:** Update the `title` value, which will be used as the dashboard name in Grafana.

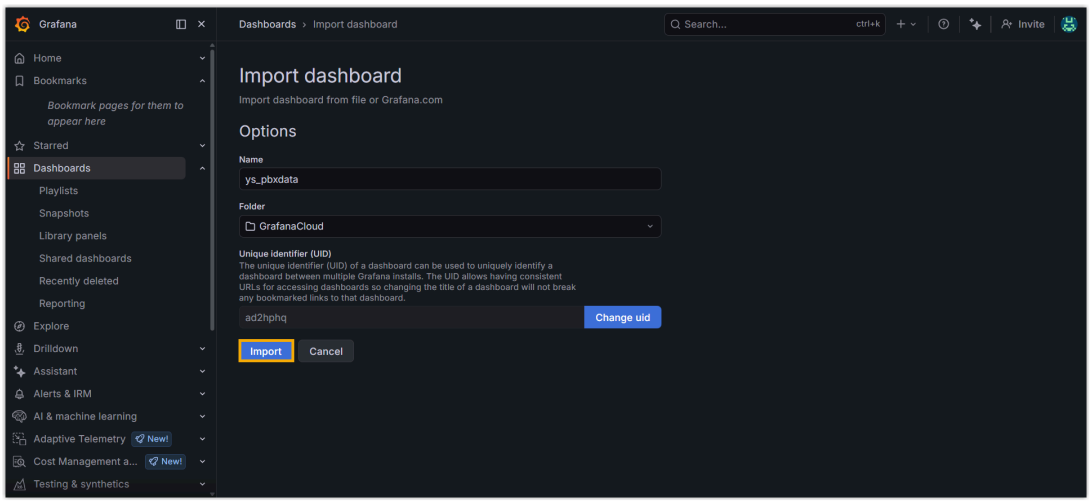


3. Import the dashboard template to Grafana.
    a. On the left navigation bar, click **Dashboards**.
    b. At the top-right corner, click + and select **Import dashboard**.



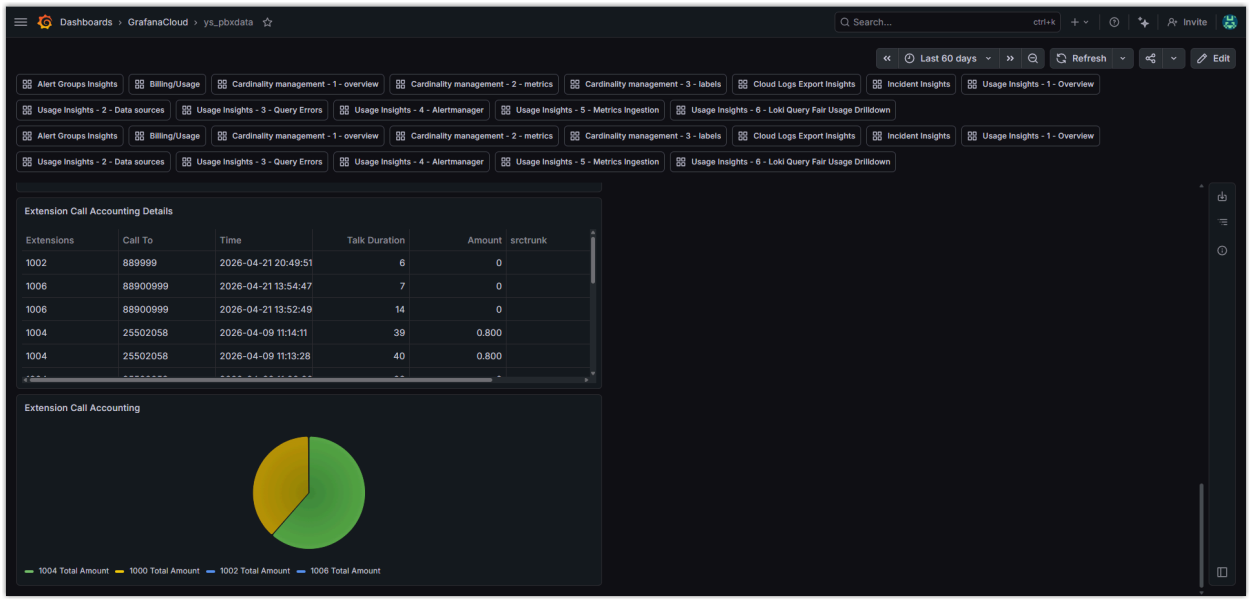
c. Click **Upload dashboard JSON file** and upload the `.json` file.

d. Click **Import**.



Result

The dashboard is successfully imported and data is visualized in the Dashboard.



What to do next

Add or modify the SQL queries to customize the data to be displayed.

Note:

By default, the Yeastar-provided dashboard template for Grafana displays data only for the following call reports, as Grafana dashboard doesn't support incorporating multiple queries from a single source.

- **Extension Call Accounting**
- **Extension Call Accounting Details**
- **Agent Missed Call Activity**
- **Queue Callback Activity**
- **Satisfaction Survey**
- **Satisfaction Survey Details**
- **IVR Report**
- **DID/Outbound Caller ID Activity**

To display additional call reports, you can add SQL queries as needed. Refer to Call Report Calculations with Multiple SQL Queries for more details on the related data logic.

Import PBX Data from a Database into Power BI

When PBX data is synchronized to a third-party database, you can add the database as a data source in Power BI, then use the Yeastar-provided template to import and visualize your data on Power BI.

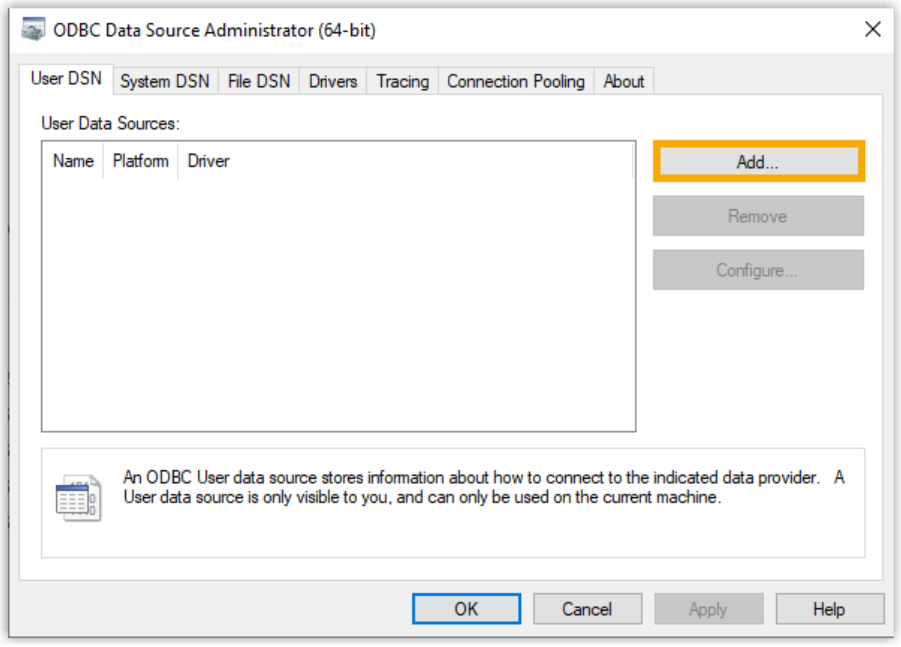
Import PBX data from PostgreSQL into Power BI

Prerequisites

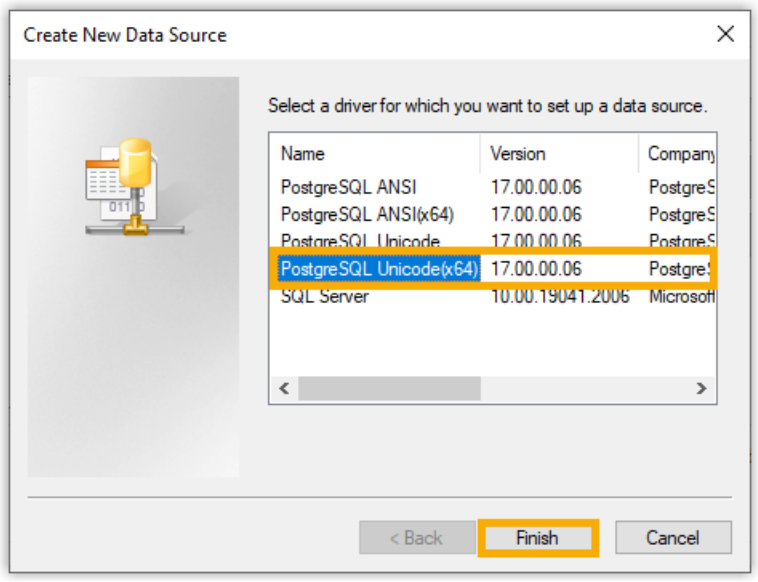
- Download and unzip the Power BI template, which includes templates for different databases, and select the appropriate one based on your database.
- Download and install Power BI desktop and PostgreSQL ODBC driver.

Procedure

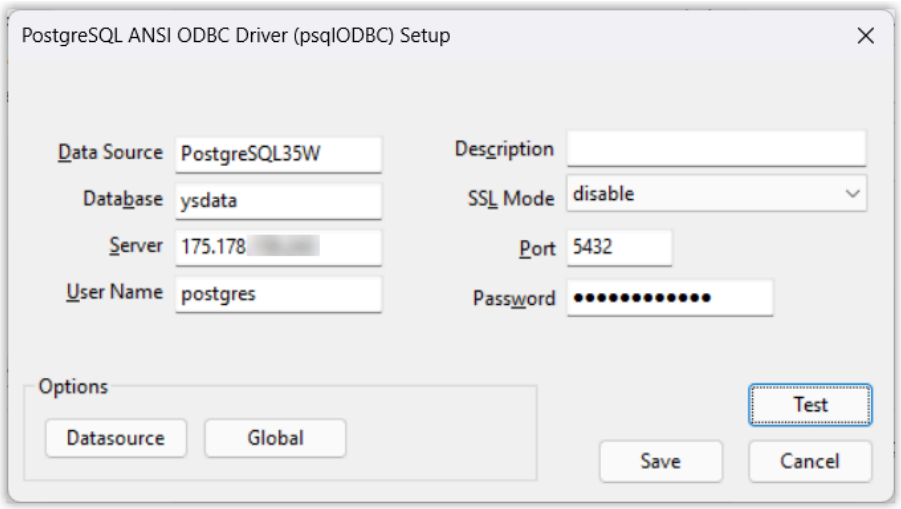
1. Connect to the database.
   a. Open **ODBC Data Sources (64-bit)**.
   b. In the **User DSN** tab, click **Add**.



   c. In the pop-up window, select **PostgreSQL Unicode(x64)**, then click **Finish**.

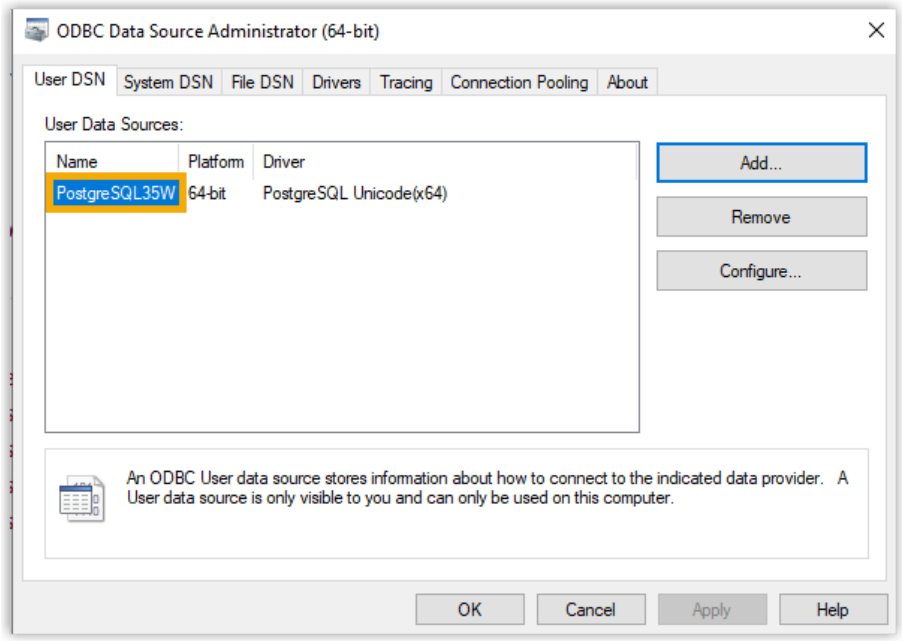


d. Fill in the following information, then click **Save**.

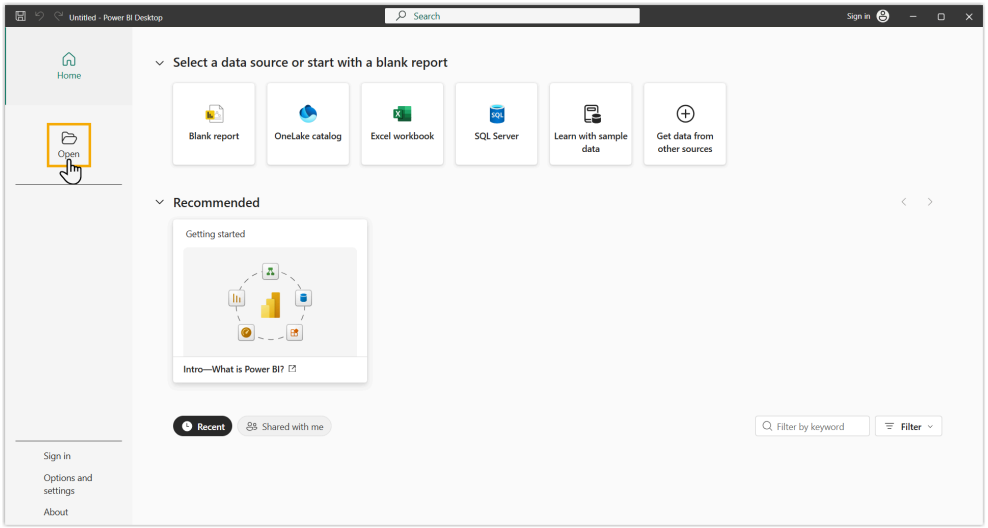


Item	Description
Database	Enter the name of the database.
Server	Enter the IP address or domain name of the host on which the database is installed.
Port	Enter the database port.
User Name	Enter the username used to connect to the database.
Password	Enter the password used to connect to the database.

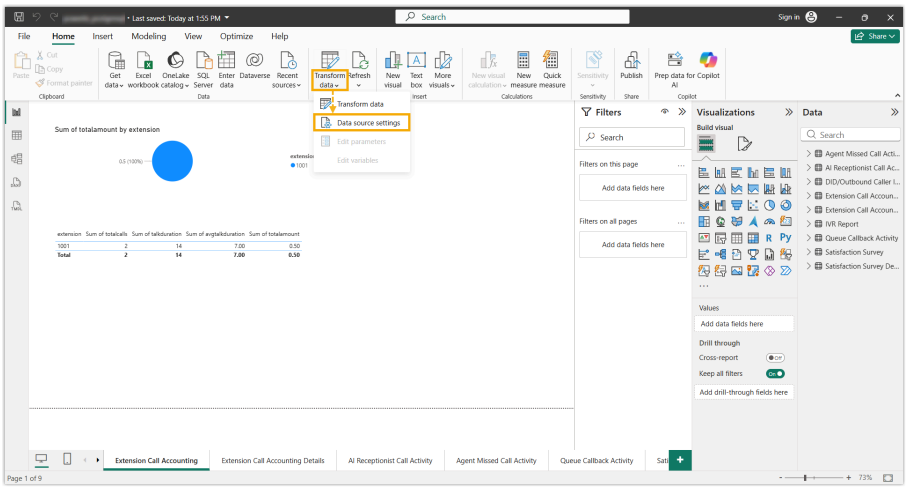
e. Remember the DSN name on the list, which will be used later.



2. Launch Power BI desktop.
3. Click **Open** to select and open the Yeastar-provided template.

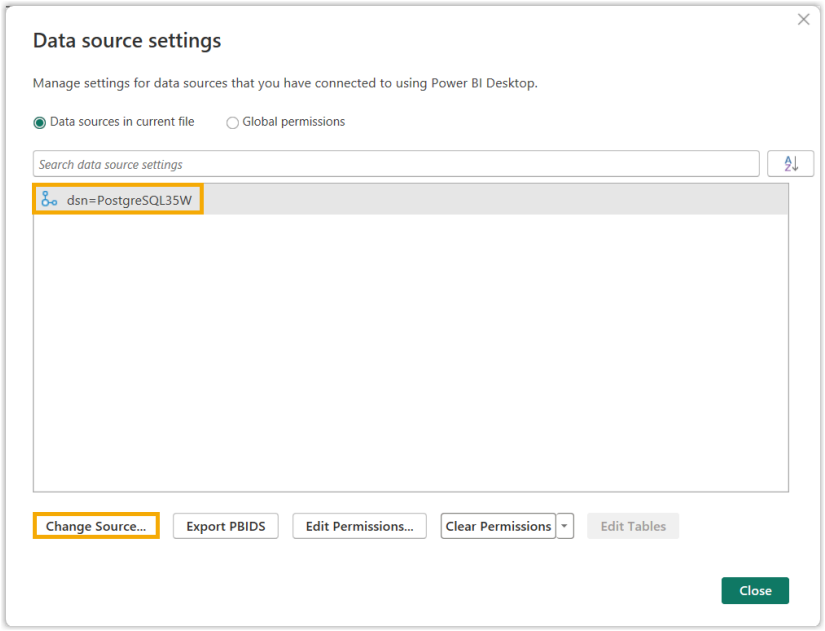


4. Change the data source and configure database access.
    a. On the top toolbar, click **Transform data** and select **Data source settings**.

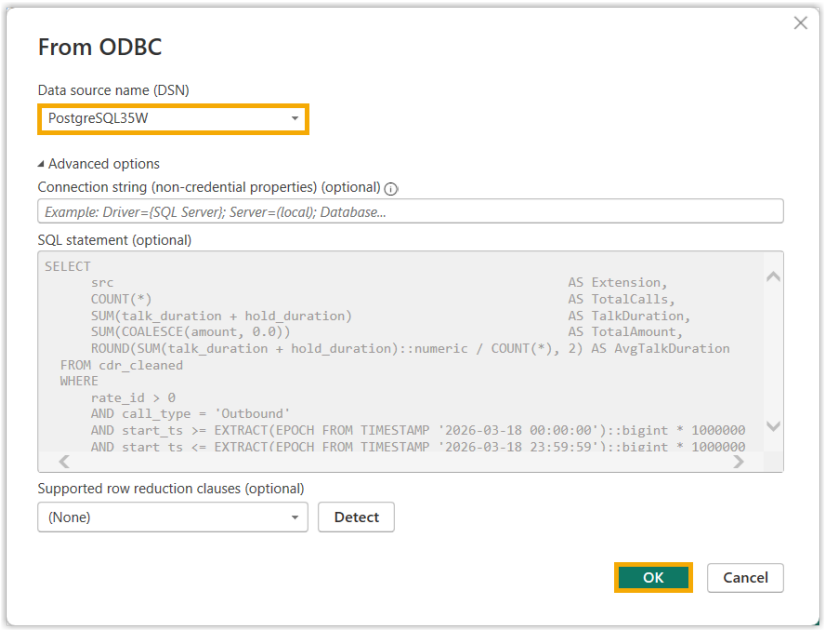


b. Change the data source.

i. Select the default data source, then click **Change Source**.

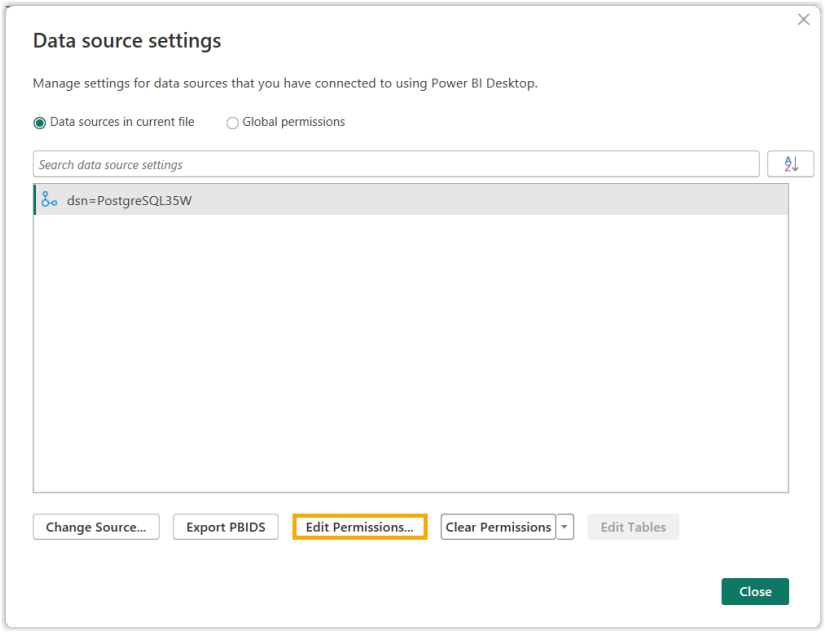


ii. In the **Data source name (DSN)** drop-down list, select the one for the database, then click **OK**.

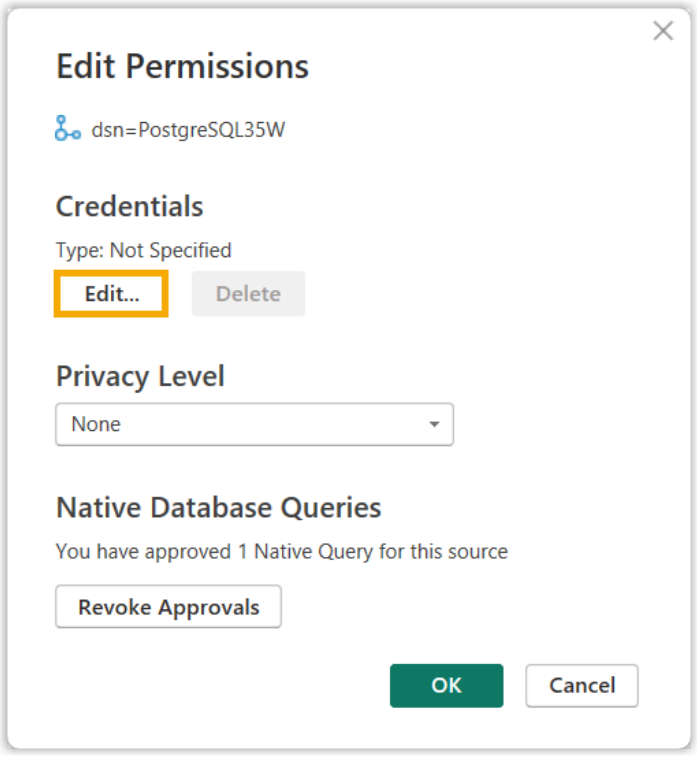


c. Configure database access credentials.

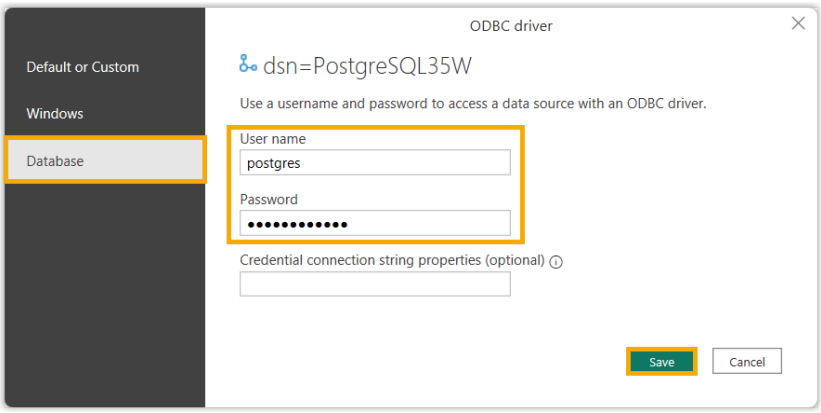
i. Click **Edit Permissions**.



ii. In the pop-up window, click **Edit**.



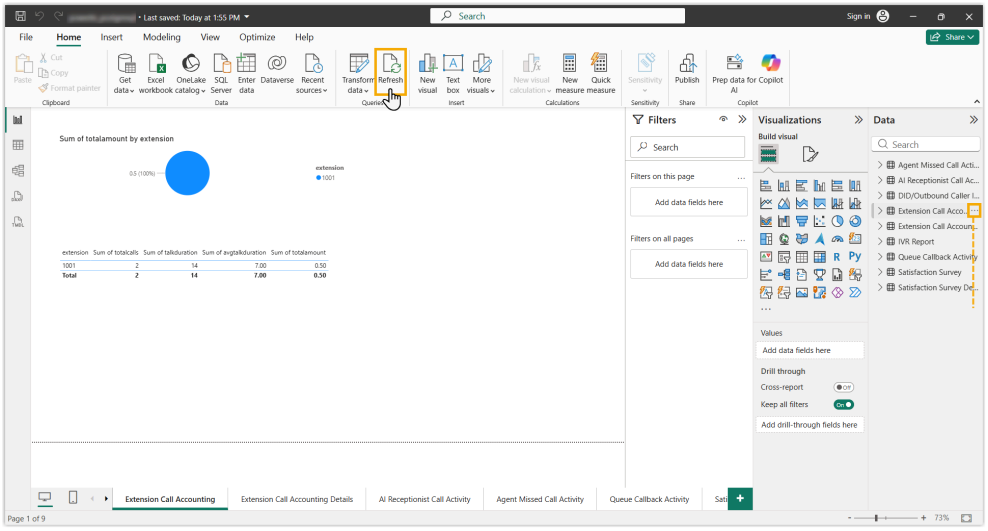
iii. Select **Database** and fill in the authentication information, then click **Save**.



Item	Description
User name	Enter the username to connect to the database.
Password	Enter the password to connect to the database.

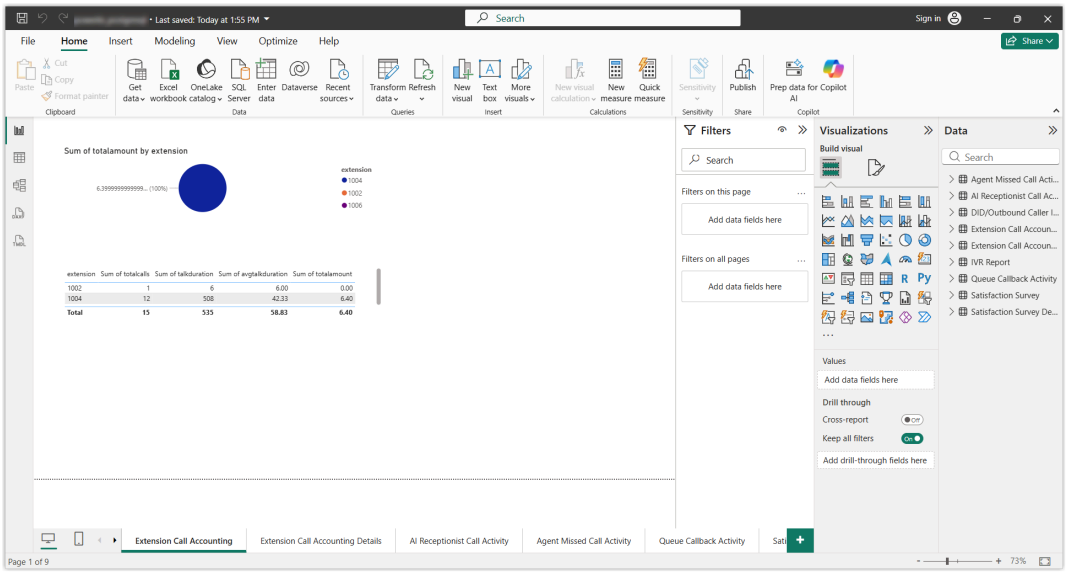
d. Save your setting .

5. On the top of the page, click **Refresh** to run all SQL queries.



Result

The data are successfully imported and visualized in the panel.



What to do next

Add or modify the SQL queries to customize the data to be displayed.

> **Note:**
> By default, the template for Power BI displays data only for the following call reports, as Power BI doesn't support incorporating multiple queries from a single source.

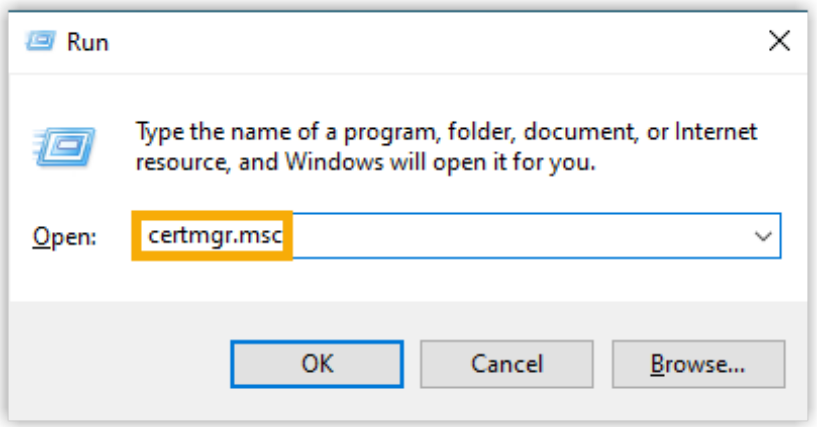
- **Extension Call Accounting**
- **Extension Call Accounting Details**
- **Agent Missed Call Activity**
- **Queue Callback Activity**
- **Satisfaction Survey**
- **Satisfaction Survey Details**
- **IVR Report**
- **DID/Outbound Caller ID Activity**

To display additional call reports, you can add SQL queries as needed. Refer to Call Report Calculations with Multiple SQL Queries for more details on the related data logic.

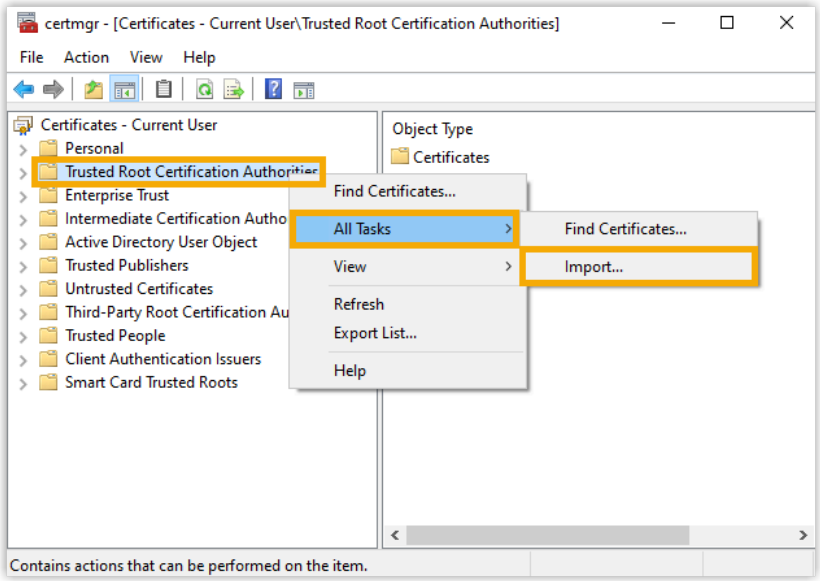
Import PBX data from MySQL into Power BI

Prerequisites

- Download and unzip the Power BI template, which includes templates for different databases, and select the appropriate one based on your database.
- Download and install Power BI desktop and MySQL Connector/NET.
- (Optional) If SSL is enabled on the MySQL server, install the certificate.
    1. Download the SSL public certificate.
    2. Install the SSL certificate in Trusted Root certification authorities store.
        a. Run `certmgr.msc` on your computer to open the Certificate Manager.



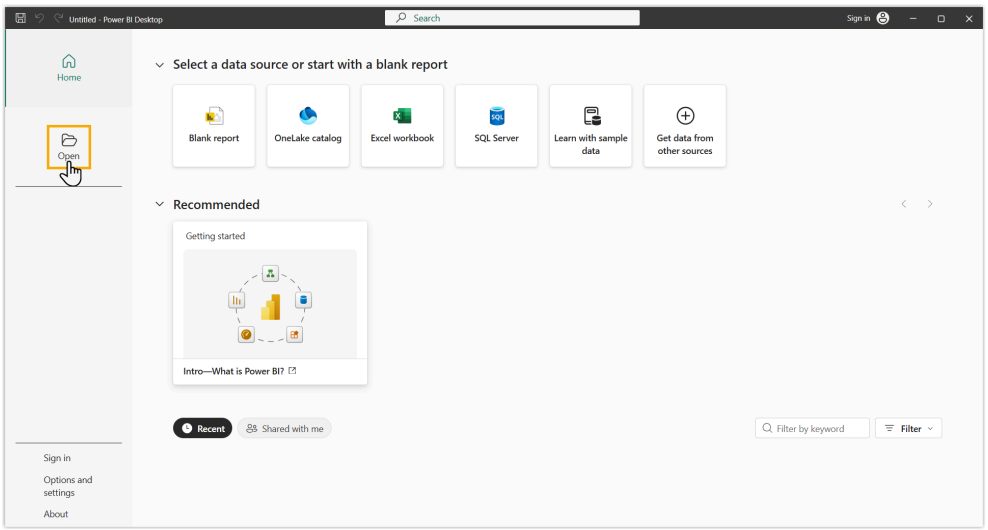
b. Right click **Trusted Root Certification Authorities** and select **Import** from **All Tasks**.



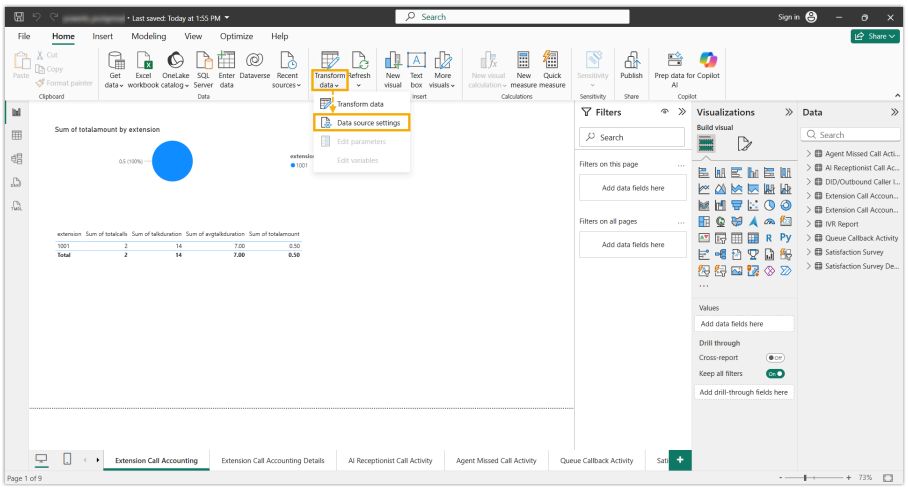
c. Follow the prompts in the wizard to import the root certificate, then click **OK**.

Procedure

1. Launch Power BI desktop.
2. Click **Open** to select and open the Yeastar-provided template.

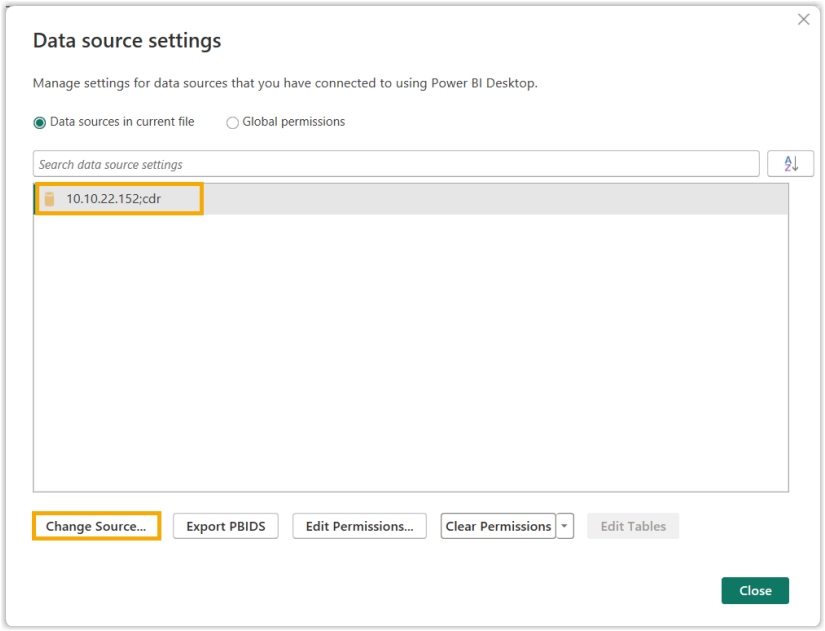


3. Change the data source and configure database access.
    a. On the top toolbar, click **Transform data** and select **Data source settings**.

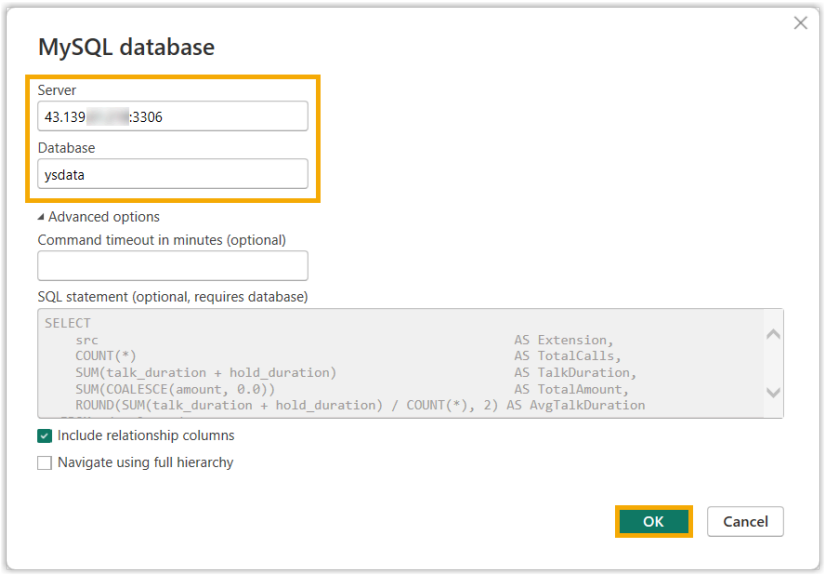


b. Change the data source.

i. Select the default data source, then click **Change Source**.



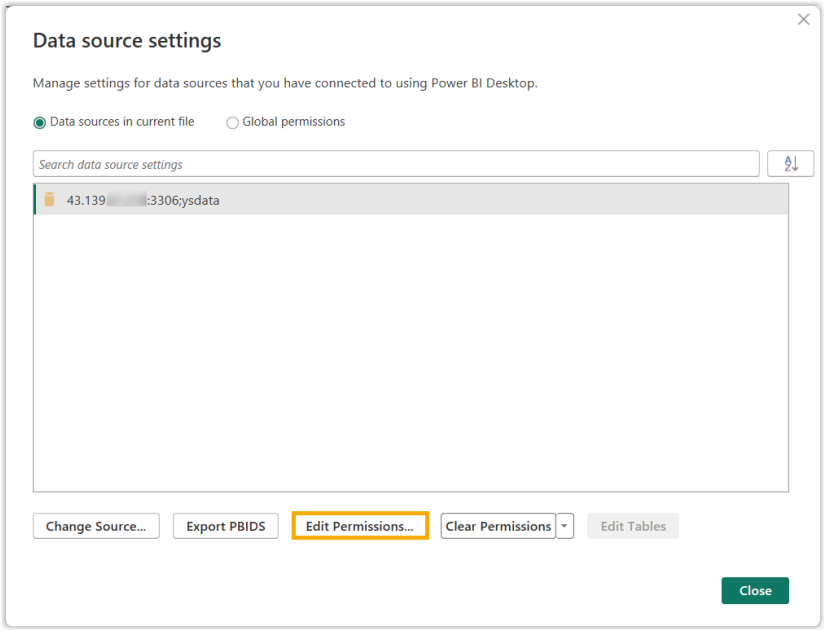
ii. Update the database connection information, then click **OK**.



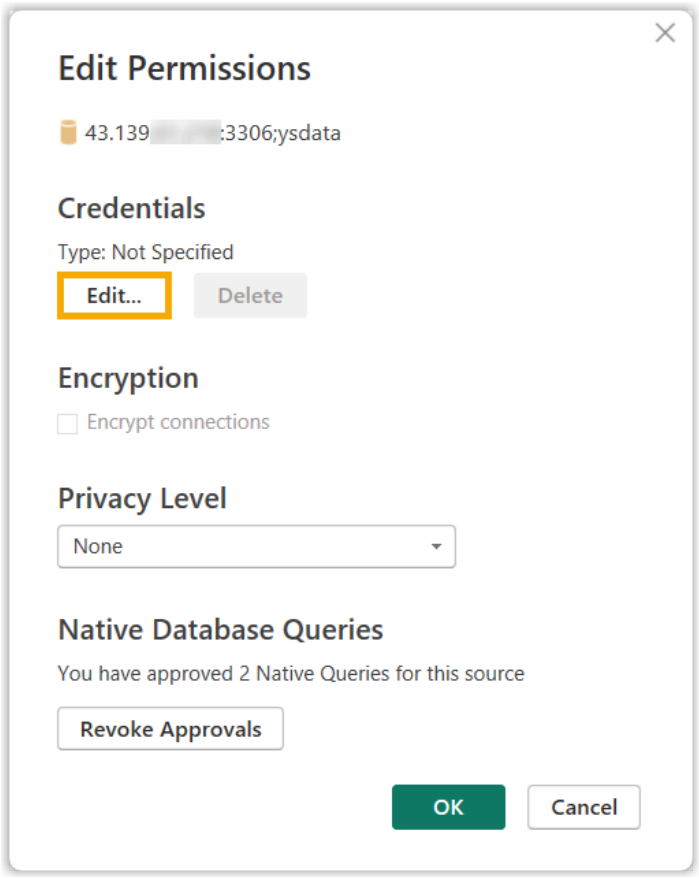
Item	Description
Server	Enter the IP address or domain name of the host on which the database is installed, as well as the database port.
Database	Enter the name of the database.

c. Configure database access credentials.

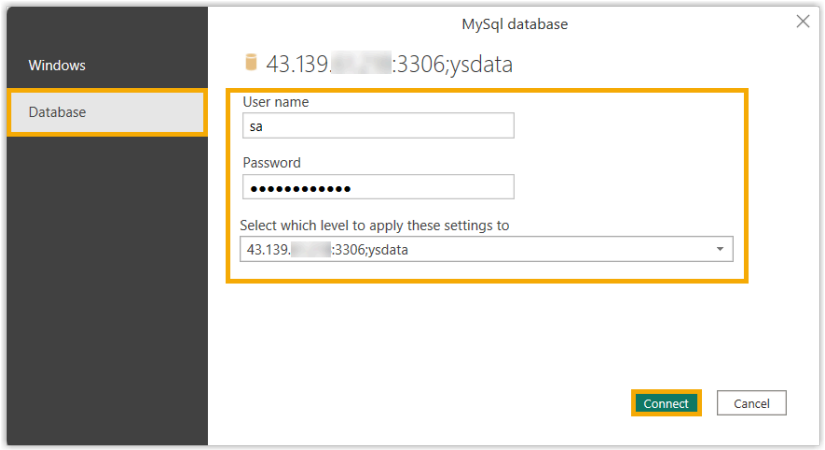
i. Click **Edit Permissions**.



ii. In the pop-up window, click **Edit**.



iii. Select **Database** and fill in the authentication information, then click **Connect**.

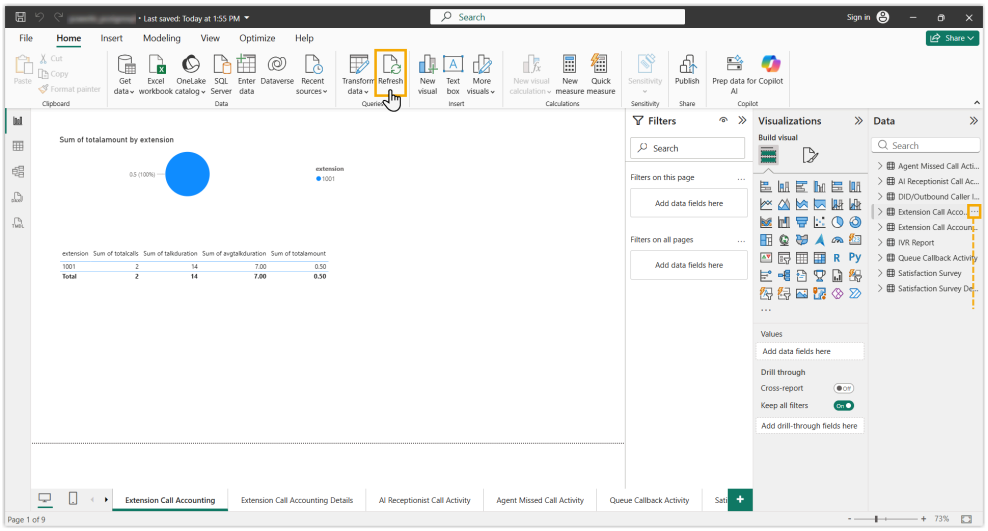


Item	Description
User name	Enter the username to connect to the database.

Item	Description
Password	Enter the password to connect to the database.
Select which level to apply these settings to	Select the database that the account can access.

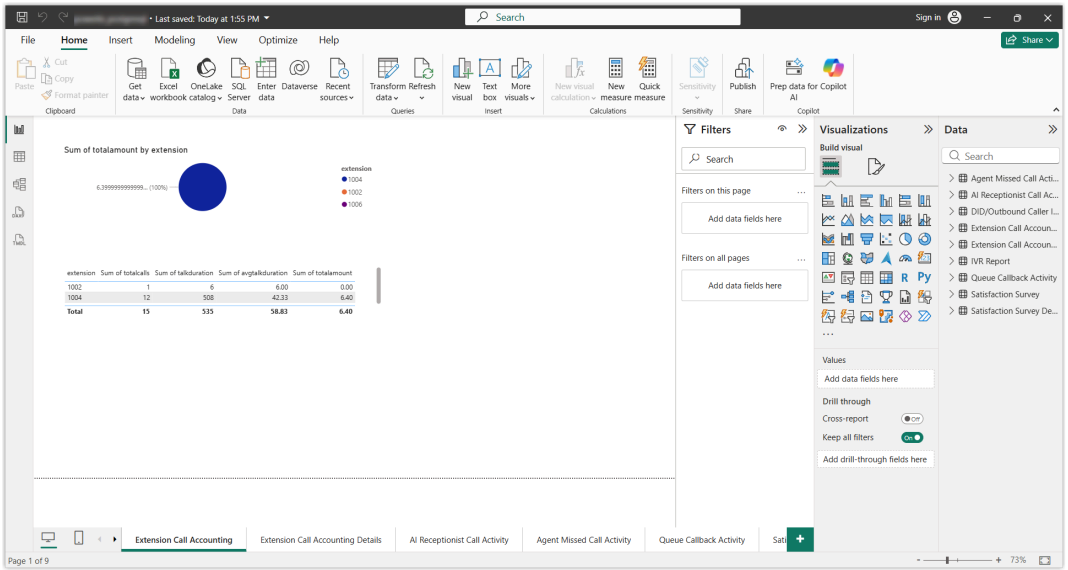
d. Save your setting .

4. On the top of the page, click **Refresh** to run all SQL queries.



Result

The data are successfully imported and visualized in the panel.



What to do next

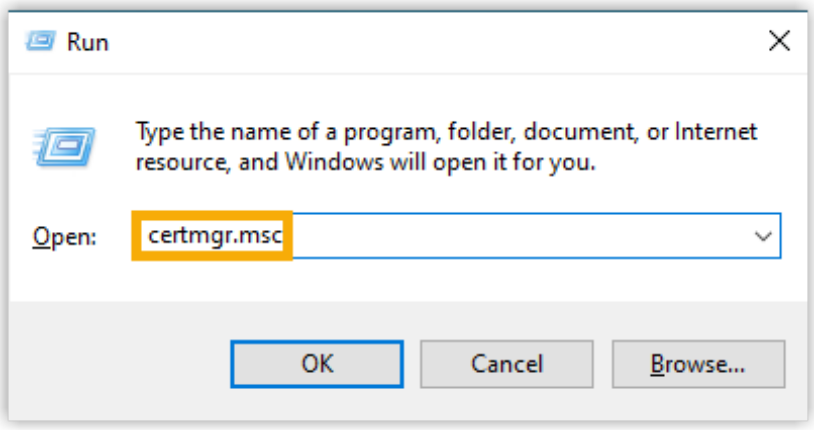
Add or modify the SQL queries to customize the data to be displayed.

> **Note:**
> By default, the template for Power BI displays data only for the following call reports, as Power BI doesn't support incorporating multiple queries from a single source.
>
> - **Extension Call Accounting**
> - **Extension Call Accounting Details**
> - **Agent Missed Call Activity**
> - **Queue Callback Activity**
> - **Satisfaction Survey**
> - **Satisfaction Survey Details**
> - **IVR Report**
> - **DID/Outbound Caller ID Activity**
>
> To display additional call reports, you can add SQL queries as needed. Refer to Call Report Calculations with Multiple SQL Queries for more details on the related data logic.

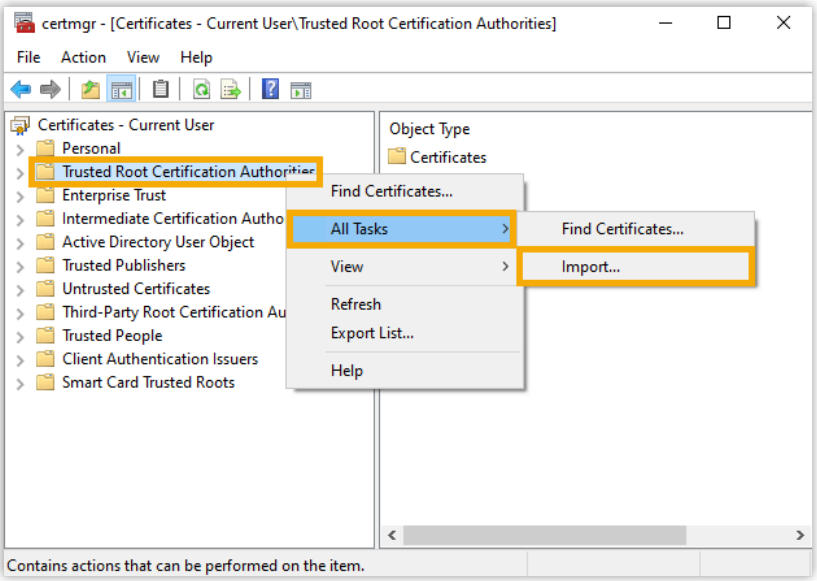
Import PBX data from Microsoft SQL into Power BI

Prerequisites

- Download and unzip the Power BI template, which includes templates for different databases, and select the appropriate one based on your database.
- Download and install Power BI desktop.
- (Optional) If SSL is enabled on the Microsoft SQL server, install the certificate.
    1. Download the SSL public certificate.
    2. Install the SSL certificate in Trusted Root certification authorities store.
        a. Run `certmgr.msc` on your computer to open the Certificate Manager.



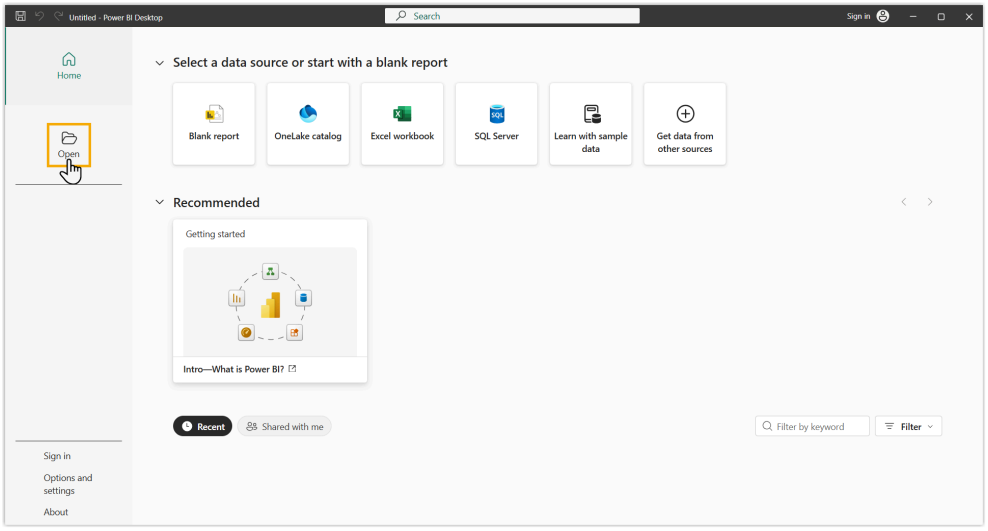
b. Right click **Trusted Root Certification Authorities** and select **Import** from **All Tasks**.



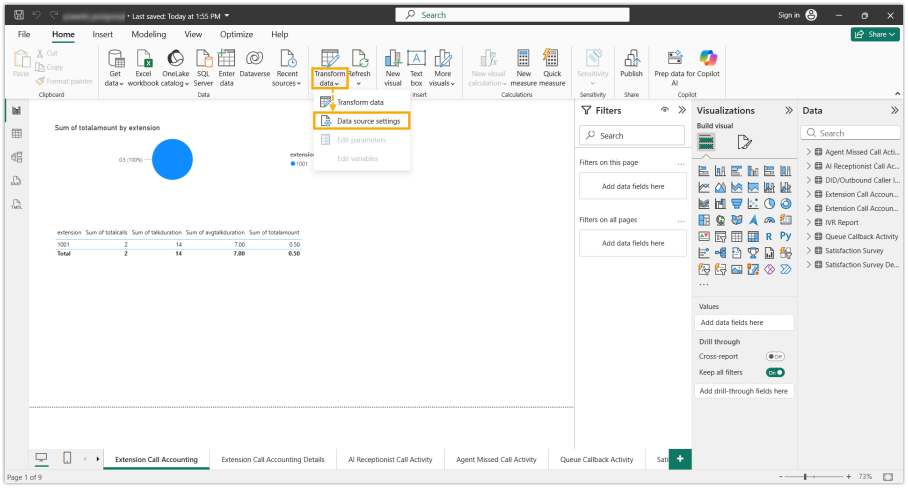
c. Follow the prompts in the wizard to import the root certificate, then click **OK**.

Procedure

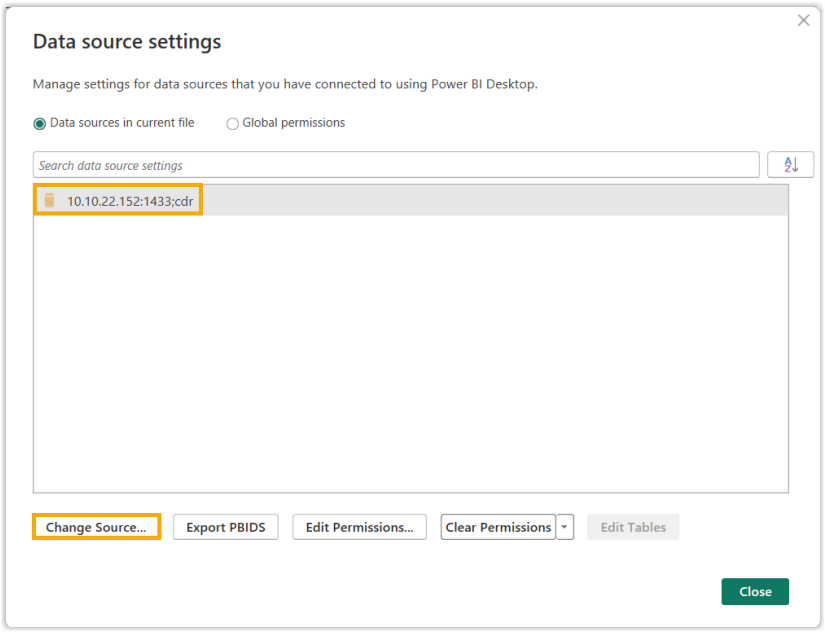
1. Launch Power BI desktop.
2. Click **Open** to select and open the Yeastar-provided template.



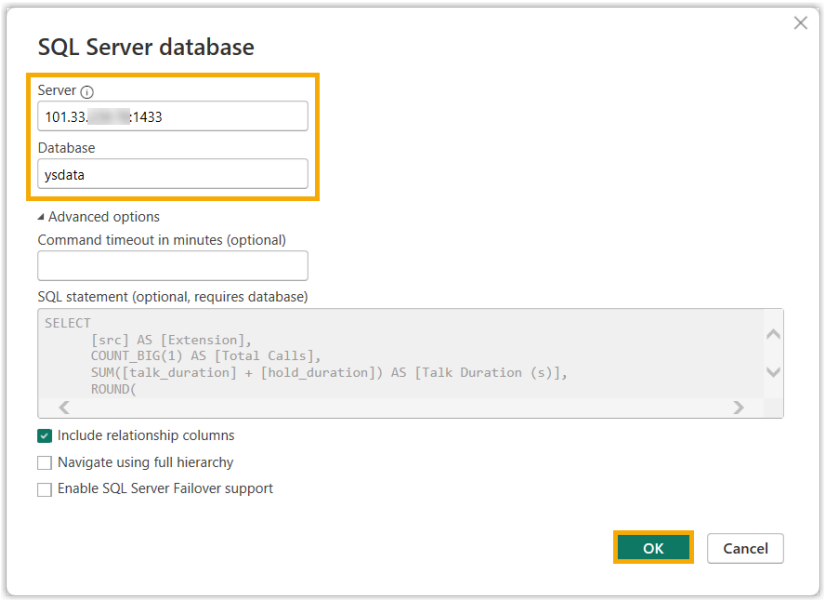
3. Change the data source and configure database access.
    a. On the top toolbar, click **Transform data** and select **Data source settings**.



    b. Change the data source.
        i. Select the default data source, then click **Change Source**.



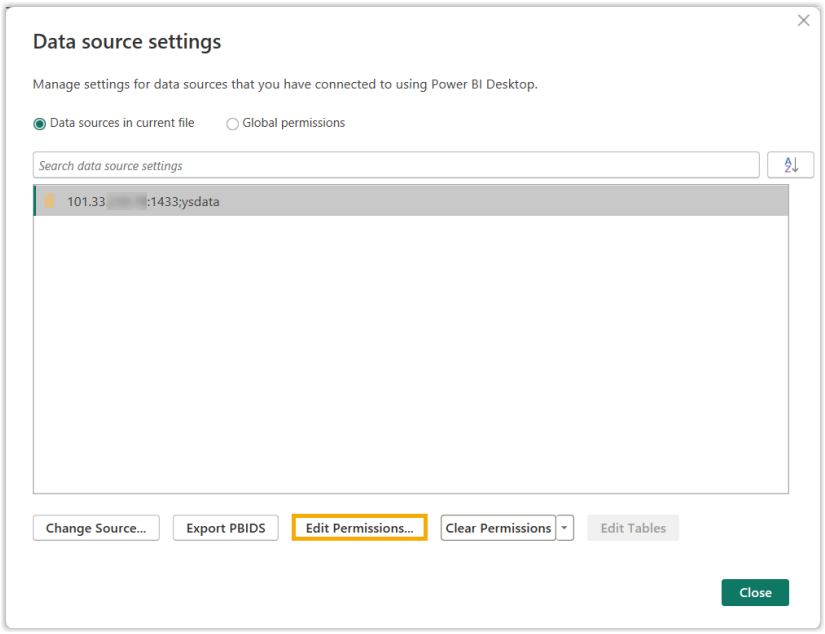
ii. Update the database connection information, then click **OK**.



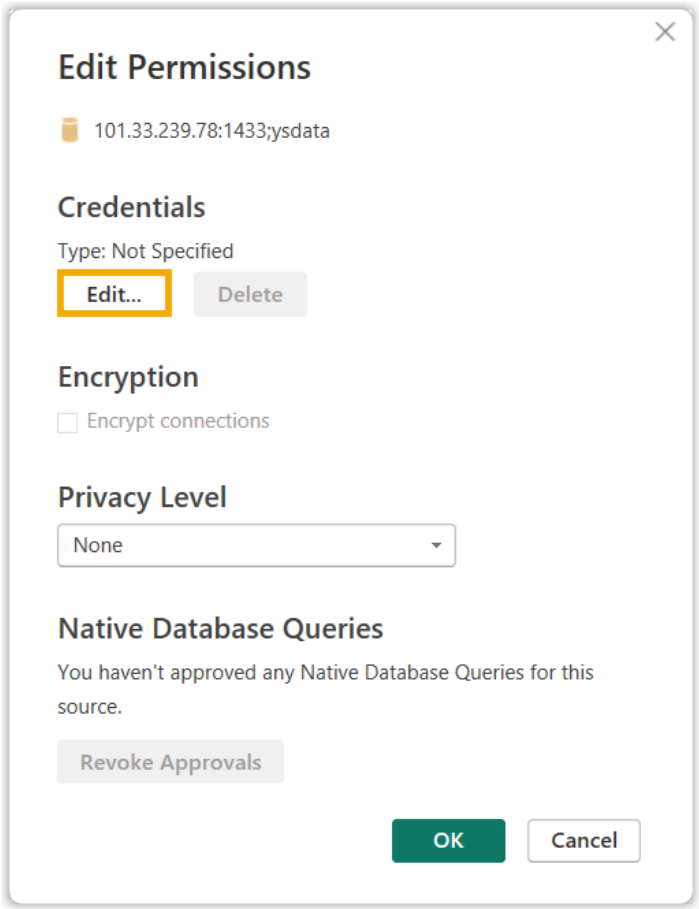
Item	Description
Server	Enter the IP address or domain name of the host on which the database is installed, as well as the database port.
Database	Enter the name of the database.

c. Configure database access credentials.

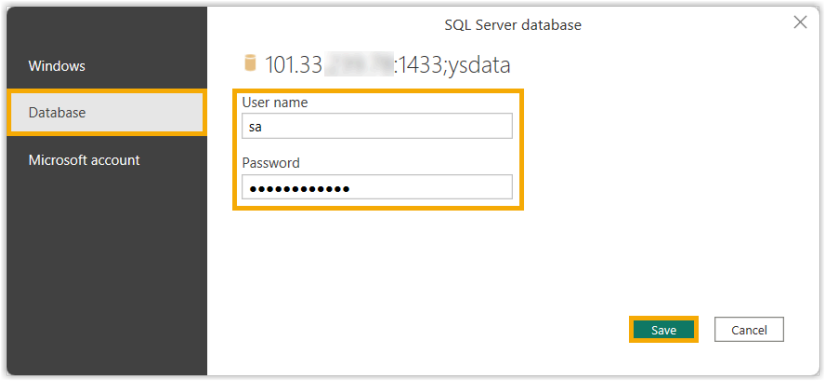
i. Click **Edit Permissions**.



ii. In the pop-up window, click **Edit**.



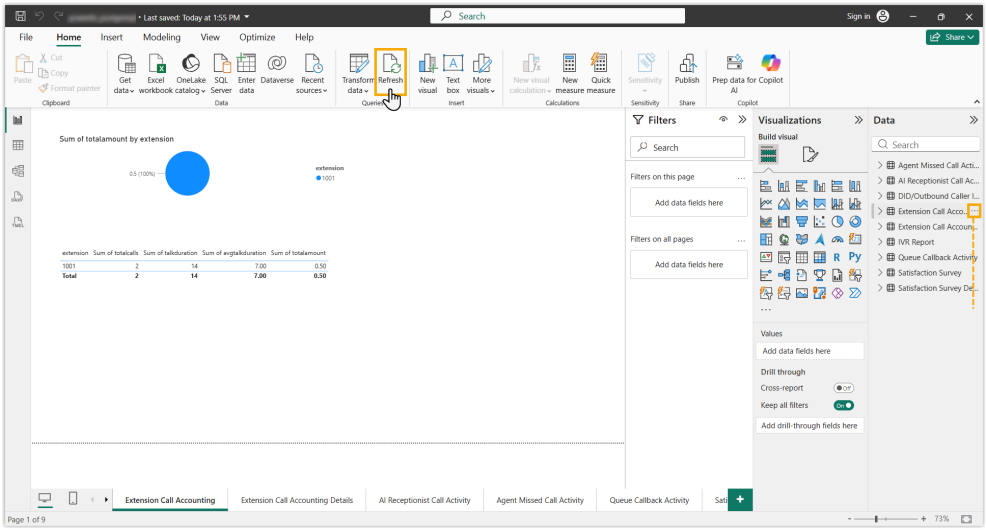
iii. Select **Database** and fill in the authentication information, then click **Save**.



Item	Description
User name	Enter the username to connect to the database.
Password	Enter the password to connect to the database.

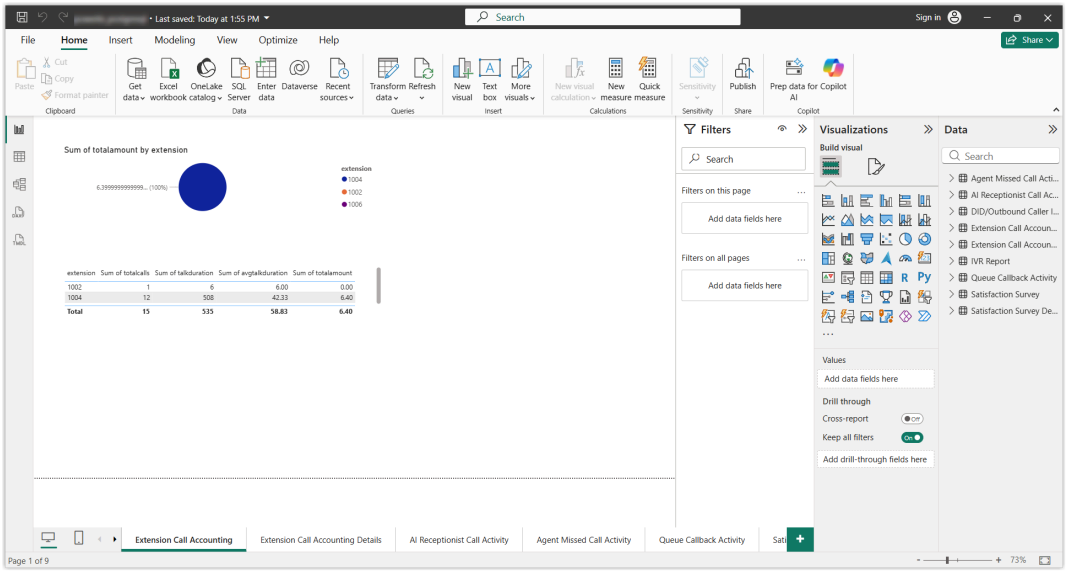
d. Save your setting .

4. On the top of the page, click **Refresh** to run all SQL queries.



Result

The data are successfully imported and visualized in the panel.



What to do next

Add or modify the SQL queries to customize the data to be displayed.

Note:
By default, the template for Power BI displays data only for the following call reports, as Power BI doesn't support incorporating multiple queries from a single source.

- **Extension Call Accounting**
- **Extension Call Accounting Details**
- **Agent Missed Call Activity**
- **Queue Callback Activity**
- **Satisfaction Survey**
- **Satisfaction Survey Details**
- **IVR Report**
- **DID/Outbound Caller ID Activity**

To display additional call reports, you can add SQL queries as needed. Refer to Call Report Calculations with Multiple SQL Queries for more details on the related data logic.

Call Report Calculations with Multiple SQL Queries

By default, the Yeastar-provided templates for Grafana and Power BI display call reports using a single query per report, due to third-party platform limitations. This topic provides reference information to help you customize SQL queries to support multi-query call reports.

Extension Call Statistics report

To display **Extension Call Statistics** report, you need to separately retrieve outbound and inbound call statistics for each extension, then merge the two result sets to generate the complete statistics for each extension.

1. Retrieve outbound call statistics for extensions.

```
SELECT
  src as ext_num,
  SUM(failed) AS failed,
  SUM(vm) AS vm,
  SUM(
    CASE WHEN c.scenario = 'mobile' THEN 0 WHEN
 COALESCE(mobile.mobile_answered, 0) = 1 THEN 1 WHEN c.call_flow = ''
 THEN c.answered WHEN c.call_flow != ''
    AND c.is_group = 1 THEN c.answered WHEN c.call_flow = 'Monitor'
 THEN c.answered ELSE 0 END
  ) AS answered,
  SUM(
    CASE WHEN c.scenario = 'mobile' THEN 0 WHEN
 COALESCE(mobile.mobile_answered, 0) = 1 THEN 0 WHEN c.call_flow = ''
 THEN c.no_answer WHEN c.call_flow != ''
    AND c.is_group = 1 THEN c.no_answer WHEN c.call_flow = 'Monitor'
 THEN c.no_answer ELSE 0 END
  ) AS total_no_answered,
  SUM(
    CASE WHEN c.scenario = 'mobile' THEN 0 WHEN
 COALESCE(mobile.mobile_answered, 0) = 1 THEN 0 WHEN c.call_flow = ''
 THEN c.busy WHEN c.call_flow != ''
    AND c.is_group = 1 THEN c.busy WHEN c.call_flow = 'Monitor' THEN
 c.busy ELSE 0 END
  ) AS busy,
  SUM(
    CASE WHEN c.scenario = 'mobile' THEN 0 WHEN
 COALESCE(mobile.mobile_answered, 0) = 1 THEN 0 WHEN c.call_flow = ''
 THEN c.abandoned WHEN c.call_flow != ''
```

```
    AND c.is_group = 1 THEN c.abandoned WHEN c.call_flow = 'Monitor'
 THEN c.abandoned ELSE 0 END
  ) AS abandoned,
  SUM(
    CASE WHEN c.scenario = 'mobile' THEN 0 WHEN
 COALESCE(mobile.mobile_answered, 0) = 1 THEN
 COALESCE(mobile.mobile_ring, 0) WHEN c.call_flow = '' THEN
 c.ring_duration WHEN c.call_flow != ''
    AND c.is_group = 1 THEN c.ring_duration WHEN c.call_flow =
 'Monitor' THEN c.ring_duration ELSE 0 END
  ) AS ring_duration,
  SUM(
    CASE WHEN c.scenario = 'mobile' THEN 0 WHEN
 COALESCE(mobile.mobile_answered, 0) = 1 THEN
 COALESCE(mobile.mobile_talk, 0) WHEN c.call_flow = '' THEN (
      c.talk_duration + c.hold_duration
    ) WHEN c.call_flow != ''
    AND c.is_group = 1 THEN (
      c.talk_duration + c.hold_duration
    ) WHEN c.call_flow = 'Monitor' THEN (
      c.talk_duration + c.hold_duration
    ) ELSE 0 END
  ) AS talk_duration
FROM
  cdr.cdr_cleaned AS c
  LEFT JOIN (
    SELECT
      m.uid,
      MAX(m.answered) AS mobile_answered,
      MAX(
        CASE WHEN m.answered = 1 THEN m.ring_duration ELSE 0 END
      ) AS mobile_ring,
      MAX(
        CASE WHEN m.answered = 1 THEN m.talk_duration +
 m.hold_duration ELSE 0 END
      ) AS mobile_talk
    FROM
      cdr.cdr_cleaned AS m
    WHERE
      m.scenario = 'mobile'
    GROUP BY
      `m`.`uid`
  ) AS mobile ON c.uid = mobile.uid
WHERE
  c.src in (
```

```
    '1028', '1016', '1003', '1004', '1006',
    '1014', '1033', '1000', '1034', '1023',
    '1027', '1021', '1019', '1022', '1020',
    '1015', '1018', '1025', '1011', '1012',
    '1029', '1024', '1032', '1008', '1007',
    '1010', '1013', '1026', '1030', '1017',
    '1001', '1009', '1031', '1002', '1005'
  )
  AND c.start_ts >= 1773590400000000
  AND c.start_ts <= 1776268799999999
GROUP BY
  `src`
```

The query returns the following data:

- `ext_num`: Extension number.
- `answered`: The number of calls that the extension answered.
- `total_no_answered`: The number of calls that the extension didn't answer.
- `busy`: The number of calls received when the extension was busy.
- `abandoned`: The number of calls abandoned by caller before connecting to the extension.
- `vm`: The number of calls routed to the extension's voicemail.
- `failed`: The number of calls that the extension failed to make.
- `ring_duration`: The total time between calls started and calls answered.
- `talk_duration`: The total time between calls answered and calls ended.

2. Retrieve inbound call statistics for extensions.

```
SELECT
  dst as ext_num,
  SUM(failed) AS failed,
  SUM(vm) AS vm,
  SUM(
    CASE WHEN c.scenario = 'mobile' THEN 0 WHEN
 COALESCE(mobile.mobile_answered, 0) = 1 THEN 1 WHEN c.call_flow = ''
 THEN c.answered WHEN c.call_flow != ''
    AND c.is_group = 0 THEN c.answered ELSE 0 END
  ) AS answered,
  SUM(
    CASE WHEN c.scenario = 'mobile' THEN 0 WHEN
 COALESCE(mobile.mobile_answered, 0) = 1 THEN 0 WHEN c.call_flow = ''
 THEN (c.no_answer) WHEN c.call_flow != ''
    AND c.is_group = 0
    AND c.is_display = 1 THEN (c.no_answer) ELSE 0 END
  ) AS total_no_answered,
  SUM(
```

```
    CASE WHEN c.scenario = 'mobile' THEN 0 WHEN
 COALESCE(mobile.mobile_answered, 0) = 1 THEN 0 WHEN c.call_flow = ''
 THEN c.busy WHEN c.call_flow != ''
    AND c.is_group = 0
    AND c.is_display = 1 THEN c.busy ELSE 0 END
  ) AS busy,
  SUM(
    CASE WHEN c.scenario = 'mobile' THEN 0 WHEN
 COALESCE(mobile.mobile_answered, 0) = 1 THEN 0 WHEN c.call_flow = ''
 THEN c.abandoned ELSE 0 END
  ) AS abandoned,
  SUM(
    CASE WHEN c.scenario = 'mobile' THEN 0 WHEN
 COALESCE(mobile.mobile_answered, 0) = 1 THEN
 COALESCE(mobile.mobile_ring, 0) WHEN c.call_flow = '' THEN
 c.ring_duration WHEN c.call_flow != ''
    AND c.is_group = 0
    AND c.is_display = 1 THEN c.ring_duration ELSE 0 END
  ) AS ring_duration,
  SUM(
    CASE WHEN c.scenario = 'mobile' THEN 0 WHEN
 COALESCE(mobile.mobile_answered, 0) = 1 THEN
 COALESCE(mobile.mobile_talk, 0) WHEN c.call_flow = '' THEN (
      c.talk_duration + c.hold_duration
    ) WHEN c.call_flow != ''
    AND c.is_group = 0 THEN (
      c.talk_duration + c.hold_duration
    ) ELSE 0 END
  ) AS talk_duration
FROM
  cdr.cdr_cleaned AS c
  LEFT JOIN (
    SELECT
      m.uid,
      MAX(m.answered) AS mobile_answered,
      MAX(
        CASE WHEN m.answered = 1 THEN m.ring_duration ELSE 0 END
      ) AS mobile_ring,
      MAX(
        CASE WHEN m.answered = 1 THEN m.talk_duration +
 m.hold_duration ELSE 0 END
      ) AS mobile_talk
    FROM
      cdr.cdr_cleaned AS m
    WHERE
```

```
      m.scenario = 'mobile'
    GROUP BY
      `m`.`uid`
  ) AS mobile ON c.uid = mobile.uid
WHERE
  c.dst in (
    '1028', '1016', '1003', '1004', '1006',
    '1014', '1033', '1000', '1034', '1023',
    '1027', '1021', '1019', '1022', '1020',
    '1015', '1018', '1025', '1011', '1012',
    '1029', '1024', '1032', '1008', '1007',
    '1010', '1013', '1026', '1030', '1017',
    '1001', '1009', '1031', '1002', '1005'
  )
  AND c.start_ts >= 1773590400000000
  AND c.start_ts <= 1776268799999999
GROUP BY
  `dst`
```

The query returns the following data:

- `ext_num`: Extension number.
- `answered`: The number of calls that the extension answered.
- `total_no_answered`: The number of calls that the extension didn't answer.
- `busy`: The number of calls received when the extension was busy.
- `abandoned`: The number of calls abandoned by caller before connecting to the extension.
- `vm`: The number of calls routed to the extension's voicemail.
- `failed`: The number of calls that the extension failed to make.
- `ring_duration`: The total time between calls started and calls answered.
- `talk_duration`: The total time between calls answered and calls ended.

3. Merge the inbound and outbound call statistics by extension number to generate the complete report.

Extension Call Activity report

To display **Extension Call Activity** report, you need to separately retrieve outbound and inbound call statistics for each extension, then merge the two result sets to generate the complete statistics for each extension.

1. Retrieve outbound call statistics for extensions.

```
SELECT
  src as ext_num,
```

```
 MONTH(datetime) AS time,
 SUM(failed) AS failed,
 SUM(vm) AS vm,
 SUM(
   CASE WHEN c.scenario = 'mobile' THEN 0 WHEN
COALESCE(mobile.mobile_answered, 0) = 1 THEN 1 WHEN c.call_flow = ''
THEN c.answered WHEN c.call_flow != ''
   AND c.is_group = 1 THEN c.answered WHEN c.call_flow = 'Monitor'
THEN c.answered ELSE 0 END
 ) AS answered,
 SUM(
   CASE WHEN c.scenario = 'mobile' THEN 0 WHEN
COALESCE(mobile.mobile_answered, 0) = 1 THEN 0 WHEN c.call_flow = ''
THEN c.no_answer WHEN c.call_flow != ''
   AND c.is_group = 1 THEN c.no_answer WHEN c.call_flow = 'Monitor'
THEN c.no_answer ELSE 0 END
 ) AS total_no_answered,
 SUM(
   CASE WHEN c.scenario = 'mobile' THEN 0 WHEN
COALESCE(mobile.mobile_answered, 0) = 1 THEN 0 WHEN c.call_flow = ''
THEN c.busy WHEN c.call_flow != ''
   AND c.is_group = 1 THEN c.busy WHEN c.call_flow = 'Monitor' THEN
c.busy ELSE 0 END
 ) AS busy,
 SUM(
   CASE WHEN c.scenario = 'mobile' THEN 0 WHEN
COALESCE(mobile.mobile_answered, 0) = 1 THEN 0 WHEN c.call_flow = ''
THEN c.abandoned WHEN c.call_flow != ''
   AND c.is_group = 1 THEN c.abandoned WHEN c.call_flow = 'Monitor'
THEN c.abandoned ELSE 0 END
 ) AS abandoned,
 SUM(
   CASE WHEN c.scenario = 'mobile' THEN 0 WHEN
COALESCE(mobile.mobile_answered, 0) = 1 THEN
COALESCE(mobile.mobile_ring, 0) WHEN c.call_flow = '' THEN
c.ring_duration WHEN c.call_flow != ''
   AND c.is_group = 1 THEN c.ring_duration WHEN c.call_flow =
'Monitor' THEN c.ring_duration ELSE 0 END
 ) AS ring_duration,
 SUM(
   CASE WHEN c.scenario = 'mobile' THEN 0 WHEN
COALESCE(mobile.mobile_answered, 0) = 1 THEN
COALESCE(mobile.mobile_talk, 0) WHEN c.call_flow = '' THEN (
     c.talk_duration + c.hold_duration
   ) WHEN c.call_flow != ''
```

```
    AND c.is_group = 1 THEN (
      c.talk_duration + c.hold_duration
    ) WHEN c.call_flow = 'Monitor' THEN (
      c.talk_duration + c.hold_duration
    ) ELSE 0 END
  ) AS talk_duration
FROM
  cdr.cdr_cleaned AS c
  LEFT JOIN (
    SELECT
      m.uid,
      MAX(m.answered) AS mobile_answered,
      MAX(
        CASE WHEN m.answered = 1 THEN m.ring_duration ELSE 0 END
      ) AS mobile_ring,
      MAX(
        CASE WHEN m.answered = 1 THEN m.talk_duration +
 m.hold_duration ELSE 0 END
      ) AS mobile_talk
    FROM
      cdr.cdr_cleaned AS m
    WHERE
      m.scenario = 'mobile'
    GROUP BY
      `m`.`uid`
  ) AS mobile ON c.uid = mobile.uid
WHERE
  c.src in (
    '1032', '1008', '1007', '1010', '1013',
    '1026', '1030', '1017', '1001', '1009',
    '1031', '1002', '1005', '1028', '1016',
    '1003', '1004', '1006', '1014', '1033',
    '1000', '1034', '1023', '1027', '1021',
    '1019', '1022', '1020', '1015', '1018',
    '1025', '1011', '1012', '1029', '1024'
  )
  AND c.start_ts >= 1767196800000000
  AND c.start_ts <= 1798732799999999
GROUP BY
  src,
  time
ORDER BY
  time,
  ext_num asc
```

The query returns the following data:

- `time`: Time.
- `ext_num`: Extension number.
- `answered`: The number of calls that the extension answered.
- `total_no_answered`: The number of calls that the extension didn't answer.
- `busy`: The number of calls received when the extension was busy.
- `abandoned`: The number of calls abandoned by caller before connecting to the extension.
- `failed`: The number of calls that the extension failed to make.
- `vm`: The number of calls routed to the extension's voicemail.
- `ring_duration`: The total time between calls started and calls answered.
- `talk_duration`: The total time between calls answered and calls ended.

2. Retrieve inbound call statistics for extensions.

```
SELECT
  dst as ext_num,
  MONTH(datetime) AS time,
  SUM(failed) AS failed,
  SUM(vm) AS vm,
  SUM(
    CASE WHEN c.scenario = 'mobile' THEN 0 WHEN
 COALESCE(mobile.mobile_answered, 0) = 1 THEN 1 WHEN c.call_flow = ''
 THEN c.answered WHEN c.call_flow != ''
    AND c.is_group = 0 THEN c.answered ELSE 0 END
  ) AS answered,
  SUM(
    CASE WHEN c.scenario = 'mobile' THEN 0 WHEN
 COALESCE(mobile.mobile_answered, 0) = 1 THEN 0 WHEN c.call_flow = ''
 THEN (c.no_answer) WHEN c.call_flow != ''
    AND c.is_group = 0
    AND c.is_display = 1 THEN (c.no_answer) ELSE 0 END
  ) AS total_no_answered,
  SUM(
    CASE WHEN c.scenario = 'mobile' THEN 0 WHEN
 COALESCE(mobile.mobile_answered, 0) = 1 THEN 0 WHEN c.call_flow = ''
 THEN c.busy WHEN c.call_flow != ''
    AND c.is_group = 0
    AND c.is_display = 1 THEN c.busy ELSE 0 END
  ) AS busy,
  SUM(
    CASE WHEN c.scenario = 'mobile' THEN 0 WHEN
 COALESCE(mobile.mobile_answered, 0) = 1 THEN 0 WHEN c.call_flow = ''
 THEN c.abandoned ELSE 0 END
  ) AS abandoned,
  SUM(
```

```
    CASE WHEN c.scenario = 'mobile' THEN 0 WHEN
 COALESCE(mobile.mobile_answered, 0) = 1 THEN
 COALESCE(mobile.mobile_ring, 0) WHEN c.call_flow = '' THEN
 c.ring_duration WHEN c.call_flow != ''
    AND c.is_group = 0
    AND c.is_display = 1 THEN c.ring_duration ELSE 0 END
  ) AS ring_duration,
  SUM(
    CASE WHEN c.scenario = 'mobile' THEN 0 WHEN
 COALESCE(mobile.mobile_answered, 0) = 1 THEN
 COALESCE(mobile.mobile_talk, 0) WHEN c.call_flow = '' THEN (
      c.talk_duration + c.hold_duration
    ) WHEN c.call_flow != ''
    AND c.is_group = 0 THEN (
      c.talk_duration + c.hold_duration
    ) ELSE 0 END
  ) AS talk_duration
FROM
  cdr.cdr_cleaned AS c
  LEFT JOIN (
    SELECT
      m.uid,
      MAX(m.answered) AS mobile_answered,
      MAX(
        CASE WHEN m.answered = 1 THEN m.ring_duration ELSE 0 END
      ) AS mobile_ring,
      MAX(
        CASE WHEN m.answered = 1 THEN m.talk_duration +
 m.hold_duration ELSE 0 END
      ) AS mobile_talk
    FROM
      cdr.cdr_cleaned AS m
    WHERE
      m.scenario = 'mobile'
    GROUP BY
      `m`.`uid`
  ) AS mobile ON c.uid = mobile.uid
WHERE
  c.dst in (
    '1032', '1008', '1007', '1010', '1013',
    '1026', '1030', '1017', '1001', '1009',
    '1031', '1002', '1005', '1028', '1016',
    '1003', '1004', '1006', '1014', '1033',
    '1000', '1034', '1023', '1027', '1021',
    '1019', '1022', '1020', '1015', '1018',
```

```
    '1025', '1011', '1012', '1029', '1024'
  )
  AND c.start_ts >= 1767196800000000
  AND c.start_ts <= 1798732799999999
GROUP BY
  dst,
  time
ORDER BY
  time,
  ext_num asc
```

The query returns the following data:

- `time`: Time.
- `ext_num`: Extension number.
- `answered`: The number of calls that the extension answered.
- `total_no_answered`: The number of calls that the extension didn't answer.
- `busy`: The number of calls received when the extension was busy.
- `abandoned`: The number of calls abandoned by caller before connecting to the extension.
- `failed`: The number of calls that the extension failed to make.
- `vm`: The number of calls routed to the extension's voicemail.
- `ring_duration`: The total time between calls started and calls answered.
- `talk_duration`: The total time between calls answered and calls ended.

3. Merge the inbound and outbound call statistics by extension number and time period to generate the complete report.

Agent Call Summary report

To display **Agent Call Summary** report, you need to separately retrieve inbound and outbound call statistics for each agent, then merge the results to calculate the overall statistics and average call durations.

1. Retrieve inbound queue call statistics for agents.

```
SELECT
  c.dst AS agent,
  count(answered) AS answered_call,
  SUM(
    CASE WHEN sub.duration != NULL THEN c.duration + sub.duration ELSE
 c.duration END
  ) AS duration,
  SUM(talk_duration) AS talking_time,
  SUM(hold_duration) AS hold_time,
```

```
  SUM(
    CASE WHEN sub.ring_duration != NULL THEN c.ring_duration +
 sub.ring_duration ELSE c.ring_duration END
  ) AS waiting_time
FROM
  cdr.cdr_cleaned as c
  LEFT JOIN (
    SELECT
      uid,
      dst,
      sum(duration) as duration,
      sum(ring_duration) AS ring_duration
    FROM
      cdr.cdr_cleaned as c
    WHERE
      start_ts >= 1773590400000000
      AND start_ts <= 1776268799999999
      AND (
        c.dst IN ('1000', '1001')
      )
      AND src != 'PBX'
      AND call_flow_number = '6404'
      AND call_flow = 'Queue'
      AND disposition != 'ANSWERED'
      AND is_group = '0'
    GROUP BY
      uid,
      dst
  ) AS sub ON c.uid = sub.uid
  and c.dst = sub.dst
WHERE
  start_ts >= 1773590400000000
  AND start_ts <= 1776268799999999
  AND (
    c.dst IN ('1000', '1001')
  )
  AND src != 'PBX'
  AND call_flow_number = '6404'
  AND call_flow = 'Queue'
  AND disposition = 'ANSWERED'
  AND is_group = '0'
GROUP BY
  `c`.`dst`
```

The query returns the following data:

- `agent`: Agent number.
- `answered_call`: The number of incoming queue calls answered by the agent.
- `duration`: The amount of time that agents spent handling incoming queue calls, from ringing to call end.
- `talking_time`: The amount of time that agents spent in incoming calls.
- `hold_time`: The amount of time that agents held incoming calls.
- `waiting_time`: The waiting time before incoming queue calls were answered.

2. Retrieve outbound queue call statistics for agents.

```
SELECT
  src AS agent,
  COUNT(uid) AS total_call,
  SUM(answered) AS answered_call,
  SUM(
    CASE WHEN disposition = 'ANSWERED' THEN duration ELSE 0 END
  ) AS duration,
  SUM(
    CASE WHEN disposition = 'ANSWERED' THEN talk_duration ELSE 0 END
  ) AS talking_time,
  SUM(
    CASE WHEN disposition = 'ANSWERED' THEN hold_duration ELSE 0 END
  ) AS hold_time,
  SUM(
    CASE WHEN disposition = 'ANSWERED' THEN ring_duration ELSE 0 END
  ) AS waiting_time
FROM
  `cdr`.`cdr_cleaned`
WHERE
  start_ts >= 1773590400000000
  AND start_ts <= 1776268799999999
  AND call_type = 'Outbound'
  AND src in ('1000', '1001')
  AND NOT(srcname = 'Conference Call')
  AND is_group = '0'
GROUP BY
  `src`
```

The query returns the following data:

- `agent`: Agent number.
- `total_call`: The number of outgoing calls placed by the agent.
- `answered_call`: The number of outgoing calls that were answered by the agent.
- `duration`: The amount of time that agents spent handling outbound queue calls, from ringing to call end.
- `talking_time`: The amount of time that agents spent in outgoing calls.

- `hold_time`: The amount of time that agents held outbound queue calls.
- `waiting_time`: The waiting time before outbound queue calls were answered.

3. Merge the inbound and outbound queue call statistics by agent's extension number to generate the overall statistics.
    - `total_duration`: Sum of `duration` for agents' inbound and outbound queue calls.
    - `total_talk_time`: Sum of `talking_time` and `hold_time` for agents' inbound and outbound queue calls.
    - `total_hold_time`: Sum of `hold_time` for agents' inbound and outbound queue calls.
    - `total_waiting_time`: Sum of `waiting_time` for agents' inbound and outbound queue calls.
    - `total_call`: Sum of `answered_call` for agents' inbound queue calls and `total_call` for agents' outbound queue calls..
    - `inbound_answered_call`: Sum of `answered_call` for agents' inbound queue calls.
    - `inbound_duration`: Sum of `talking_time` and `hold_time` for agents' inbound queue calls.
    - `outbound_total_call`: Sum of `total_call` for agents' outbound queue calls.
    - `outbound_answered_call`: Sum of `answered_call` for agents' outbound queue calls.
    - `outbound_duration`: Sum of `talking_time` and `hold_time` for agents' outbound queue calls.
4. Calculate average call durations for each agent.
    - `avg_server_time` = `total_duration` / (`inbound_answered_call` + `outbound_answered_call`)
    - `avg_talking_time` = `total_talk_time` / (`inbound_answered_call` + `outbound_answered_call`)
    - `avg_hold_time` = `total_hold_time` / (`inbound_answered_call` + `outbound_answered_call`)
    - `avg_waiting_time` = `total_waiting_time` /(`inbound_answered_call`+ `outbound_answered_call`)

Agent Login Activity report

To display **Agent Login Activity** report, you need to retrieve agent login and logout events from the queue log.

```
SELECT
  agent,
```

```
  timestamp,
  event
FROM
  `queue_log`
WHERE
  (
    queuename = 'queue-6402'
    and event in('ADDMEMBER', 'REMOVEMEMBER')
  )
  AND (timestamp >= 1773590400)
  AND (timestamp <= 1776268799)
GROUP BY
  agent,
  timestamp,
  event
ORDER BY
  timestamp asc
```

The query returns the following data:

- `agent`: Agent number.
- `timestamp`: The date and time when the agent logged in to or logged out of a queue.
- `event`:
  - `ADDMEMBER`: Agent logged in to the queue.
  - `REMOVEMEMBER`: Agent logged out of the queue.

Note:
Total login time is calculated as the difference between the timestamps of `REMOVEMEMBER` and `ADDMEMBER`.

Agent Pause Activity report

To display **Agent Pause Activity** report, you need to retrieve agent pause and unpause events from the queue log.

```
SELECT
  agent,
  timestamp,
  event,
  data1
FROM
  `queue_log`
WHERE
```

```
  (
    queuename = 'queue-6404'
    and event in(
      'PAUSE', 'UNPAUSE', 'REMOVEMEMBER'
    )
  )
  AND (timestamp >= 1774022400)
  AND (timestamp <= 1776700799)
GROUP BY
  agent,
  timestamp,
  event,
  data1
ORDER BY
  timestamp asc
```

The query returns the following data:

- `agent`: Agent number.
- `timestamp`: The date and time when agents paused or resumed service.
- `data1`: Pause reason.
- `event`:
  - `PAUSE`: Agent paused service.
  - `UNPAUSE`: Agent resumed service.

> **Note:**
> Total pause time is calculated as the difference between the timestamps of `UNPAUSE` and `PAUSE`.

Agent Performance report

To display **Agent Performance** report, you need to aggregate queue call records from multiple datasets and then merge the results to calculate final agent-level metric results.

1. Aggregate queue performance call data excluding `q_half_consult` records.

```
SELECT
  `dst`,
  Count(*) as total_calls,
  sum(answered) as answered_calls,
  sum(missed) as missed_calls,
  sum(abandoned) as abandoned_calls,
  max(ring_duration) as max_ring_time,
```

```
  sum(answered_ring_duration) as total_answered_ring_time,
  sum(ring_duration) as total_ring_time,
  sum(hold_duration) as total_hold_time,
  sum(talk_duration) as total_talking_time,
  sum(member_hold_duration) as total_member_hold_duration,
  sum(member_answered_count) as total_member_answered
FROM
  cdr.cdr_cleaned as c
WHERE
  c.is_group = 1
  AND (
    not (
      c.ring_duration <= 1
      and c.abandoned = 1
    )
  )
  AND flag != 'q_half_consult'
  AND c.start_ts >= 1776614400000000
  AND c.start_ts <= 1776700799999999
  AND call_flow = 'Queue'
  AND call_flow_number = '6404'
GROUP BY
  `dst`
```

The query returns the following data:

- `dst`: Queue number.
- `total_calls`: The number of calls that the queue received.
- `answered_calls`: The number of calls that the queue answered.
- `missed_calls`: The number of calls that the queue missed.
- `abandoned_calls`: The number of calls that callers abandoned.
- `max_ring_time`: The maximum amount of time that caller waited in the queue before being connected to an agent.
- `total_answered_ring_time`: The time between the call started and the call was answered.
- `total_ring_time`: The time between the call started and the call was answered.
- `total_hold_time`: The amount of time that the call was held.
- `total_talking_time`: The time between the call answered and the call ended.
- `total_member_hold_duration`: The time that agent held calls.
- `total_member_answered` : The number of calls that agent answered.

2. Aggregate queue performance call records with `q_half_consult` flag.

```
SELECT
  c.dst,
  sum(c.ring_duration) as total_ring_time,
```

```
  sum(
    CASE WHEN h.answered > 0 THEN c.ring_duration ELSE 0 END
  ) as total_answered_ring_time,
  max(
    c.ring_duration + h.ring_duration
  ) as max_ring_time
FROM
  cdr.cdr_cleaned as c
  LEFT JOIN (
    SELECT
      uid,
      answered,
      ring_duration
    FROM
      cdr.cdr_cleaned as c
    WHERE
      c.is_group = 1
      AND flag != 'q_half_consult'
      AND c.start_ts >= 1776614400000000
      AND c.start_ts <= 1776700799999999
      AND call_flow = 'Queue'
      AND call_flow_number = '6404'
  ) AS h ON h.uid = c.another_uid
WHERE
  c.is_group = 1
  AND flag = 'q_half_consult'
  AND c.start_ts >= 1776614400000000
  AND c.start_ts <= 1776700799999999
  AND call_flow = 'Queue'
  AND call_flow_number = '6404'
GROUP BY
  `c`.`dst`
```

The query returns the following data:

- `dst`: Queue number.
- `total_ring_time`: The time between the call started and the call was answered.
- `total_answered_ring_time`: The time between the call started and the call was answered for answered calls.
- `max_ring_time`: The maximum amount of time that caller waited in the queue before being connected to an agent.

3. Merge queue performance statistics from both datasets to generate the final queue-level statistics.

    - `total_answered_ring_time` = Sum of `total_answered_ring_time` from both datasets

- `total_ring_time` = Sum of `total_ring_time` from both datasets
- `max_ring_time` = Maximum value of `max_ring_time` from both datasets

4. Calculate queue-level performance metrics based on the aggregated data.
   - `average_waiting_time` = `total_answered_ring_time` / `answered_calls`
   - `average_talking_time` = (`total_talking_time` + `total_hold_time`) / `answered_calls`
   - `average_handle_time` = (`total_answered_ring_time` + `total_talking_time` + `total_hold_time`) / `answered_calls`
   - `average_hold_time` = `total_hold_time` / `answered_calls`
   - `anwsered_rate` = `answered_calls` / `total_calls`
   - `missed_rate` = `missed_calls` / `total_calls`
   - `abandoned_rate` = `abandoned_calls` / `total_calls`
   - `all_call_average_waiting_time` = `total_ring_time` / `total_calls`
5. Aggregate agent-level queue call records excluding callback-related calls and invalid abandoned records.

```
SELECT
  c.uid,
  c.call_flow_number as queue,
  c.dst as agent,
  sum(c.answered) as answered_calls,
  sum(c.missed) as missed_calls,
  max(c.ring_duration) as max_ring_time,
  sum(c.answered_ring_duration) as total_answered_ring_time,
  sum(c.ring_duration) as total_ring_time,
  sum(c.talk_duration) as total_talking_time,
  sum(c.hold_duration) as total_hold_time
FROM
  cdr.cdr_cleaned as c
  LEFT JOIN cdr.cdr_cleaned AS h ON h.uid = c.uid
  AND h.is_group = 1
  AND h.ring_duration <= 1
  AND h.disposition = 'ABANDONED'
WHERE
  c.call_flow IN ('Queue', 'QCB')
  AND c.call_flow_number = '6404'
  AND c.dst IN ('1000', '1001')
  AND c.is_group != 1
  AND c.disposition != 'ABANDONED'
  AND c.src != 'PBX'
  AND h.uid IS NULL
  AND c.start_ts >= 1776614400000000
  AND c.start_ts <= 1776700799999999
```

```
GROUP BY
  c.call_flow_number,
  c.uid,
  c.dst
```

The query returns the following data:

- `uid`: The unique ID of CDR.
- `queue`: Queue number.
- `agent`: Agent number.
- `answered_calls`: The number of calls that agent answered.
- `missed_calls`: The number of calls that agent missed.
- `max_ring_time`: The maximum ring time before a call is connected to an agent.
- `total_answered_ring_time`: The time between call started and call answered for answered calls.
- `total_ring_time`: The time between call started and call answered.
- `total_talking_time`: The time between call answered and call ended.
- `total_hold_time`: The amount of time that agent held calls.

6. Aggregate agent-level queue call records related to callback calls.

```
SELECT
  c.uid,
  c.call_flow_number as queue,
  c.dst as agent,
  sum(c.answered) as answered_calls,
  sum(c.missed) as missed_calls,
  max(c.ring_duration) as max_ring_time,
  sum(c.answered_ring_duration) as total_answered_ring_time,
  sum(c.ring_duration) as total_ring_time,
  sum(c.talk_duration) as total_talking_time,
  sum(c.hold_duration) as total_hold_time
FROM
  cdr.cdr_cleaned as c
  LEFT JOIN cdr.cdr_cleaned AS h ON h.uid = c.uid
  AND h.is_group = 1
  AND h.ring_duration <= 1
  AND h.disposition = 'ABANDONED'
WHERE
  c.call_flow IN ('Queue', 'QCB')
  AND c.call_flow_number = '6404'
  AND c.dst IN ('1000', '1001')
  AND c.is_group != 1
  AND c.disposition != 'ABANDONED'
  AND c.src = 'PBX'
  AND h.uid IS NULL
```

```
  AND c.start_ts >= 1776614400000000
  AND c.start_ts <= 1776700799999999
GROUP BY
  c.call_flow_number,
  c.uid,
  c.dst
```

The query returns the following data:

- `uid`: The unique ID of CDR.
- `queue`: Queue number.
- `agent`: Agent number.
- `answered_calls`: The number of calls that agent answered.
- `missed_calls`: The number of calls that agent missed.
- `max_ring_time`: The maximum ring time before a call is connected to an agent.
- `total_answered_ring_time`: The time between the call started and the call was answered.
- `total_ring_time`: The time between the call started and the call was answered.
- `total_talking_time`: The time between call answered and call ended.
- `total_hold_time`: The amount of time that agent held calls.

7. Retrieve agent-level callback (QCB) statistics.

```
SELECT
  call_flow_number as queue,
  src as agent,
  count(*) as total_calls,
  sum(answered) as answered_calls,
  sum(missed) as missed_calls,
  max(ring_duration) as max_ring_time,
  sum(answered_ring_duration) as total_answered_ring_time,
  sum(ring_duration) as total_ring_time,
  sum(duration) as total_talking_time,
  sum(hold_duration) as total_hold_time
FROM
  `cdr`.`cdr_cleaned`
WHERE
  call_flow_number = '6404'
  AND is_qcb = 1
  AND src IN ('1000', '1001')
  AND start_ts >= 1776614400000000
  AND start_ts <= 1776700799999999
  AND call_flow = ('Queue')
GROUP BY
  call_flow_number,
  src
```

The query returns the following data:

- `queue`: Queue number.
- `agent`: Agent number.
- `total_calls`: The number of calls received.
- `answered_calls`: The number of calls that agent answered.
- `missed_calls`: The number of calls that agent missed.
- `max_ring_time`: The maximum ring time before a call is connected to an agent.
- `total_answered_ring_time`: The time between the call started and the call was answered.
- `total_ring_time`: The time between the call started and the call was answered.
- `total_talking_time`: The time between call answered and call ended.
- `total_hold_time`: The amount of time that agent held calls.

8. Add callback talking time to the agent statistics.

```
agent_talk_time_map[queue_agent]=total_talking_time
```

9. Aggregate call statistics at queue and agent level.

```
agentMap[queue_agent].dst=agent
agentMap[queue_agent].total_calls+=answered_calls+missed_calls
agentMap[queue_agent].answered_calls+=answered_calls
agentMap[queue_agent].missed_calls+=missed_calls
agentMap[queue_agent].total_ring_time+=total_ring_time

agentMap[queue_agent].total_talking_time+=agent_talk_time_map[queue_ag
ent]//只加一次
agent_answered_ring_duration[queue_uid_agent] =
 total_answered_ring_time
if answered_calls > 0 {
    agent_answered[queue_uid_agent] = 1
}
```

10. Calculate agent-level performance metrics.

```
 //
answered_ring_time = total_answered_ring_time
if answered_calls > 0 || agent_answered[queue_uid_agent] == 1 {
    answered_ring_time = total_ring_time
}
agentMap.total_calls += answered_calls + missed_calls
agentMap.answered_calls += answered_calls
agentMap.missed_calls += missed_calls
agentMap.total_answered_ring_time += answered_ring_time
agentMap.total_ring_time += total_ring_time
agentMap.total_talking_time += total_talking_time + total_hold_time
```

```
//
agent_answered_ring_duration[queue_uid_agent] += answered_ring_time

//
agentMap.max_ring_time =
 max(agent_answered_ring_duration[queue_agent]...)
//
agentMap.avg_talking_time = agentMap.total_talking_time /
 agentMap.answered_calls
agentMap.avg_waiting_time = agentMap.total_answered_ring_time /
 agentMap.answered_calls
agentMap.miss_rate=agentMap.missed_calls / agentMap.total_calls
```

Queue AVG Wait and Talk Time report

To display **Queue AVG Wait and Talk Time** report, you need to aggregate queue call data and calculate related metrics.

1. Aggregate queue call data excluding `q_half_consult` records to build the dataset `normal_data`.

   ```
   SELECT
     DAY(datetime) AS time,
     Count(*) as total_calls,
     sum(answered) as answered_calls,
     sum(answered_ring_duration) as total_answered_ring_time,
     sum(ring_duration) as total_ring_time,
     sum(talk_duration + hold_duration) as total_talking_time
   FROM
     `cdr`.`cdr_cleaned`
   WHERE
     is_group = 1
     AND flag != 'q_half_consult'
     AND start_ts >= 1774972800000000
     AND start_ts <= 1777564799999999
     AND call_flow = 'Queue'
     AND call_flow_number = '6404'
   GROUP BY
     `time`
   ```

   The query returns the following data:
   - `time`: When the call was received.
   - `total_calls`: The total number of calls that the queue received.
   - `answered_calls`: The number of calls that the queue answered.

- `total_answered_ring_time`: The time between the call started and the call was answered.
- `total_ring_time`: The time between the call started and the call was answered.
- `total_talking_time`: The time between the call answered and the call ended.

2. Aggregate queue call data with the `q_half_consult` records to build the dataset `q_half_consult_data`.

```
SELECT
  day AS time,
  sum(c.ring_duration) as total_ring_time,
  sum(
    CASE WHEN h.answered > 0 THEN c.ring_duration ELSE 0 END
  ) as total_answered_ring_time,
  max(
    c.ring_duration + h.ring_duration
  ) as max_ring_time
FROM
  cdr.cdr_cleaned as c
  LEFT JOIN (
    SELECT
      uid,
      answered,
      ring_duration
    FROM
      cdr.cdr_cleaned as c
    WHERE
      c.is_group = 1
      AND flag != 'q_half_consult'
      AND c.start_ts >= 1774972800000000
      AND c.start_ts <= 1777564799999999
      AND call_flow = 'Queue'
      AND call_flow_number = '6404'
  ) AS h ON h.uid = c.another_uid
WHERE
  is_group = 1
  AND flag = 'q_half_consult'
  AND start_ts >= 1774972800000000
  AND start_ts <= 1777564799999999
  AND call_flow = 'Queue'
  AND call_flow_number = '6404'
GROUP BY
  `time`
```

The query returns the following data:

- `time`: When the call was received.

- `total_ring_time`: The time between the call started and the call was answered.
- `total_answered_ring_time`: The time between the call started and the call was answered.
- `max_ring_time`: The maximum amount of time that callers waited in the queue.

3. Merge `normal_data` and `q_half_consult_data` to generate the final metrics data.

```
nornal_data[time].time
nornal_data[time].total_calls
nornal_data[time].answered_calls
nornal_data[time].total_talking_time
nornal_data[time].total_answered_ring_time +=
 q_half_consult_data[time].total_answered_ring_time
nornal_data[time].total_ring_time+=
 q_half_consult_data[time].total_ring_time
nornal_data[time].average_waiting_time =
 nornal_data[time].total_answered_ring_time /
 nornal_data[time].answered_calls
nornal_data[time].average_talking_time
 = nornal_data[time].total_taking_time /
 nornal_data[time].answered_calls
nornal_data[time].all_call_average_waiting_time =
 nornal_data[time].total_ring_time/ nornal_data[time].total_calls
```

Queue Callback Summary report

To display **Queue Callback Summary** report, you need to aggregate queue call records based on callback-related fields.

```
SELECT
  dst,
  COUNT(*) AS total_count,
  SUM(CASE WHEN is_qcb = 1 THEN 1 ELSE 0 END) AS qcb_count,
  SUM(
    CASE WHEN is_qcb = 1
    AND qcb_status = 'success' THEN 1 ELSE 0 END
  ) AS qcb_success_count,
  SUM(
    CASE WHEN is_qcb = 1
    AND qcb_status != 'success' THEN 1 ELSE 0 END
  ) AS qcb_failed_count
FROM
  `cdr`.`cdr_cleaned`
WHERE
  start_ts >= 1774108800000000
  AND start_ts <= 1776787199999999
```

```
  AND call_flow_number in ('6404')
  AND call_flow = 'Queue'
  AND is_group = '1'
GROUP BY
  `dst`
```

The query returns the following data:

- `dst`: Queue number.
- `qcb_failed_count`: The number of failed callbacks.
- `qcb_success_count`: The number of successful callbacks.
- `qcb_count`: The total number of callbacks for which callers requested successfully.
- `total_count`: The total number of calls that the queue received.

Queue Performance report

To display **Queue Performance** report, you need to aggregate queue call data and calculate related metrics.

1. Aggregate queue call data excluding `q_half_consult` records to build the dataset `normal_data`.

```
SELECT
  `dst`,
  Count(*) as total_calls,
  sum(answered) as answered_calls,
  sum(missed) as missed_calls,
  sum(abandoned) as abandoned_calls,
  max(ring_duration) as max_ring_time,
  sum(answered_ring_duration) as total_answered_ring_time,
  sum(ring_duration) as total_ring_time,
  sum(hold_duration) as total_hold_time,
  sum(talk_duration) as total_talking_time,
  sum(member_hold_duration) as total_member_hold_duration,
  sum(member_answered_count) as total_member_answered,
  SUM(
    CASE WHEN dst = '6404'
    AND disposition = 'ANSWERED'
    AND (
      ring_duration - COALESCE(join_announcement_duration, 0)
    )< 60 THEN 1 ELSE 0 END
  ) AS total_sla_call
FROM
  cdr.cdr_cleaned as c
```

```
WHERE
  c.is_group = 1
  AND (
    not (
      c.ring_duration <= 1
      and c.abandoned = 1
    )
  )
  AND (
    not (
      c.talk_duration < 1
      and c.answered = 1
    )
  )
  AND flag != 'q_half_consult'
  AND c.start_ts >= 1774108800000000
  AND c.start_ts <= 1776787199999999
  AND call_flow = 'Queue'
  AND call_flow_number = '6404'
GROUP BY
  `dst`
```

2. Aggregate queue call data with the `q_half_consult` records to build the dataset `q_half_consult_data`.

```
SELECT
  c.dst,
  sum(c.abandoned) as abandoned_calls,
  sum(c.ring_duration) as total_ring_time,
  sum(
    CASE WHEN h.answered > 0 THEN c.ring_duration ELSE 0 END
  ) as total_answered_ring_time,
  max(
    c.ring_duration + h.ring_duration
  ) as max_ring_time
FROM
  cdr.cdr_cleaned as c
  LEFT JOIN (
    SELECT
      uid,
      answered,
      ring_duration
    FROM
      cdr.cdr_cleaned as c
    WHERE
      c.is_group = 1
```

```
      AND flag != 'q_half_consult'
      AND c.start_ts >= 1774108800000000
      AND c.start_ts <= 1776787199999999
      AND call_flow = 'Queue'
      AND call_flow_number = '6404'
  ) AS h ON h.uid = c.another_uid
WHERE
  c.is_group = 1
  AND flag = 'q_half_consult'
  AND c.start_ts >= 1774108800000000
  AND c.start_ts <= 1776787199999999
  AND call_flow = 'Queue'
  AND call_flow_number = '6404'
GROUP BY
  `c`.`dst`
```

3. Merge `normal_data` and `q_half_consult_data` to generate the final metrics data.

```
nornal_data[dst].queue_num = nornal_data[dst].dst
nornal_data[dst].total_calls = nornal_data[dst].total_calls
nornal_data[dst].answered_calls = nornal_data[dst].answered_calls
nornal_data[dst].abandoned_calls = nornal_data[dst].abandoned_calls
nornal_data[dst].missed_calls = nornal_data[dst].missed_calls
nornal_data[dst].total_sla_call = nornal_data[dst].total_sla_call
nornal_data[dst].total_answered_ring_time +=
 q_half_consult_data[dst].total_answered_ring_time
nornal_data[dst].total_ring_time +=
 q_half_consult_data[dst].total_ring_time
if q_half_consult_data[dst].max_ring_time >
 nornal_data[dst].max_ring_time {
    nornal_data[dst].max_waiting_time =
 q_half_consult_data[dst].max_ring_time
}
nornal_data[dst].average_waiting_time=nornal_data[dst].total_answered_
ring_time/nornal_data[dst].answered_calls
nornal_data[dst].average_talking_time=(nornal_data[dst].total_talking_
time+nornal_data[dst].total_hold_time)/nornal_data[dst].answered_calls
nornal_data[dst].average_handle_time=(nornal_data[dst].total_answered_
ring_time+nornal_data[dst].total_talking_time+nornal_data[dst].total_h
old_time)/nornal_data[dst].answered_calls
nornal_data[dst].average_hold_time=(nornal_data[dst].total_hold_time)
/nornal_data[dst].answered_calls

nornal_data[dst].anwsered_rate=nornal_data[dst].answered_calls/nornal_
data[dst].total_calls
```

```
nornal_data[dst].missed_rate=nornal_data[dst].missed_calls/nornal_data
[dst].total_calls
nornal_data[dst].abandoned_rate=nornal_data[dst].abandoned_calls/norna
l_data[dst].total_calls
nornal_data[dst].all_call_average_waiting_time=nornal_data[dst].total_
ring_time/nornal_data[dst].total_calls
```

Queue Performance Activity report

To display **Queue Performance Activity** report, you need to aggregate queue call data from different call scenarios and compute performance metrics.

1. Aggregate queue call records excluding `q_half_consult` records to build the base dataset `normal_data`.

```
SELECT
  day AS time,
  Count(*) as total_calls,
  sum(answered) as answered_calls,
  sum(missed) as missed_calls,
  sum(abandoned) as abandoned_calls,
  max(ring_duration) as max_ring_time,
  sum(answered_ring_duration) as total_answered_ring_time,
  sum(ring_duration) as total_ring_time,
  sum(hold_duration) as total_hold_time,
  sum(talk_duration) as total_talking_time,
  sum(member_hold_duration) as total_member_hold_duration,
  sum(member_answered_count) as total_member_answered,
  SUM(
    CASE WHEN dst = '6404'
    AND disposition = 'ANSWERED'
    AND (
      ring_duration - COALESCE(join_announcement_duration, 0)
    )< 60 THEN 1 ELSE 0 END
  ) AS total_sla_call
FROM
  `cdr`.`cdr_cleaned`
WHERE
  is_group = 1
  AND flag != 'q_half_consult'
  AND (
    not (
      ring_duration <= 1
      and abandoned = 1
    )
```

```
  )
  AND start_ts >= 1774972800000000
  AND start_ts <= 1777564799999999
  AND call_flow = 'Queue'
  AND call_flow_number = '6404'
GROUP BY
  `time`
```

2. Aggregate queue call records with the `q_half_consult` flag to build the dataset `q_half_consult_data`.

```
SELECT
  day AS time,
  sum(c.abandoned) as abandoned_calls,
  sum(c.ring_duration) as total_ring_time,
  sum(
    CASE WHEN h.answered > 0 THEN c.ring_duration ELSE 0 END
  ) as total_answered_ring_time,
  max(
    c.ring_duration + h.ring_duration
  ) as max_ring_time
FROM
  cdr.cdr_cleaned as c
  LEFT JOIN (
    SELECT
      uid,
      answered,
      ring_duration
    FROM
      cdr.cdr_cleaned as c
    WHERE
      c.is_group = 1
      AND flag != 'q_half_consult'
      AND c.start_ts >= 1774972800000000
      AND c.start_ts <= 1777564799999999
      AND call_flow = 'Queue'
      AND call_flow_number = '6404'
  ) AS h ON h.uid = c.another_uid
WHERE
  is_group = 1
  AND flag = 'q_half_consult'
  AND start_ts >= 1774972800000000
  AND start_ts <= 1777564799999999
  AND call_flow = 'Queue'
  AND call_flow_number = '6404'
GROUP BY
```

```
  `time`
```

3. Combine `normal_data` and `q_half_consult_data` to generate final queue performance metrics.

```
nornal_data[time].time = nornal_data[dst].time
nornal_data[time].total_calls = nornal_data[time].total_calls
nornal_data[time].answered_calls = nornal_data[time].answered_calls
nornal_data[time].abandoned_calls = nornal_data[time].abandoned_calls
nornal_data[time].missed_calls = nornal_data[time].missed_calls
nornal_data[time].total_sla_call = nornal_data[time].total_sla_call
nornal_data[time].total_answered_ring_time +=
 q_half_consult_data[time].total_answered_ring_time
nornal_data[time].total_ring_time +=
 q_half_consult_data[time].total_ring_time
if q_half_consult_data[time].max_ring_time >
 nornal_data[time].max_ring_time {
    nornal_data[time].max_waiting_time =
 q_half_consult_data[time].max_ring_time
}
nornal_data[time].answered_hold_time =
 nornal_data[time].total_hold_time
nornal_data[time].answered_call_time =
 nornal_data[time].total_answered_ring_time+nornal_data[time].total_ta
lking_time+nornal_data[time].total_hold_time
nornal_data[time].total_talk_time =
 nornal_data[time].total_talking_time+nornal_data[time].total_hold_tim
e
nornal_data[time].answered_waiting_time =
 nornal_data[time].total_answered_ring_time
nornal_data[time].average_waiting_time=nornal_data[time].total_answere
d_ring_time/nornal_data[time].answered_calls
nornal_data[time].average_talking_time=(nornal_data[time].total_talkin
g_time+nornal_data[time].total_hold_time)/nornal_data[time].answered_c
alls
nornal_data[time].average_handle_time=(nornal_data[time].total_answere
d_ring_time+nornal_data[time].total_talking_time+nornal_data[time].tot
al_hold_time)/nornal_data[time].answered_calls
nornal_data[time].average_hold_time=(nornal_data[time].total_hold_time
)/nornal_data[time].answered_calls
nornal_data[time].total_waiting_time =
 nornal_data[time].total_ring_time
nornal_data[time].sla = nornal_data[time].total_sla_call /
 nornal_data[time].total_calls
nornal_data[time].anwsered_rate=nornal_data[time].answered_calls/norna
l_data[time].total_calls
```

```
nornal_data[time].missed_rate=nornal_data[time].missed_calls/nornal_da
ta[time].total_calls
nornal_data[time].abandoned_rate=nornal_data[time].abandoned_calls/nor
nal_data[time].total_calls
nornal_data[time].all_call_average_waiting_time=nornal_data[time].tota
l_ring_time/nornal_data[time].total_calls
```

Unreturned Missed Call report

To display **Unreturned Missed Call** report, you need to query missed call records and correlate them with answered and callback records to determine whether each missed call has been returned.

1. Retrieve missed call records.

```
SELECT
  id,
  uid,
  timestamp,
  src,
  srcname,
  dst,
  dstname,
  ring_duration,
  dstnameprefix,
  disposition,
  call_flow,
  call_type
FROM
  `cdr`.`cdr_cleaned`
WHERE
  call_type = 'Inbound'
  AND is_display = 1
  AND disposition in ('ABANDONED', 'NO ANSWER', 'BUSY')
  AND (
    NOT(
      disposition = 'ABANDONED'
      and ring_duration < 5
    )
  )
  AND start_ts >= 1774108800000000
  AND start_ts <= 1776787199999999
ORDER BY
  uid,
```

```
  timestamp;
```

2. Remove duplicate records with the same `uid` from missed call records, keeping only the record with the latest `timestamp` for each `uid`, and build the base dataset `list_-map`.

```
list_map[uid].uid = miss_call_data.uid
list_map[uid].timestamp = miss_call_data.timestamp
list_map[uid].src = miss_call_data.src
list_map[uid].srcname = miss_call_data.srcname
list_map[uid].dst= miss_call_data.dst
list_map[uid].dstname = miss_call_data.dstname
list_map[uid].call_to_type = miss_call_data.call_flow
list_map[uid].unreturn_status = 0
list_map[uid].return_time = 0
list_map[uid].miss_call_type = miss_call_data.disposition
list_map[uid].call_type = miss_call_data.call_type
list_map[uid].ring_duration = miss_call_data.ring_duration
```

3. Retrieve answered call records related to caller number.

```
SELECT
  max(timestamp) as timestamp,
  src,
  dst,
  disposition
FROM
  `cdr`.`cdr_cleaned`
WHERE
  timestamp > 1775534367893043
  AND is_display = 1
  AND disposition = 'ANSWERED'
  AND (
    src = '2233123456'
    or dst = '2233123456'
  )
GROUP BY
  src,
  dst,
  disposition
```

4. Determine the latest timestamp of answered calls for each caller number.

```
//has_return_time_map
if answered_call_data[src].timestamp > has_return_time_map[src] {
    has_return_time_map[src] = answered_call_data[src].timestamp
}
```

5. Retrieve callback records associated with the missed caller number.

```
SELECT
  max(timestamp) as timestamp,
  src,
  dst,
  disposition
FROM
  `cdr`.`cdr_cleaned`
WHERE
  timestamp >= 1775534367893043
  AND is_display = 1
  AND (
    (dst = '2233123456')
    or (
      src = '2233123456'
      and disposition = 'ANSWERED'
    )
  )
GROUP BY
  src,
  dst,
  disposition
```

6. Determine the latest callback timestamp for each missed caller number.

```
//last_return_time_map
if last_answered_call_data[src].timestamp > last_return_time_map[src]
 {
    last_return_time_map[src] = last_answered_call_data[src].timestamp
}
```

7. Determine unreturned call status for each missed call record.

```
list_map[uid].unreturn_status=2
src = list_map[uid].src
//
if has_return_time_map[src] > list_map[uid].timestamp {
    list_map[uid].unreturn_status = 1
}
//
if last_return_time_map[src] > list_map[uid].timestamp {
    list_map[uid].return_time = last_return_time_map[src]
}
```

Ring Group Statistics report

To display **Ring Group Statistics** report, you need to aggregate ring group call data from both the ring group level and its members, then calculate call statistics and build the final structured result.

1. Aggregate call statistics from ring group level to generate the list dataset.

```
SELECT
  dst as ring_group_num,
  SUM(
    CASE WHEN disposition = 'ANSWERED' THEN 1 ELSE 0 END
  ) AS answered_calls,
  count(*) as total_calls
FROM
  `cdr`.`cdr_cleaned`
WHERE
  is_group = 1
  AND start_ts >= 1776441600000000
  AND start_ts <= 1779119999999999
  AND call_flow = 'RingGroup'
  AND call_flow_number in ('6300', '6301', '6302')
GROUP BY
  `dst`
```

2. Aggregate call statistics from member-level to build the member_list dataset.

```
SELECT
  `dst`,
  call_flow_number as ring_group_num,
  SUM(
    CASE WHEN disposition = 'ANSWERED' THEN 1 ELSE 0 END
  ) AS answered_calls,
  count(*) as total_calls
FROM
  `cdr`.`cdr_cleaned`
WHERE
  is_group = 0
  AND start_ts >= 1776441600000000
  AND start_ts <= 1779119999999999
  AND call_flow = 'RingGroup'
  AND call_flow_number in ('6300', '6301', '6302')
GROUP BY
  dst,
  call_flow_number
```

3. Sort both the ring group list and the member_list by answered rate in descending order, then generate the final structured output.

```
{"ring_group_num":"6200","answered_calls":10,"total_calls":20,"member_
list":[{"ext_num":"1000","answered_calls":10,"total_calls":20}...]}
```

PBX Call Activity report

To display **PBX Call Activity** report, you need to aggregate call activity data from three sources: trunk calls, device calls, and internal calls. Each dataset is calculated separately and then used for overall call analysis.

1. Retrieve data for calls made or received via trunks.
    a. Retrieve trunk call activity data, excluding multi-party calls.

```
SELECT
  max(day) AS time,
  max(srctrunk) as srctrunk,
  max(dsttrunk) as dsttrunk,
  SUM(talk_duration + hold_duration) as total_talk_duration,
  MAX(concurrent) as max_concurrent_call,
  uid,
  0 as total_calls
FROM
  `cdr`.`cdr_cleaned`
WHERE
  (
    srctrunk in (
      '97账号中继注册段238', '235点对点kk'
    )
    or dsttrunk in (
      '97账号中继注册段238', '235点对点kk'
    )
  )
  AND is_display = 1
  AND (
    NOT(
      dstnameprefix in ('Queue', 'RingGroup')
      and disposition = 'ANSWERED'
    )
  )
  AND NOT(srcname = 'Conference Call')
  AND start_ts >= 1774972800000000
  AND start_ts <= 1777564799999999
```

```
GROUP BY
  `uid`;
```

b. Calculate call volume, total talk duration, and maximum concurrent calls for the same trunk within the same time period, excluding multi-party calls.

c. Retrieve trunk call activity data, including multi-party calls.

```
SELECT
  max(day) AS time,
  srctrunk,
  dsttrunk,
  SUM(talk_duration + hold_duration) as total_talk_duration,
  MAX(concurrent) as max_concurrent_call,
  uid,
  COUNT(*) as total_calls
FROM
  `cdr`.`cdr_cleaned`
WHERE
  (
    srctrunk in (
      '97账号中继注册段238', '235点对点kk'
    )
    or dsttrunk in (
      '97账号中继注册段238', '235点对点kk'
    )
  )
  AND is_display = 1
  AND (
    NOT(
      dstnameprefix in ('Queue', 'RingGroup')
      and disposition = 'ANSWERED'
    )
  )
  AND srcname = 'Conference Call'
  AND start_ts >= 1774972800000000
  AND start_ts <= 1777564799999999
GROUP BY
  srctrunk,
  dsttrunk,
  uid;
```

d. Calculate call volume, total talk duration, and maximum concurrent calls for the same trunk within the same time period, including multi-party calls.

2. Retrieve call activity data for the entire system, including internal calls, inbound calls, and outbound calls across all trunks.

a. Retrieve device call activity data, excluding audio conference and paging.

```
SELECT
  max(day) AS time,
  uid,
  SUM(talk_duration + hold_duration) as total_talk_duration,
  MAX(concurrent) as max_concurrent_call
FROM
  `cdr`.`cdr_cleaned`
WHERE
  (
    NOT(
      dstnameprefix in ('Queue', 'RingGroup')
      and disposition = 'ANSWERED'
    )
  )
  AND call_flow != 'Audio Conference'
  AND call_flow != 'Paging'
  AND dstnameprefix != 'Paging'
  AND start_ts >= 1774972800000000
  AND start_ts <= 1777564799999999
GROUP BY
  `uid`;
```

b. Calculate call volume, total talk duration, and maximum concurrent calls within the same time period, excluding multi-party calls.
c. Query call data for audio conference calls and paging calls.

```
SELECT
  uid,
  src,
  dst,
  timestamp,
  talk_duration,
  hold_duration,
  concurrent,
  dstnameprefix,
  call_flow,
  call_type,
  month,
  day,
  hour
FROM
  `cdr`.`cdr_cleaned`
WHERE
  (
    NOT(
```

```
      dstnameprefix in ('Queue', 'RingGroup')
      and disposition = 'ANSWERED'
    )
  )
  AND (
    call_flow = 'Audio Conference'
    OR call_flow = 'Paging'
    OR dstnameprefix = 'Paging'
  )
  AND start_ts >= 1774972800000000
  AND start_ts <= 1777564799999999
ORDER BY
  timestamp asc,
  dstnameprefix desc
LIMIT
  5000;
```

d. Apply the following aggregation rules:

- **Paging Calls**: Records with the same `uid` are treated as a single call; device call volume is incremented by 1 per record; call duration is taken only from the Paging record (`talk_duration` + `hold_duration` where `dstnameprefix = Paging`); maximum concurrent calls are calculated based on the paging record.
- **Audio Conference Calls**: Records with the same `uid_src_dst` are treated as a single call; device call volume is incremented by 1 per record; call duration is the sum of `talk_duration` and `hold_duration`; maximum concurrent calls are calculated based on the aggregated audio conference call records.

e. Calculate call volume, total talk duration, and maximum concurrent calls for device calls within the same time period, including multi-party calls.

3. Retrieve data for internal calls.

a. Retrieve internal calls data, excluding audio conference calls and paging calls.

```
SELECT
  max(day) AS time,
  uid,
  SUM(talk_duration + hold_duration) as total_talk_duration,
  MAX(concurrent) as max_concurrent_call
FROM
  `cdr`.`cdr_cleaned`
WHERE
  call_type = 'Internal'
  AND src != 'PBX'
  AND (
```

```
    NOT(
      dstnameprefix in ('Queue', 'RingGroup')
      and disposition = 'ANSWERED'
    )
  )
  AND call_flow != 'Audio Conference'
  AND call_flow != 'Paging'
  AND dstnameprefix != 'Paging'
  AND start_ts >= 1774972800000000
  AND start_ts <= 1777564799999999
GROUP BY
  `uid`;
```

b. Calculate call volume, total talk duration, and maximum concurrent calls for internal calls within the same time period, excluding multi-party calls.
c. Query call data for audio conference calls and paging calls.

```
SELECT
  uid,
  src,
  dst,
  timestamp,
  talk_duration,
  hold_duration,
  concurrent,
  dstnameprefix,
  call_flow,
  call_type,
  month,
  day,
  hour
FROM
  `cdr`.`cdr_cleaned`
WHERE
  call_type = 'Internal'
  AND (
    NOT(
      dstnameprefix in ('Queue', 'RingGroup')
      and disposition = 'ANSWERED'
    )
  )
  AND (
    call_flow = 'Audio Conference'
    OR call_flow = 'Paging'
    OR dstnameprefix = 'Paging'
  )
```

```
  AND start_ts >= 1774972800000000
  AND start_ts <= 1777564799999999
ORDER BY
  timestamp asc,
  dstnameprefix desc
LIMIT
  5000;
```

d. Apply the following aggregation rules:

- **Paging Calls**: Records with the same `uid` are treated as a single call; internal call volume is incremented by 1 per record; call duration is taken only from the Paging record (`talk_duration` + `hold_duration` where `dstnameprefix = Paging`); maximum concurrent calls are calculated based on the paging record.
- **Audio Conference Calls**: Records with the same `uid_src_dst` are treated as a single call; internal call volume is incremented by 1 per record; call duration is the sum of `talk_duration` and `hold_duration`; maximum concurrent calls are calculated based on the aggregated audio conference call records.

e. Calculate call volume, total talk duration, and maximum concurrent calls for internal calls within the same time period, including multi-party calls.

Transcription Usage Details

To display **Transcription Usage Details** report, you need to retrieve the total transcription usage grouped by usage type.

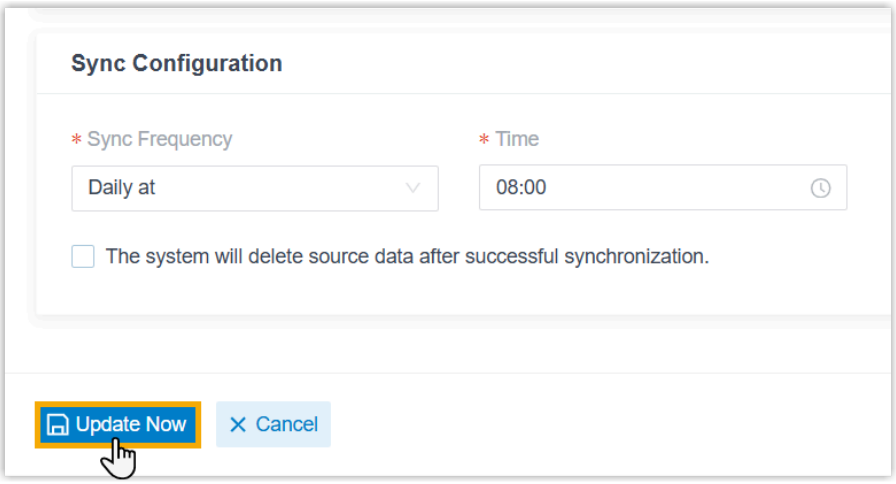
```
SELECT
  usage_type,
  SUM(duration) as total_duration
FROM
  `pbx`.`ai_usage_record`
WHERE
  (
    usage_type in (
      'call_transcription', 'voicemail_transcription'
    )
  )
  AND (timestamp >= 1767196800)
  AND (timestamp < 1779206399)
GROUP BY
  usage_type
```

Manually Trigger Data Synchronization to a Third-party Database

By default, Yeastar P-Series PBX System synchronizes the specified data to the third-party database at a scheduled interval. You can also initiate synchronization manually at any time, any data generated after the last synchronization will be synchronized to the database immediately.

Procedure

1. Log in to PBX web portal, go to **Integrations > Data Connectors**.
2. Manually trigger data synchronization.
    a. On the bottom of the page, click **Update Now**.



    b. In the pop-up window, click **OK**.

    The system will verify the database connection.

    c. In the pop-up window, click **Confirm**.

Result

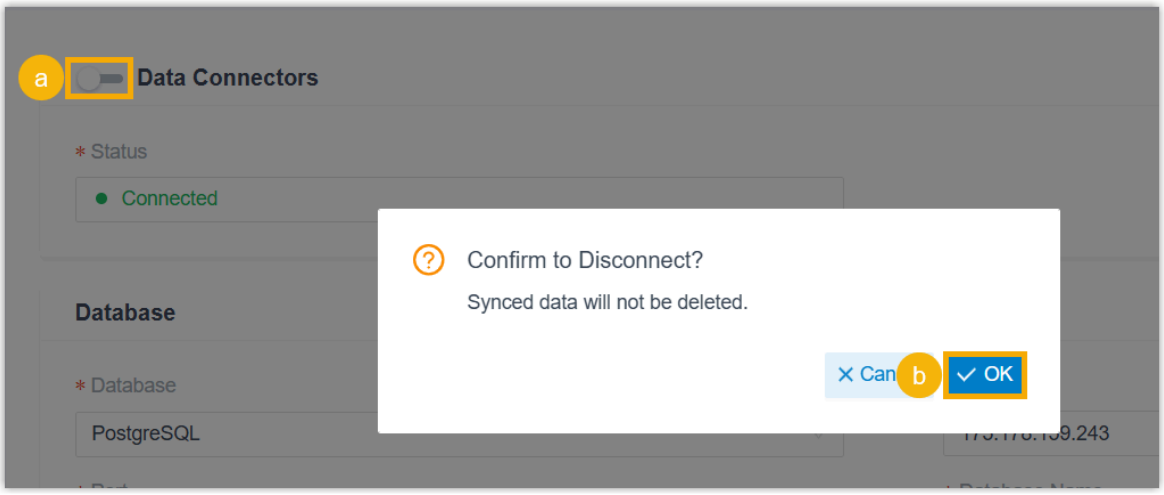
Any data generated after the last synchronization will be synchronized to the database immediately.

Disable Data Connector Integration

This topic describes how to disable data connector integration on Yeastar P-Series PBX System.

Procedure

1. Log in to PBX web portal, go to **Integrations > Data Connectors**.
2. Disable data connector integration.



   a. On the top of the integration page, turn off the switch.
   b. In the pop-up window, click **OK**.

Result

- The **Status** is displayed as "Disconnect", indicating that the integration is disconnected.
- The synchronized data will not be deleted from the third-party database.