Lakeflow Connect: PostgreSQL Connector

Private Preview Documentation: November 2025

Introduction

As part of Lakeflow Connect, Databricks is introducing a private preview of the PostgreSQL connector, allowing you to ingest data directly from your PostgreSQL database into Databricks.

NOTE: You must run the gateway in continuous mode to ensure that the WAL does not become full. This can result in your database going down.

Support and Feedback

We welcome your questions, feature requests, and feedback. Please contact your account team and CC connector-support@databricks.com.

Enabling Private Preview

To join the private preview, go to your workspace's **Previews** page and enable the **Lakeflow Connect for PostgreSQL** flag.

Prerequisites

Current Limitations

- Connector cannot connect to a read replica i.e. replication from the read replica is not supported.
- Only one replication configuration is supported per Database catalog.
- Replication applies to DML operations only: `INSERT`, `UPDATE`, and `DELETE`.
  - Schema changes (DDL) are detected in some cases but not automatically applied.
  - To reflect detected changes, you must trigger a full table refresh.
  - See details on supported schema changes.
- Not all PostgreSQL data types are supported. Refer to the FAQ for type mapping details.
- The maximum supported row size is 10MB.
- Table/schema/catatlog names starting with quotes are not currently supported.

Database Requirements

A **user with the `REPLICATION` role** is required. This user:

- Creates the publication and replication slot on the source PostgreSQL server.
- Reads from the replication slot during ingestion.
- See PostgreSQL REPLICATION role for more details.

Server configuration recommendation:
It is recommended not to set `max_slot_wal_keep_size` to -1 (the default value), as this may lead to unbounded WAL bloat due to the retention of slots. Depending on your workload, it is advised to set this parameter to a finite value. (In future connector releases, this configuration may become mandatory).

Supported variations: AWS RDS for PostgreSQL, AWS Aurora PostgreSQL, PostgreSQL on EC2, PostgreSQL on Azure VMs, Azure Cloud SQL, Google Cloud SQL for PostgreSQL, PostgreSQL on-premise via Express Route/Direct Connect/VPN.

Databricks Requirements

To use the PostgreSQL connector with Lakeflow Connect, ensure the following workspace and permission requirements are met:

Workspace Configuration

- **Unity Catalog** is enabled.
- **Serverless compute** is enabled for **notebooks**, **workflows**, and **DLT**. See Enable serverless compute.

Required Privileges

To create a new connection:

- `CREATE CONNECTION` on the metastore

To use an existing connection:

- `USE CONNECTION` or `ALL PRIVILEGES` on the connection

To read/write data:

- `USE CATALOG` on the **target catalog**
- Either
  - USE SCHEMA, CREATE TABLE, and CREATE VOLUME on an existing schema, or
  - CREATE SCHEMA on the target catalog

Cluster Permissions

- You must have either:
  - **Unrestricted permissions** to create clusters, or
  - A **custom cluster policy** with the following requirements:
    - Family: Job Compute
    - Policy family overrides:

```
{
 "cluster_type": {
    "type": "fixed",
    "value": "dlt"
  },
  "num_workers": {
    "type": "unlimited",
    "defaultValue": 1,
    "isOptional": true
  },
  "runtime_engine": {
    "type": "fixed",
    "value": "STANDARD",
    "hidden": true
  }
}
```

For more information about cluster policies, see Select a cluster policy.

Source Database Setup

Cleaning up replication slots

The Databricks PostgreSQL connector uses logical replication via **pgoutput**, which requires replication slots and publications:

- A **replication slot** retains WAL (Write-Ahead Log) records on the primary server until they are consumed.
- A **publication** defines which tables are available for replication.

*Note: The connector **automatically creates** replication slots and publications, but due to a temporary limitation in the private preview, **cleanup must be done manually**.*

Manual replication slot cleanup

If a pipeline is deleted, you must also delete its associated replication slot to avoid:

- WAL bloating
- Disk usage issues
- Errors such as: `maximum number of replication slots already in use`

Replication slots are named:

```
dbx_{DB_OID}_{PIPELINE_ID}
```

Publications are named:

```
dbx_pub_{DB_OID}_{PIPELINE_ID}
```

Example command to delete the replication slot:

```
SELECT
pg_drop_replication_slot('dbx_1406676_3cfcaccf_6633_497c_a5ef_1
a4e7786ab55');
```

Preparing the replication user

Logical replication requires a **user with replication privileges**. This user creates the publication and replication slot, and streams WAL data.

Note for AWS RDS: *Logical replication is not enabled by default.You must attach a custom parameter group with the necessary settings. See the AWS RDS documentation before proceeding.*

1. Log into the PostgreSQL client:

Vanilla PostgreSQL:

```
psql -U $POSTGRESQL_ROOT_USER
```

AWS RDS:

```
psql -h
postgres-prpr.abvstewht5ukt.ap-south-1.rds.amazonaws.com -p
5432 -d postgres -U postgres
```

2. Create a user for replication in the source PostgreSQL instance. For example, the following creates a user called `replicate_user`:

```
CREATE USER replicate_user PASSWORD 'replicate#123';
```

3. Grant the replication role to the replication user using following command:

Vanilla PostgreSQL, Azure PG:

```
ALTER USER replicate_user WITH REPLICATION;
```

AWS RDS:

```
GRANT rds_replication TO replicate_user;
```

4. Grant this user CREATE permissions on the source database so that it can create publications for those tables. For example, the following grants `replicate_user` permissions on the `db1` database:

```
GRANT CREATE ON DATABASE db1 TO replicate_user;
```

5. Grant USAGE on all the schemas of which tables are to be replicated:

```
GRANT USAGE ON SCHEMA myschema TO replicate_user;
```

6. Grant ownership of source tables to allow both the original owner and the replication user to manage the tables.

```
CREATE ROLE data_ownership_role;

GRANT data_ownership_role TO original_owner_of_the_table;
GRANT data_ownership_role TO replicate_user;

ALTER TABLE myschema.t1 OWNER TO data_ownership_role;
```

Setting up PostgreSQL for replication

To use logical replication with the Databricks PostgreSQL connector, you'll need to adjust certain database settings.

1. Edit the PostgreSQL Configuration

Vanilla PostgreSQL:

```
vi $PGDATA/postgresql.conf
```

AWS RDS:
RDS uses parameter groups instead of direct config files.Edit the associated parameter group in the RDS console.

2. Enable logical replication

Vanilla PostgreSQL:

```
wal_level = logical
```

AWS RDS:
RDS does not expose wal_level. Instead, set in the associated parameter group:

```
rds.logical_replication = 1
```

3. Configure replication slot capacity

Vanilla PostgreSQL:

```
max_replication_slots = 10 //default setting
```

AWS RDS:
Edit the value of max_replication_slots in the parameter's group.

Important Note: *Before starting a pipeline, ensure there are no long-running transactions on the source database. These can block the creation of replication slots and prevent ingestion from starting.*

Setting Replica Identities

To enable incremental ingestion using logical replication, source tables must be configured with the appropriate replica identity. This setting determines what information PostgreSQL includes in the Write-Ahead Log (WAL) when rows are updated or deleted.

The correct settings will depend on the table structure.

A. Tables with a primary key and no TOASTable (variable length) columns

If the table has a primary key and does not contain TOASTable columns (e.g., TEXT, BYTEA, VARCHAR(n) with large values), ensure the replica identity is set to DEFAULT.

Check the current configuration by running the following:

```
SELECT relname AS table_name, relreplident AS replica_identity
FROM pg_class WHERE relname = 't1';
```

If the result shows anything other than `'d'` (default), run:

```
ALTER TABLE your_table_name REPLICA IDENTITY DEFAULT;
```

B. Tables with a primary key and TOASTable (variable length) columns

For tables that include large variable-length columns, set the replica identity to FULL to ensure complete row data is written to the WAL:

```
ALTER TABLE your_table_name REPLICA IDENTITY FULL;
```

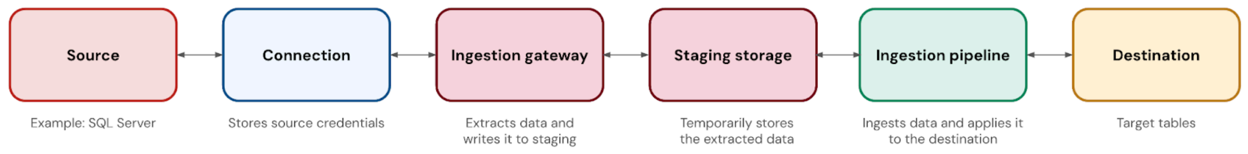
C. Tables without a primary key

If the table lacks a primary key, logical replication requires REPLICA IDENTITY FULL to track changes:

```
ALTER TABLE your_table_name REPLICA IDENTITY FULL;
```

Connector Setup

In this next step, you'll create the components to ingest from the database using the PostgreSQL connector.


Source
Example: SQL Server
Connection
Stores source credentials
Ingestion gateway
Extracts data and writes it to staging
Staging storage
Temporarily stores the extracted data
Ingestion pipeline
Ingests data and applies it to the destination
Destination
Target tables

There are five components:

- **Connection:** A Unity Catalog securable object that stores authentication details for the database.
- **Ingestion gateway:** A pipeline that extracts snapshots, change logs, and metadata from the source database. The gateway runs on classic compute, and it runs continuously to capture changes before change logs can be truncated in the source.
- **Staging storage:** A Unity Catalog volume that temporarily stores extracted data before it's applied to the destination table. This allows you to run your ingestion pipeline at whatever schedule you'd like, even as the gateway continuously captures changes. It also helps with failure recovery. You automatically create a staging storage volume when you deploy the gateway, and you can customize the catalog and schema where it lives. Data is automatically purged from staging after 30 days.
- **Ingestion pipeline:** A pipeline that moves the data from staging storage into the destination tables. The pipeline runs on serverless compute. Currently, each ingestion pipeline must be associated with exactly one ingestion gateway.
- **Destination tables:** The tables where the ingestion pipeline writes the data. These are streaming tables, which are Delta tables with extra support for incremental data processing.

Create a PostgreSQL connection

1. In the Databricks workspace, click **Catalog > External Data > Connections.**
2. Click **Create connection**. (If this option is missing, you may lack connection privileges.)
3. Enter a unique **Connection name**.
4. For **Connection type**, select **PostgreSQL**.
5. For **Host**, specify the PostgreSQL domain name.
6. For **User** and **Password**, enter the PostgreSQL login credentials of the replication user.
7. Click Create.

Create a target catalog and schema

1. In the Databricks workspace, click **Catalog**.
2. On the **Catalog** tab, do one of the following:
3. Click **Create catalog**. If you don't see this button, you don't have CREATE CATALOG privileges.
4. Enter a unique name for the catalog, and then click **Create**.
5. Select the catalog you created.
6. Click **Create schema**. If you don't see this button, you don't have CREATE SCHEMA privileges.
7. Enter a unique name for the schema, and then click **Create**.

Create the ingestion gateway and ingestion pipeline

1. Import the provided notebook
Download and import the notebook shared in your onboarding email into your Databricks workspace.

2. Configure ingestion configuration
In the notebook, locate Cell 6, titled Ingestion Configuration.

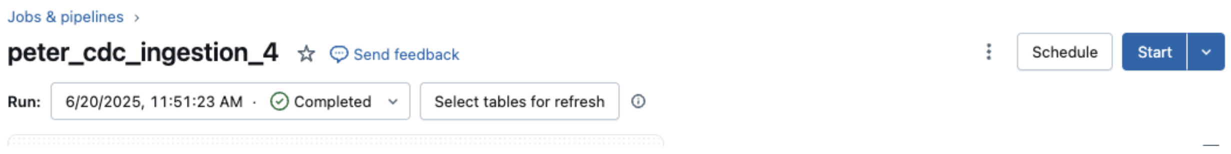
Update the following values:

- **Connection**: Use the name of the PostgreSQL connection you created earlier.
- **Target catalog and schema:** Set these to the target catalog and schema you configured in the previous step.
- **Gateway pipeline name:** Any name you would like to use for the gateway pipeline.
- **Ingestion pipeline name:** Any name you would like to use for the ingestion pipeline.
- **Tables to replicate**: The tables you would like to replicate from your source database into Databricks.

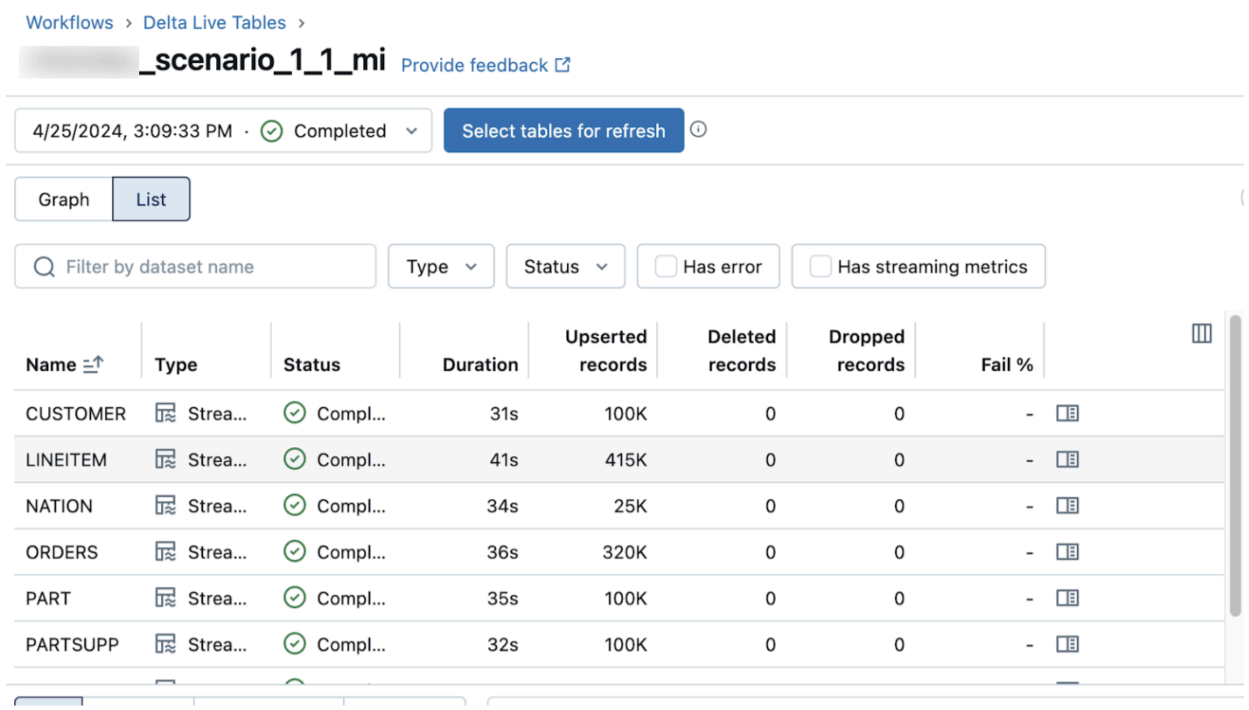
Note*: The ingestion pipeline currently supports only one destination catalog and schema. To ingest into multiple catalogs or schemas, you must create separate gateway-pipeline pairs.*

3. Run all cells in the notebook.
- Cell 7 outputs the ingestion gateway pipeline
- Cell 8 outputs the ingestion pipeline

Run the data ingestion pipeline
1. Navigate to the investigation pipeline created by the notebook.
2. Click Start to initiate the initial ingestion run.
3. Optionally, click schedule to configure recurring runs.
4.



Verify successful data ingestion
Once the pipeline run completes:
- You'll see a DAG view and a table list view showing tables had data ingested.
- These views will update with each subsequent run.



Workflows › Delta Live Tables ›

[redacted]_scenario_1_1_mi Provide feedback

4/25/2024, 3:09:33 PM · Completed | Select tables for refresh

Graph | List

Filter by dataset name | Type | Status | Has error | Has streaming metrics

Name	Type	Status	Duration	Upserted records	Deleted records	Dropped records	Fail %
CUSTOMER	Strea...	Compl...	31s	100K	0	0	-
LINEITEM	Strea...	Compl...	41s	415K	0	0	-
NATION	Strea...	Compl...	34s	25K	0	0	-
ORDERS	Strea...	Compl...	36s	320K	0	0	-
PART	Strea...	Compl...	35s	100K	0	0	-
PARTSUPP	Strea...	Compl...	32s	100K	0	0	-

FAQ

Type mapping

Databricks transforms the following PostgreSQL data types to Delta-compatible data types as follows:

PostgreSQL	Databricks (Current Mappings)
SERIAL	INT
SMALLSERIAL	SMALLINT
BIGSERIAL	BIGINT
SMALLINT	SMALLINT
INT	INT
BIGINT	BIGINT
DECIMAL	DECIMAL
NUMERIC	DECIMAL
REAL	FLOAT

DOUBLE PRECISION	DOUBLE
MONEY	DOUBLE
CHAR	STRING
VARCHAR	STRING
BPCHAR	STRING
TEXT	STRING
BYTEA	BINARY
DATE	DATE
TIMESTAMP	TIMESTAMP_NTZ
TIMESTAMPTZ	TIMESTAMP
BOOLEAN	BOOLEAN
BIT	BINARY
VARBIT	BINARY
UUID	BINARY
OID	DECIMAL

Types that are mapped to STRING

POSTGRESQL	**DATABRICKS**
TIME	STRING
TIMETZ	STRING
INTERVAL	STRING
XML	STRING
JSON	STRING
JSONB	STRING
CIDR	STRING
INET	STRING

MACADDR	STRING
MACADDR8	STRING
POINT	STRING
LINE	STRING
LSEG	STRING
BOX	STRING
PATH	STRING
POLYGON	STRING
CIRCLE	STRING
ARRAY	STRING
ENUM	STRING
NUMRANGE	STRING
TSQUERY	STRING
TSVECTOR	STRING
TSRANGE	STRING
TSTZRANGE	STRING

If attempting to stream to a Delta streaming table with a PostgreSQL table containing unsupported types (CITEXT, GEOGRAPHY, GEOMETRY, HSTORE, LINE SEGMENT, LTREE) behavior will be unknown.

DDL limitations

- TRUNCATE Table is detected and a full refresh will be needed.
- Alter table ADD/DROP columns are detected on a subsequent DML on the table, if the table has a FULL replica identity. If the table has a primary key and default replica identity, then the change will be detected on the next INSERT or UPDATE on the table. Thereafter, a full refresh is required.
- Alter table rename column is detected on the next DML(or INSERT/UPDATE if no FULL identity).
- Alter table ADD/DROP generated column is not detected, and will need a manual full refresh.
- Alter column with default value is not detected.
- Alter table ADD/DROP constraints/check constraints are not detected.
- Replication of generated columns is not supported.
- CREATE INDEX on table is not detected.

- PK widening/shrinking is not supported, as replicant cannot detect add/drop constraints. A manual full refresh is needed.
- If the schema owner or table owner is changed, unless the replication user still has the permission on the table, a manual refresh will be needed after giving the necessary permissions to the replication user.
- ALTER TABLE...SET SCHEMA... to a different schema, will need a manual refresh.
- ALTER COLUMN with USING expression is not detected.
- Attach partition - New changes on the attached partition table will be replicated, existing rows will not be replicated. If older rows are needed a manual full refresh should be triggered.
- Detached partitions will not delete the rows on target. If the rows need to be deleted on target, a manual full refresh should be triggered.