

A Technical Overview to:

Unlock Seamless Logical Replication From Heterogeneous Databases

Presented by Cary Huang







2024-11-06



Empowering Data Integration with Confidence

Agenda



-  Few words about me.
-  Why heterogeneous replication to PostgreSQL.
-  Existing heterogeneous replication tools.
-  Introduce a new heterogeneous tool tailored to PostgreSQL.
-  Discuss architecture and how it works.
-  Q&A and future improvements.



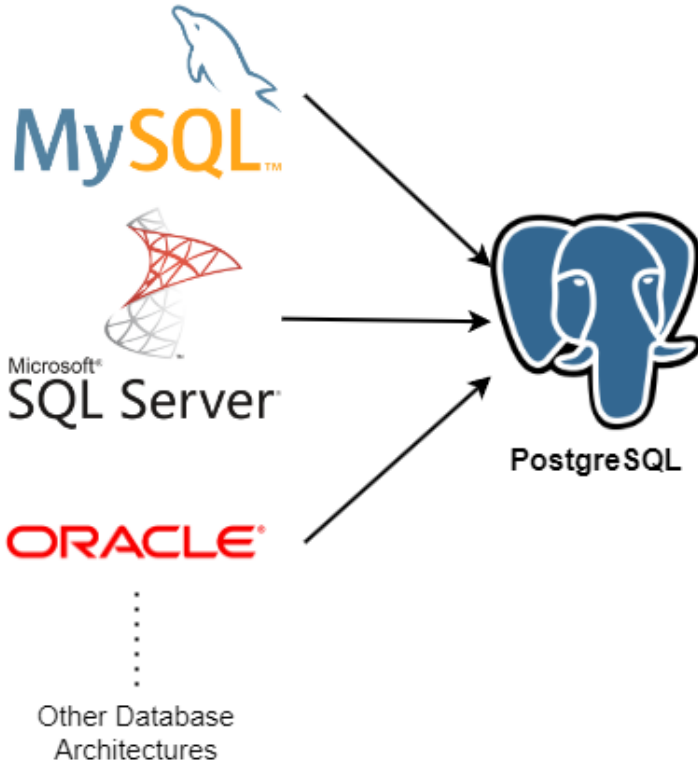
• • • Few Words About Cary



- Electrical Engineering graduate from University of British Columbia (UBC) in 2012.
- Worked in the smart metering and energy sector right out of school.
- Joined Highgo Software in 2019 to start my PostgreSQL journey.
- Post-graduate instructor at Peking University to promote PostgreSQL open-source ecosystem.
- Worked on several aspects of PostgreSQL including sharding enhancements, distributed database, HA, shared storage, serverless, security..etc.
- Now with Hornetlabs to continue my PostgreSQL journey.



Why do we Need Heterogeneous Database Replication to PostgreSQL?



Analytics

PostgreSQL is a strong choice for data consolidation for analytics, reporting, and data warehousing.



Cost Reduction

PostgreSQL is a powerful open-source database to migrate data to from legacy proprietary databases like SQLServer and Oracle.



Modernization

PostgreSQL's extensive feature set, along with JSONB, GIS, geospatial, vector and full-text search that other databases may lack.



Cross-Platform Integrations

Replicating data from diverse sources into PostgreSQL can centralize the data for easier and unified application integration.

Options for Heterogeneous Database Replication to PostgreSQL?



Advantages

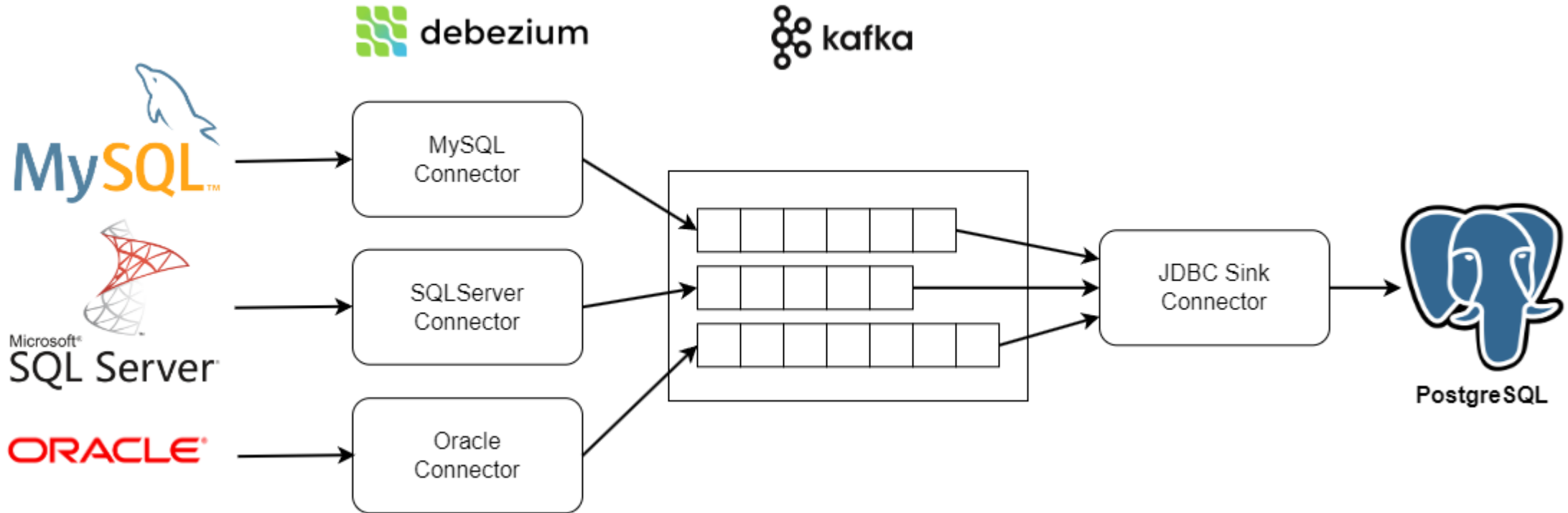
- Real-time change data capture (CDC).
- Broad database support.
- Transactional consistency.
- Fault tolerance and scalability with Kafka.
- Decoupled architecture with Kafka in the middle.
- Open-source.

Disadvantages

- Complex setup (Kafka, Kafka Connect, Zookeeper, Kafka Sink)
- Resource intensive.
- Latency from multiple stages (Source -> Kafka topic -> Kafka Sink -> PostgreSQL)
- Schema management and data transform limitations.



Heterogeneous Replication via Debezium and Kafka



<https://debezium.io/documentation/reference/stable/architecture.html>



Options for Heterogeneous Database Replication to PostgreSQL?



Advantages

- Real-time Change Data Capture (CDC).
- Broad database support.
- Transactional consistency.
- Granular control and customization.
- Robust monitoring and management tools
- High availability topologies (active-active, active-passive).




Disadvantages




- High licensing and operational costs.
- Complex setup, configurations.
- High learning curve.
- Limited flexibility with non-Oracle databases.



Options for Heterogeneous Database Replication to PostgreSQL?



- Both can do real-time Change Data Capture (CDC). 
- Both can ensure transactional consistency. 
- Both can support broad database types. 

- They are not tailored to PostgreSQL (though it is supported). 
- They require complex setups and intensive resource requirements. 
- They have limited data transform flexibilities for PostgreSQL. 

Is there an option more tailored to PostgreSQL?



Introducing 'SynchDB' - A PostgreSQL Extension



- **SynchDB** is a PostgreSQL extension for synchronizing data from different heterogeneous database sources – controlled and managed all within PostgreSQL.



Heterogeneous
Databases Support



Simple Setup and
Installation



Flexible Data
Transform Rules



Simple State and
Error Provisioning



High Performance



DDL and DML
Replications



Real-time Change
Data Capture



Open-source and
Community Support




Concurrent Connector
Workers



How does SynchDB Work?



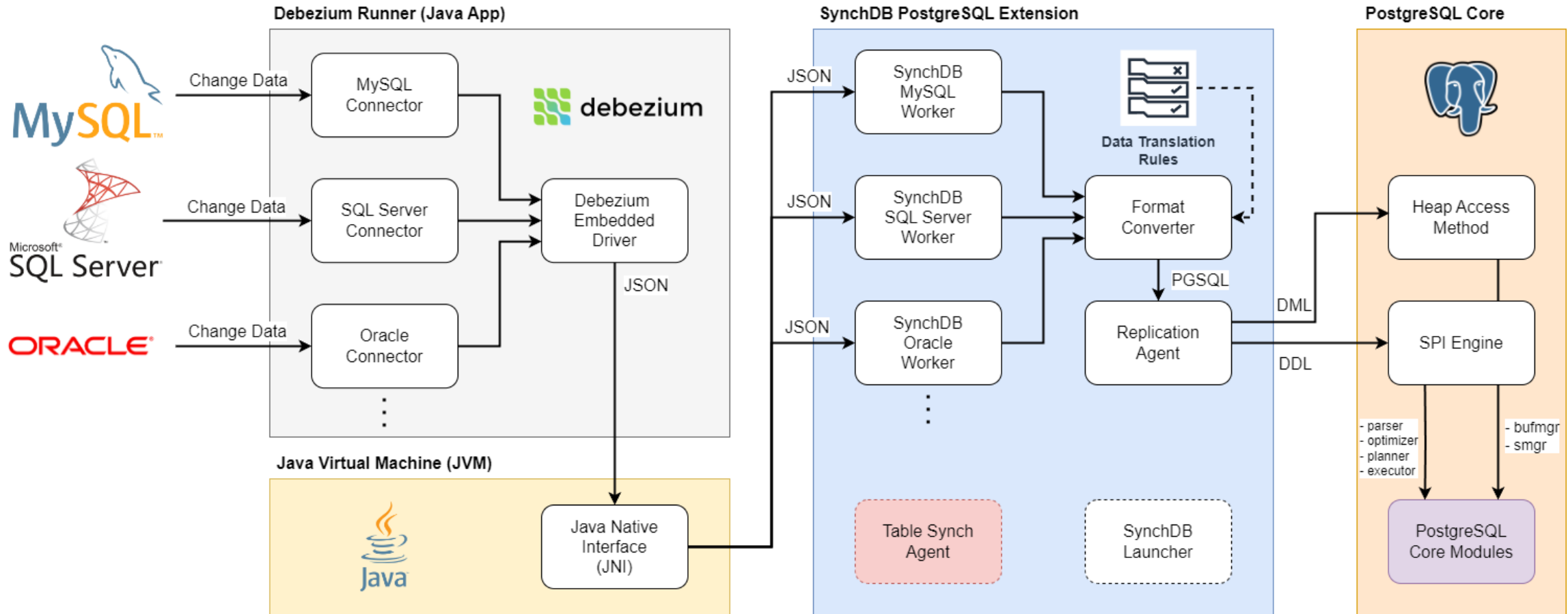
- Written in C and built as a PostgreSQL extension.  **debezium**
- Powered by **Debezium Embedded Engine** that grants PostgreSQL:
 - Access to heterogeneous database connectors for CDC. (MySQL, Oracle, SQLServer, DB2...et)
 - Schema tracking, initial snapshot, offset management...etc.
- No Kafka connect or sink connectors – simplified setup and installation...

But... Debezium is written in Java...

So Java Native Interface (JNI) is required to harness the power of Debezium



SynchDB Architecture



• SynchDB Setup – We Want it Simple

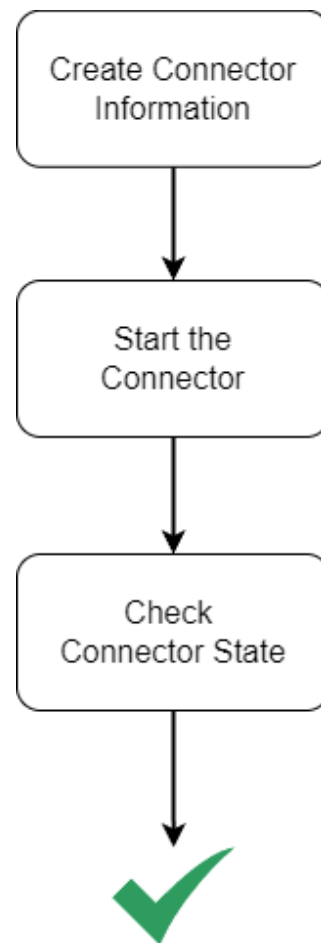
- We envision SynchDB to require minimum efforts to:
 - Install SynchDB to PostgreSQL.
 - Get started quickly.
- SynchDB requires JRE version 17+ to run.



Get Started with SynchDB – We Want it Simple Too!



```
SELECT synchdb_start_engine_bgw('mysqlconn');
```



```
SELECT synchdb_add_conninfo(  
    'mysqlconn',    ← unique connector name  
    '127.0.0.1',   ← remote hostname  
    3306,          ← remote port  
    'mysqluser',   ← remote username  
    'mysqlpwd',    ← remote password  
    'inventory',   ← remote source database  
    'postgres',    ← destination database (PG)  
    'inventory.orders', ← table to replicate  
    'mysql',       ← connector type  
    'myrule.json'); ← custom transform rule file
```

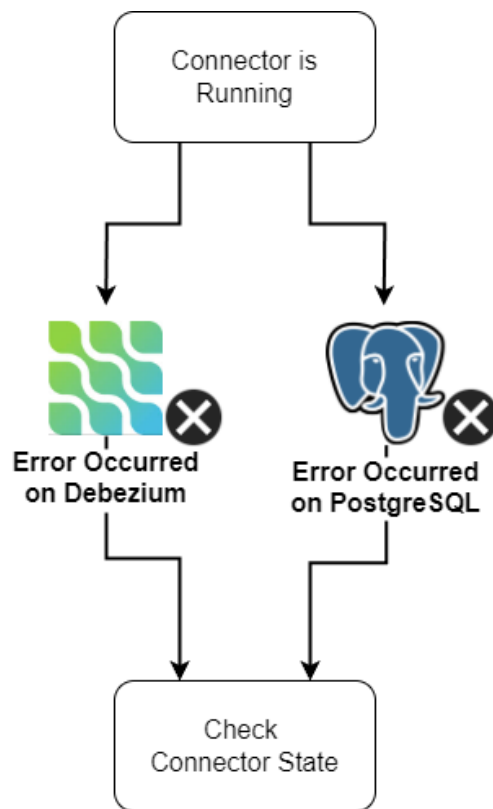
```
SELECT * from synchdb_state_view;
```

Change data incoming...



Provision Connector State and Error We Want it Straightforward!

- Server disconnected.
- Schema incompatibility.
- False configuration.
- Snapshot mode error.
- Permission issues.
- Resource issues
- ...etc.



- Duplicate key violation.
- Table does not exist.
- Permission denied on relation.
- Column does not exist.
- invalid data or data type.
- ...etc.

```
SELECT connector, conninfo_name, state, err from synchdb_state_view LIMIT 5;
```

connector	conninfo_name	state	err
mysql	mysqlconn	syncing	no error
mysql	mysqlconn2	stopped	table 1539111: duplicate key value violates unique constraint "orders_pkey"
mysql	mysqlconn3	paused	no error
sqlserver	sqlsvrconn	syncing	no error
sqlserver	sqlsvrconn2	stopped	Connector configuration is not valid. Unable to connect: Access denied for user 'mysqluser'@'172.20.0.1' (using password: YES)

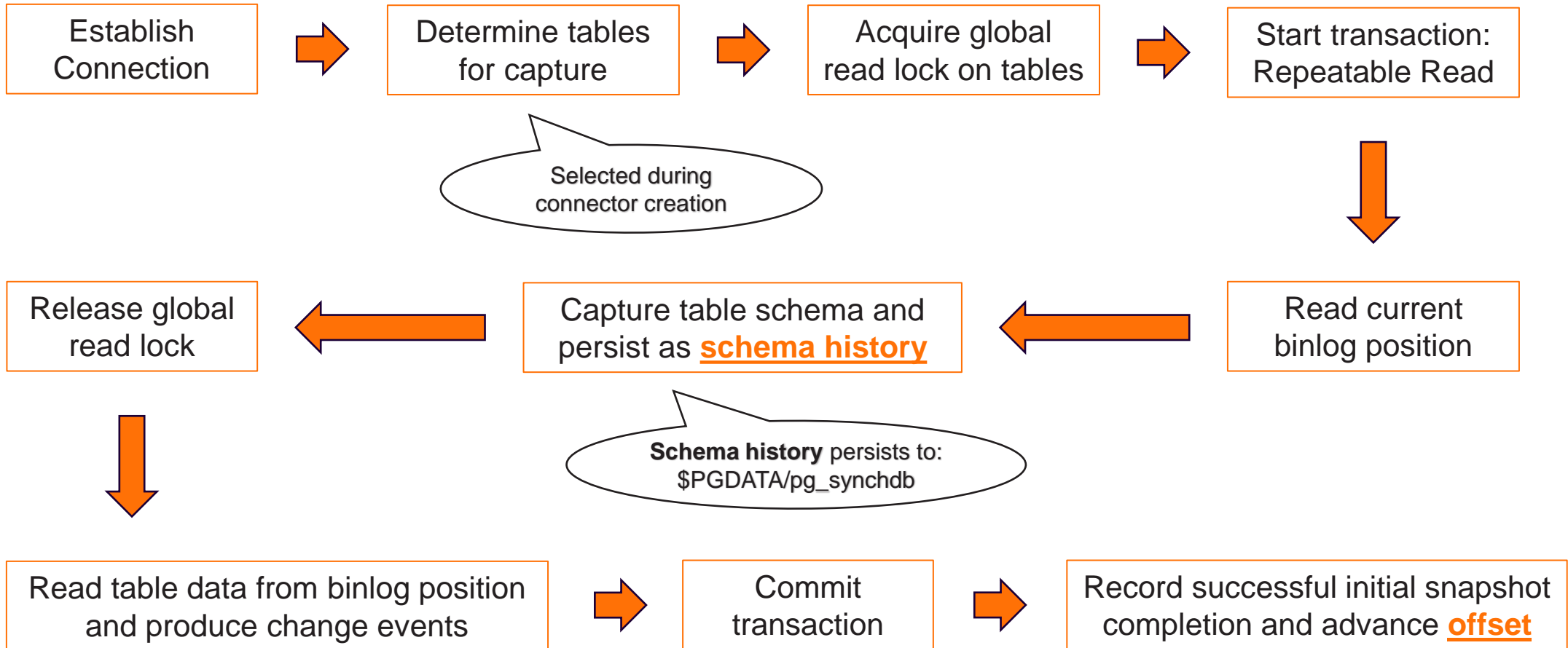
(5 rows)

Resolve the reported Issue



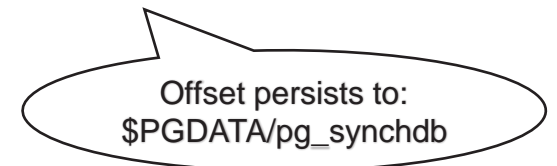


SynchDB Workflow – Initial Snapshot



Initial Snapshot is performed when connector is started for the very first time

CDC follows immediately after that.



• Add Tables to Replicate in Run-time – Redo Initial Snapshot?

- After the initial snapshot is completed, SynchDB knows the structure of selected tables and can decode subsequent changes applied to it (CDC).
- But.... What if you changed your mind and need to add more tables to replicate after initial snapshot has been completed?
- Well... we need to instruct Debezium to redo initial snapshot on the new table lists. Similar to PostgreSQL's REFRESH PUBLICATION.

Alter connector info to select more tables

```
UPDATE synchdb_conninfo SET  
  data = jsonb_set(data, '{table}',  
  '"inventory.orders,inventory.products,inventory.customers"')  
WHERE name = 'mysqlconn';
```

Restart the Connector in different snapshot mode

```
SELECT synchdb_restart_connector('mysqlconn', 'always');
```

Check Connector State



Change data incoming...





Supported Snapshot Modes

Mode	Description
always	<ul style="list-style-type: none">• Always perform initial snapshot (table structures + their data).• Then CDC begins.
initial (default)	<ul style="list-style-type: none">• Perform initial snapshot (table structures + their data) if not already done.• Then CDC begins.
initial_only	<ul style="list-style-type: none">• Perform initial snapshot (table structures + their data) if not already done.• Then it will shutdown and will not begin CDC.
no_data	<ul style="list-style-type: none">• Performs initial snapshot (table structures only, no data) if not already done.• Then CDC begins.
never	<ul style="list-style-type: none">• No initial snapshot done• Just Begins CDC
recovery	<ul style="list-style-type: none">• Performs snapshot to restore lost or corrupted database schema• No CDC.
when_needed	<ul style="list-style-type: none">• Performs initial snapshot only if it cannot detect topic offsets, or previously recorded offset not available on the server.



SynchDB Workflow – Change Events

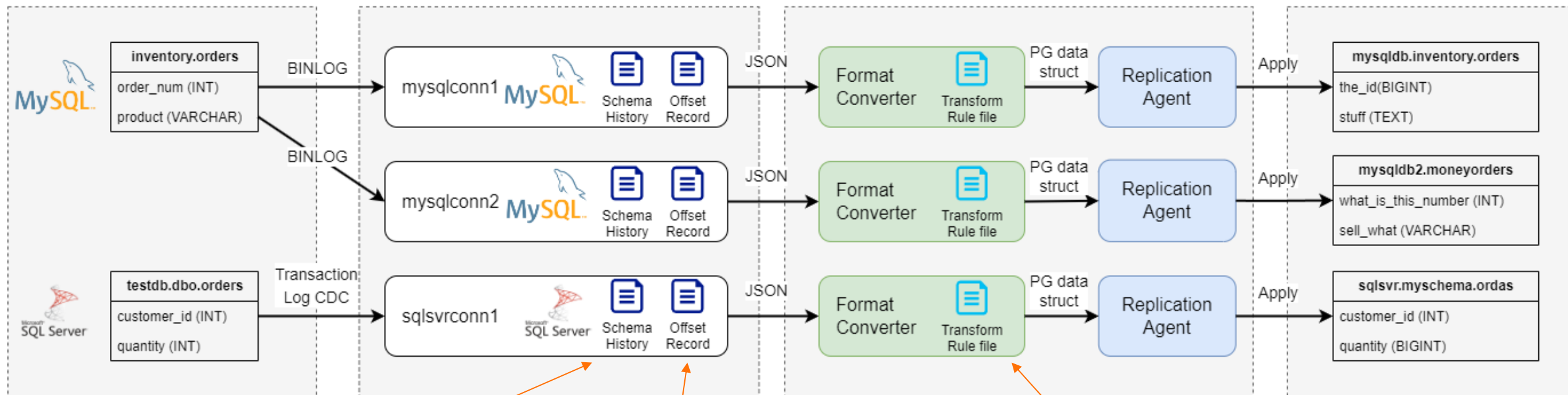


Heterogeneous Databases

Debezium Runner

SynchDB

PostgreSQL Core



Schema history

contains table structure that Debezium requires to produce a data change event

Offset Record

contains the offset (similar to LSN in PG) that Debezium should start to read change records

Transform Rule File

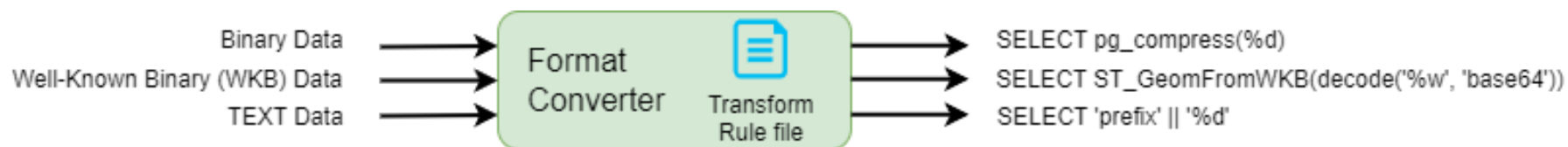
Tells SynchDB format converter how to transform incoming table, column, schema names and even the **data** before applying to PostgreSQL



SynchDB Workflow – Data Transform



- The transform rule file not only transforms between data types, table names and column names.
- It can do data expression transform as well.
- Useful when the replicated data needs further processing before applying to PostgreSQL.
- Not possible, or not flexible enough if we were built as a middleware outside of PostgreSQL.



Received column data is fed through user-defined expression before applying to PostgreSQL:

- %d replaced with data value
- %w replaced with WKB value (geometry type)
- %s replaced with SRID value (geometry type)





SynchDB Workflow – Rule File



- Written in JSON and is to be put under \$PGDATA.
- Consists of 3 JSON arrays:
- Uses Fully Qualified Name (FQN) to indicate an object to avoid ambiguity.

```
"transform_datatype_rules":  
  [  
    {  
      "translate_from": "GEOMETRY",  
      "translate_from_autoinc": false,  
      "translate_to": "TEXT",  
      "translate_to_size": -1  
    },  
    {  
      "translate_from": "inventory.geom.g.GEOMETRY",  
      "translate_from_autoinc": false,  
      "translate_to": "GEOMETRY",  
      "translate_to_size": 0  
    }  
  ]
```

```
"transform_objectname_rules":  
  [  
    {  
      "object_type": "table",  
      "source_object": "inventory.orders",  
      "destination_object": "schema1.orders"  
    },  
    {  
      "object_type": "column",  
      "source_object": "testDB.dbo.customers.first_name",  
      "destination_object": "the_awesome_first_name"  
    }  
  ]
```

```
"transform_expression_rules":  
  [  
    {  
      "transform_from": "inventory.orders.quantity",  
      "transform_expression": "case when %d < 500 then 0 else %d end"  
    },  
    {  
      "transform_from": "inventory.geom.g",  
      "transform_expression": "ST_SetSRID(ST_GeomFromWKB(decode('%w', 'base64')),%s)"  
    }  
  ]
```



SynchDB Workflow – Fetch Changes

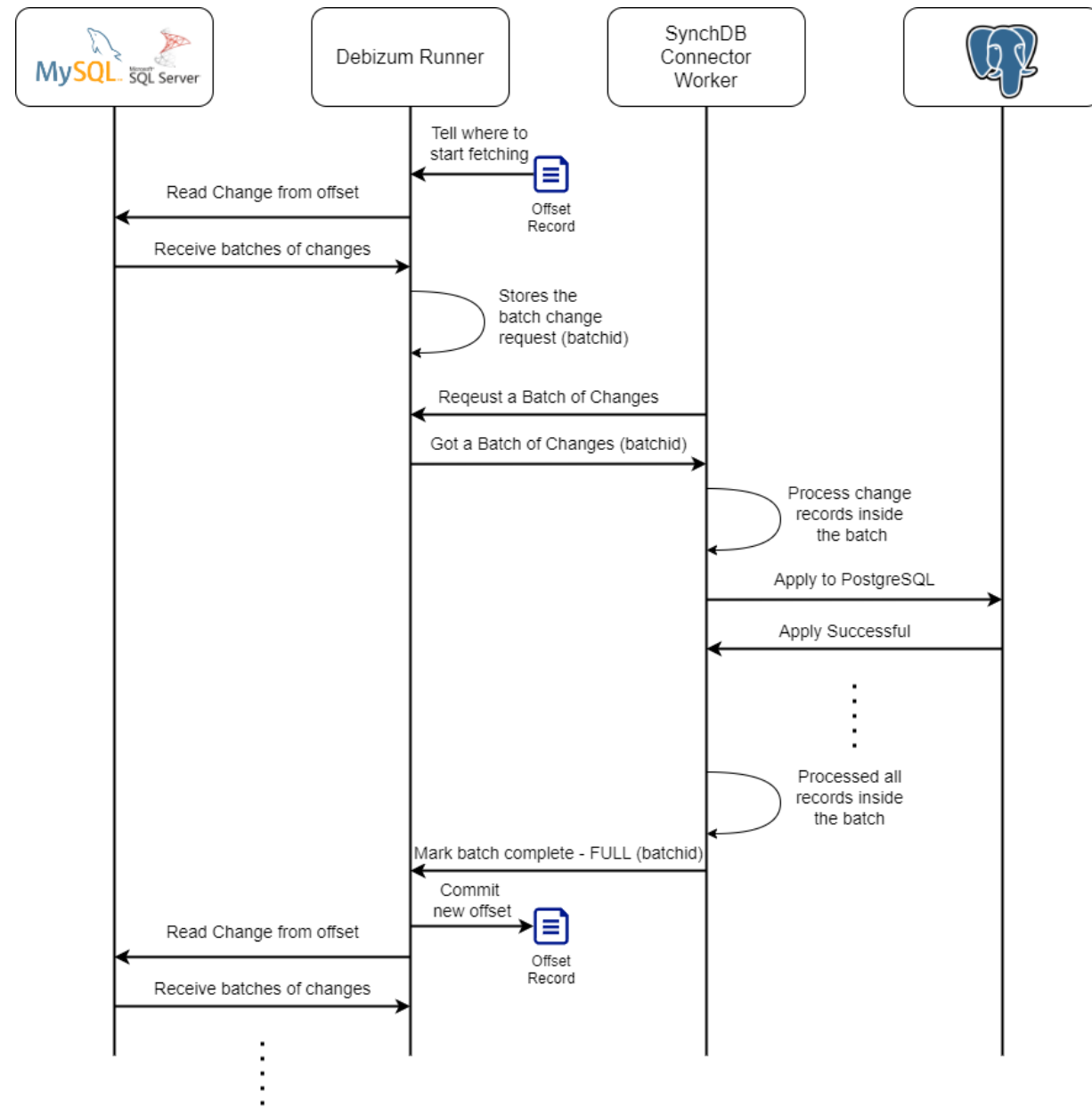


- The Debezium runner engine periodically fetches changes from remote heterogeneous databases.
- The SynchDB extension also periodically fetches changes from the Debezium Runner engine.
- So... we have 2 layers of change propagation before it can be applied to PostgreSQL.
- How to properly coordinate change event propagation becomes important, as to:
 - Prevent duplicate records.
 - Prevent missing records.
 - Re-fetch when error occurs.
- This is similar to PostgreSQL's replication slot concept to coordinate between publisher and subscriber where the current LSN is at, and which LSN to resume during restarts.



SynchDB Workflow – Fetch Changes With Batch Management

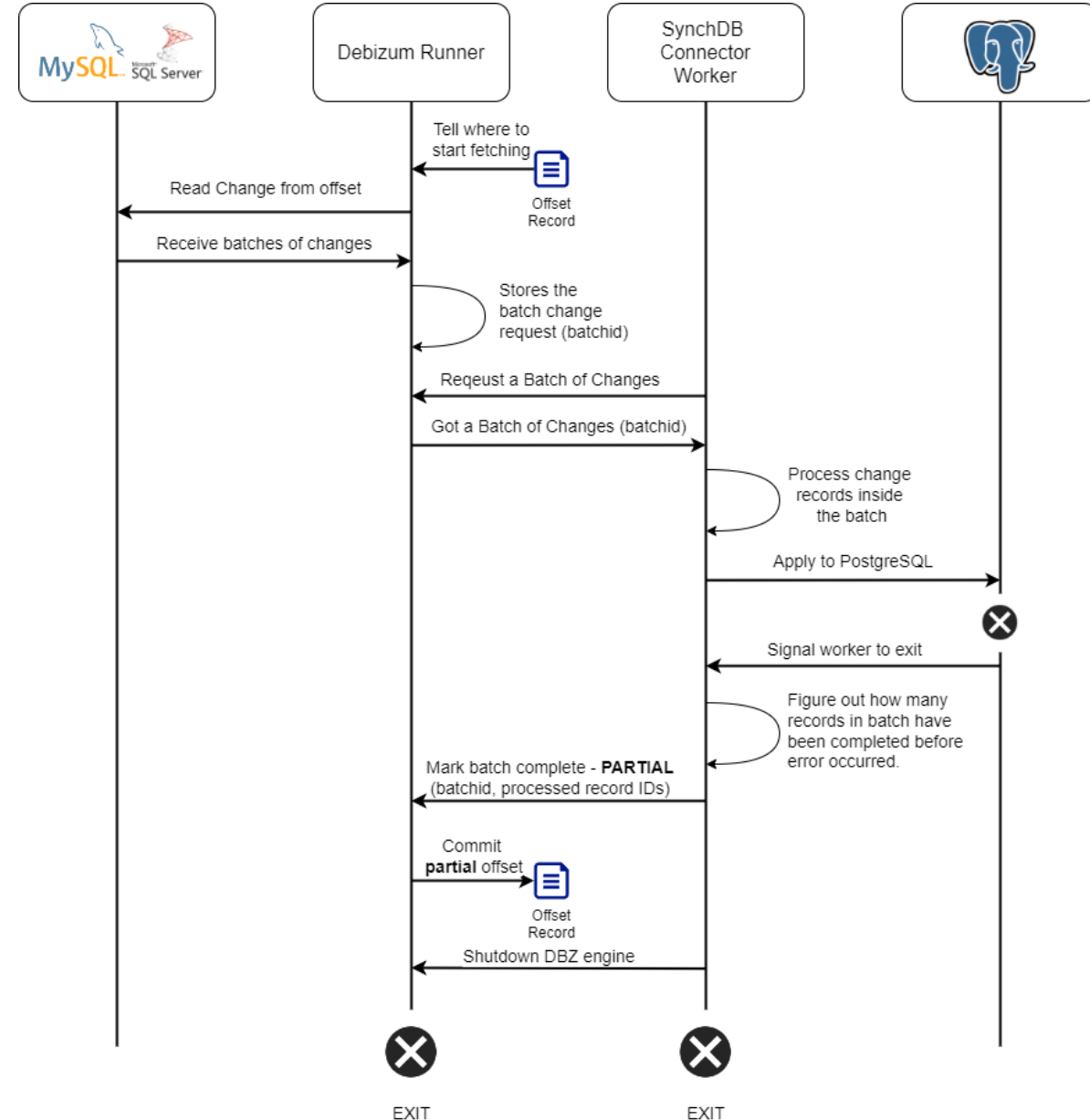
- A change record inside a batch is considered completed when it is successfully applied to PostgreSQL.
- A completed change record inside a batch allows Debezium to “advance the offset”.



SynchDB Workflow – Error During Batch Processing



- In case of error, SynchDB connector worker is still required to mark a batch as **partially** completed.
- This causes Debezium runner to partially advance the offset up until the point of error.
- Upon restart, the first change record inside the first batch will be the one that caused the error from the previous run.



SynchDB v1.0Beta



- SynchDB v1.0 Beta is released on our github page here: <https://github.com/Hornetlabs/synchdb>.
- Documentation page here: <https://docs.synchdb.com/>.

Heterogeneous

Databases:

- MySQL
- SQLServer

DDLs:

- CREATE TABLE
- DROP TABLE
- ALTER TABLE ADD COLUMN
- ALTER TABLE DROP COLUMN
- ALTER TABLE ALTER COLUMN

DMLs:

- INSERT
- UPDATE
- DELETE

Transforms:

- Data types
- Table names
- Column names
- Data values

Apply Modes:

- SPI
- HeapAM API

Core Features

- Automatic connector launcher at PostgreSQL startup.
- Global connector state and error provisioning.
- Selective table replication.
- Batch handling.
- Offset management.
- JSON rule fine definition per connector.
- Utility functions to manage connectors.
- 30 concurrent connector workers.
- Connector restart in different snapshot modes.



SynchDB Future Improvements

- More optimized initial snapshot task (more efficient table data “copying” to PostgreSQL).
- Support more authentication methods such as TLS or via third-party auth apps (RADIUS, GSSAPI...etc).
- More heterogeneous databases support. (Oracle, db2...etc)
- Obtain performance figures and tuning.
- More error handling configurations. (skip error records or exit on error...etc).
- Include transaction boundaries in change events to optimize apply. (multi-inserts...etc).
- Selective operations: (ignore update, insert or delete...etc).
- More DDLs: CREATE INDEX, TRUNCATE...etc.
- Strict query ordering with transaction metadata.
- ... many more waiting to be discovered!

Join SynchDB development on github to make it better together!

<https://github.com/Hornetlabs/synchdb>.



SynchDB



Thank You!