EUROPEAN MiCROSOFT FABRiC Community Conference

VIENNA 15-18 SEPTEMBER 2025

JOIN THE CONVERSATION

#FABCONEUROPE25

Mastering Open Mirroring in Microsoft Fabric

Cristian Urbina Guerra

Senior Data Engineer
Bsgroup Data Analytics AG
Switzerland

Cristian Urbina Guerra

- Chilean-Spanish living in Switzerland
- Photography, my second passion
- Metal Rock fanatic
- Senior Data Engineer / Data Platform Architect / Data enthusiastic

ABSGROUP
DATA ANALYTICS

https://github.com/metxito/session-open-mirroring

Agenda

- What is Mirroring?
- Open Mirroring
- Configuration and Designs
- Examples
- How to capture changes

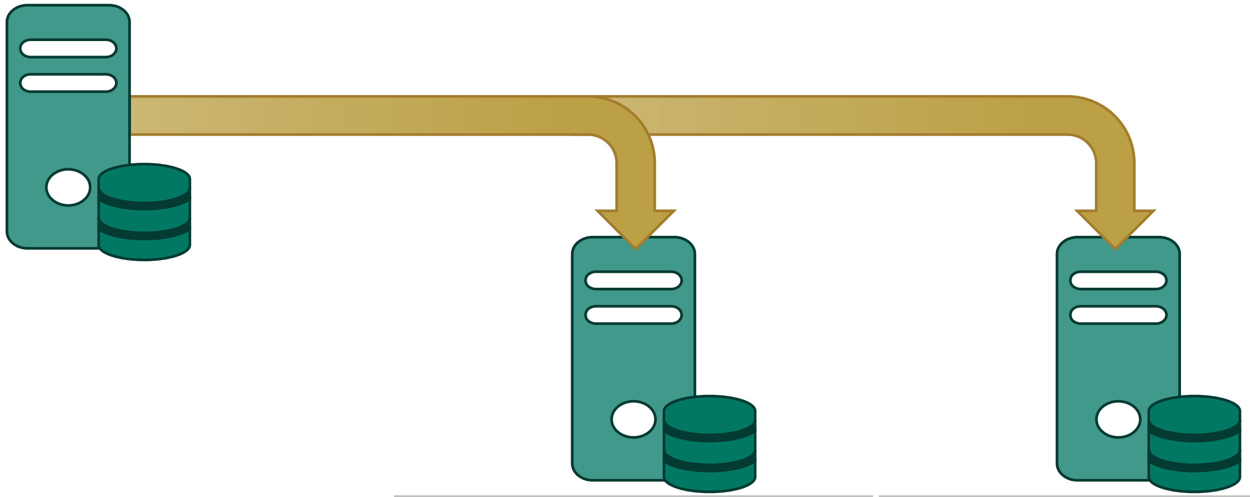
What is mirroring?

What is Database Mirroring

Database mirroring is a technique to **create and maintain a ~~real~~ near real-time copy of a database** on a different server or storage system.

The idea is that every transaction (insert, update, delete) performed on the **primary database** is **replicated** to a **mirrored database**.

```
UPDATE[dbo].[account]
SET [CurrentBalance] = (
    SELECT SUM([TransactionAmmount])
    FROM [dbo].[transactions]
    WHERE [AccountId] = 2354
          AND Status = 'Valid'
          AND [AuthorizationCode] = 'DER12'
)
WHERE [AccountId] = 2354
```

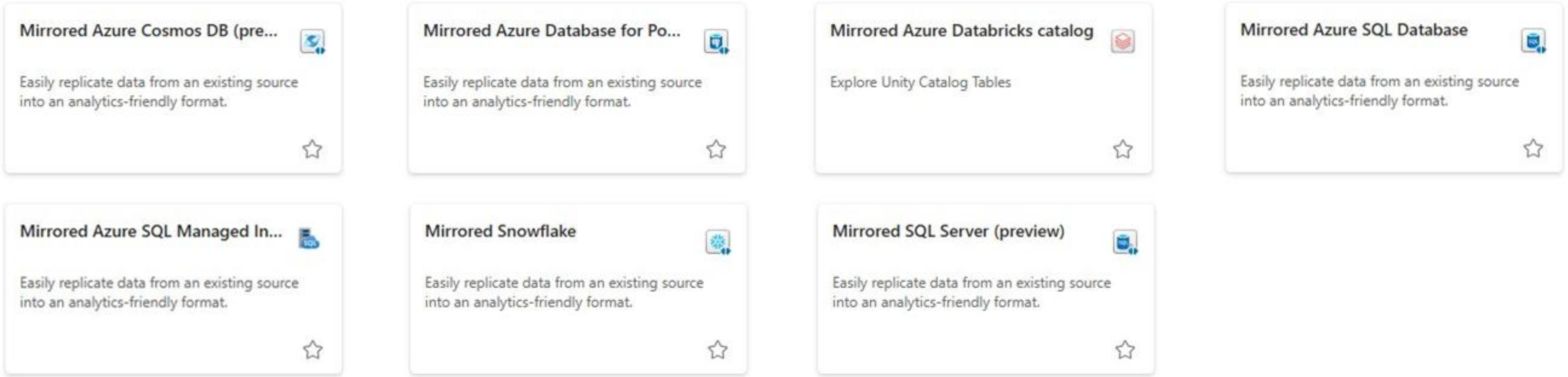


```
UPDATE[dbo].[account]
SET [CurrentBalance] = 2536583.25
WHERE [AccountId] = 2354
```

```
UPDATE[dbo].[account]
SET [CurrentBalance] = 2536583.25
WHERE [AccountId] = 2354
```

Why / When mirroring a database?

- High availability / Disaster recovery
- Data redundancy / Read scalability
- Minimize the workload on the ETL/ELT



Mirrored Azure Cosmos DB (pre...

Easily replicate data from an existing source into an analytics-friendly format.

Mirrored Azure Database for Po...

Easily replicate data from an existing source into an analytics-friendly format.

Mirrored Azure Databricks catalog

Explore Unity Catalog Tables

Mirrored Azure SQL Database

Easily replicate data from an existing source into an analytics-friendly format.

Mirrored Azure SQL Managed In...

Easily replicate data from an existing source into an analytics-friendly format.

Mirrored Snowflake

Easily replicate data from an existing source into an analytics-friendly format.

Mirrored SQL Server (preview)

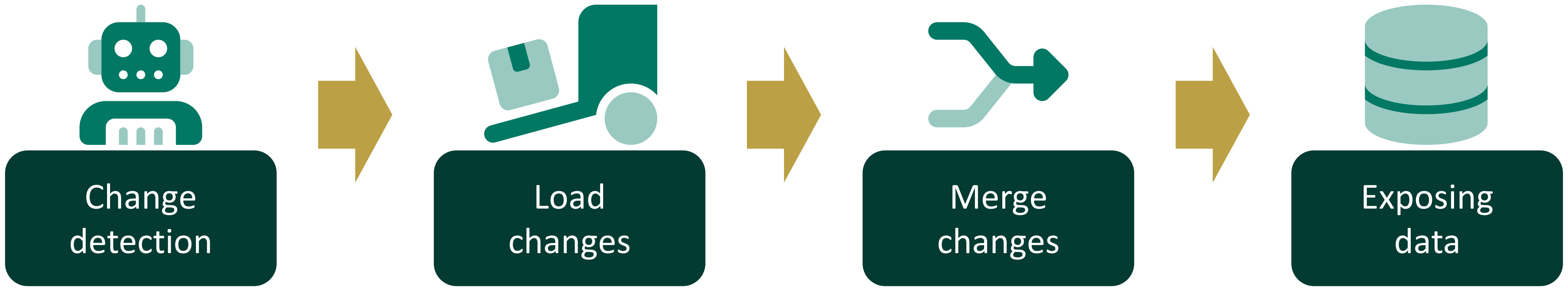
Easily replicate data from an existing source into an analytics-friendly format.

Microsoft Fabric Open Mirroring

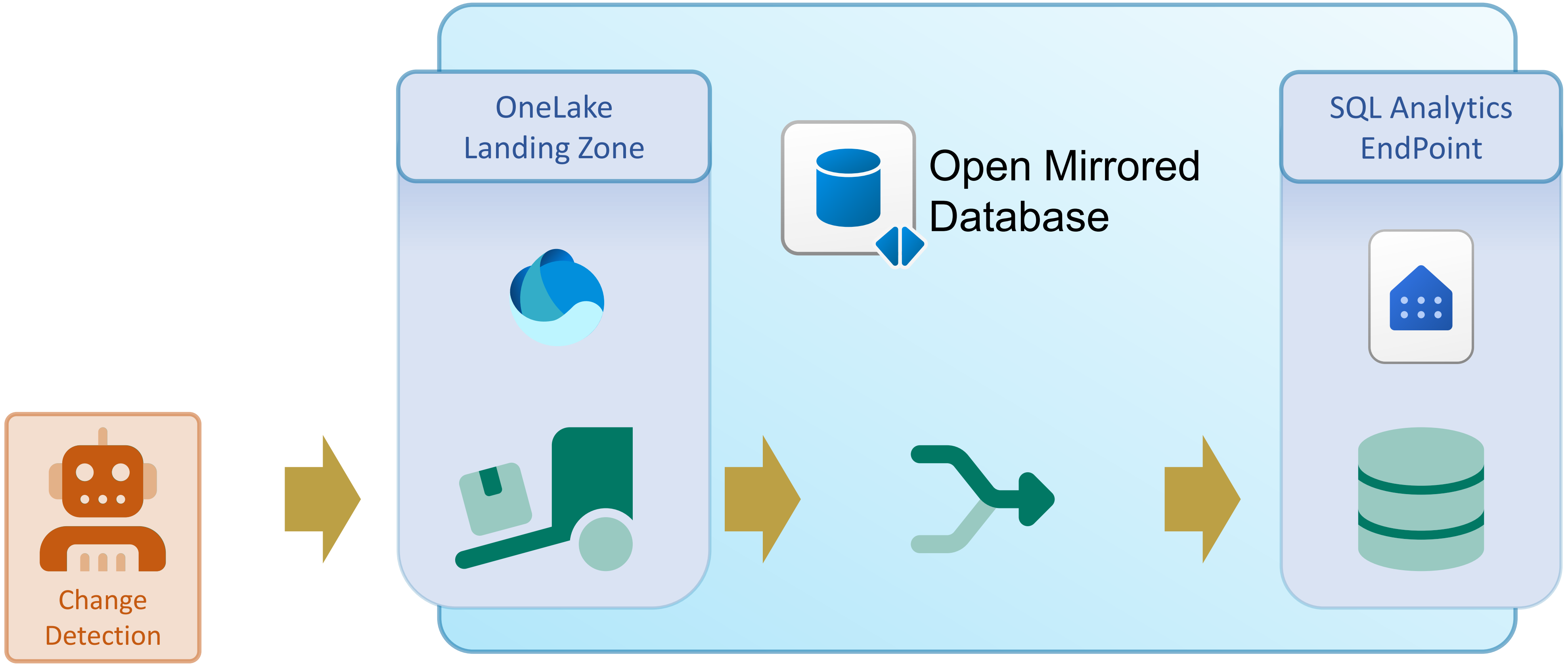
Open Mirroring

It is a database used as target replication in Microsoft Fabric.

You can upload transactional files to continuously replicate your data directly into Fabric's OneLake.



Open Mirroring



Data

00000000000000000001.parquet

	zone	id	name	email	__rowMarker__
INSERT	ch	1	John Catier	juan@hotmail.com	0
INSERT	de	1	Claudio Müller	culler.c@msn.com	0
INSERT	de	2	Pedro Tono	pedro.tono@mycompany.com	0

00000000000000000002.parquet

	zone	id	name	email	__rowMarker__
INSERT	ch	2	Raul Massana	r.massana@yonunca.ch	0
INSERT	de	3	Roberto Carlos	roberto.carlos@elsobrigo.de	0
UPDATE	de	2	Pedro Antonio	pedro.antonio@mycompany.com	1

00000000000000000002.parquet

	zone	id	name	email	__rowMarker__
INSERT	es	1	Cristian Urbina	cristianurbina@viverominta.es	0
DELETE	de	3			2

__rowMarker__ 0 : INSERT 1 : UPDATE 2 : DELETE

Files/LandingZone/Sales.schema/Contact

- _metadata.json
- 00000000000000000001.parquet
- 00000000000000000002.parquet
- 00000000000000000003.parquet
- 00000000000000000005.parquet
- 00000000000000000006.parquet
- 00000000000000000007.parquet
- . . . .
- 99999999999999999999.parquet

Ninety-nine quintillion, nine hundred ninety-nine quadrillion, nine hundred ninety-nine trillion, nine hundred ninety-nine billion, nine hundred ninety-nine million, nine hundred ninety-nine thousand, nine hundred ninety-nine

Metadata

_metadata.json   for .csv, .tsv files

```
{
  "KeyColumns": ["zone", "id"],
  "SchemaDefinition": {
    "Columns": [
       {"Name":"zone", "DataType":"String" },
       {"Name":"id", "DataType":"Int32" },
       {"Name":"name", "DataType":"String", "IsNullable":true },
       {"Name":"email", "DataType":"String", "IsNullable":true }
    ]
  },
  "FileFormat": "DelimitedText",
  "FileExtension": "csv",
  "FileFormatTypeProperties": {
    "FirstRowAsHeader": true,
    "RowSeparator": "\r\n",
    "ColumnSeparator": ";",
    "QuoteCharacter": "\"",
    "EscapeCharacter": "\\",
    "NullValue": "N/A",
    "Encoding": "UTF-8"
  }
}
```

_metadata.json   for .parquet files

```
{
 "KeyColumns": ["zone", "id"]
}
```

More information

Tables Management

- How to create a new table?

Create a folder in LandingZone and upload files:

- `_metadata.json`
- `00000000000000000001.parquet`

```
/Files/LandingZone/TransactionType              >  dbo.TransactionType
/Files/LandingZone/sales.schema/Customers       >  sales.Customers
/Files/LandingZone/erp.schema/Products          >  erp.Products
```

- How to drop a table?

Delete the subfolder in the LandingZone

- How to rename a table?

Delete the subfolder in the LandingZone and create it again

DEMO

CREATE MIRRORED DATABASE

Costs

- All computation is included in the capacity. (Save money by avoiding running pipelines)
- Storage is included, but ....

SKU	Free up to (TB)
F2	2
F4	4
F8	8
F16	16
F32	32
F64/P1	64
F128/P2	128
F256/P3	256
F512/P4	512
F1024/P5	1,024
F2048	2,048

Open Mirroring

- **Simplifies integration**: Avoids complex ETL, brings existing data into OneLake.
- **Continuous replication**: Keeps mirrored data always up-to-date.
- **Open, extensible**: Any app can write changes into Fabric.
- **Ready for analysis**: Automatically handles inserts, updates, deletes.

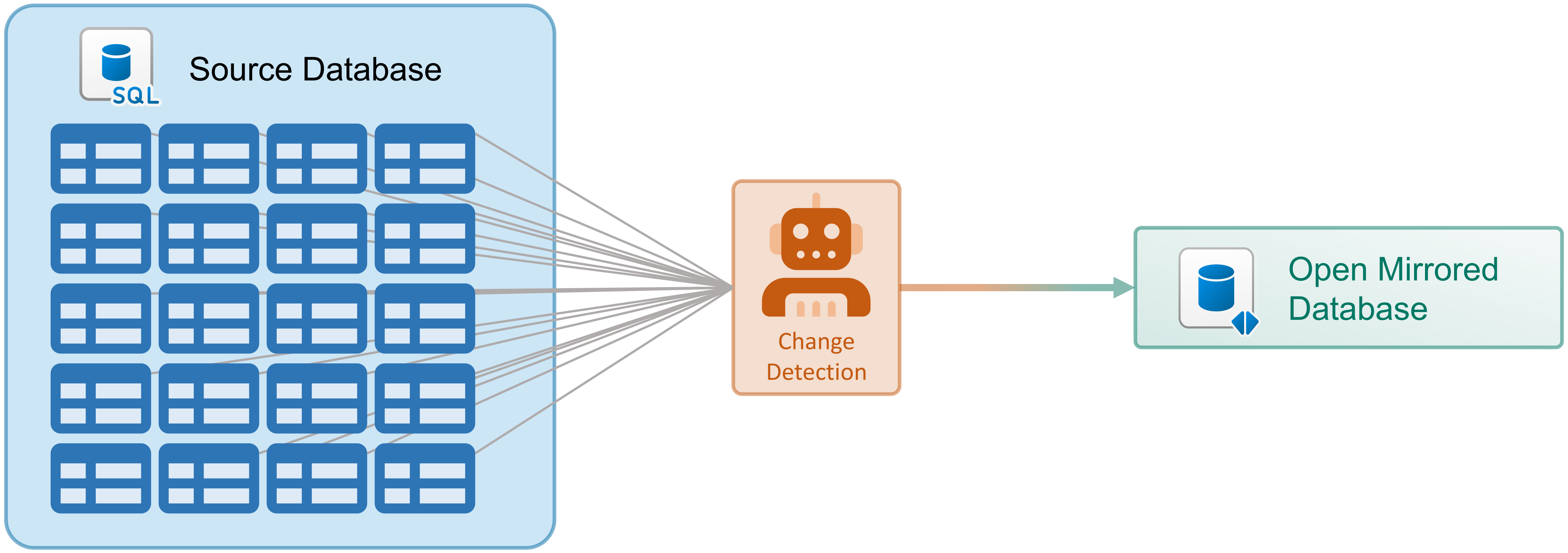
What if

- … is the capacity off?
- ... has you uploaded a file with a wrong name?
- ... has you uploaded in no sequential order?
- ... the new file has different structure?

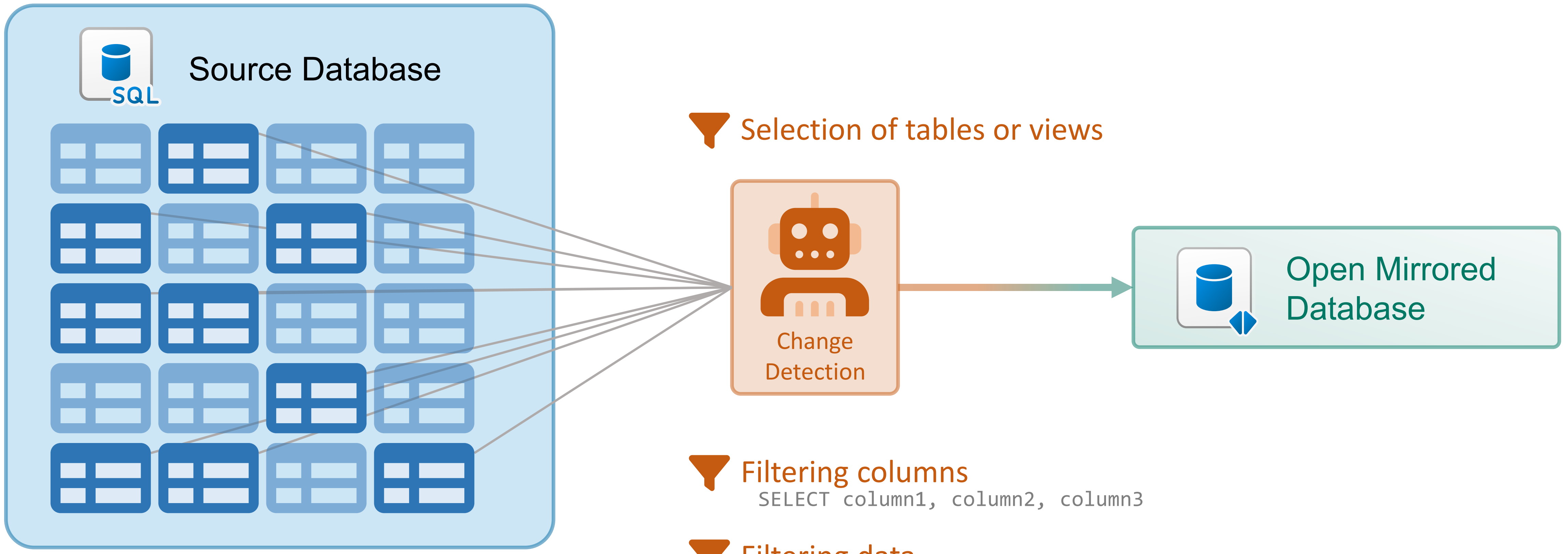
- You can not apply modifications on LandingZone
- This file will no be considered
- Fabric will 'old' this file
- Same rules as Delta Tables

Configuration and Designs

Designs : Full Mirroring



Designs : Partial Mirroring

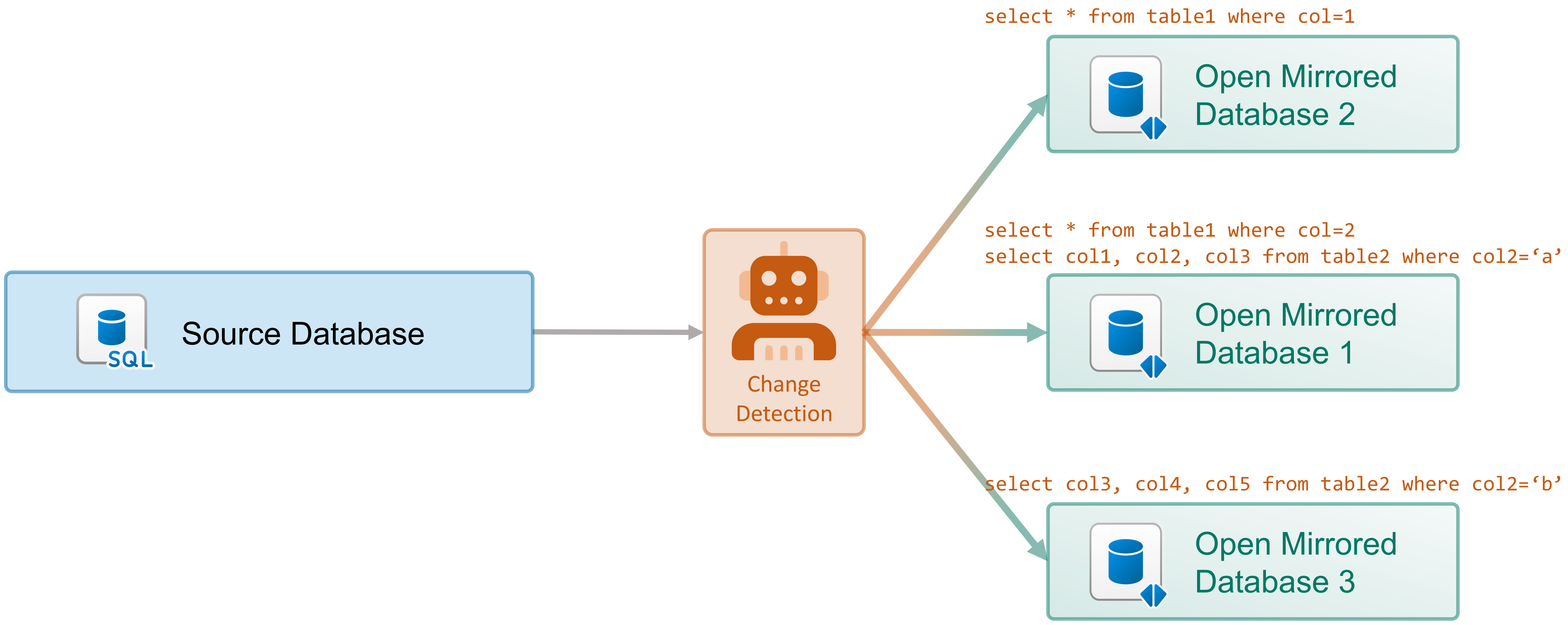


Selection of tables or views

Filtering columns
```
SELECT column1, column2, column3
```

Filtering data
```
WHERE type = 3 AND flag = 0
```

Designs : Multi Targets

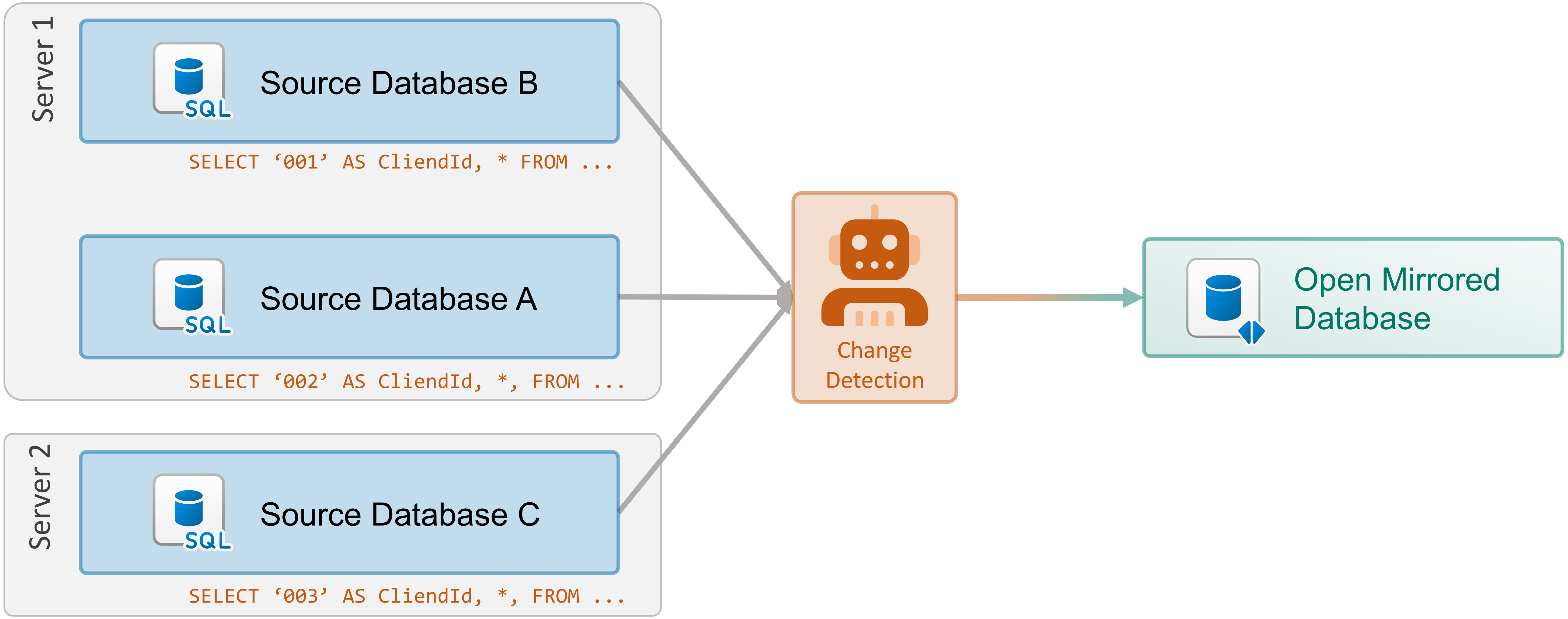


```
select * from table1 where col=1
```

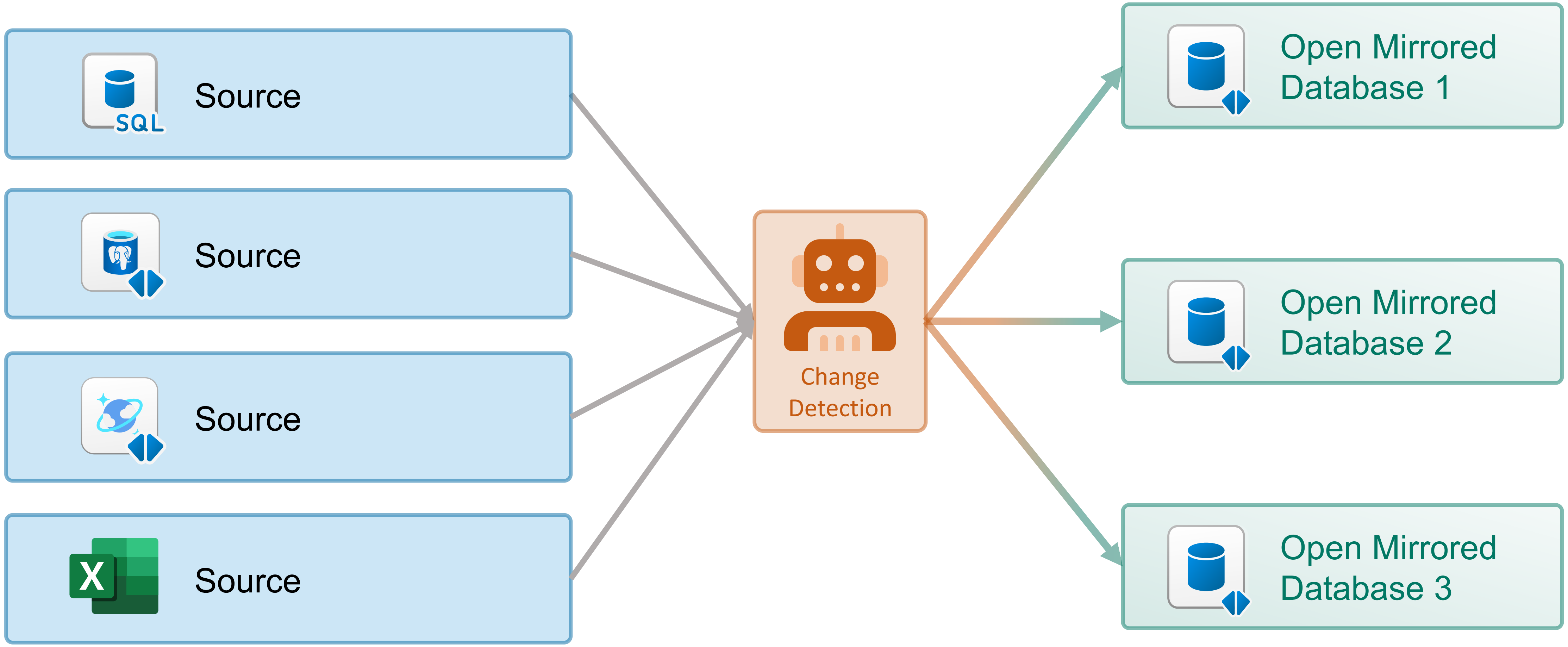
```
select * from table1 where col=2
select col1, col2, col3 from table2 where col2='a'
```

```
select col3, col4, col5 from table2 where col2='b'
```

Designs : Multi Sources



Designs : Multi Sources – Multi Targets



Examples

Code Examples : Python

Unix Environment Preparation

```
> sudo apt-get install -y unixodbc
> sudo apt-get install -y unixodbc-dev
> sudo ACCEPT_EULA=Y apt-get install -y msodbcsql18
> sudo ACCEPT_EULA=Y apt-get install -y mssql-tools18
> pip install pyarrow fastparquet sqlalchemy pyodbc
```

Python : Save a .parquet file

```python
import pandas as pd
from sqlalchemy import create_engine

# local path where the .parquet file will be stored
parquet_full_path = "c:\tmp\00000000000000000001.parquet"

# connection string to a Microsoft SQL server
connstring = f"mssql+pyodbc://{user}:{password}@{server},{port}/{catalog}?driver=ODBC+Driver+18+for+SQL+Server&TrustServerCertificate=yes"
sql_source_engine = create_engine(connstring)
with sql_source_engine.connect() as conn:

    # query to execute
    data_query = f"SELECT * FROM [dbo].[my_table]"

    # read the data
    data_df = pd.read_sql(data_query, conn)

    # write into .parque file
    data_df.to_parquet(parquet_full_path, engine="pyarrow", index=False)
```

Code Examples : Python

Unix Environment Preparation

```
> pip install azure-identity azure-storage-file-datalake azure-storage-blob
```

Python : Save a .parquet file

```python
from azure.identity import ClientSecretCredential
from azure.storage.filedatalake import DataLakeServiceClient

# local path where the .parquet file is stored
parquet_full_path = "c:\tmp\00000000000000000001.parquet"
# OneLake location where the folder for the table is
landing_table = f"{mirrored_database_guid}/Files/LandingZone/sch.schema/my_table“

# connect to OneLake
client_credential = ClientSecretCredential(
    tenant_id = my_tenant_id,
    client_id = my_client_id,
    client_secret = my_client_secret
)
onelake_service_client = DataLakeServiceClient(account_url="https://onelake.dfs.fabric.microsoft.com", credential=client_credential)
onelake_filesystem = onelake_service_client.get_file_system_client(file_system=my_workspace_guid)
onelake_table_directory = onelake_filesystem.get_directory_client(landing_table)

# Upload file into OneLake
with open(parque_full_path, "rb") as data:
    file_client = onelake_table_directory.get_file_client("00000000000000000001.parquet")
    file_client.upload_data(data, overwrite=True)
```

How to capture changes

How to capture changes

Option		Capture Inserts	Capture Updates	Capture Deletes	Overload Source	Extra Storage
Datetime	- Needs trigger - Time change (Summer) - Needs table modification	🟢 Yes	🟢 Yes	🔴 No	🔴 Heavy	🟢 Medium
Rowversion (timestamp)	- Detect a change (Insert or update) - Independ of time - Needs table modification	🟡 Yes*	🟡 Yes*	🔴 No	🟢 Light	🟢 Medium
Change Tracking	- Apply at Low level	🟢 Yes	🟢 Yes	🟢 Yes	🟢 Light	🟡 Heavy*
Change Date Capture	- Apply at Low level - Track all changes and values	🟢 Yes	🟢 Yes	🟢 Yes	🟢 Light	🔴 Heavy

SQL Server Change Tracking

```
USE [myDB];
GO
CREATE TABLE [dbo].[Employees] (
    [EmpID]  INT          PRIMARY KEY,
    [Name]   NVARCHAR(50),
    [Salary] INT
);
GO


ALTER DATABASE [myDB]
SET CHANGE_TRACKING = ON
(CHANGE_RETENTION = 2 DAYS, AUTO_CLEANUP = ON);
GO


ALTER TABLE [dbo].[Employees]
ENABLE CHANGE_TRACKING
WITH (TRACK_COLUMNS_UPDATED = ON);


INSERT INTO [dbo].[Employees] ([EmpID], [Name], [Salary]) VALUES (1, 'Alice', 5000);
INSERT INTO [dbo].[Employees] ([EmpID], [Name], [Salary]) VALUES (2, 'Pablo', 4800);


UPDATE [dbo].[Employees] SET [Salary] = 6000 WHERE [EmpID] = 1;
UPDATE [dbo].[Employees] SET [Salary] = 7000 WHERE [EmpID] = 1;
```

SQL Server Change Tracking

```
SELECT *
FROM CHANGETABLE(CHANGES [dbo].[Employees], 0) AS ct;
```

SYS_CHANGE_VERSION	SYS_CHANGE_CREATION_VERSION	SYS_CHANGE_OPERATION	SYS_CHANGE_COLUMNS	SYS_CHANGE_CONTEXT	EmpID
4	1	I	NULL	NULL	1
2	2	I	NULL	NULL	2

SQL Server Change Date Capture

```
USE [myDB];
GO
CREATE TABLE [dbo].[Employees] (
    [EmpID]  INT          PRIMARY KEY,
    [Name]   NVARCHAR(50),
    [Salary] INT
);
GO

EXEC [sys].[sp_cdc_enable_db]

SELECT [name], [is_cdc_enabled] FROM [sys].[databases]
GO

EXEC [sys].[sp_cdc_enable_table]
     @source_schema = N'dbo',
     @source_name = N'Employees',
     @role_name = NULL,
     @supports_net_changes = 1

SELECT s.[name] AS [schema], t.[name] AS [table], t.[is_tracked_by_cdc]
FROM [sys].[tables] AS t
JOIN [sys].[schemas] AS s ON t.[schema_id] = s.[schema_id]
ORDER BY 1, 2

INSERT INTO [dbo].[Employees] ([EmpID], [Name], [Salary]) VALUES (1, 'Alice', 5000)
INSERT INTO [dbo].[Employees] ([EmpID], [Name], [Salary]) VALUES (2, 'Pablo', 4800)

UPDATE [dbo].[Employees] SET [Salary] = 6000 WHERE [EmpID] = 1
UPDATE [dbo].[Employees] SET [Salary] = 7000 WHERE [EmpID] = 1
UPDATE [dbo].[Employees] SET [Salary] = 5800, [Name]='Pablo R.' WHERE [EmpID] = 2

DELETE FROM [dbo].[Employees] WHERE [EmpID] = 1
```

@role_name:
Which SQL Role is allow to query the changes.
Null means open

@supports_net_changes
- 0: create only ‘ALL’ function
- 1: create ‘ALL’ and ‘NET’ function

SQL Server Change Date Capture

```sql
SELECT [EmpID], [Name], [Salary], [__$operation]
FROM [cdc].[fn_cdc_get_all_changes_dbo_Employees](
    [sys].[fn_cdc_get_min_lsn]('dbo_Employees'),
    [sys].[fn_cdc_get_max_lsn](),
    'all'
);
```

__$start_lsn	EmpID	Name	Salary	__$operation	
0x0000003400001ED0001F	1	Alice	5000	2	Insert
0x00000034000020E80003	2	Pablo	4800	2	Insert
0x00000034000020F80003	1	Alice	6000	4	Update
0x00000034000021100003	1	Alice	7000	4	Update
0x00000034000021400003	2	Pablo R.	5800	4	Update
0x00000034000021480005	1	Alice	7000	1	Delete

```sql
SELECT [EmpID], [Name], [Salary], [__$operation]
FROM [cdc].[fn_cdc_get_net_changes_dbo_Employees](
    [sys].[fn_cdc_get_min_lsn]('dbo_Employees'),
    [sys].[fn_cdc_get_max_lsn](),
    'all'
);
```

__$start_lsn	EmpID	Name	Salary	__$operation	
0x00000034000021400003	2	Pablo R.	5800	2	Insert

```sql
SELECT [EmpID], [Name], [Salary], [__$operation]
FROM [cdc].[fn_cdc_get_all_changes_dbo_Employees](
    0x00000034000020E80003,
    [sys].[fn_cdc_get_max_lsn](),
    'all'
);
```

__$start_lsn	EmpID	Name	Salary	__$operation	
0x00000034000021100003	1	Alice	7000	4	Update
0x00000034000021400003	2	Pablo R.	5800	4	Update
0x00000034000021480005	1	Alice	7000	1	Delete

```sql
SELECT [EmpID], [Name], [Salary], [__$operation]
FROM [cdc].[fn_cdc_get_net_changes_dbo_Employees](
    0x00000034000020E80003,
    [sys].[fn_cdc_get_max_lsn](),
    'all'
);
```

__$start_lsn	EmpID	Name	Salary	__$operation	
0x00000034000021400003	2	Pablo R.	5800	4	Update
0x00000034000021480005	1	Alice	7000	1	Delete

lsn: Log Sequence Number
'all' → every change row, full detail.

'all with mask' → same as 'all' + column update bitmap.

'all with merge' → simplified view, one row per update.

Change Date Capture vs Change Tracking

Feature	Change Tracking (CT)	Change Data Capture (CDC)
Introduced	SQL Server 2008	SQL Server 2008
SQL Agent required?	❌ No	✅ Yes
CLR required?	❌ No	✅ Yes
Tracks before values?	❌ No	✅ Yes
Tracks column changes?	✅ Yes (bitmap)	✅ Yes (full values)

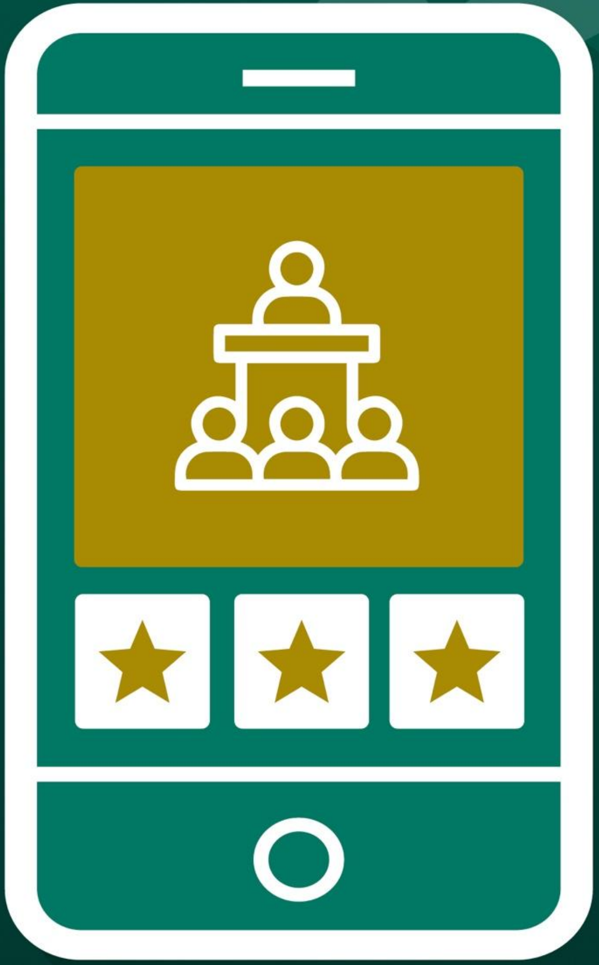
Change Date Capture : History

SQL Server Version	CDC at Database Level	CDC at Table Level	Notes
2008 (Enterprise & Dev)	✅ Yes	✅ Yes	First introduction of CDC. Not in Standard/Express.
2008 R2 (Enterprise & Dev)	✅ Yes	✅ Yes	Same limitations as 2008.
2012 – 2016 (Enterprise & Dev)	✅ Yes	✅ Yes	Still **Enterprise-only** until 2016 SP1.
2016 SP1+ (Standard & above)	✅ Yes	✅ Yes	Big change: CDC available in **Standard Edition** (not just Enterprise).
2017 (Standard & above)	✅ Yes	✅ Yes	Fully supported.
2019 (Standard & above)	✅ Yes	✅ Yes	Fully supported.
2022 (Standard & above)	✅ Yes	✅ Yes	Still supported. Works with **Azure SQL Managed Instance** too.
Azure SQL Database (PaaS)	❌ No	❌ No	CDC **not supported** in single Azure SQL DBs.
Azure SQL Managed Instance	✅ Yes	✅ Yes	Supported since 2020.

Other database tools

Database	Database-Level Enablement	Table-Level Enablement	Equivalent Feature(s)	Notes
Oracle	✅ Yes (DB options need enabling)	✅ Yes (per table with supplemental logging)	Oracle GoldenGate, Flashback Data Archive	GoldenGate = replication/CDC tool; Flashback = time travel queries. Requires extra licensing.
PostgreSQL	❌ No (not a DB toggle)	✅ Yes (per table via logical decoding or triggers)	Logical Replication, WAL Decoding, Triggers	No built-in CDC like SQL Server, but WAL (Write Ahead Log) can be decoded (e.g., Debezium, pgoutput).
MySQL / MariaDB	❌ No	❌ No (at table level)	Binary Log (binlog), row-based replication	CDC is done via binlog readers (Debezium, Maxwell, etc.), not native per-table toggles.
DB2 (IBM)	✅ Yes (at DB/subsystem level)	✅ Yes (per table registration)	IBM InfoSphere Data Replication (IIDR) / DB2 CDC	Strong native CDC, often paired with replication.
SAP HANA	✅ Yes (logging enabled at DB)	✅ Yes (per table configuration)	Table-Based Data Capture, SAP SLT Replication	Supports log-based CDC.
Snowflake	❌ No	✅ Yes (per table streams)	Snowflake Streams	You create a **stream** on a table to capture inserts/updates/deletes.
BigQuery	❌ No	✅ Yes (per table change streams)	Change Streams	Works at table level only, must be enabled.
MongoDB (NoSQL)	✅ Yes (at cluster level via oplog/Change Streams)	✅ Yes (per collection)	Change Streams	More like CDC but document-based.

Please rate this session on the app



cvent

GET IT ON Google Play

Download on the App Store