

**Apache NiFi Data Pipeline with PostgreSQL Integration & Airbyte**

**Table of Contents**

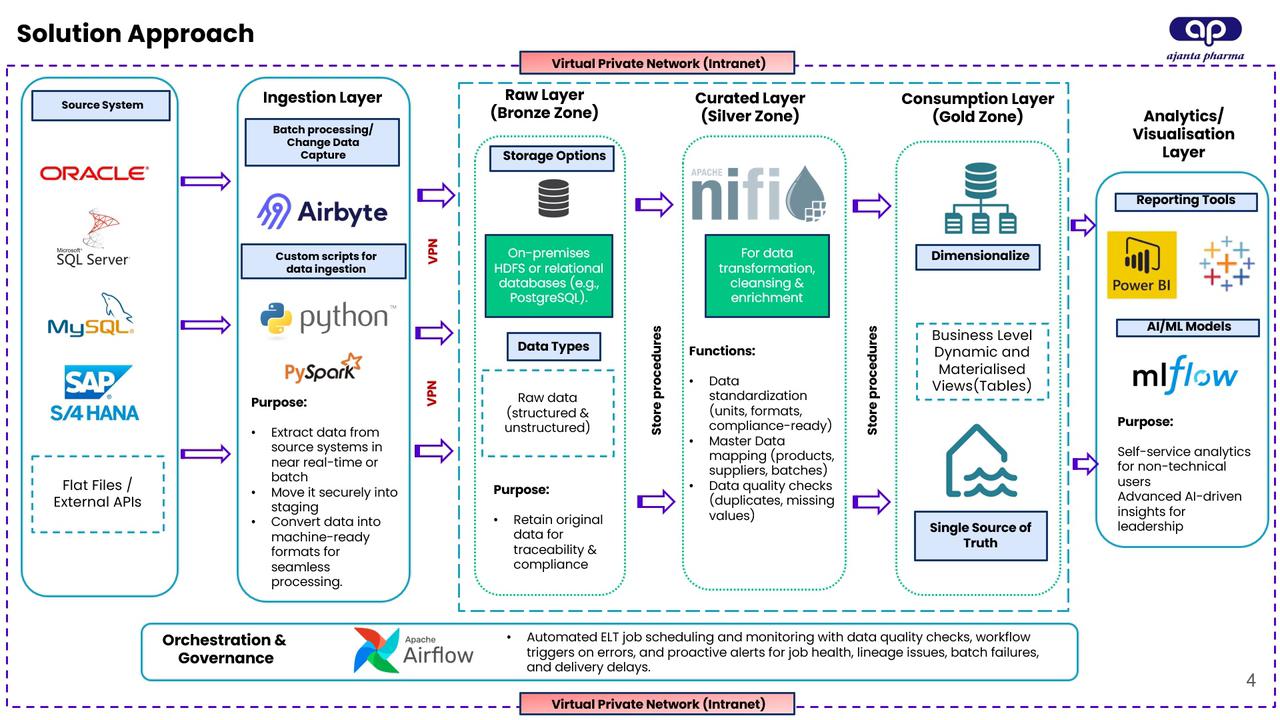
1. [Project Overview](#project-overview)
2. [Prerequisites and Setup](#prerequisites-and-setup)
3. [PostgreSQL Database Configuration](#postgresql-database-configuration)
4. [Apache NiFi Setup and Configuration](#apache-nifi-setup-and-configuration)
5. [Controller Services Configuration](#controller-services-configuration)
6. [NiFi Processor Flow Implementation](#nifi-processor-flow-implementation)
7. [Incremental Processing with Airbyte Integration](#incremental-processing-with-airbyte-integration)
8. [Testing and Validation](#testing-and-validation)
9. [Troubleshooting Guide](#troubleshooting-guide)
10. [Performance Optimization](#performance-optimization)

**Project Overview**

This PoC project was implemented in the span of 2 Weeks (25-07-2025 to 08-08-2025) by **Yatin Kumar Reddy** **K** & **Sriman Satwik Reddy Chinnam**, both being B.Tech students from Mahindra University, Hyderabad. This is an automated data pipeline using Apache NiFi to perform incremental batch processing of data synchronized by Airbyte. The pipeline fetches data from a PostgreSQL source table (scms\_data\_2) and performs UPSERT operations into a target table (scms\_data\_1) using Airbyte's metadata columns for efficient incremental processing.

**Key Features**

* **Incremental Processing**: Uses \_airbyte\_extracted\_at timestamp for efficient data fetching.
* **UPSERT Operations**: Automatic INSERT for new records and UPDATE for existing records.
* **State Management**: NiFi automatically tracks processing state for resumable operations.



**Prerequisites and Setup**

**System Requirements**

* Apache NiFi 2.4.0 or later
* PostgreSQL 15+ with JDBC driver
* Java 8 or later
* Minimum 8GB RAM for NiFi
* Database user with SELECT/INSERT/UPDATE permissions

**Required Software**

1. **Apache NiFi**: Download from <https://nifi.apache.org/download.html>
2. **PostgreSQL JDBC Driver**: Download postgresql-42.7.3.jar from <https://jdbc.postgresql.org/download.html>

**Installation Steps**

1. **Install Apache NiFi**:

# Extract NiFi archive  
tar -xzf nifi-1.28.0-bin.tar.gz  
cd nifi-1.28.0  
  
# Start NiFi service  
./bin/nifi.sh start

1. **Install PostgreSQL JDBC Driver**:

# Copy JDBC driver to NiFi lib directory  
cp postgresql-42.7.3.jar /path/to/nifi/lib/  
  
# Restart NiFi to load the driver  
./bin/nifi.sh restart

1. **Verify Installation**:
   * Access NiFi UI at http://localhost:8080/nifi
   * Check that PostgreSQL driver is loaded in Controller Services

**PostgreSQL Database Configuration**

**Database Schema Setup**

**Source Table (Airbyte Output)**

-- Create source table with Airbyte metadata columns  
CREATE TABLE public.scms\_data\_2 (  
 id INTEGER PRIMARY KEY,  
 name VARCHAR(255),  
 email VARCHAR(255),  
 phone VARCHAR(20),  
 address TEXT,  
 created\_date DATE,  
 -- Airbyte metadata columns  
 \_airbyte\_raw\_id UUID NOT NULL,  
 \_airbyte\_extracted\_at TIMESTAMP WITH TIME ZONE NOT NULL,  
 \_airbyte\_generation\_id BIGINT,  
 \_airbyte\_meta JSONB  
);  
  
-- Create index for efficient incremental processing  
CREATE INDEX idx\_scms\_data\_2\_extracted\_at ON public.scms\_data\_2(\_airbyte\_extracted\_at);  
CREATE INDEX idx\_scms\_data\_2\_id ON public.scms\_data\_2(id);

**Target Table (NiFi Output)**

-- Create target table for processed data  
CREATE TABLE public.scms\_data\_1 (  
 id INTEGER PRIMARY KEY,  
 name VARCHAR(255),  
 email VARCHAR(255),  
 phone VARCHAR(20),  
 address TEXT,  
 created\_date DATE,  
 -- Processing metadata  
 updated\_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT\_TIMESTAMP  
);  
  
-- Create index for efficient UPSERT operations  
CREATE INDEX idx\_scms\_data\_1\_id ON public.scms\_data\_1(id);

**Database User and Permissions**

-- Create dedicated user for NiFi  
CREATE USER nifi\_user WITH PASSWORD 'secure\_password';  
  
-- Grant necessary permissions  
GRANT USAGE ON SCHEMA public TO nifi\_user;  
GRANT SELECT ON public.scms\_data\_2 TO nifi\_user;  
GRANT INSERT, UPDATE, DELETE ON public.scms\_data\_1 TO nifi\_user;  
GRANT USAGE, SELECT ON ALL SEQUENCES IN SCHEMA public TO nifi\_user;

**Verify Database Setup**

-- Check table structure  
\d public.scms\_data\_2  
\d public.scms\_data\_1  
  
-- Verify sample data with Airbyte columns  
SELECT id, name, \_airbyte\_extracted\_at   
FROM public.scms\_data\_2   
ORDER BY \_airbyte\_extracted\_at DESC   
LIMIT 5;

**Apache NiFi Setup and Configuration**

**Accessing NiFi Interface**

1. Open web browser and navigate to http://localhost:8080/nifi
2. Default credentials (if security is enabled):
   * Username: admin
   * Password: Check nifi-app.log for generated password, if else the username also

**Basic NiFi Configuration**

1. **Verify NiFi Status**:
   * Check that NiFi is running: Green status indicator
   * Verify available processors in the processor toolbar
   * Confirm JDBC driver is loaded
2. **Configure NiFi Properties** (if needed):

# In nifi.properties file  
nifi.web.http.port=8080  
nifi.web.http.host=localhost  
nifi.content.repository.implementation=org.apache.nifi.controller.repository.FileSystemRepository

**Controller Services Configuration**

**Step 1: Create DBCPConnectionPool Service**

1. **Access Controller Services**:
   * Right-click on canvas → "Configure"
   * Navigate to "Controller Services" tab
   * Click "+" to add new service
2. **Add DBCPConnectionPool**:
   * Service Type: DBCPConnectionPool
   * Service Name: PostgreSQL-DBCPConnectionPool
3. **Configure DBCPConnectionPool Properties**:

Database Connection URL: jdbc:postgresql://localhost:5432/your\_database?currentSchema=public  
Database Driver Class Name: org.postgresql.Driver  
Database User: nifi\_user  
Database Password: secure\_password  
Max Wait Time: 500 millis  
Max Total Connections: 8  
Min Idle: 2  
Max Idle: 8  
Validation Query: SELECT 1  
Min Evictable Idle Time: 30 mins  
Time Between Eviction Runs: 5 mins

1. **Enable the Service**:
   * Click "Apply" to save configuration
   * Click "Enable" (lightning bolt icon)
   * Verify status shows "Enabled"

**Step 2: Create Record Reader and Writer Services**

**AvroRecordSetWriter Configuration**

1. **Add Service**:
   * Service Type: AvroRecordSetWriter
   * Service Name: AvroRecordSetWriter
2. **Configure Properties**:

Schema Write Strategy: Embedded Avro Schema  
Schema Access Strategy: Infer Schema  
Schema Name: ${schema.name}  
Schema Version: ${schema.version}  
Schema Branch: ${schema.branch}  
Cache Size: 1000  
Compression Format: NONE

1. **Enable the Service**

**AvroRecordReader Configuration**

1. **Add Service**:
   * Service Type: AvroRecordReader
   * Service Name: AvroRecordReader
2. **Configure Properties**:

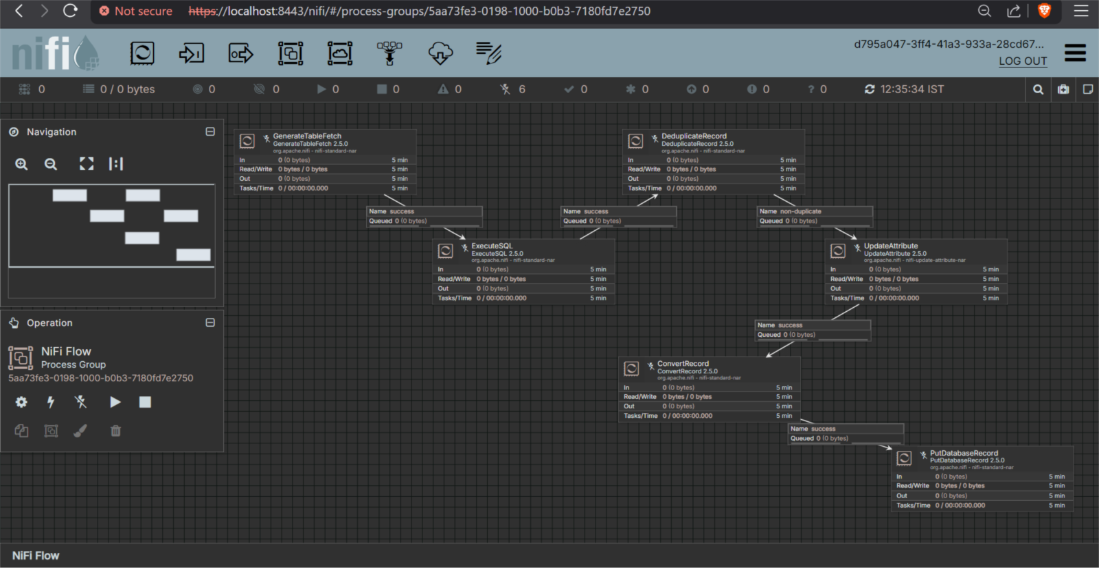
Schema Access Strategy: Embedded Avro Schema  
Schema Name: ${schema.name}  
Schema Version: ${schema.version}  
Schema Branch: ${schema.branch}  
Cache Size: 1000

1. **Enable the Service**

**Step 3: Verify Controller Services**

* All three services should show "Enabled" status
* No error bulletins should be present
* Test database connection using "Verify" button on DBCPConnectionPool

**NiFi Processor Flow Implementation**

**Processor Flow Overview**

QueryDatabaseTableRecord → PutDatabaseRecord

**Step 1: Add QueryDatabaseTableRecord Processor**

1. **Add Processor to Canvas**:
   * Drag processor icon from toolbar
   * Search for "QueryDatabaseTableRecord"
   * Add to canvas
2. **Configure QueryDatabaseTableRecord Properties**:

Database Connection Pooling Service: PostgreSQL-DBCPConnectionPool  
Database Type: Generic  
Table Name: public.scms\_data\_2  
Columns to Return: [Leave empty for all columns]  
Maximum-Value Columns: \_airbyte\_extracted\_at  
Where Clause: [Leave empty - auto-generated]  
Max Wait Time: 0 seconds  
Partition Size: 1000  
Record Writer: AvroRecordSetWriter  
Normalize Table/Column Names: false  
Use Avro Logical Types: false  
Default Decimal Precision: 10  
Default Decimal Scale: 0  
Output Empty FlowFile on Zero Results: false

1. **Configure Scheduling**:

Scheduling Strategy: Timer driven  
Run Schedule: 0 0 17 \* \* ? (Daily at 5 PM)  
Concurrent Tasks: 1  
Run Duration: 0 sec

1. **Configure Settings**:

Automatically Terminate Relationships: [None]  
Penalty Duration: 30 sec  
Yield Duration: 1 sec  
Bulletin Level: WARN

**Step 2: Add PutDatabaseRecord Processor**

1. **Add Processor to Canvas**:
   * Search for "PutDatabaseRecord"
   * Add to canvas
2. **Configure PutDatabaseRecord Properties**:

Record Reader: AvroRecordReader  
Database Connection Pooling Service: PostgreSQL-DBCPConnectionPool  
Statement Type: UPSERT  
Table Name: public.scms\_data\_1  
Update Keys: id  
Field Containing SQL: [Leave empty]  
Allow Multiple Statements: false  
Quote Column Identifiers: false  
Quote Table Identifiers: false  
SQL Parameter Attribute Prefix: sql  
Table Schema Cache Size: 100  
Max Batch Size: 100

1. **Configure Settings**:

Automatically Terminate Relationships: success, retry, failure  
Penalty Duration: 30 sec  
Yield Duration: 1 sec  
Bulletin Level: WARN

**Step 3: Connect Processors**

1. **Create Connection**:
   * Click and drag from QueryDatabaseTableRecord to PutDatabaseRecord
   * Select "success" relationship
   * Configure connection settings:

Name: success  
FlowFile Expiration: 0 sec  
Back Pressure Object Threshold: 10000  
Back Pressure Data Size Threshold: 1 GB  
Load Balance Strategy: DO\_NOT\_LOAD\_BALANCE

1. **Configure Auto-Terminate Relationships**:
   * For QueryDatabaseTableRecord: Check "failure"
   * For PutDatabaseRecord: Check "success", "retry", "failure"

**Step 4: Validate Configuration**

1. **Check Processor Status**:
   * Both processors should show valid configuration (no red warning icons)
   * Controller services should be enabled and accessible
2. **Verify Connections**:
   * Connection should show no back pressure
   * Relationship should be properly configured

**Incremental Processing with Airbyte Integration**

**Understanding Airbyte Metadata Columns**

**Key Columns for Integration**

-- Airbyte automatically adds these columns to synchronized tables  
\_airbyte\_raw\_id -- UUID: Unique identifier for each record  
\_airbyte\_extracted\_at -- TIMESTAMP WITH TIME ZONE: When record was extracted  
\_airbyte\_generation\_id -- BIGINT: Version/generation number  
\_airbyte\_meta -- JSONB: Additional metadata from Airbyte

**Sample Data Structure**

-- Example data in scms\_data\_2 after Airbyte sync  
SELECT   
 id,  
 name,  
 \_airbyte\_raw\_id,  
 \_airbyte\_extracted\_at,  
 \_airbyte\_generation\_id  
FROM public.scms\_data\_2  
ORDER BY \_airbyte\_extracted\_at DESC  
LIMIT 3;  
  
-- Expected output:  
-- id | name | \_airbyte\_raw\_id | \_airbyte\_extracted\_at | generation\_id  
-- 1 | John | 123e4567-e89b-12d3-a456-426614174000 | 2025-08-31 17:30:00+05:30 | 0  
-- 2 | Jane | 123e4567-e89b-12d3-a456-426614174001 | 2025-08-31 17:30:00+05:30 | 0  
-- 3 | Bob | 123e4567-e89b-12d3-a456-426614174002 | 2025-08-31 17:30:00+05:30 | 0

**How Incremental Processing Works**

**First Run (Initial Load)**

1. **NiFi State**: Empty (no previous processing)
2. **Generated Query**:

SELECT \* FROM public.scms\_data\_2   
WHERE \_airbyte\_extracted\_at IS NOT NULL   
ORDER BY \_airbyte\_extracted\_at ASC

1. **Processing**: All records are processed
2. **State Storage**: NiFi stores maximum \_airbyte\_extracted\_at value

**Subsequent Runs (Incremental)**

1. **NiFi State**: Contains last processed timestamp (e.g., '2025-08-31 17:30:00+05:30')
2. **Generated Query**:

SELECT \* FROM public.scms\_data\_2   
WHERE \_airbyte\_extracted\_at > '2025-08-31 17:30:00+05:30'  
ORDER BY \_airbyte\_extracted\_at ASC

1. **Processing**: Only new/changed records since last run
2. **State Update**: NiFi updates stored timestamp with new maximum

**UPSERT Operation Logic**

**PostgreSQL UPSERT Implementation**

-- Generated by PutDatabaseRecord processor  
INSERT INTO public.scms\_data\_1 (id, name, email, phone, address, created\_date)  
VALUES (?, ?, ?, ?, ?, ?)  
ON CONFLICT (id) DO UPDATE SET  
 name = EXCLUDED.name,  
 email = EXCLUDED.email,  
 phone = EXCLUDED.phone,  
 address = EXCLUDED.address,  
 created\_date = EXCLUDED.created\_date,  
 updated\_at = CURRENT\_TIMESTAMP;

**Operation Types**

* **INSERT**: For records with new IDs not existing in target table
* **UPDATE**: For records with existing IDs in target table
* **Conflict Resolution**: Based on primary key (id) column

**Coordination with Airbyte Schedule**

**Recommended Schedule Coordination**

Airbyte Sync Schedule: Every 6 hours (00:00, 06:00, 12:00, 18:00)  
NiFi Processing Schedule: Daily at 19:00 (after 18:00 Airbyte sync)  
  
Timeline Example:  
00:00 - Airbyte syncs batch #1  
06:00 - Airbyte syncs batch #2   
12:00 - Airbyte syncs batch #3  
18:00 - Airbyte syncs batch #4  
19:00 - NiFi processes all changes since last run (batches #1-4)

**Testing and Validation**

**Step 1: Initial Setup Validation**

**Verify Database Connectivity**

-- Test connection and data presence  
SELECT   
 COUNT(\*) as total\_records,  
 MIN(\_airbyte\_extracted\_at) as earliest\_record,  
 MAX(\_airbyte\_extracted\_at) as latest\_record  
FROM public.scms\_data\_2;  
  
-- Check target table initial state  
SELECT COUNT(\*) as target\_count FROM public.scms\_data\_1;

**Validate NiFi Configuration**

1. **Controller Services Check**:
   * All services should be "Enabled"
   * DBCPConnectionPool should validate successfully
   * No error bulletins present
2. **Processor Configuration Check**:
   * QueryDatabaseTableRecord shows valid configuration
   * PutDatabaseRecord shows valid configuration
   * Connection between processors is properly configured

**Step 2: Test Initial Data Load**

**Clear Processor State (if needed)**

1. **Right-click QueryDatabaseTableRecord** → "View State"
2. **Click "Clear State"** if you want to reprocess all data
3. **Confirm** the state clearing

**Run Initial Load Test**

1. **Start QueryDatabaseTableRecord processor**
2. **Monitor Data Provenance**:
   * Right-click processor → "View Data Provenance"
   * Check for FlowFiles being created
   * Verify record count matches expected
3. **Start PutDatabaseRecord processor**
4. **Monitor Processing**:
   * Check connection queue for FlowFiles
   * Monitor processor bulletins for any errors
   * Verify success relationship receives FlowFiles

**Validate Results**

-- Check data transfer success  
SELECT   
 (SELECT COUNT(\*) FROM public.scms\_data\_2) as source\_count,  
 (SELECT COUNT(\*) FROM public.scms\_data\_1) as target\_count;  
  
-- Verify data consistency (sample check)  
SELECT s1.id, s1.name, s2.name, s1.updated\_at  
FROM public.scms\_data\_1 s1  
JOIN public.scms\_data\_2 s2 ON s1.id = s2.id  
ORDER BY s1.id  
LIMIT 10;

**Step 3: Test Incremental Processing**

**Add New Test Data**

-- Insert new records with recent timestamp  
INSERT INTO public.scms\_data\_2 (  
 id, name, email, phone, address, created\_date,  
 \_airbyte\_raw\_id, \_airbyte\_extracted\_at, \_airbyte\_generation\_id, \_airbyte\_meta  
) VALUES   
(9999, 'Test User 1', 'test1@example.com', '555-0001', '123 Test St', CURRENT\_DATE,  
 gen\_random\_uuid(), CURRENT\_TIMESTAMP, 0, '{"sync\_id": "test\_batch"}'),  
(9998, 'Test User 2', 'test2@example.com', '555-0002', '456 Test Ave', CURRENT\_DATE,  
 gen\_random\_uuid(), CURRENT\_TIMESTAMP, 0, '{"sync\_id": "test\_batch"}');

**Run Incremental Test**

1. **Start processors** (if not already running)
2. **Wait for processing to complete**
3. **Check processor state**:
   * Right-click QueryDatabaseTableRecord → "View State"
   * Verify new maximum timestamp is stored

**Validate Incremental Results**

-- Verify new records were processed  
SELECT \* FROM public.scms\_data\_1   
WHERE id IN (9999, 9998);  
  
-- Check total count increased  
SELECT COUNT(\*) FROM public.scms\_data\_1;

**Step 4: Test UPSERT Functionality**

**Update Existing Records**

-- Update existing records in source table  
UPDATE public.scms\_data\_2   
SET   
 email = 'updated\_' || email,  
 \_airbyte\_extracted\_at = CURRENT\_TIMESTAMP  
WHERE id IN (9999, 9998);

**Run UPSERT Test**

1. **Trigger processor run** (or wait for scheduled run)
2. **Monitor processing**

**Validate UPSERT Results**

-- Verify records were updated, not duplicated  
SELECT id, name, email, updated\_at   
FROM public.scms\_data\_1   
WHERE id IN (9999, 9998);  
  
-- Confirm no duplicate records  
SELECT id, COUNT(\*)   
FROM public.scms\_data\_1   
GROUP BY id   
HAVING COUNT(\*) > 1;

**Troubleshooting Guide**

**Common Issues and Solutions**

**Issue 1: "Relation does not exist" Error**

**Symptoms**: QueryDatabaseTableRecord fails with relation error

**Solutions**:

1. **Check table name qualification**:

Correct: public.scms\_data\_2  
Incorrect: scms\_data\_2

1. **Verify database connection**:

-- Test connection and schema  
SELECT table\_schema, table\_name   
FROM information\_schema.tables   
WHERE table\_name = 'scms\_data\_2';

1. **Check user permissions**:

-- Verify user can access table  
SELECT \* FROM public.scms\_data\_2 LIMIT 1;

**Issue 2: No Data Being Processed**

**Symptoms**: QueryDatabaseTableRecord runs but processes 0 records

**Solutions**:

1. **Check processor state**:
   * Right-click processor → "View State"
   * Clear state if timestamp is ahead of data
2. **Verify Maximum-Value Column**:

-- Check if column exists and has data  
SELECT   
 COUNT(\*) as total\_records,  
 COUNT(\_airbyte\_extracted\_at) as records\_with\_timestamp  
FROM public.scms\_data\_2;

1. **Check data in time range**:

-- Find records newer than stored state  
SELECT COUNT(\*) FROM public.scms\_data\_2   
WHERE \_airbyte\_extracted\_at > 'YOUR\_LAST\_PROCESSED\_TIMESTAMP';

**Issue 3: UPSERT Failures in PutDatabaseRecord**

**Symptoms**: Records reach PutDatabaseRecord but UPSERT fails

**Solutions**:

1. **Verify Update Keys configuration**:

Update Keys: id (must match exactly, case-sensitive)

1. **Check PostgreSQL version supports UPSERT**:

-- PostgreSQL 9.5+ required for INSERT...ON CONFLICT  
SELECT version();

1. **Verify target table structure**:

-- Ensure primary key exists  
\d public.scms\_data\_1

**Issue 4: Connection Pool Exhaustion**

**Symptoms**: Database connection errors or timeouts

**Solutions**:

1. **Increase connection pool size**:

Max Total Connections: 8 → 16

1. **Reduce concurrent tasks**:

Concurrent Tasks: 1 (recommended for database processors)

1. **Check for connection leaks**:

-- Monitor active connections  
SELECT count(\*) FROM pg\_stat\_activity   
WHERE usename = 'nifi\_user';

**Issue 5: Schema or Data Type Mismatches**

**Symptoms**: Records fail to process due to schema issues

**Solutions**:

1. **Check column data types**:

-- Compare source and target schemas  
SELECT column\_name, data\_type   
FROM information\_schema.columns   
WHERE table\_name IN ('scms\_data\_2', 'scms\_data\_1')  
ORDER BY table\_name, ordinal\_position;

1. **Verify Avro schema handling**:
   * Use "Infer Schema" strategy
   * Check for special characters in column names
2. **Handle NULL values**:

-- Check for NULL values in key columns  
SELECT COUNT(\*) FROM public.scms\_data\_2   
WHERE id IS NULL OR \_airbyte\_extracted\_at IS NULL;

**Debugging Steps**

**Enable Debug Logging**

1. **Configure NiFi logging**:

# In logback.xml  
<logger name="org.apache.nifi.processors.standard" level="DEBUG"/>  
<logger name="org.apache.nifi.controller.service" level="DEBUG"/>

1. **Monitor NiFi logs**:

tail -f /path/to/nifi/logs/nifi-app.log

**Use Data Provenance**

1. **Access Provenance**:
   * Right-click processor → "View Data Provenance"
   * Check FlowFile attributes and content
2. **Examine FlowFile Content**:
   * Click on provenance event
   * Select "View Content" to see actual data

**Monitor Performance**

1. **Check processor statistics**:
   * Right-click processor → "View Configuration" → "Stats"
   * Monitor processing times and throughput
2. **Database performance monitoring**:

-- Check query performance  
SELECT query, mean\_exec\_time, calls   
FROM pg\_stat\_statements   
WHERE query LIKE '%scms\_data%'  
ORDER BY mean\_exec\_time DESC;

**Performance Optimization**

**NiFi Configuration Optimization**

**Processor-Level Optimization**

QueryDatabaseTableRecord:  
├── Partition Size: 1000 (adjust based on memory)  
├── Max Rows Per Flow File: 10000 (balance memory vs. processing)  
├── Concurrent Tasks: 1 (avoid conflicts)  
└── Run Schedule: Align with business requirements  
  
PutDatabaseRecord:  
├── Max Batch Size: 100 (balance transaction size vs. performance)  
├── Concurrent Tasks: 1 (maintain transaction integrity)  
└── Statement Type: UPSERT (most efficient for mixed operations)

**Connection Pool Tuning**

DBCPConnectionPool:  
├── Max Total Connections: 8-16 (based on database capacity)  
├── Min Idle: 2-4 (maintain ready connections)  
├── Max Idle: 8 (balance resource usage)  
├── Max Wait Time: 500ms-1s (reasonable timeout)  
└── Validation Query: SELECT 1 (lightweight validation)

**Database Optimization**

**Index Strategy**

-- Essential indexes for performance  
CREATE INDEX CONCURRENTLY idx\_scms\_data\_2\_extracted\_at   
ON public.scms\_data\_2(\_airbyte\_extracted\_at);  
  
CREATE INDEX CONCURRENTLY idx\_scms\_data\_2\_id   
ON public.scms\_data\_2(id);  
  
CREATE INDEX CONCURRENTLY idx\_scms\_data\_1\_id   
ON public.scms\_data\_1(id);  
  
-- Composite index for complex queries (if needed)  
CREATE INDEX CONCURRENTLY idx\_scms\_data\_2\_composite   
ON public.scms\_data\_2(\_airbyte\_extracted\_at, id);

**Database Configuration**

-- Optimize for batch processing  
SET work\_mem = '256MB';  
SET maintenance\_work\_mem = '1GB';  
SET checkpoint\_completion\_target = 0.9;  
SET wal\_buffers = '16MB';  
SET effective\_cache\_size = '4GB';

**Monitoring and Alerts**

**Key Metrics to Monitor**

NiFi Metrics:  
├── FlowFiles processed per minute  
├── Processing time per batch  
├── Memory usage and garbage collection  
├── Connection pool utilization  
└── Error rates and retry attempts  
  
Database Metrics:  
├── Query execution times  
├── Connection count and wait times  
├── Index usage statistics  
├── Lock contention and blocking  
└── Disk I/O and buffer hit ratios

**Performance Baselines**

Expected Performance (1000 records):  
├── Query Processing: 10-30 seconds  
├── UPSERT Operations: 20-60 seconds  
├── Memory Usage: 200-500MB  
├── CPU Usage: 20-40% during processing  
└── Database Connections: 2-4 concurrent

Follow each section carefully to ensure successful deployment and operation of your data processing system.