

# Module 9

## Implementing a Data Extraction Solution



**Zyeed Ahmed.**  
**Aspiring To Learning Data Engineer.**

# Module Overview

- Introduction to Incremental ETL
- Extracting Modified Data
- Loading Modified Data
- Temporal Tables

# Lesson 1: Introduction to Incremental ETL

- Overview of Data Warehouse Load Cycles
- Considerations for Incremental ETL
- Common ETL Data Flow Architectures
- Planning Extraction Windows
- Planning Transformations
- Documenting Data Flows
- Slowly Changing Dimensions

# Overview of Data Warehouse Load Cycles

- Extract changes from data sources
- Refresh the data warehouse based on changes

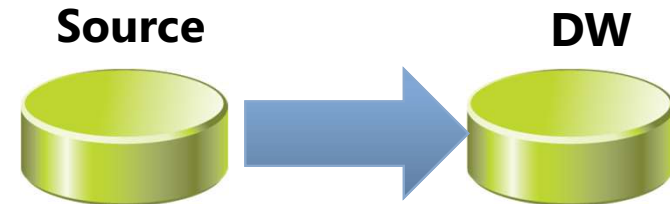
# Considerations for Incremental ETL

- Data modifications to be tracked
- Load order
- Dimension keys
- Updating dimension members
- Updating fact records

# Common ETL Data Flow Architectures

- Single-stage ETL:

- Data is transferred directly from source to data warehouse
- Transformations and validations occur in-flight or on extraction



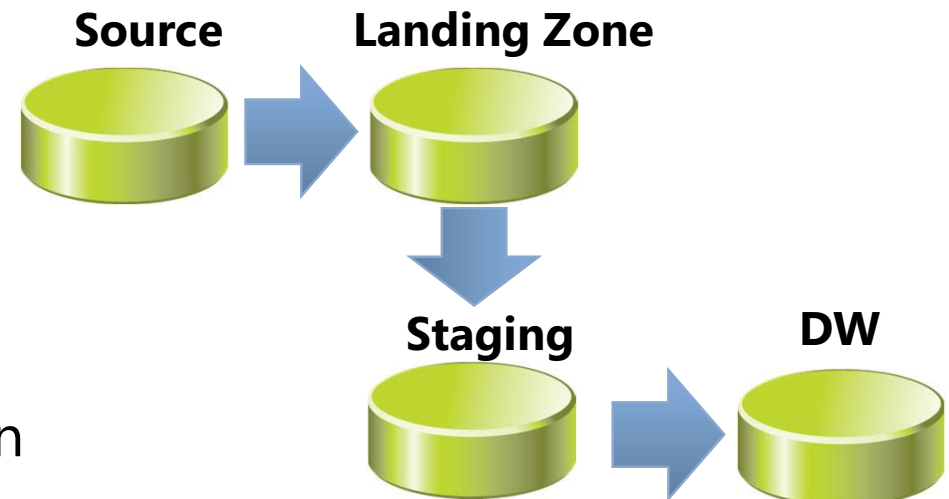
- Two-stage ETL:

- Data is staged for a coordinated load
- Transformations and validations occur in-flight, or on staged data

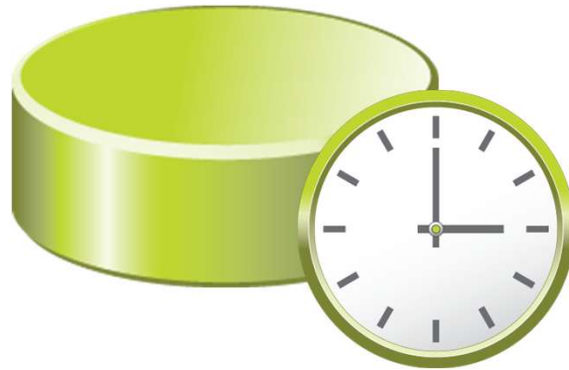


- Three-stage ETL:

- Data is extracted quickly to a landing zone, and then staged prior to loading
- Transformations and validation can occur throughout the data flow



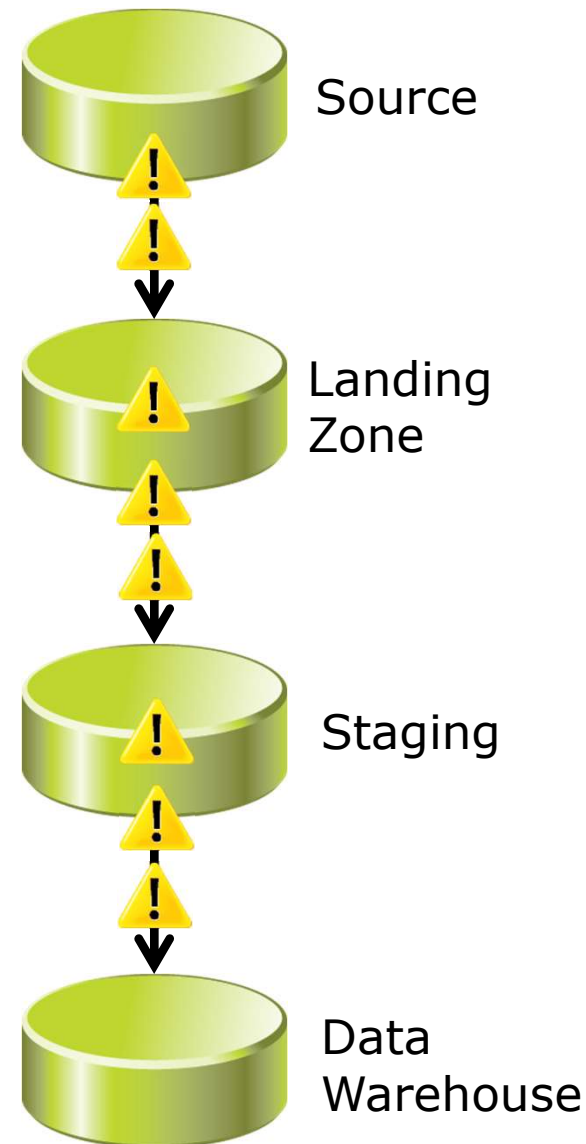
# Planning Extraction Windows



- How frequently is new data generated in the source systems, and for how long is it retained?
- What latency between changes in source system and reporting is tolerable?
- How long does data extraction take?
- During what time periods are source systems least heavily used?

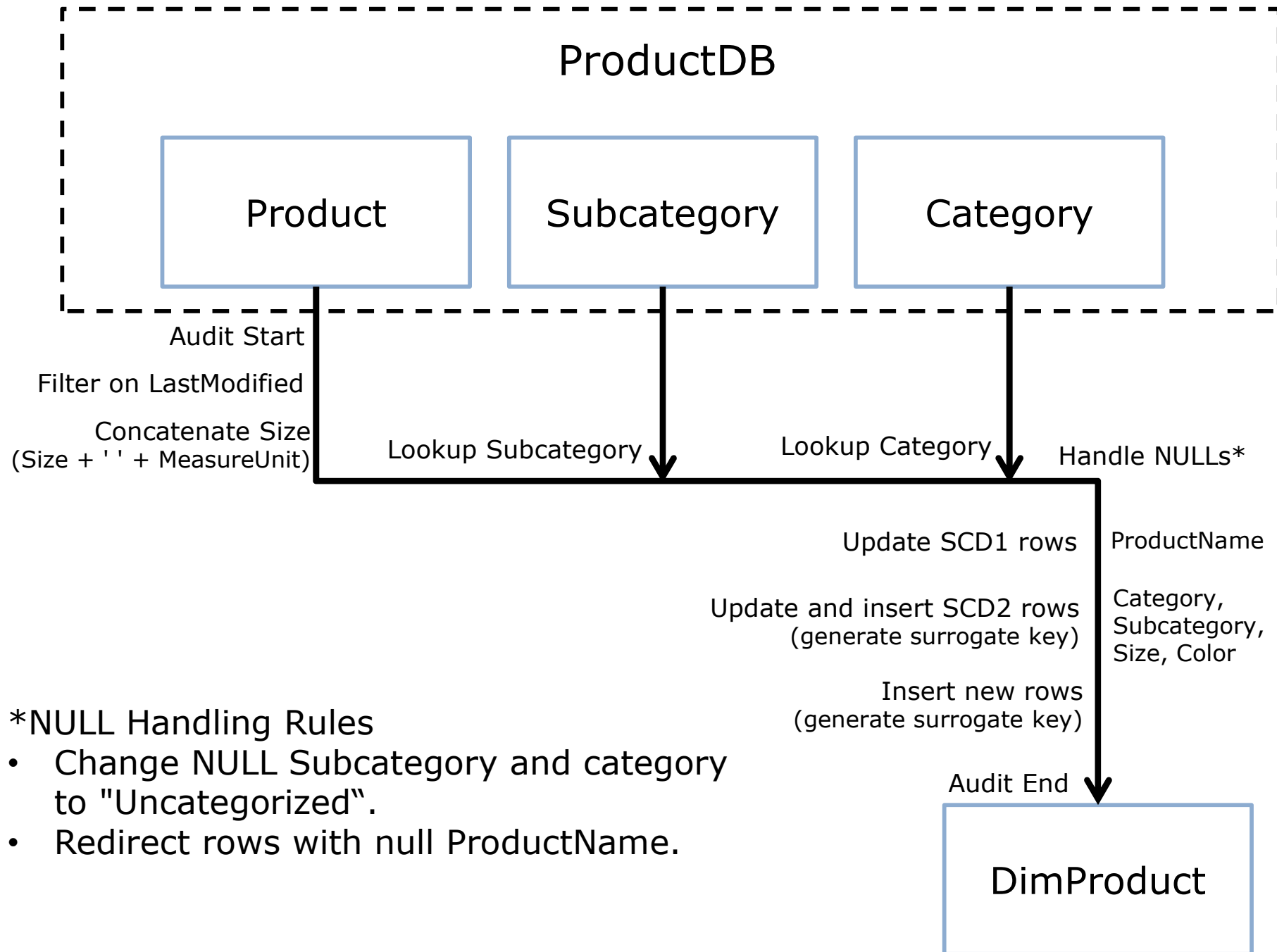
# Planning Transformations

- On extraction:
  - From source
  - From landing zone
  - From staging
- In data flow:
  - Source to landing zone
  - Landing zone to staging
  - Staging to data warehouse
- In-place:
  - In landing zone
  - In staging






# Documenting Data Flows



# Slowly Changing Dimensions

- Types of change to a dimension member:
  - Type 1: Changing attributes are updated in the dimension record


Key	AltKey	Name	Phone	City
101	C123	Mary	5551234	New York



Key	AltKey	Name	Phone	City
101	C123	Mary	5554321	New York

- Type 2: Historical attribute changes result in a new record


Key	AltKey	Name	Phone	City	Current
101	C123	Mary	5551234	New York	True



Key	AltKey	Name	Phone	City	Current
101	C123	Mary	5551234	New York	False
102	C123	Mary	5551234	Seattle	True

- Type 3: The original and current values of historical attributes are stored in the dimension record

Key	AltKey	Name	Phone	OriginalCity	CurrentCity	EffectiveDate
101	C123	Mary	5551234	New York	New York	1/1/00



Key	AltKey	Name	Phone	OriginalCity	CurrentCity	EffectiveDate
101	C123	Mary	5551234	New York	<b>Seattle</b>	<b>6/7/11</b>

## Lesson 2: Extracting Modified Data

- Options for Extracting Modified Data
- Extracting Rows Based on a Datetime Column
- Demonstration: Using a Datetime Column
- Change Data Capture
- Demonstration: Using Change Data Capture
- Extracting Data with Change Data Capture
- The CDC Control Task and Data Flow Components
- Demonstration: Using CDC Components
- Change Tracking
- Demonstration: Using Change Tracking
- Extracting Data with Change Tracking

# Options for Extracting Modified Data

- Extract all records
- Store a primary key and checksum
- Use a datetime column as a “high water mark”
- Use Change Data Capture
- Use Change Tracking

# Extracting Rows Based on a Datetime Column

1. Note the current time
2. Retrieve the last extraction time from an extraction log
3. Extract and transfer records that were modified between the last extraction and the current time
4. Replace the stored last extraction value with the current time

# Demonstration: Using a Datetime Column

In this demonstration, you will see how to use a Datetime Column to Extract Modified Data

# Change Data Capture

## 1. Enable Change Data Capture:

```
EXEC sys.sp_cdc_enable_db  
  
EXEC sys.sp_cdc_enable_table @source_schema = N'dbo', @source_name = N'Customers',  
                             @role_name = NULL, @supports_net_changes = 1
```

## 2. Map start and end times to log sequence numbers:

```
DECLARE @from_lsn binary(10), @to_lsn binary(10);  
SET @from_lsn = sys.fn_cdc_map_time_to_lsn('smallest greater than', @StartDate)  
SET @to_lsn = sys.fn_cdc_map_time_to_lsn('largest less than or equal', @EndDate)
```

## 3. Handle null log sequence numbers:

```
IF (@from_lsn IS NULL) OR (@to_lsn IS NULL)  
-- There may have been no transactions in the timeframe
```

## 4. Extract changes between log sequence numbers:

```
SELECT * FROM cdc.fn_cdc_get_net_changes_dbo_Customers(@from_lsn, @to_lsn, 'all')
```

# Demonstration: Using Change Data Capture

In this demonstration, you will see how to:

- Enable Change Data Capture
- Use Change Data Capture to Extract Modified Data

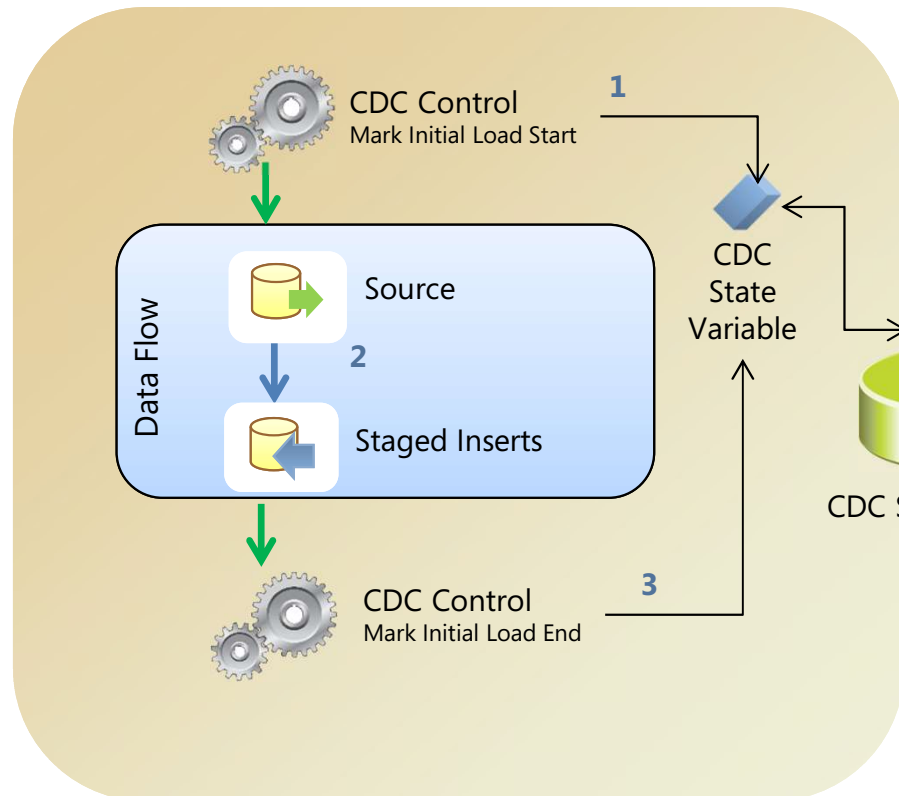


# Extracting Data with Change Data Capture

1. Identify the endpoint for the extraction (LSN or DateTime)
2. Retrieve the last extraction endpoint from an extraction log
3. Extract and transfer records that were modified during the LSN range defined by the previous extraction endpoint and the current endpoint
4. Replace the logged endpoint value with the current endpoint

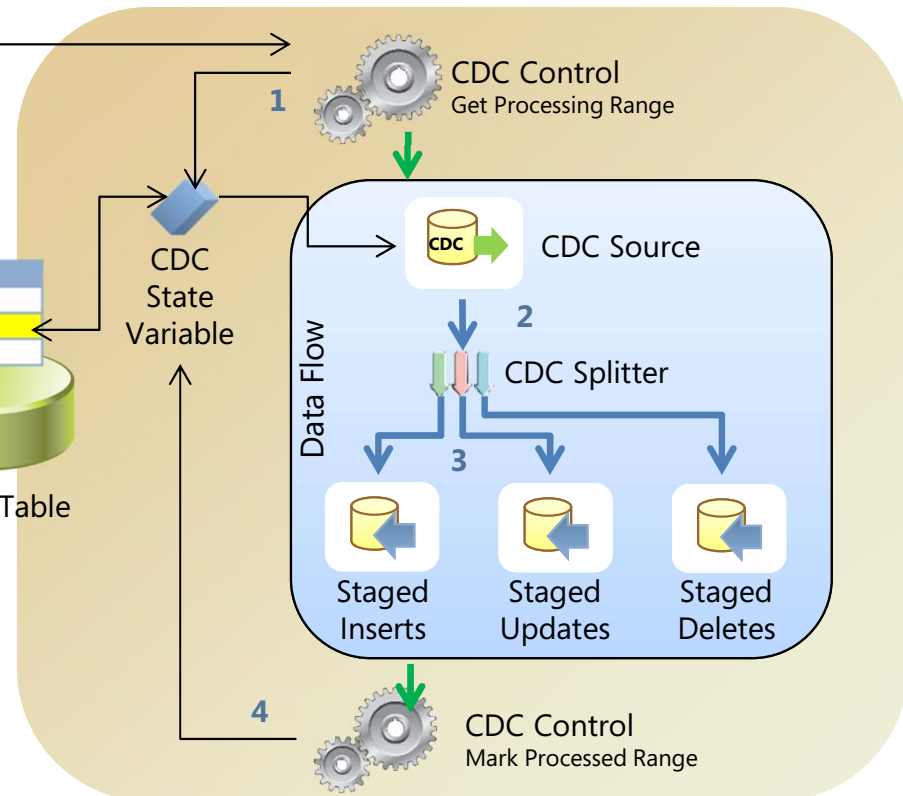
# The CDC Control Task and Data Flow Components

## Initial Extraction



1. A CDC Control Task records the starting LSN.
2. A data flow extracts all records.
3. A CDC Control task records the ending LSN.

## Incremental Extraction



1. CDC Control Task establishes the range of LSNs to be extracted.
2. A CDC Source extracts records and CDC metadata.
3. Optionally, a CDC Splitter splits the data flow into inserts, updates, and deletes.
4. A CDC Control task records the ending LSN.

# Demonstration: Using CDC Components

In this demonstration, you will see how to use the CDC Control Task to:

- Perform an Initial Extraction
- Extract Changes

# Change Tracking

## 1. Enable Change Tracking

```
ALTER DATABASE Sales
SET CHANGE_TRACKING = ON (CHANGE_RETENTION = 7 DAYS, AUTO_CLEANUP = ON)

ALTER TABLE Salespeople
ENABLE CHANGE_TRACKING WITH (TRACK_COLUMNS_UPDATED = OFF)
```

## 2. Record the current version and extract the initial data

```
SET @CurrentVersion = CHANGE_TRACKING_CURRENT_VERSION();
SELECT * FROM Salespeople
SET @LastExtractedVersion = @CurrentVersion
```

## 3. Extract changes since the last extracted version, and then update the last extracted version

```
SET @CurrentVersion = CHANGE_TRACKING_CURRENT_VERSION();
SELECT * FROM CHANGETABLE(CHANGES Salespeople, @LastExtractedVersion) CT
    INNER JOIN Salespeople s ON CT.SalespersonID = s.SalespersonID
SET @LastExtractedVersion = @CurrentVersion
```

**Tip:** Use snapshot isolation to ensure consistency

# Demonstration: Using Change Tracking

In this demonstration, you will see how to:

- Enable Change Tracking
- Use Change Tracking

# Extracting Data with Change Tracking

1. Retrieve the last version number that was extracted from an extraction log
2. Extract and transfer records that were modified since the last version, retrieving the current version number
3. Replace the logged version number with the current version number

# Lab A: Extracting Modified Data

- Exercise 1: Using a Datetime Column to Incrementally Extract Data
- Exercise 2: Using Change Data Capture
- Exercise 3: Using the CDC Control Task
- Exercise 4: Using Change Tracking

## **Logon Information**

Virtual machine: **20767C-MIA-SQL**

User name: **ADVENTUREWORKS\Student**

Password: **Pa55w.rd**

**Estimated Time: 60 minutes.**

## Lab Scenario

You have developed SSIS packages that extract data from various data sources and load it into a staging database. However, the current solution extracts all source records each time the ETL process is run. This results in unnecessary processing of records that have already been extracted and consumes a sizeable amount of network bandwidth to transfer a large volume of data. To resolve this problem, you must modify the SSIS packages to extract only data that has been added or modified since the previous extraction.



# Lab Review

Having completed this lab, you will now be able to:

- Use a datetime column to extract modified rows
- Use Change Data Capture to extract modified rows
- Use the CDC Control Task to extract modified rows
- Use Change Tracking to extract modified rows

## Lesson 3: Loading Modified Data

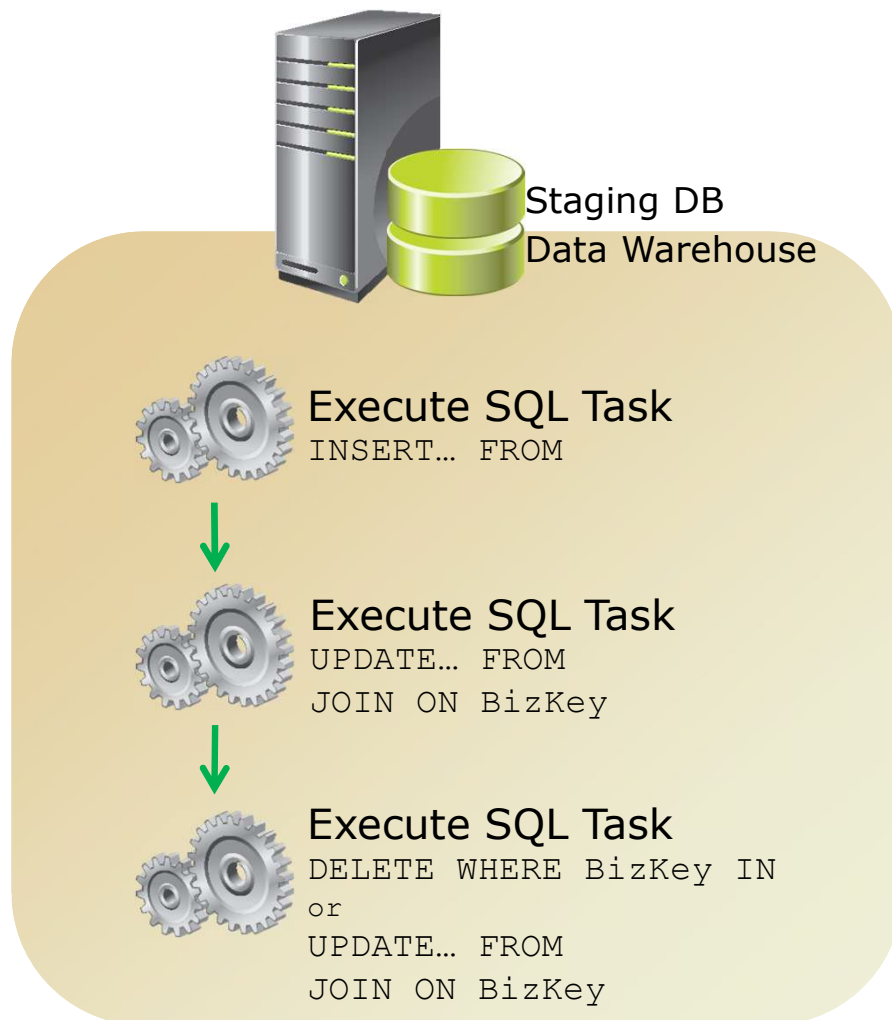
- Options for Incrementally Loading Data
- Using CDC Output Tables
- Demonstration: Using CDC Output Tables
- The Lookup Transformation
- Demonstration: Using the Lookup Transformation
- The Slowly Changing Dimension Transformation
- The MERGE Statement
- Demonstration: Using the Merge Statement
- Partition Switching
- Demonstration: Partition Switching

# Options for Incrementally Loading Data

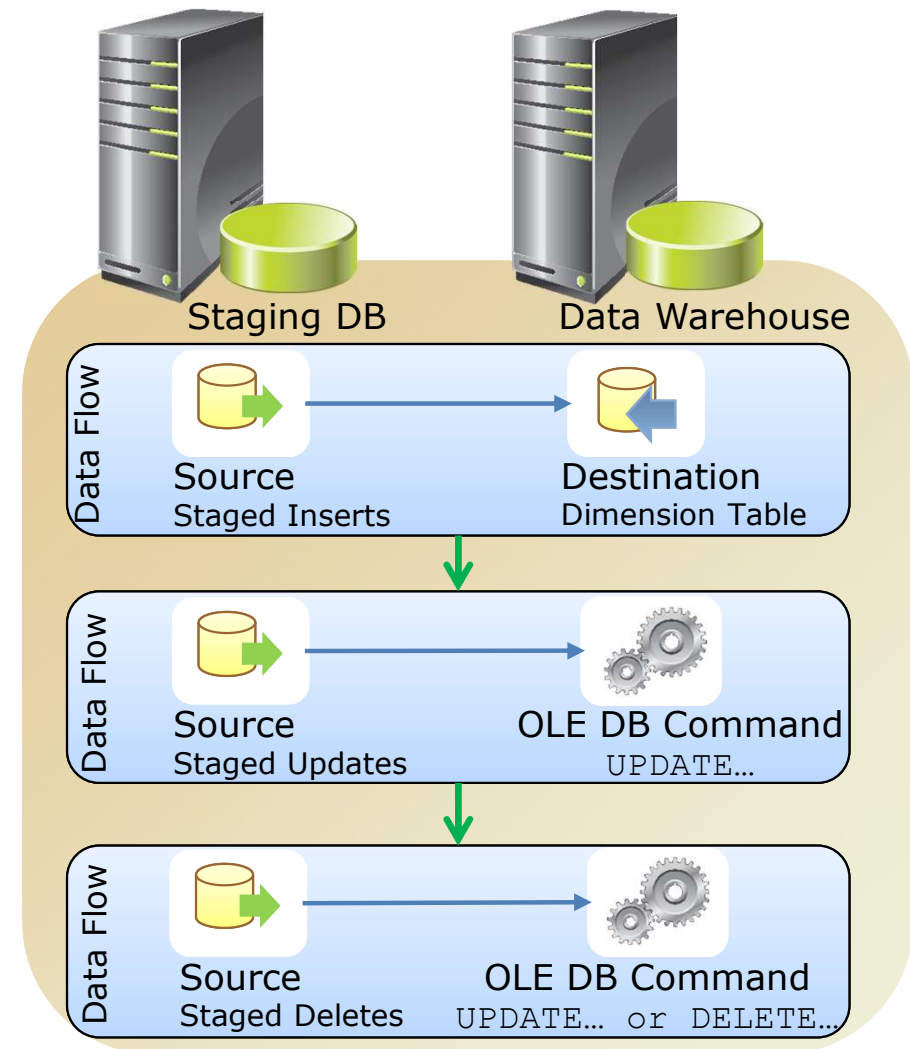
- Insert, Update, or Delete from CDC Output Tables
- Use a Lookup transformation
- Use the Slowly Changing Dimension transformation
- Use the MERGE statement
- Use a checksum
- Considerations for Deleting Data Warehouse records:
  - Use a logical deletion technique
  - Technique depends on how deleted records are staged

# Using CDC Output Tables

## Staging and Data Warehouse Co-Located



## Remote Data Warehouse



# Demonstration: Using CDC Output Tables

In this demonstration, you will see how to load data from CDC output tables

# The Lookup Transformation

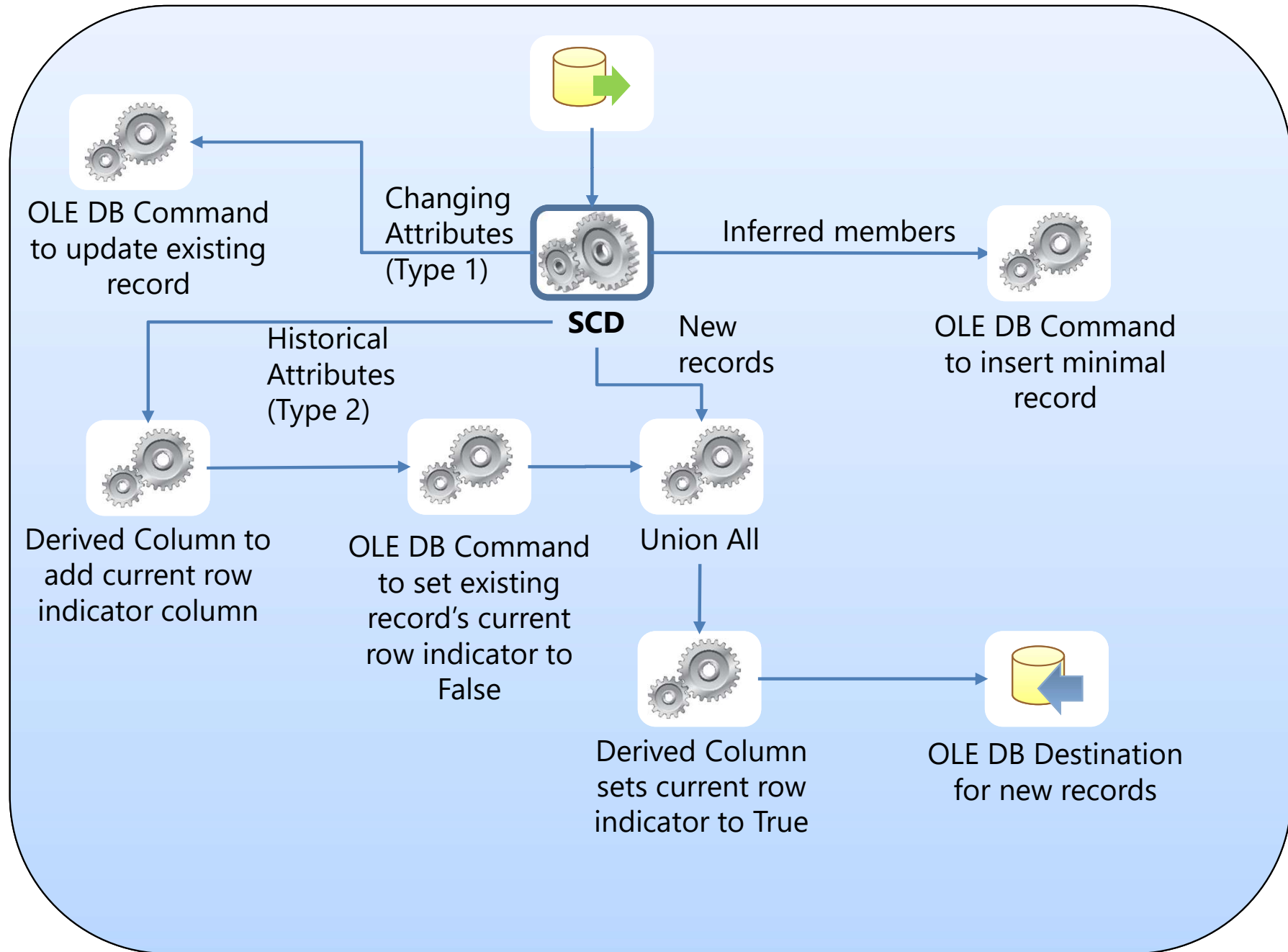
- Redirect nonmatched rows to the *no match* output
- Look up extracted data in a dimension or fact table based on a business key or unique combination of keys
- If no match is found, insert a new record
- Optionally, if a match is found, update non-key columns to apply a type 1 change

# Demonstration: Using the Lookup Transformation

In this demonstration, you will see how to:

- Use a Lookup Transformation to Insert Rows
- Use a Lookup Transformation to Insert and Update Rows

# The Slowly Changing Dimension Transformation





# The MERGE Statement

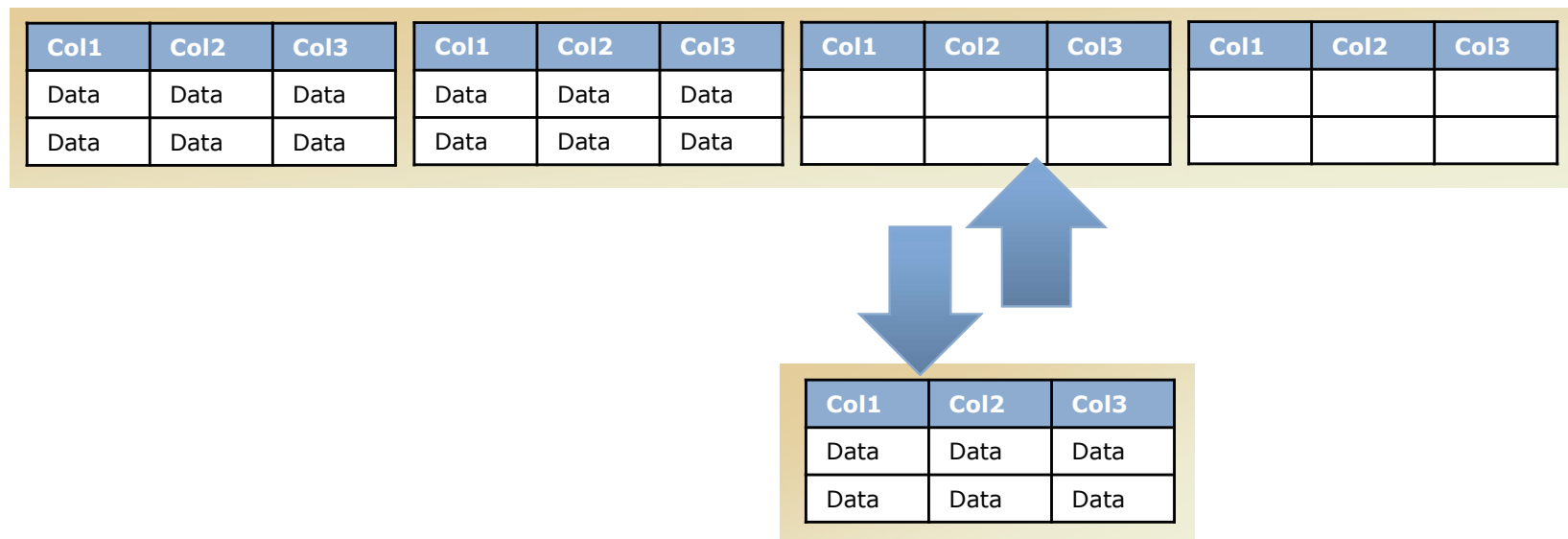
- Matches source and target rows
- Performs insert, update, or delete operations based on row matching results

# Demonstration: Using the Merge Statement

In this demonstration, you will see how to use the MERGE Statement

# Partition Switching

- Switch loaded tables into partitions
- Partition-align indexed views



# Demonstration: Partition Switching

In this demonstration, you will see how to:

- Split a partition
- Create a load table
- Switch a partition

# Lesson 4: Temporal Tables

- About System-Versioned Tables
- Considerations for System-Versioned Tables
- Creating System-Versioned Tables
- Querying System-Versioned Tables
- Demonstration: Creating System-Versioned Tables
- Using System-Versioned Tables to Implement Slowly Changing Dimensions
- Change Data Capture Compared to System-Versioned Tables

# About System-Versioned Tables

- System-Versioned tables summary:
  - Enable a full history of data changes
  - Current and historical tables operate as a pair
  - Feature can be added to existing tables
  - Historical table can be named, or take system name
  - Current table must have a primary key
- System-versioned tables operation:
  - Versioning is automatic
  - SysStartTime and SysEndTime columns define the validity period for data versions
  - History specific queries are used to obtain version information

# Considerations for System-Versioned Tables

- Main considerations when using System-Versioned tables:
  - Current table must have a primary key
  - SysStartTime and SysEndTime must be datetime2
  - Data in the historical table cannot be directly modified
  - Current table cannot be truncated when SYSTEM\_VERSIONING is ON
  - FILETABLE or FILESTREAM are not supported
  - History table is PAGE compressed by default

# Creating System-Versioned Tables

- Create a new System-Versioned table:

```
CREATE TABLE dbo.Employee
(
    EmployeeID int NOT NULL PRIMARY KEY CLUSTERED,
    ManagerID int NULL,
    FirstName varchar(50) NOT NULL,
    LastName varchar(50) NOT NULL,
    SysStartTime datetime2 GENERATED ALWAYS AS ROW START NOT NULL,
    SysEndTime datetime2 GENERATED ALWAYS AS ROW END NOT NULL,
    PERIOD FOR SYSTEM_TIME (SysStartTime, SysEndTime)
)
WITH (SYSTEM_VERSIONING = ON (HISTORY_TABLE = dbo.EmployeeHistory));
```



# Querying System-Versioned Tables

- System-Versioned tables can be queried using the FOR SYSTEM\_TIME clause and one of the following four subclauses:
  - AS OF <date\_time>
  - FROM <start\_date\_time> TO <end\_date\_time>
  - BETWEEN <start\_date\_time> AND <end\_date\_time>
  - CONTAINED IN (<start\_date\_time>, <end\_date\_time>)
- Or use ALL to return everything

# Demonstration: Creating System-Versioned Tables

In this demonstration you will learn:

- How to create a System-Versioned table
- How to update an existing table to make it system-versioned
- You will also look at the structure of the tables in SSMS

# Using System-Versioned Tables to Implement Slowly Changing Dimensions

## Slowly Changing Dimensions (SCD):

- SCD manages incoming data:
  - Changed, historical, fixed, disallowed, inferred
- SCD change types and transformation outputs:
  - Changing attributes updates
  - Historical attribute insert output and new
  - Fixed attribute inserts
  - Inferred member
- The SCD wizard:
  - Choose data source and dimension table
  - Configure mapping
  - Set attribute options
  - Review, run and update

# Change Data Capture Compared to System-Versioned Tables

Parameter	CDC	System-Versioning
Usage	Rapid data change, short retention. ETL, data checks, fact tables. Transact-SQL options, SSMS, and Visual Studio.	Slower data change, long retention. SCD, audit, fix corrupt data, period reporting, and comparison. Transact SQL and SSMS.
Deployment	Database and table levels.	Table level.
Methodology	Transaction logs many tables.	One historical table per source table.
Scope	Down to row/column level.	Source table tracked at row level.
Performance	Handles large data amounts quickly.	Large data amounts slower.
Dependencies	SQL Server Agent.	None.
Operations	Can be quickly enabled/disabled at database level.	Requires each table to be enabled/disabled.
Operational	Easy to enable/disable for maintenance, and so on.	More difficult.
Primary Tables	No extra columns in primary.	No extra columns in primary.
Primary Keys	Not required.	Required.