06: Joins

CONFLUENT
Global Education

Module Overview

This module contains five lessons:

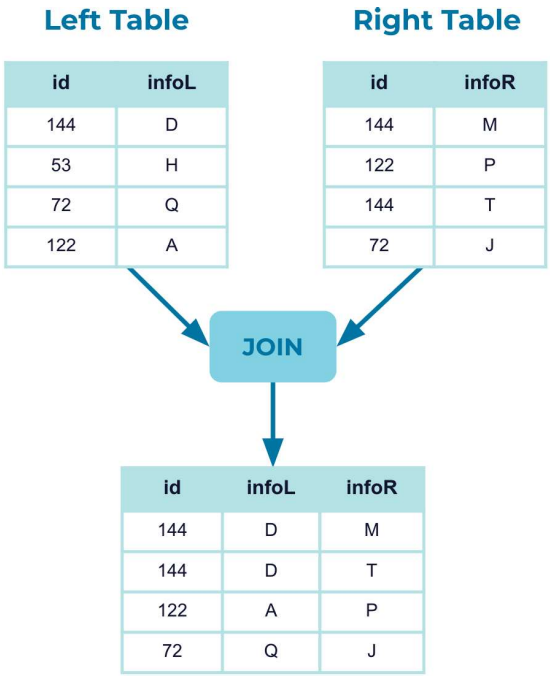
- Introduction to Joins
- Regular Joins
- Optimized Joins
- Window Joins
- Other Joins

06a: Introduction to Joins

Description

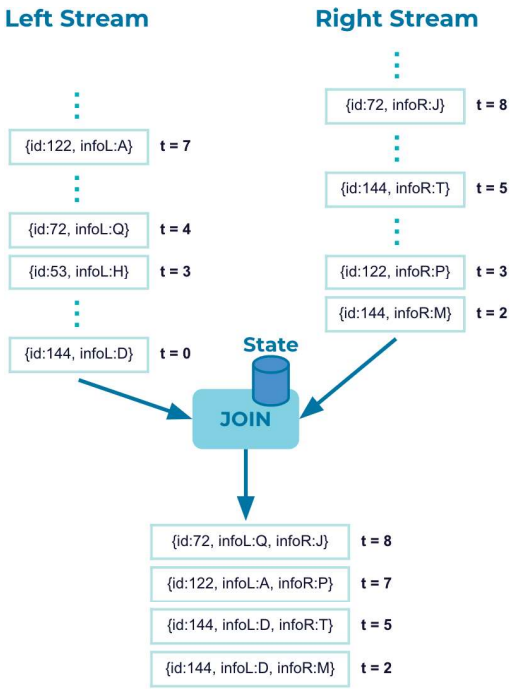
Explaining fundamental concepts about Joins, types of joins and differences between batch and streaming.

Joining Tables



Joining static tables is well understood. You have different types of joins that operate on some join predicates. There are many different available algorithms and since they are static tables all the data is available when a join is performed.

Joining Dynamic Tables



Joining dynamic tables is more complex since the tables are constantly changing. For example a regular streaming join is potentially quite expensive due to the requirement to retain both tables in state. Flink supports many ways to join dynamic tables and each supports special optimization joins for cases that can be handled without unbounded state retention.

Joins: Batch vs. Streaming

Key differences between batch data joining and streaming data joining:

- Data Availability
- State Management
- Handling Data Updates
- System Complexity and Performance

- Data Availability:
  - Batch Data: The dataset has a clear beginning and end
  - Streaming Data: Data continuously arrives over time with no predefined end
- State Management
  - Batch Data: No state maintenance needed as all data is available upfront
  - Streaming Data: Requires maintaining state for intermediate results and updates.
- Handling Data Updates:
  - Batch Data: Updates are applied in subsequent batch runs, not in real-time.
  - Streaming Data: Dynamically handles updates, deletions, and insertions.
- Complexity and Performance:
  - Batch Data: Simpler implementation and optimization with static data and predictable volumes.
  - Streaming Data: Higher complexity due to state management, fault tolerance, and out-of-order data handling.

Types of Joins in Flink SQL

- Regular Joins
- Interval Joins
- Temporal Joins
- Window Joins
- Cross Joins (Unnest)
- Multi-Joins

Flink supports many ways to join dynamic tables, and some support special optimization joins for cases that can be handled without unbounded state retention.

In this module, we will look at different optimized joins. If Flink's query planner can recognize your query as one of these special joins, it will use an efficient execution plan.

Lesson 06b

06b: Regular Joins

Description

Diving into regular joins. We'll explore Inner, Left Outer, Right Outer, and Full Outer Joins, providing examples to clarify these concepts.

Regular Joins: Features

- Generic join type
- Flexible grammar for streaming queries: supports insert, update, and delete
- Any new or updated row in either table affects the join result
- Requires both sides of the join to be kept in state indefinitely
- Suitable for small to medium-sized tables to avoid state size issues
- Equi-JOINs (at least one equality condition)

```
SELECT *
FROM Orders
JOIN Customers
ON Orders.customerId = Customers.id;
```

DANGER: State might grow indefinitely

Regular joins are the most flexible type of join, where any new row or change to either side of the join is immediately visible and affects the entire join result. For instance, if a new record appears on the left side, it will be joined with all previous and future records on the right side, provided the join fields match.

In streaming queries, the syntax for regular joins is highly flexible, allowing for any kind of updates (inserts, updates, deletes) on the input tables. However, this flexibility comes with significant implications: it requires maintaining the state of both sides of the join indefinitely. This means the state required for computing the query result might grow without bound, depending on the number of distinct input rows and intermediate join results.

A regular join does not have any temporal conditions; it only requires an equality predicate. Both tables involved in the join can be updated, and the results will be emitted downstream as updates are received from either table. This means the join result is updated whenever records in either input table are inserted, deleted, or updated.

The join operator fully materializes both input tables in state, allowing new records to join with any record from the other table. Regular joins work best when both input tables remain relatively small. If the tables grow too large and need to be kept in memory, you may start experiencing performance degradation.

Regular Joins: Example

```
SELECT
    p.user_id,
    p.timestamp AS purchase_timestamp,
    p.product_id,
    p.amount,
    c.advertisement_type,
    c.page_id
FROM
    clicks AS c
JOIN
    purchases AS p
ON
    c.user_id = p.user_id
WHERE
    c.advertisement_type = 'Special Offer';
```

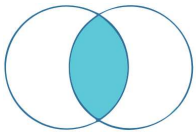
This query allows for the analysis of user behavior in response to specific marketing campaigns or promotions.

`clicks` stream: Contains information about user clicks.
`purchases` stream: Contains information about user purchases.

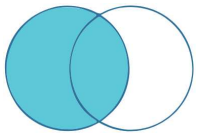
The result aggregates information about user purchases along with details of the corresponding clicks, specifically focusing on users who clicked on a specific type of advertisement (e.g., 'Special Offer'). We can effectively analyze user behavior in response to specific marketing campaigns or promotions. This allows us to gain insights into the effectiveness of advertising efforts by correlating user clicks with subsequent purchases.

Regular Joins: Types

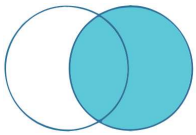
- Inner Joins
- Outer Joins:
  - Left Outer Join
  - Right Outer Join
  - Full Outer Join



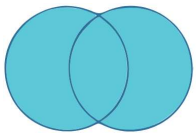
Inner



Left



Right



Full Outer

Regular Joins: Join Semantics

Left Table		Right Table		Joined Table
Append-Only	+	Append-Only	=	Append-Only
Append-Only	+	Updating	=	Updating
Updating	+	Append-Only	=	Updating
Updating	+	Updating	=	Updating

- Updating (upsert or retract)

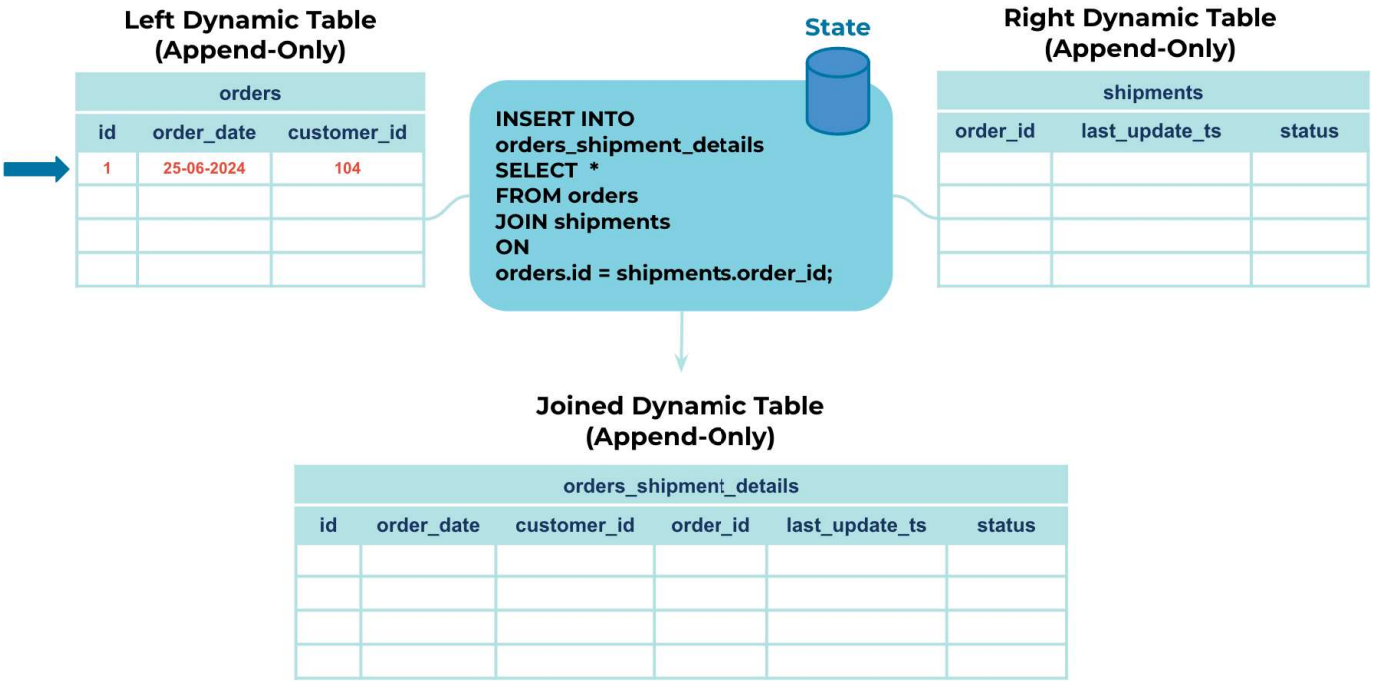
When two append-only tables are joined, the result remains append-only, meaning new records are continuously appended without updates. However, when an append-only table is joined with an updating table (upsert or retract), or when two updating tables are joined, the result is an updating table.

Inner Joins

- Combines records from two tables where the join condition is met
- Returns only the matching rows
- Syntax: `INNER JOIN` or `JOIN`

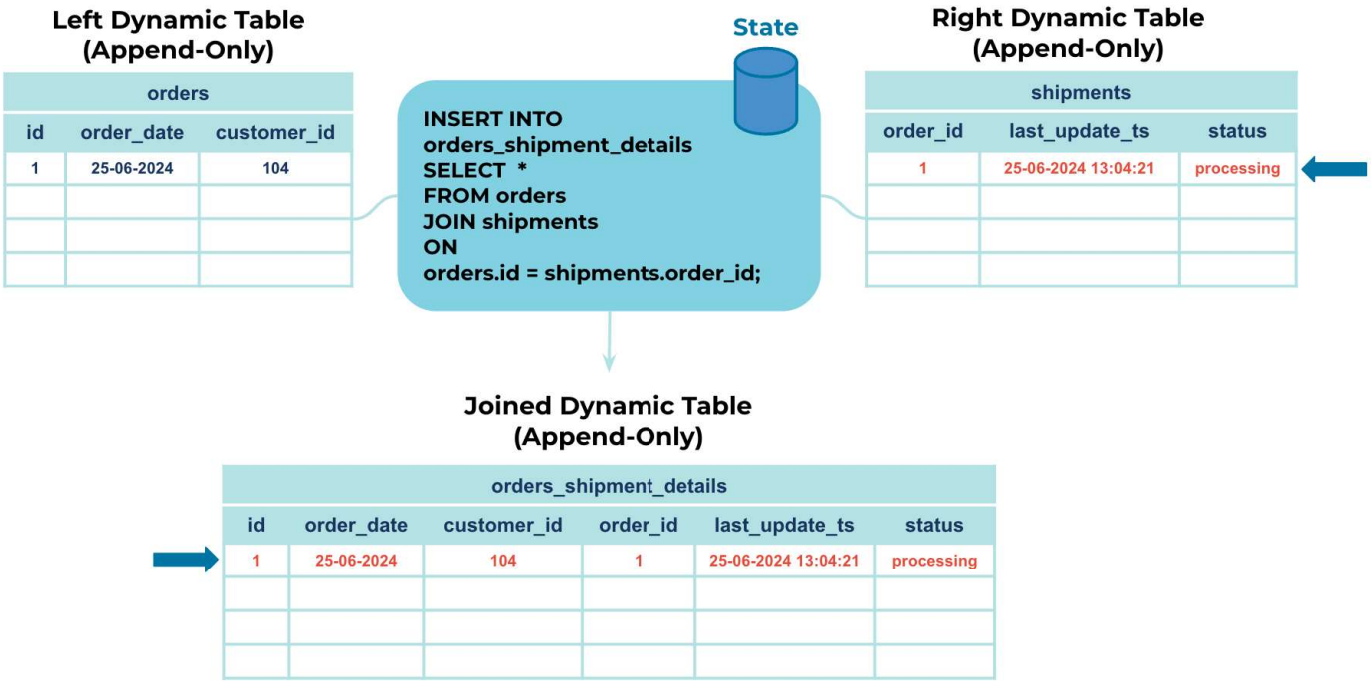
```
SELECT *
FROM Orders
[INNER] JOIN Customers
ON Orders.customerId = Customers.id;
```

Inner Joins: Example (I)

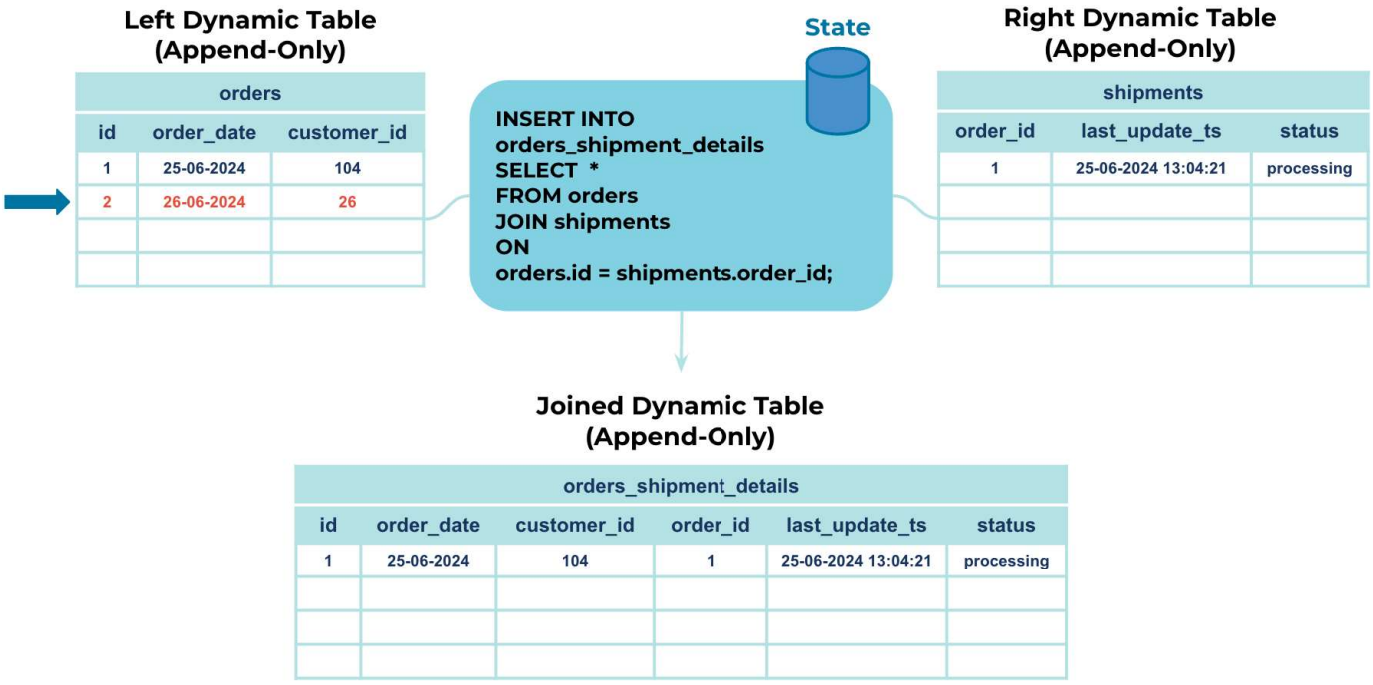


Display all orders along with their shipment status, including orders not yet shipped.

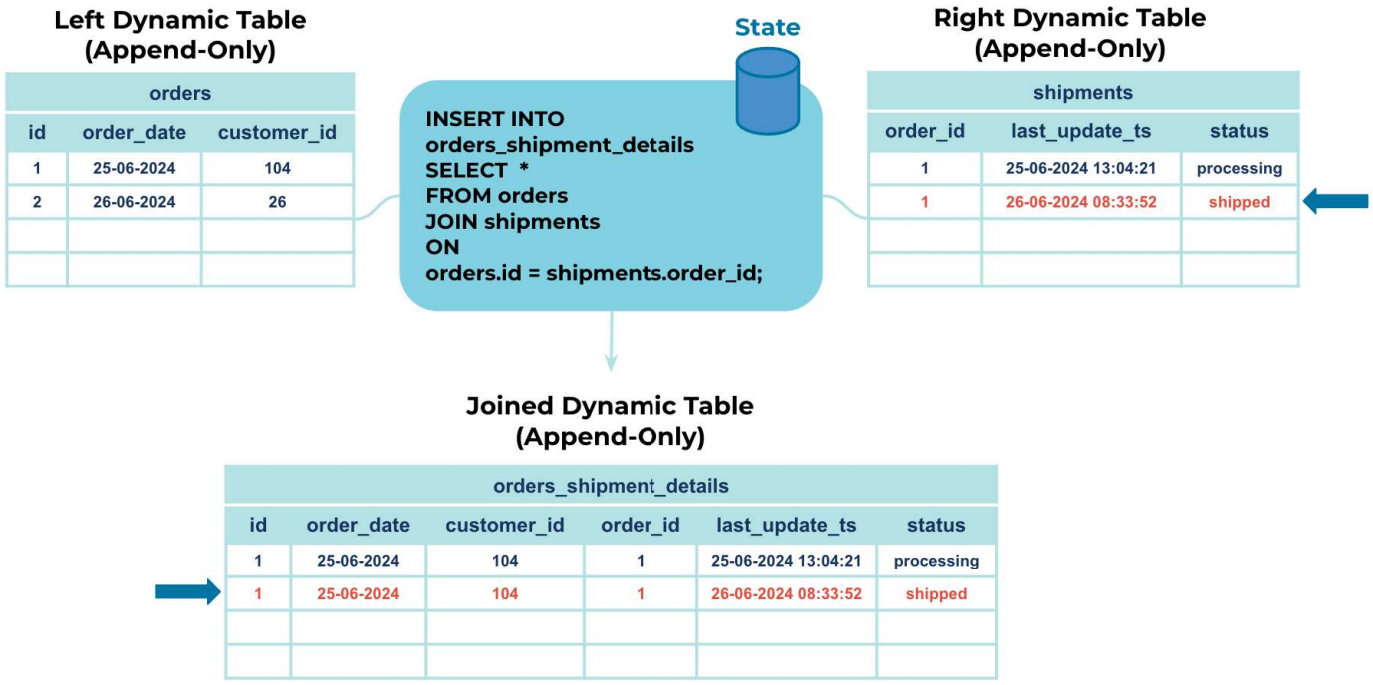
Inner Joins: Example (II)



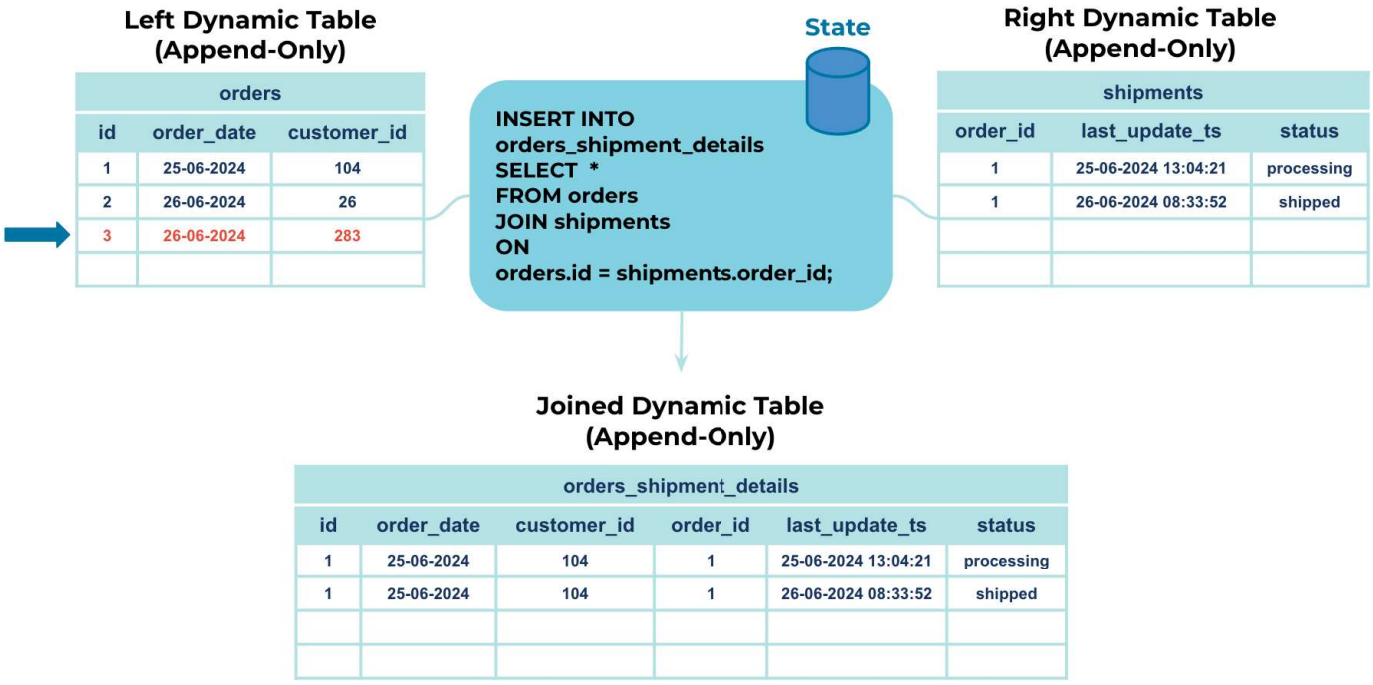
Inner Joins: Example (III)



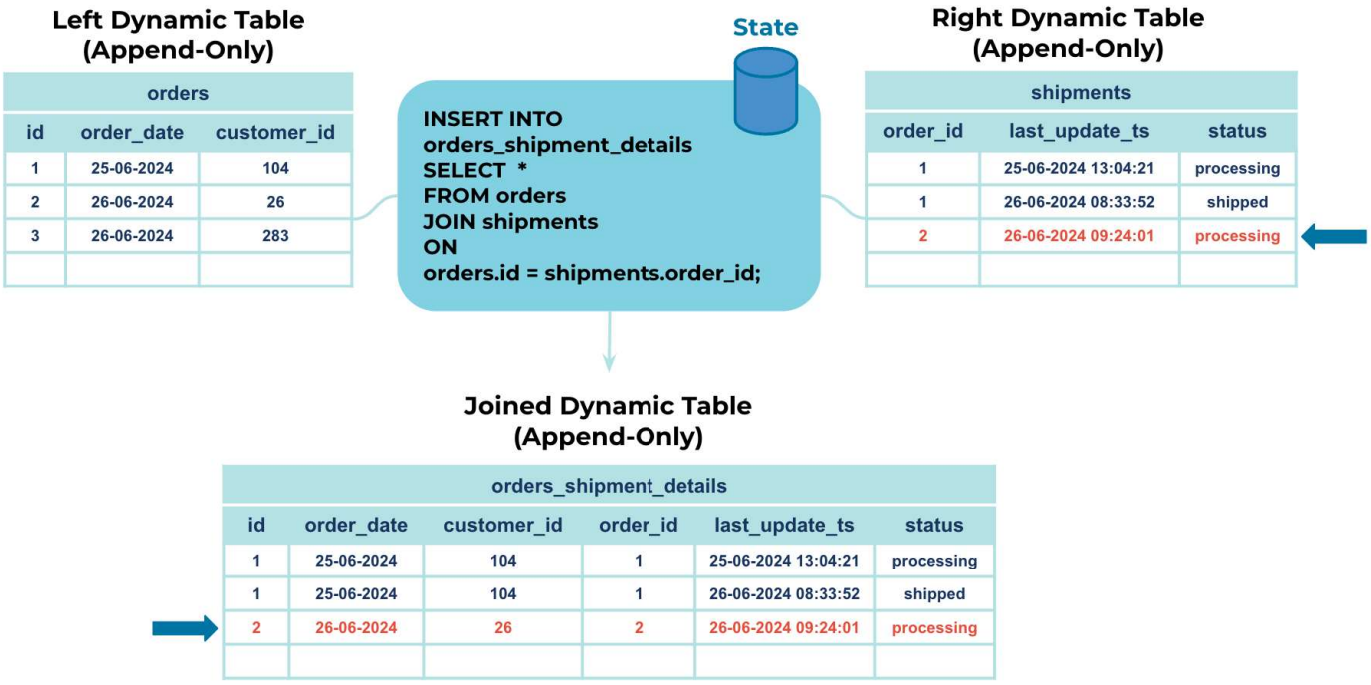
Inner Joins: Example (IV)



Inner Joins: Example (V)



Inner Joins: Example (VI)



Outer Joins

- Includes LEFT, RIGHT, and FULL outer joins.
- Combines all rows that pass the join condition plus unmatched rows from one or both tables.
- Syntax:
  - `LEFT JOIN`
  - `RIGHT JOIN`
  - `FULL OUTER JOIN`

```
SELECT *
FROM Orders
[LEFT,RIGHT,FULL OUTER] JOIN Shipments
ON Orders.orderId = Shipments.orderId;
```

Examples:

LEFT OUTER JOIN: Display all orders along with their shipment status, including orders not yet shipped

```
SELECT *
FROM Orders
LEFT JOIN Shipments
ON Orders.orderId = Shipments.orderId;
```

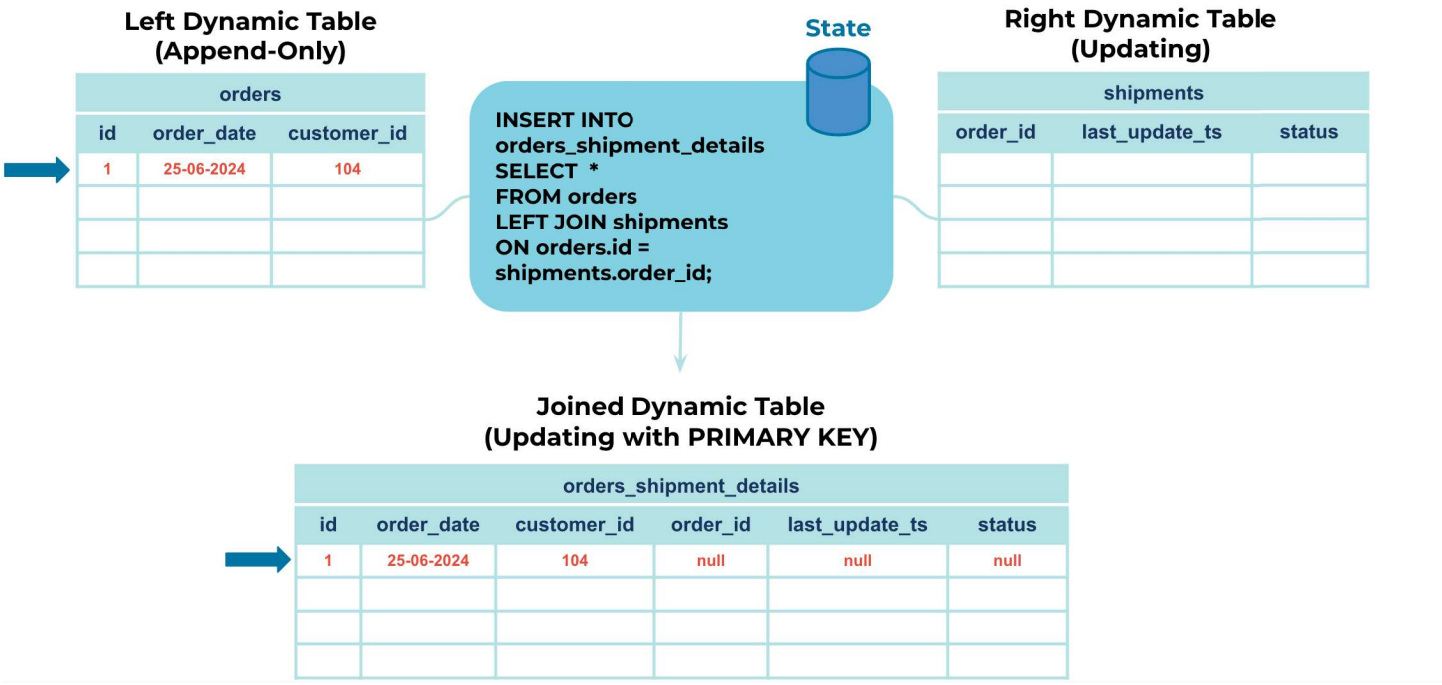
RIGHT OUTER JOIN: List all shipments including those not associated with an order (e.g., erroneous shipments)

```
SELECT *
FROM Orders
RIGHT JOIN Shipments
ON Orders.orderId = Shipments.orderId;
```

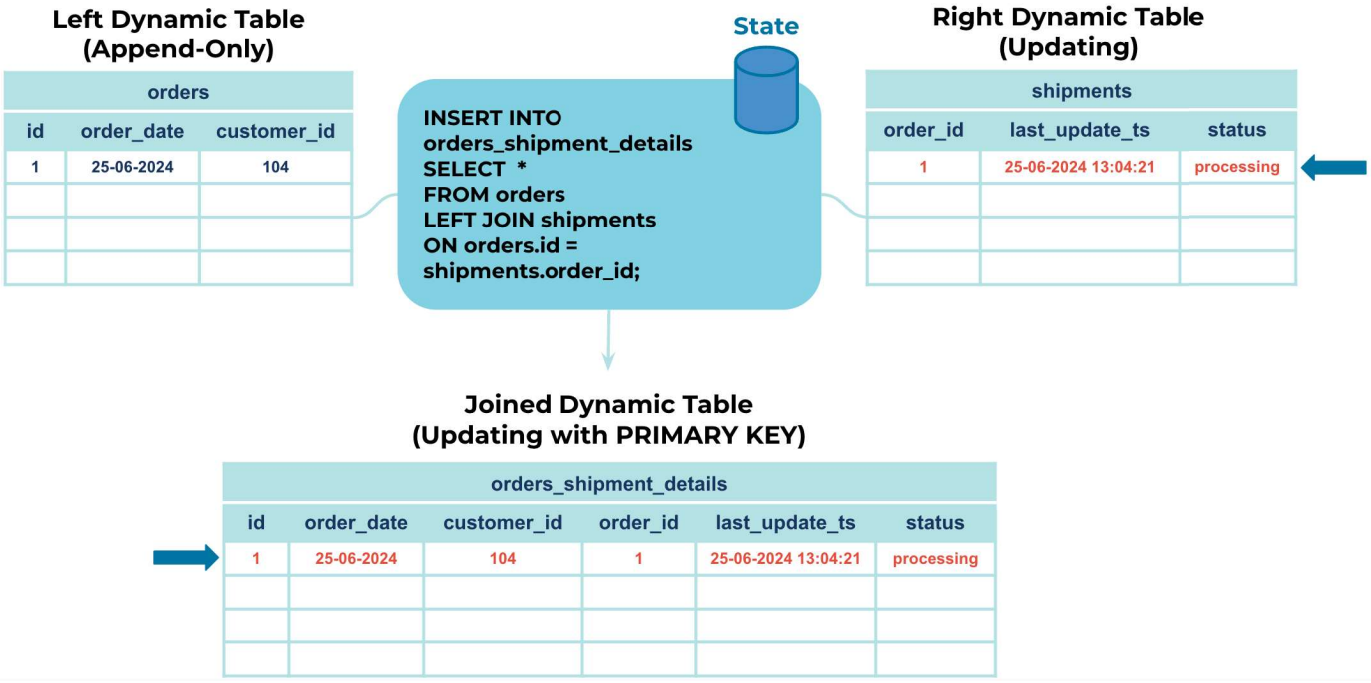
FULL OUTER JOIN: Combine order and shipment data, showing all records from both sides regardless of match

```
SELECT *
FROM Orders
FULL OUTER JOIN Shipments
ON Orders.orderId = Shipments.orderId;
```

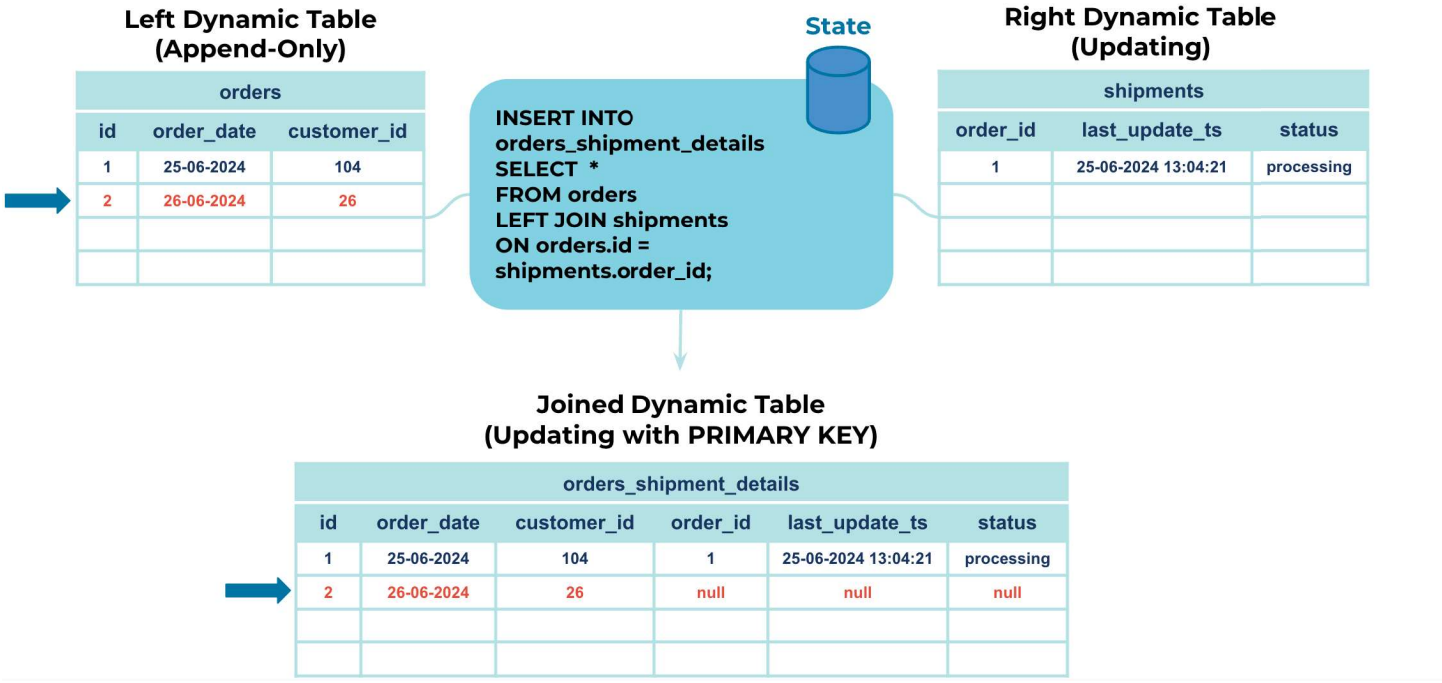
Left Outer Join: Example (I)



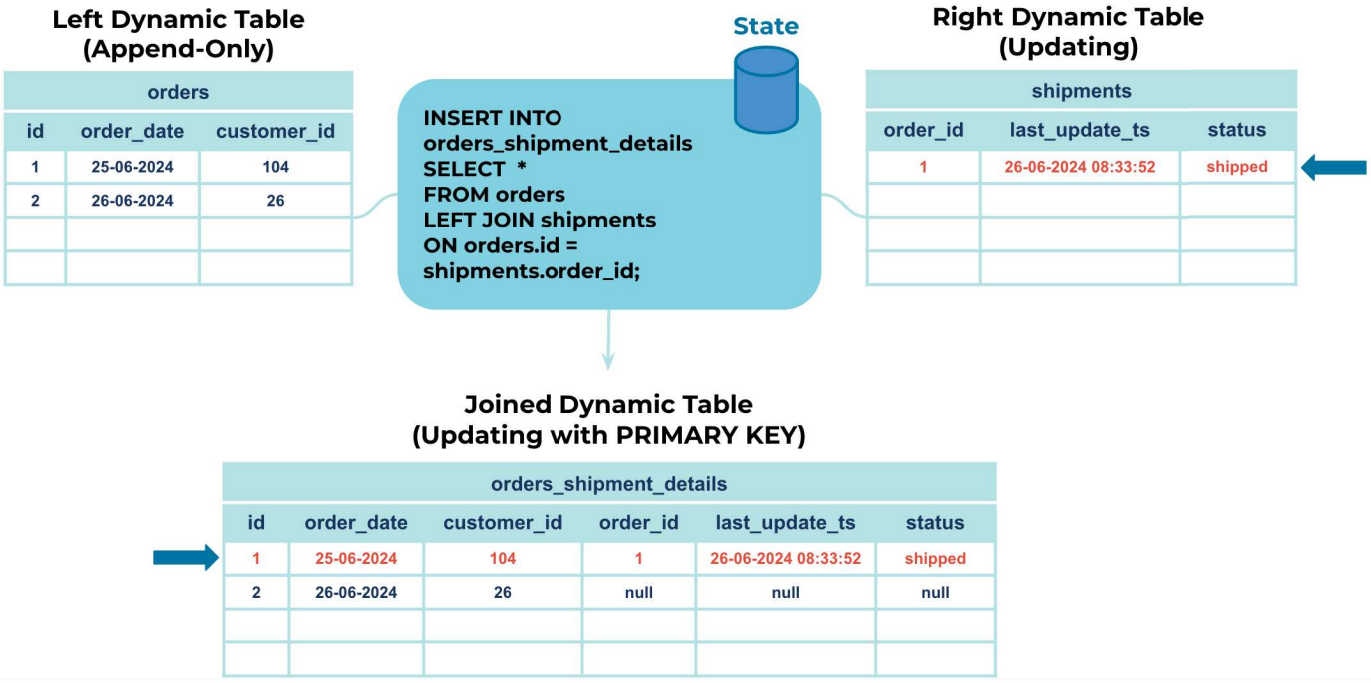
Left Outer Join: Example (II)



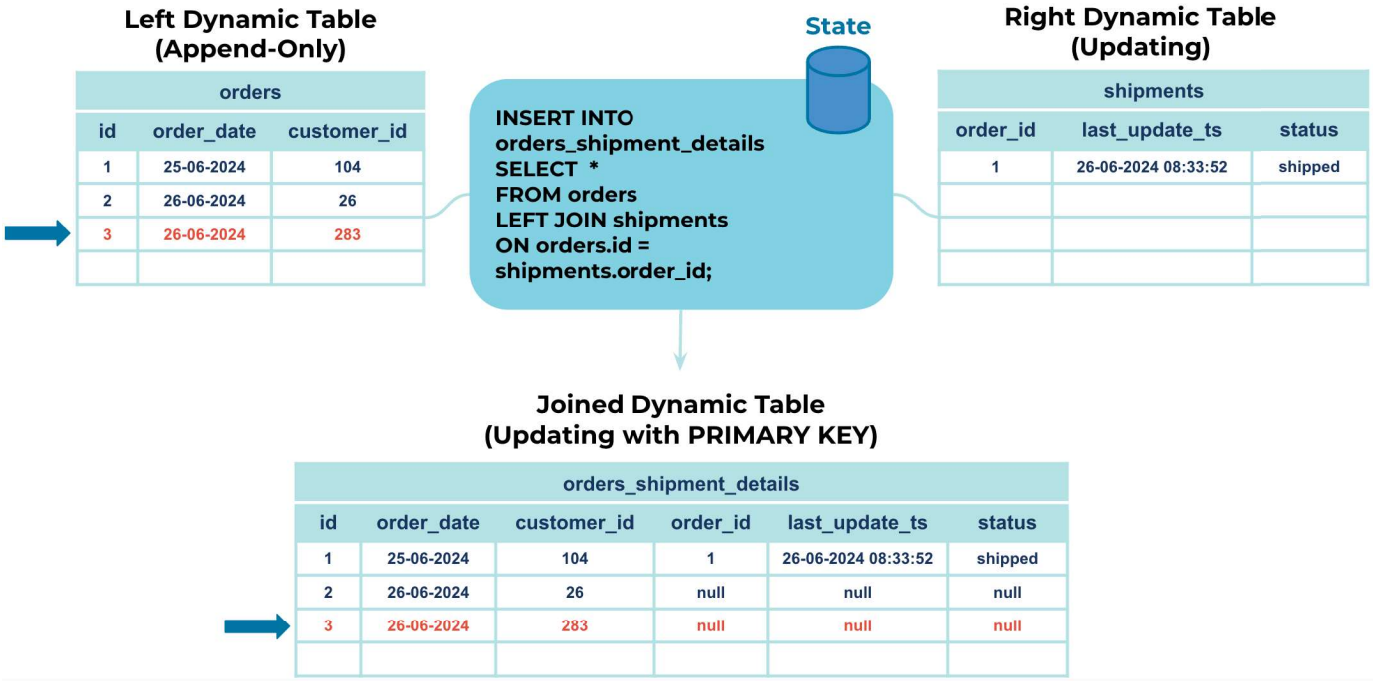
Left Outer Join: Example (III)



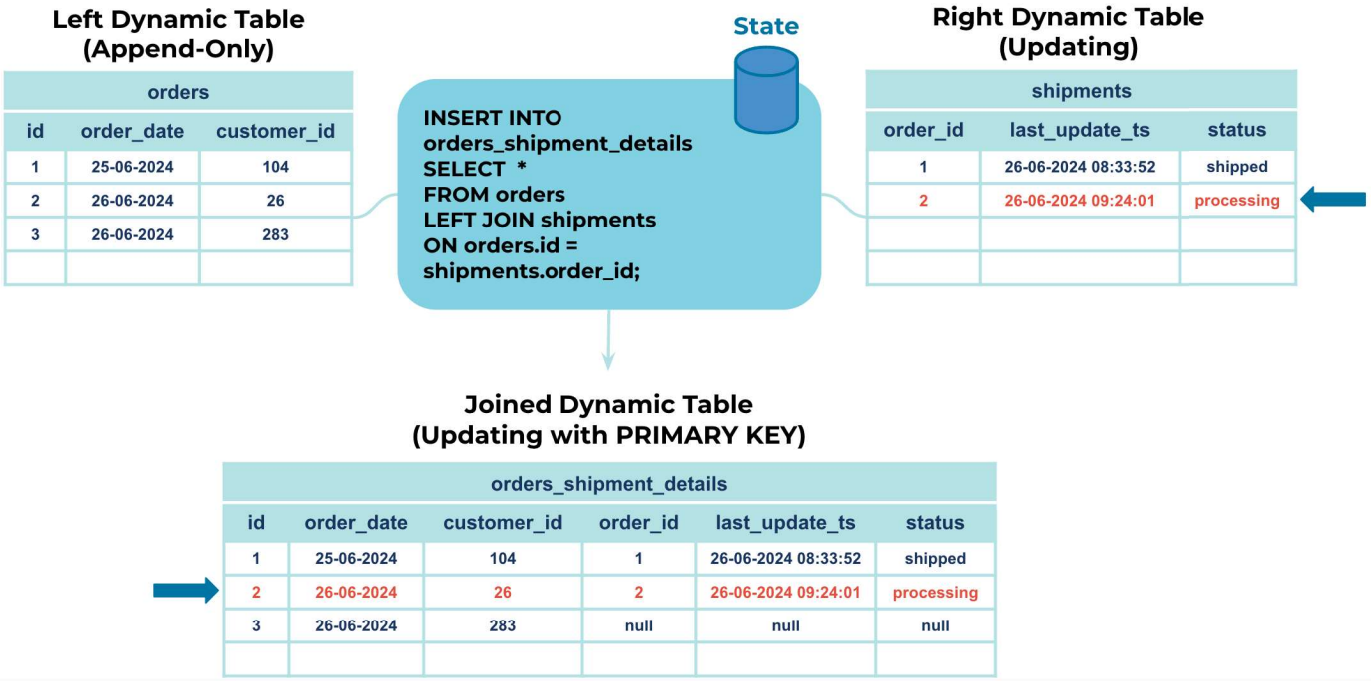
Left Outer Join: Example (IV)



Left Outer Join: Example (V)

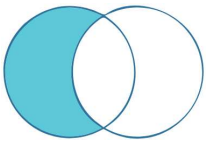


Left Outer Join: Example (VI)

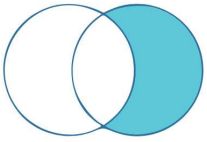


Activity: Exclusive Joins

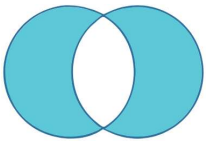
Check these JOIN types:



Left Exclusive



Right Exclusive

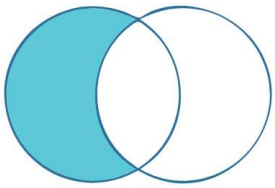


Full Outer Exclusive

Is it possible to get these results? If so, what would the queries be?

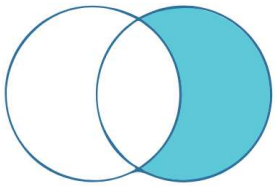
Activity: Exclusive Joins - Solution

Left Exclusive:



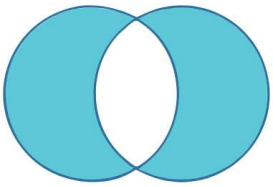
```
SELECT *
FROM A
LEFT JOIN B
ON A.key = B.key
WHERE B.key IS NULL;
```

Right Exclusive:



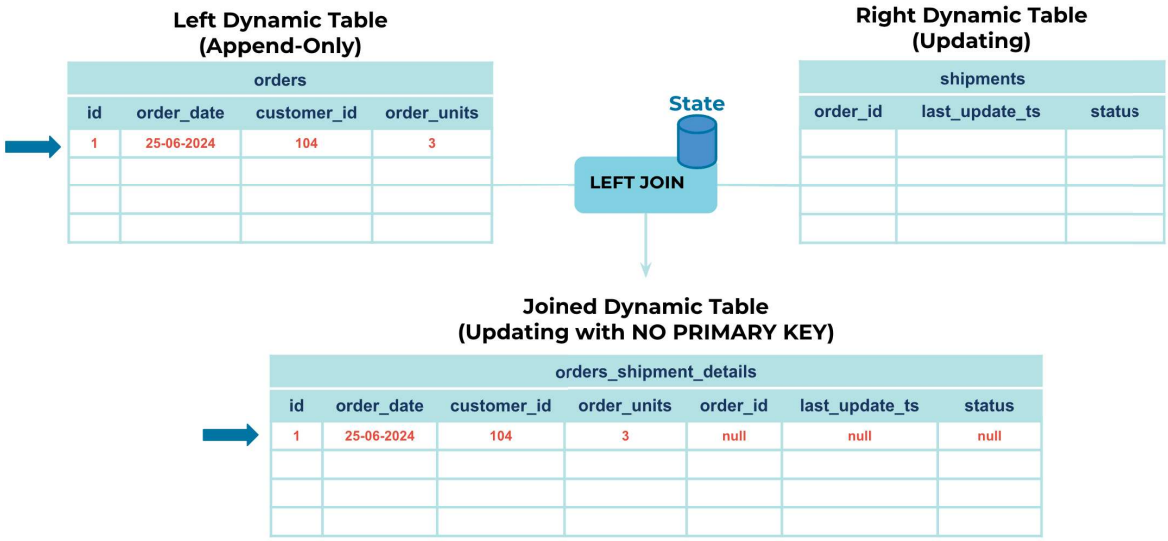
```
SELECT *
FROM A
RIGHT JOIN B
ON A.key = B.key
WHERE A.key IS NULL;
```

Full Outer Exclusive:

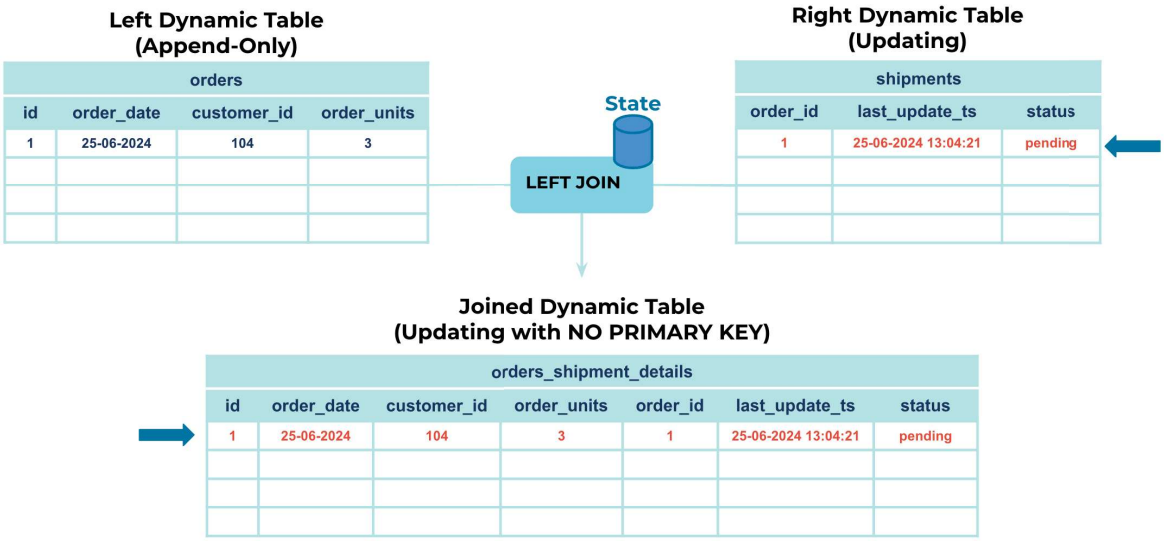


```
SELECT *
FROM A
FULL OUTER JOIN B
ON A.key = B.key
WHERE A.key IS NULL
OR B.key IS NULL;
```

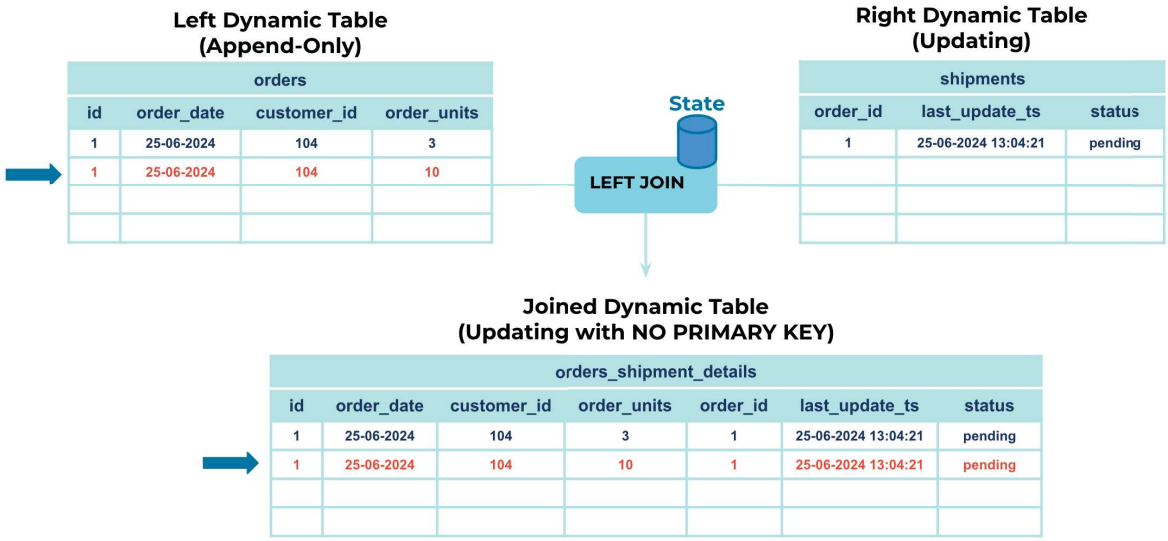
Left Outer Join with NO PRIMARY KEY (I)



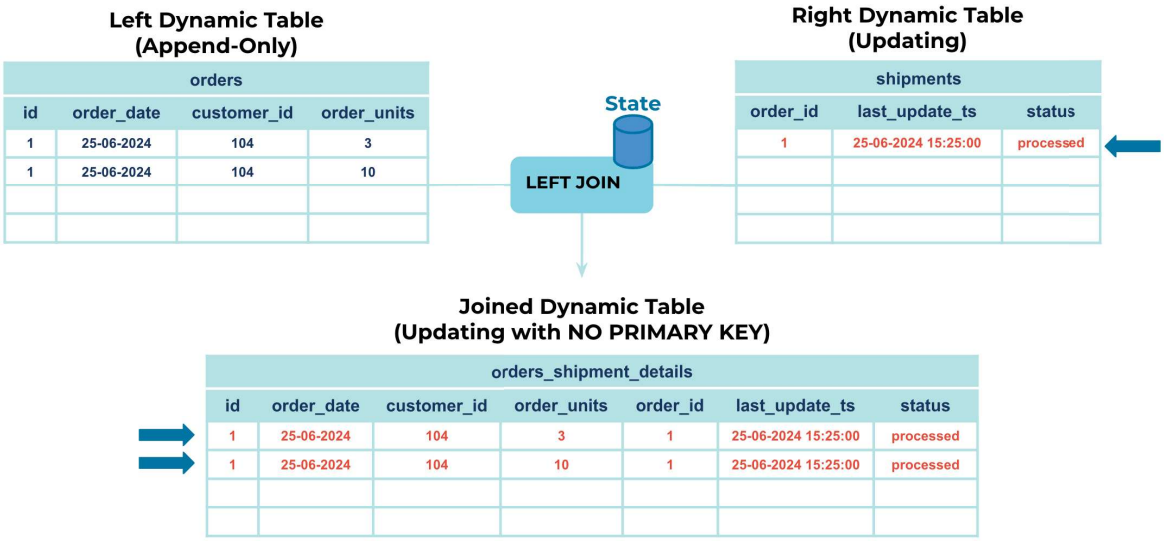
Left Outer Join with NO PRIMARY KEY (II)



Left Outer Join with NO PRIMARY KEY (III)



Left Outer Join with NO PRIMARY KEY (IV)



Lesson 06c

06c: Optimized Joins

Description

Understanding of how Interval and Temporal Joins work and how they provide state management optimizations.

Optimized Joins: Overview

In this lesson, we will talk about:

- Interval Joins
- Temporal Joins

Interval Join: Overview

- Joins records from two **append-only** tables
- Requires an **equality predicate** and a **time-bound condition**
- Efficient state management by removing outdated records
- Similar to `INNER JOIN` but with time constraint

Example

```
SELECT *
FROM orders AS o, shipments AS s
WHERE o.id = s.order_id
AND o.order_time BETWEEN s.ship_time - INTERVAL '1' HOUR AND s.ship_time;
```

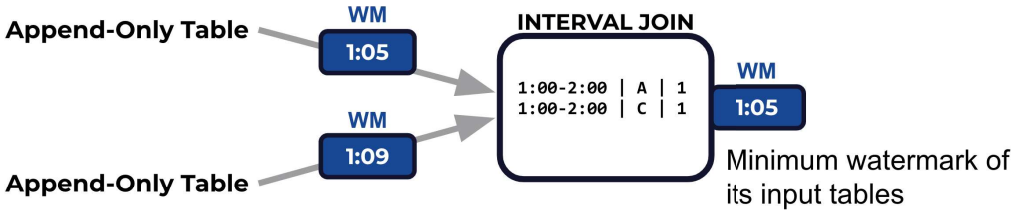
The keyword `JOIN` is not used in Interval Joins

Features:

- Time Interval Constraint: Joins records from two append-only tables where the time attributes of joined records are within a specified time interval.
- SQL Syntax: Requires an equality predicate and a specified time interval in the SQL syntax.
- Append-Only Tables: Tables involved in the interval join must be append-only; no rows can be updated.
- State Management: Relevant tails of both append-only tables are kept in Flink state during the join operation.
- Dynamic State Removal: Rows are immediately removed from the state when they are no longer relevant due to time constraints.
- Event Time Skew Consideration: Event time skew between streams can increase state size, as rows may be kept longer to ensure correct joins.

Interval Join: State Management

- State retention based on **Watermarks**
- As Watermarks advance, Flink deletes outdated data based on the time-bound conditions
- Time skew between streams can increase state size



Interval join only supports append-only tables with time attributes. Since time attributes are quasi-monotonic increasing, Flink can remove old values from its state without affecting the correctness of the result.

Watermarks play a critical role in state management for interval joins. Watermarks are used to track the progress of event time in the stream. As watermarks advance, Flink can determine which records have become outdated based on the join interval conditions and safely remove them from the state.

Event Time Skew Consideration: Event time skew between streams can increase state size, as rows may be kept longer to ensure correct joins

Temporal Join: Overview

- Joins records from an **append-only** table and a **versioned** table (upsert)
- Joins tables based on their states at **specific points in time**
- Requires an **equality predicate** and the `FOR SYSTEM_TIME AS OF` syntax
- Efficient state management by removing outdated records based on Watermarks
- Only INNER and LEFT joins

Syntax

```
SELECT [column_list]
FROM table1 [AS <alias1>]
[LEFT] JOIN table2 FOR SYSTEM_TIME AS OF table1.{ time_attribute }
ON table1.column = table2.column;
```

The result of the Temporal Join is an **append-only** table

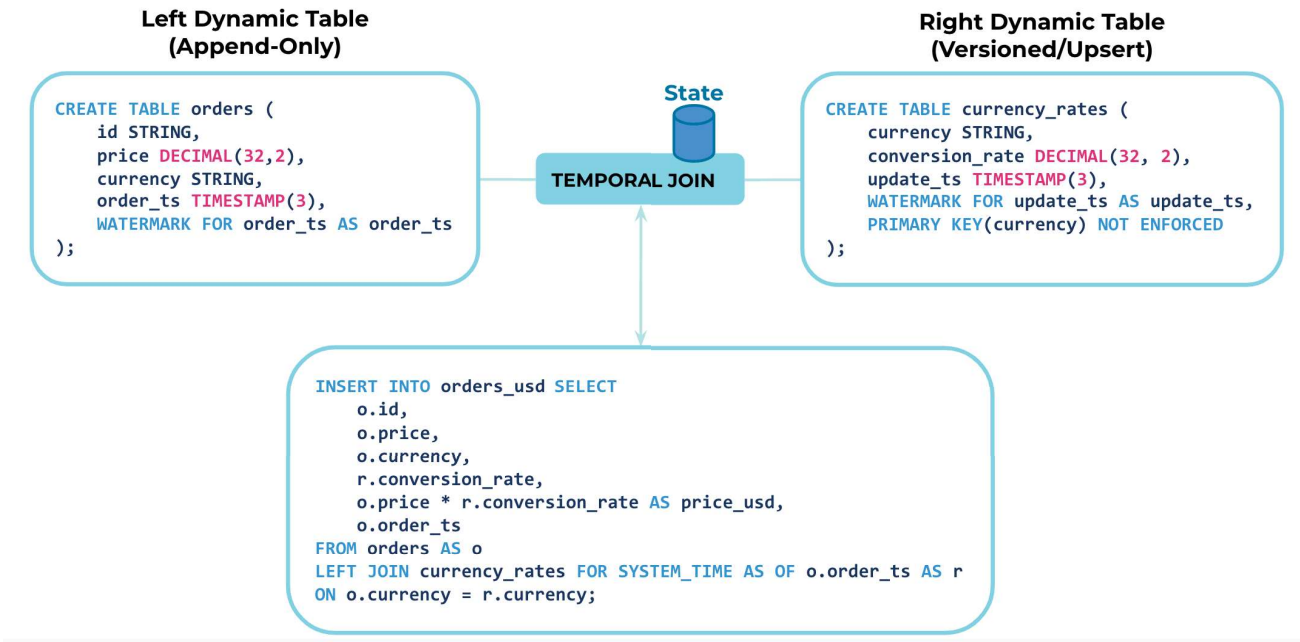
Temporal joins are crucial for scenarios where you need to understand the state of data at different points in time, such as financial data analysis or inventory tracking. Each record from an append-only table is matched with the corresponding version in the versioned table based on the record's timestamp.

Features:

- Time-based Correlation: Temporal joins correlate rows from one table (the primary table) with rows from another table (the versioned table) based on a time attribute. This allows for joining current records with the appropriate historical versions from the versioned table.
- `FOR SYSTEM_TIME AS OF`: This SQL syntax allows querying historical data as it was at a specific point in time. It ensures that each row from the primary table is joined with the corresponding versioned row from the versioned table at the specified time.
- Time Attributes: Temporal joins require time attributes (event time or processing time) in both tables to perform the join operation correctly.
- Data Consistency: By joining with historical versions of the data, temporal joins ensure that the join results are consistent with the state of the data at the specified time.
- Optimized State Management: Flink efficiently manages the state by removing outdated versions of the data that are no longer needed for the join, optimizing memory usage.

Temporal Join: Example Overview

Normalizing Order Prices in USD

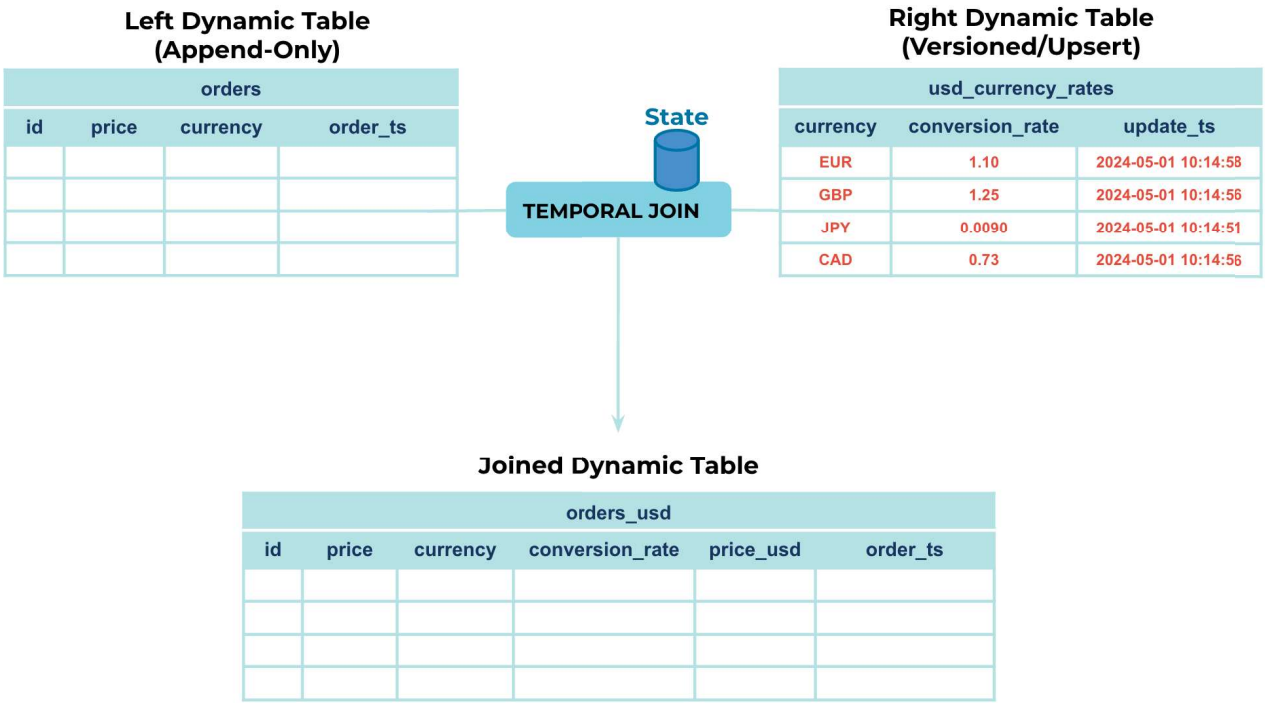


In this example, we are receiving different orders in various currencies in the orders table, and we want to normalize all the orders to a single currency (USD). To achieve this, we have another table, currency_rates, which contains the exact conversion rates at specific points in time.

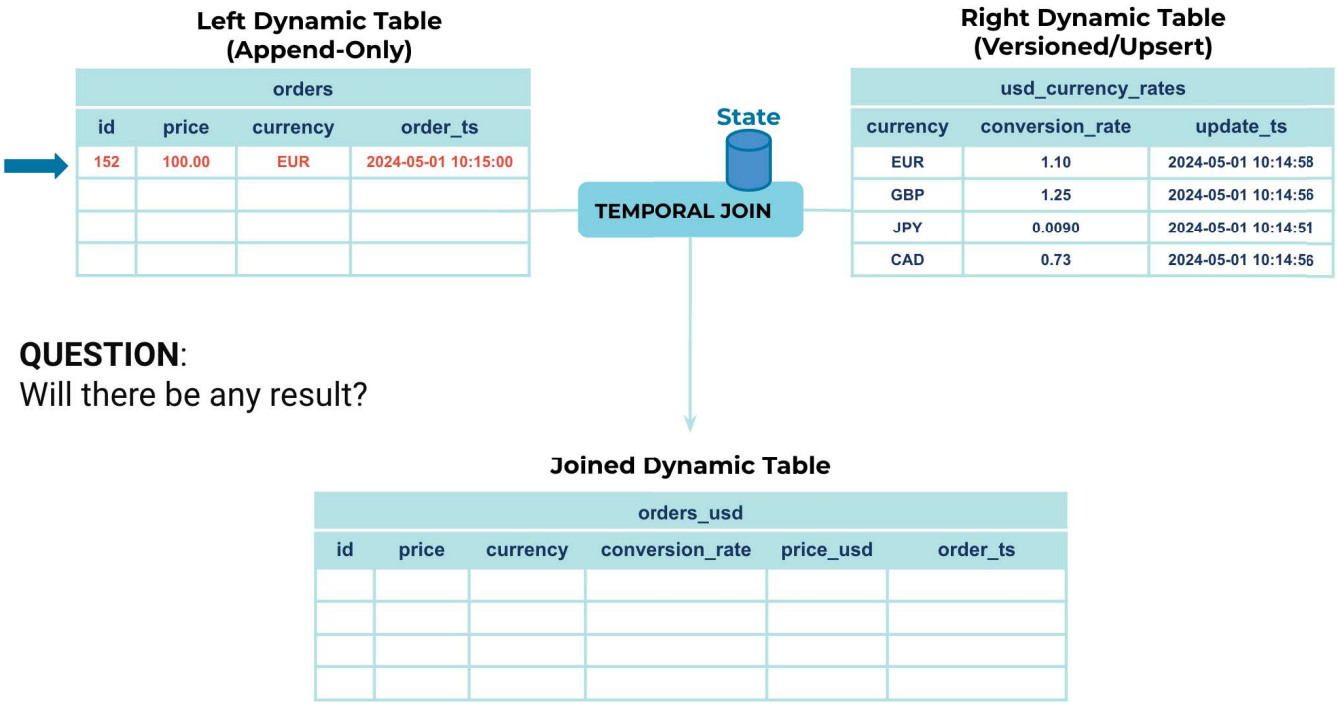
The objective of the join is to provide the order details along with the conversion rate at the time of the order and calculate the price in USD (price_usd) using the conversion rate.

Note that the watermarks of the input tables are simply `order_ts` and `update_ts` respectively.

Temporal Join: Example (I)



Temporal Join: Example (II)

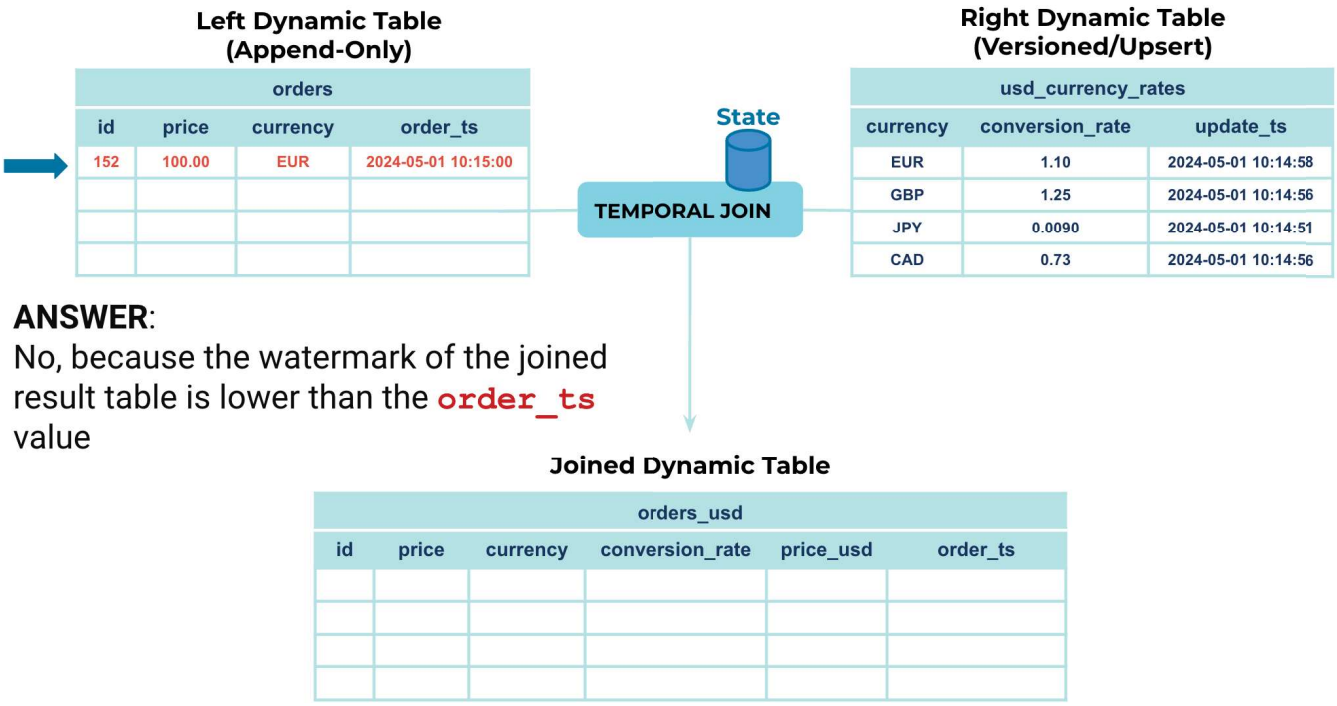


QUESTION:
Will there be any result?

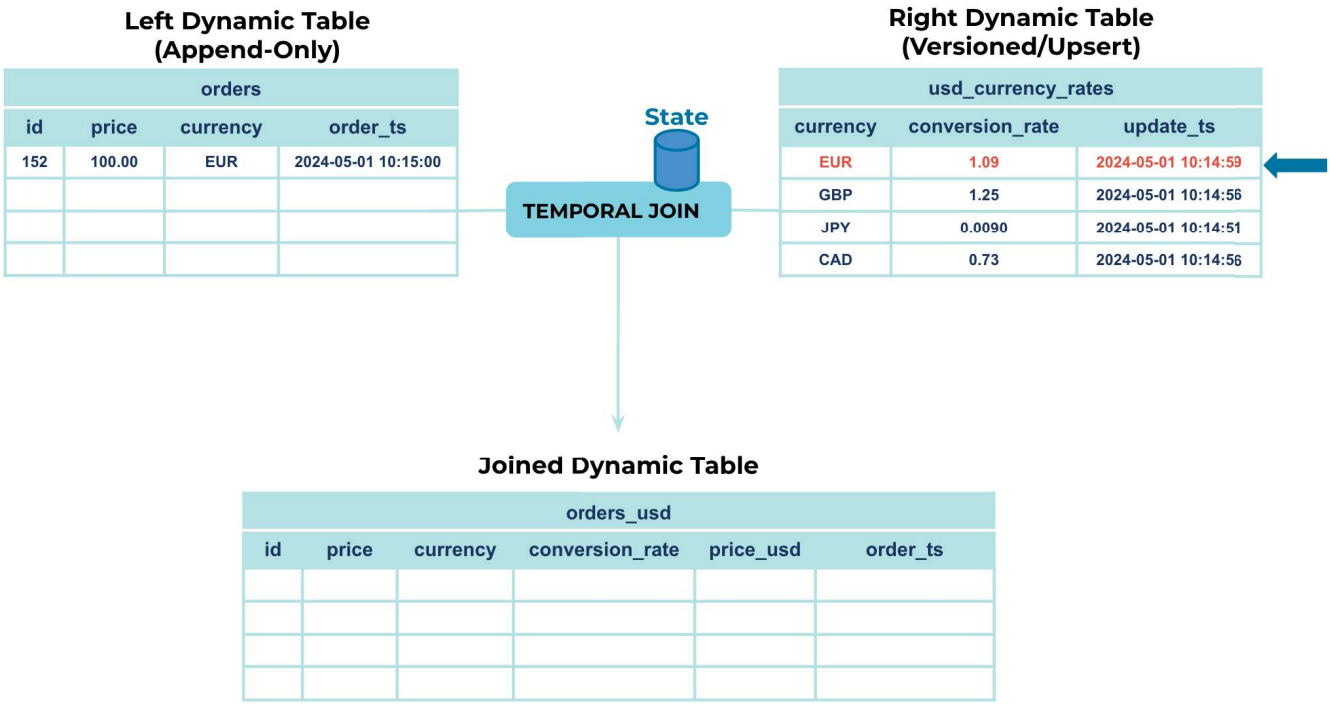
Joined Dynamic Table

orders_usd					
id	price	currency	conversion_rate	price_usd	order_ts

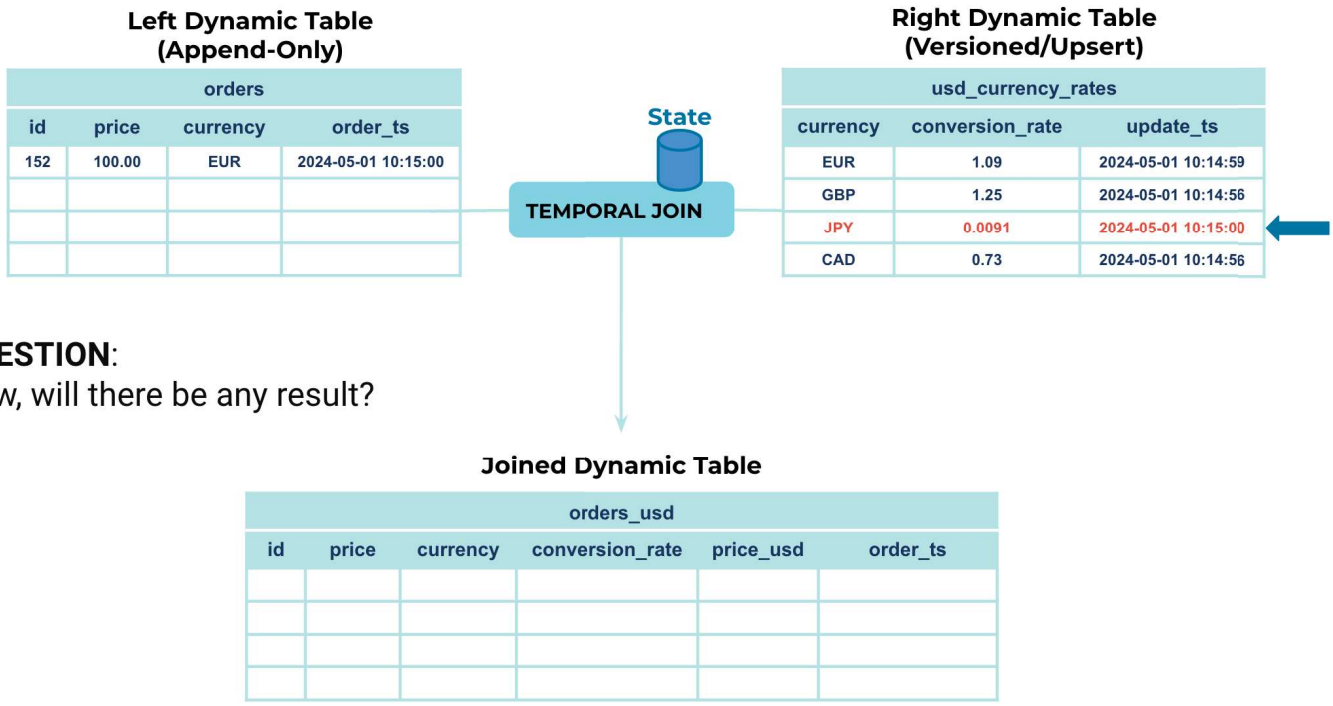
Temporal Join: Example (III)



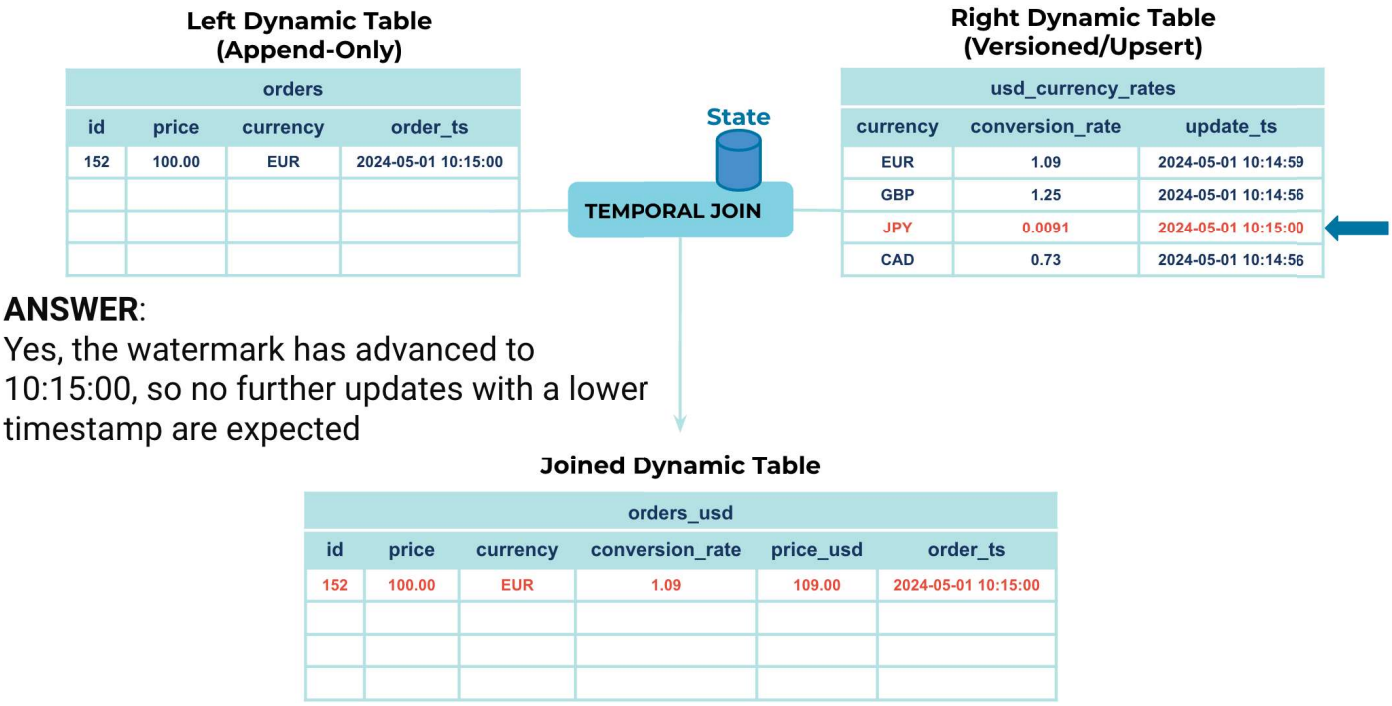
Temporal Join: Example (IV)



Temporal Join: Example (V)

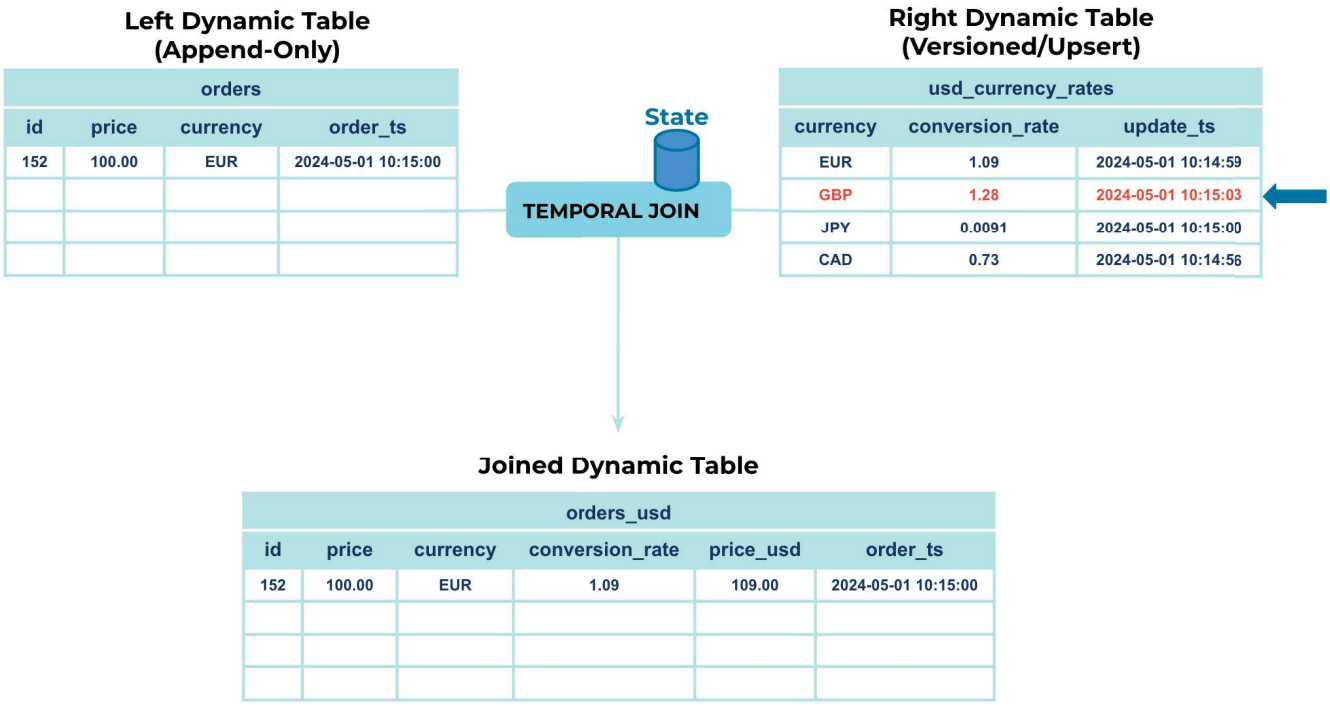


Temporal Join: Example (VI)

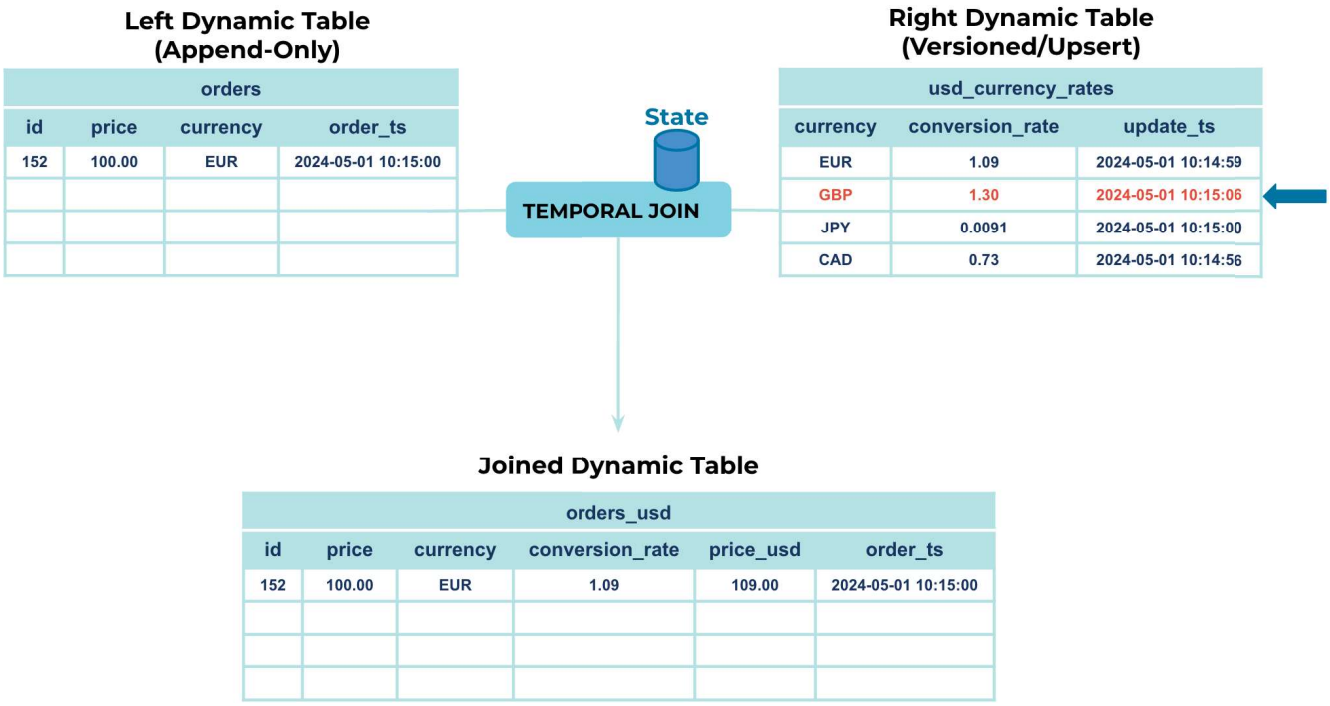


ANSWER:
Yes, the watermark has advanced to 10:15:00, so no further updates with a lower timestamp are expected

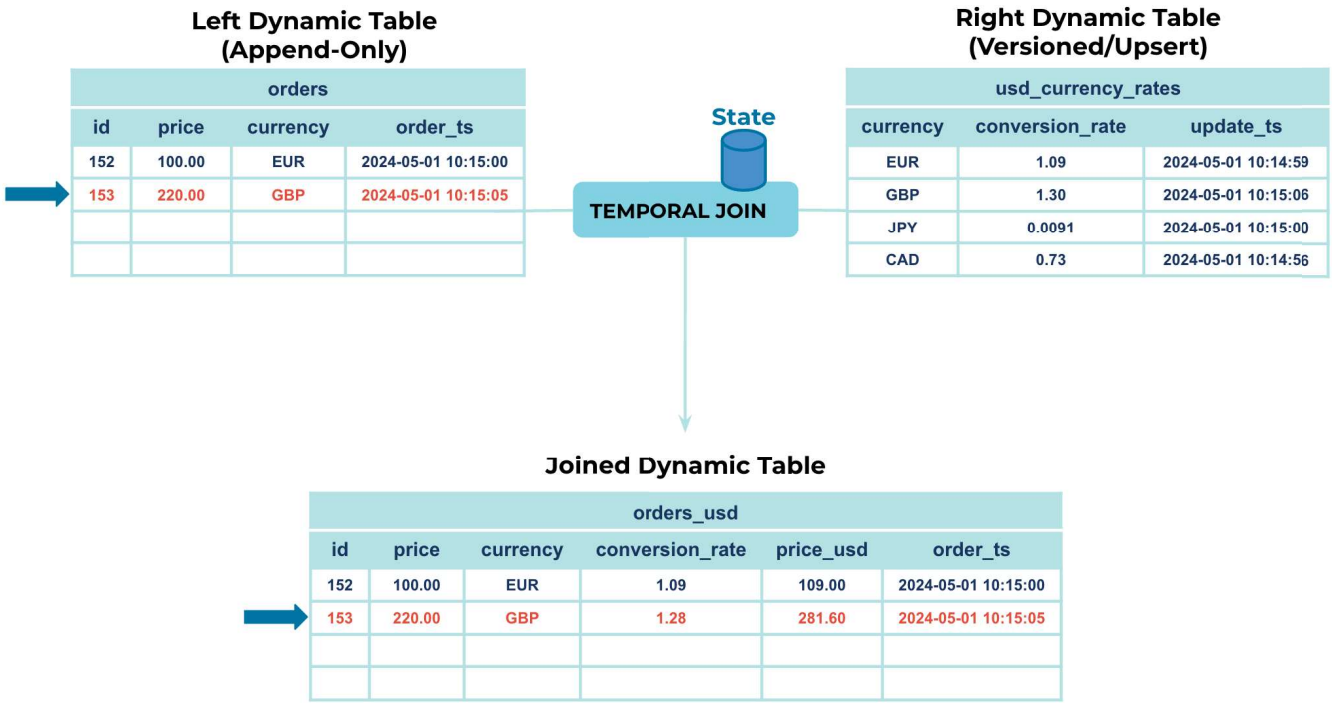
Temporal Join: Example (VII)



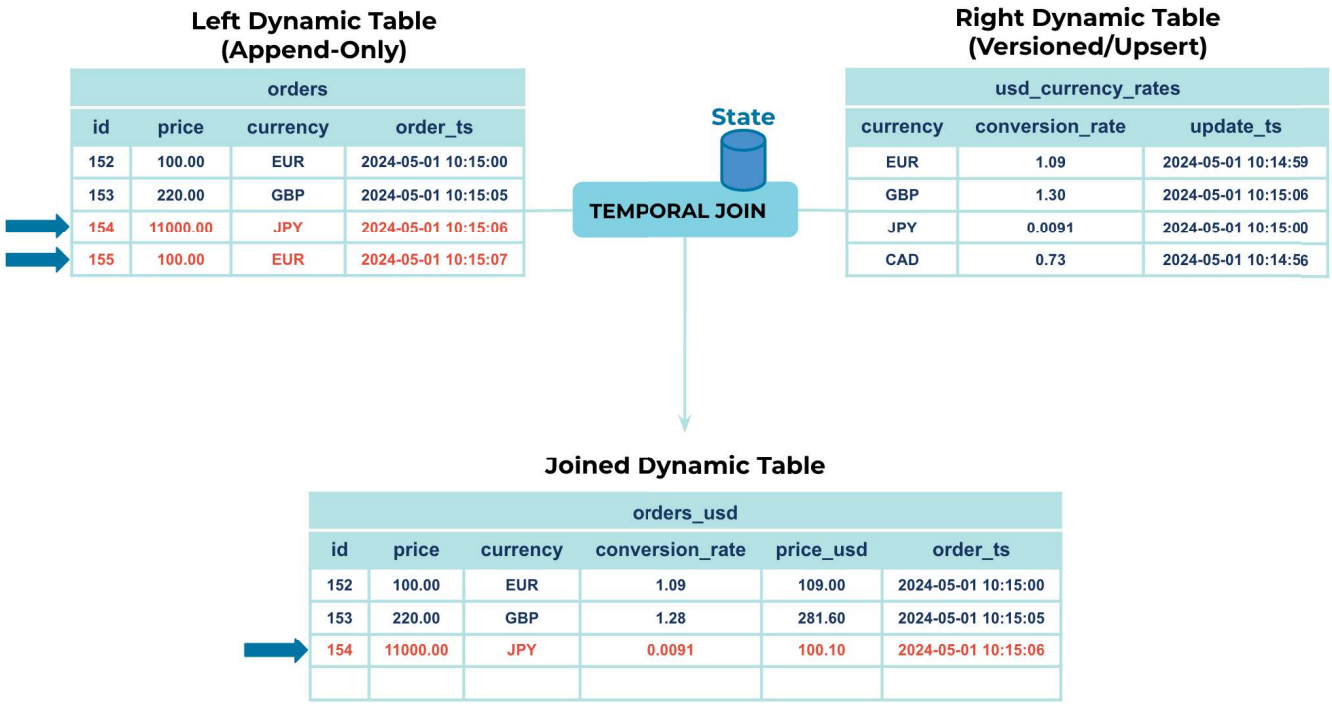
Temporal Join: Example (VIII)



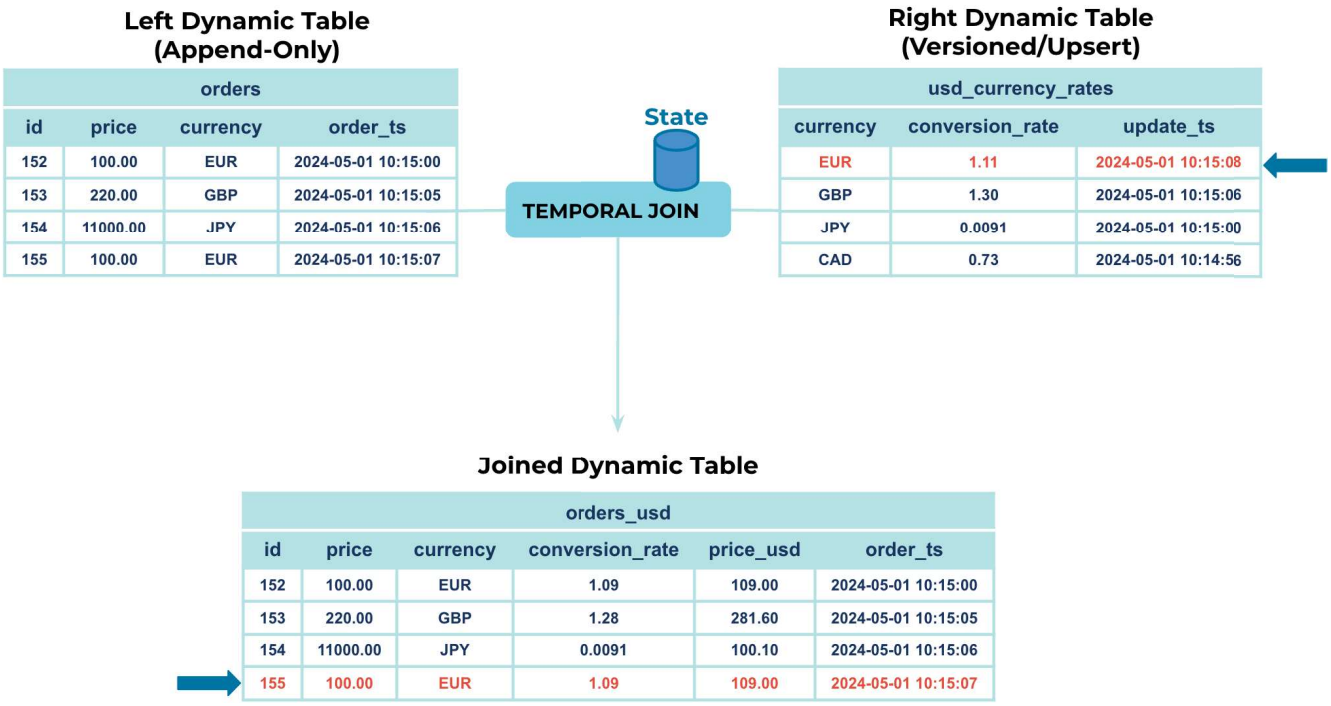
Temporal Join: Example (IX)



Temporal Join: Example (X)



Temporal Join: Example (XI)



Lesson 06d

06d: Window Joins

Description

Exploring Window Joins with examples and learning what SEMI and ANTI joins are.

Window Joins: Overview

- Joins the elements of two streams that share a **common key** and are in the **same window**:
  - `L.col = R.col`
  - `L.window_start = R.window_start`
  - `L.window_end = R.window_end`
- Supports INNER, LEFT, RIGHT and FULL OUTER joins
- Only emits final results at the end of the window
- Purges all intermediate state when no longer needed

A window join is a type of join operation in stream processing where data from two streams are joined within a specific time window. This is particularly useful for correlating events that happen within a certain timeframe.

For streaming queries, unlike other joins on continuous tables, window join does not emit intermediate results but only emits final results at the end of the window. Moreover, window join purge all intermediate state when no longer needed.

- Use Cases:
  - Real-time analytics on streaming data
  - Event correlation across multiple data streams
  - Monitoring and alerting systems

Lab Module 06: Exploring Various Types of Joins

Please work on **Lab Module 06: Exploring Various Types of Joins**.

Refer to the Exercise Guide.

Course Conclusion

CONFLUENT
Global Education

Course Contents

Now that you have completed this course, you should have the skills to:

- Understand the fundamentals of Apache Flink and its relevance to stream processing
- Write and execute Flink SQL queries on Confluent Cloud
- Differentiate between streaming and batch processing
- Work with dynamic tables and understand stream-table duality
- Manage time attributes and windows for effective stream processing
- Perform complex windowed aggregations in real-time with Flink SQL
- Utilize Flink SQL for efficient joining of streaming data
- Apply pattern-matching techniques to identify complex event sequences

Other Confluent Training Courses

- Confluent Developer Skills for Building Apache Kafka®
- Confluent Stream Processing Using Apache Kafka®Streams & ksqlDB
- Apache Kafka® Administration by Confluent
- Confluent Advanced Skills for Optimizing Apache Kafka®
- Setting Data in Motion with Confluent Cloud

For more details, see https://confluent.io/training

- **Setting Data in Motion with Confluent Cloud** covers:
  - Administer and secure their Confluent Cloud cluster
  - Manage their data and schemas in Confluent Cloud
  - Integrate their Confluent Cloud cluster with external data systems using Kafka Connect in Confluent Cloud
  - Create streaming applications using ksqlDB in Confluent Cloud
  - Link their Confluent Cloud cluster with their existing Kafka infrastructures (Kafka Connect, Schema Registry, ksqlDB servers)
  - Achieve VPC peering between Confluent Cloud and AWS/Azure/GCP
  - Monitor their Confluent Cloud cluster
- **Confluent Stream Processing Using Apache Kafka® Streams & ksqlDB** covers:
  - Identify common patterns and use cases for real-time stream processing
  - Understand the high level architecture of Kafka Streams
  - Write real-time applications with the Kafka Streams API to filter, transform, enrich, aggregate, and join data streams
  - Describe how ksqlDB combines the elastic, fault-tolerant, high-performance stream processing capabilities of Kafka Streams with the simplicity of a SQL-like syntax
  - Author ksqlDB queries that showcase its balance of power and simplicity
  - Test, secure, deploy, and monitor Kafka Streams applications and ksqlDB queries

- **Apache Kafka® Administration by Confluent** covers:
  - Data Durability in Kafka
  - Replication and log management
  - How to optimize Kafka performance
  - How to secure the Kafka cluster
  - Basic cluster management
  - Design principles for high availability
  - Inter-cluster design

- **Confluent Advanced Skills for Optimizing Apache Kafka**
  - Formulate the Apache Kafka® Confluent Platform specific needs of your organization
  - Monitor all essential aspects of your Confluent Platform
  - Tune the Confluent Platform according to your specific needs
  - Provide first level production support for your Confluent Platform

Confluent Certified Developer for Apache Kafka

Duration: 90 minutes

Qualifications: Solid understanding of Apache Kafka and Confluent products, and 6-to-9 months hands-on experience

Availability: Live, online, 24-hours a day!

Cost: $150

Register online: `www.confluent.io/certification`

Benefits:

- Recognition for your Confluent skills with an official credential
- Digital certificate and use of the official Confluent Certified Developer Associate logo
Exam Details:
- The exam is linked to the current Confluent Platform version
- Multiple choice questions
- 90 minutes
- Designed to validate professionals with a minimum of 6-to-9 months hands-on experience
- Remotely proctored on your computer
- Available globally in English

Confluent Certified Administrator for Apache Kafka

Duration: 90 minutes

Qualifications: Solid work foundation in Confluent products and 6-to-9 months hands-on experience

Availability: Live, online, 24-hours per day!

Cost: $150

Register online: `www.confluent.io/certification`

This course prepares you to manage a production-level Kafka environment, but does not guarantee success on the Confluent Certified Administrator Certification exam. We recommend running Kafka in Production for a few months and studying these materials thoroughly before attempting the exam.

Benefits:

- Recognition for your Confluent skills with an official credential
- Digital certificate and use of the official Confluent Certified Administrator Associate logo

Exam Details:

- The exam is linked to the current Confluent Platform version
- Multiple choice and multiple select questions
- 90 minutes
- Designed to validate professionals with a minimum of 6 - 12 months of Confluent experience
- Remotely proctored on your computer
- Available globally in English

Confluent Certified Operator for Confluent Cloud

Duration: 90 minutes

Qualifications: Solid work foundation in Confluent products and 6-to-9 months hands-on experience

Availability: Live, online, 24-hours per day!

Cost: $150

Register online: `www.confluent.io/certification`

We Appreciate Your Feedback!

Please complete the course survey now.

Your instructor will give you details on how to access the survey.

