
DATACOUCH
CREATING INFINITE POSSIBILITIES
DSAI
Data Science and AI Academy
Piramal
Finance
Flink Training for Real-Time Data Engineering
Working with Windows in Apache Flink

AGENDA

- Types: Inner, Outer, Interval, Temporal Joins
- Keyed vs. Broadcast joins
- Handling skewed joins and out-of-order events

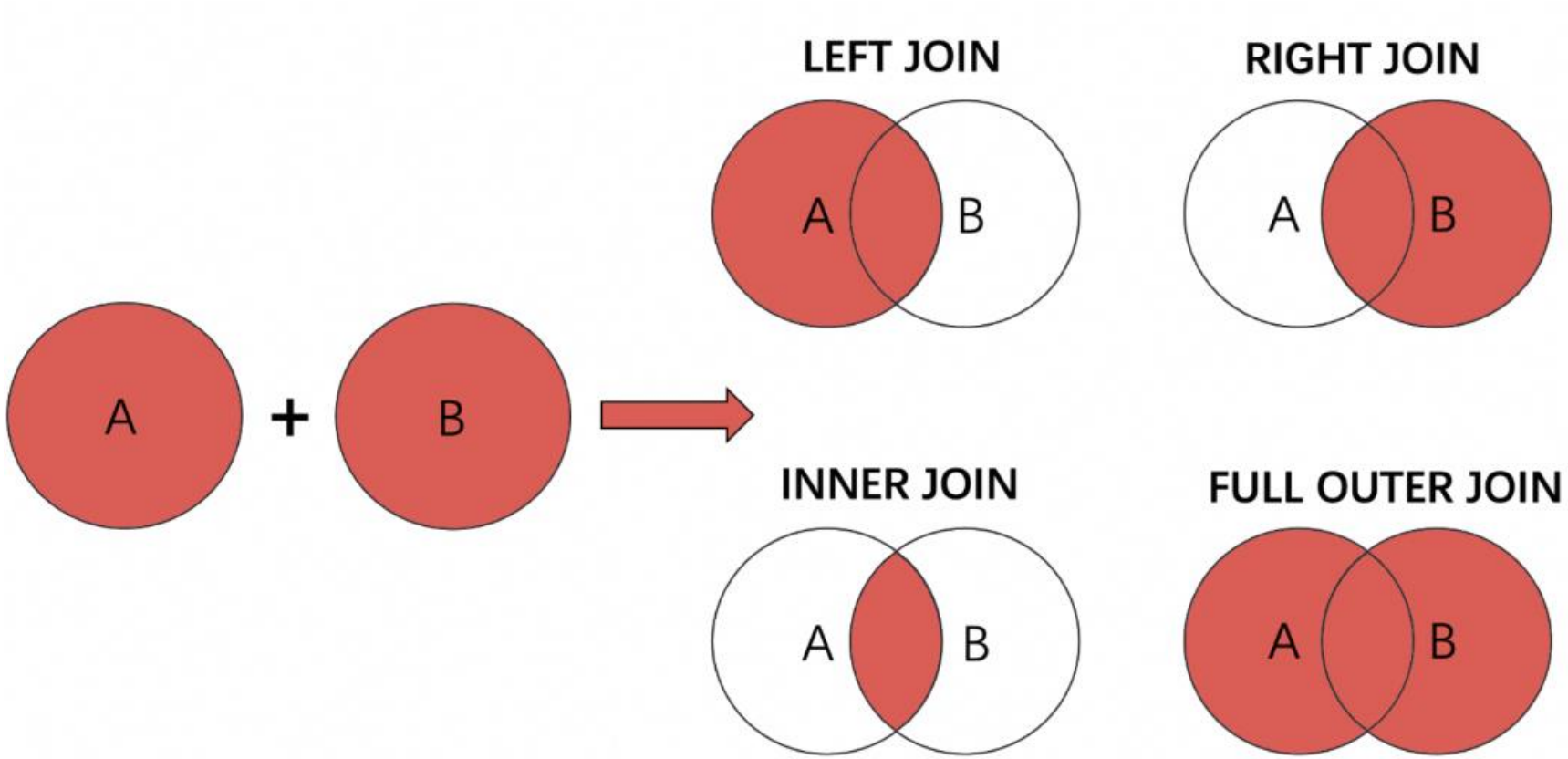
Joins in Apache Flink

Joins in Apache Flink support a variety of join types inner, outer, interval, and temporal combined with mechanics for keyed vs. broadcast joins, handling skew and late events, and powerful real-time enrichment use cases. Below are structured explanations, source links, and references to diagrams for instructional clarity.

Join Types in Flink

- Inner Join: Emits records matching the join condition in both streams/tables.
- Outer Join: LEFT, RIGHT, and FULL OUTER joins include unmatched records from one or both streams.
- Interval Join: Joins records from two keyed streams if their timestamps fall within a specified interval, useful for event correlation in time-based scenarios.
- Temporal Join (a.k.a. Lookup Join): Matches real-time or streaming records with point-in-time snapshots from external, slower-moving side tables (dimension tables)—essential for use cases like user/profile enrichment.

Joins in Apache Flink



INNER Equi-JOIN

Returns a simple Cartesian product restricted by the join condition. Currently, only equi-joins are supported, i.e., joins that have at least one conjunctive condition with an equality predicate. Arbitrary cross or theta joins are not supported.

```
SELECT *
FROM Orders
INNER JOIN Product
ON Orders.product_id = Product.id
```

OUTER Equi-JOIN

Returns all rows in the qualified Cartesian product (i.e., all combined rows that pass its join condition), plus one copy of each row in an outer table for which the join condition did not match with any row of the other table. Flink supports LEFT, RIGHT, and FULL outer joins. Currently, only equi-joins are supported, i.e., joins with at least one conjunctive condition with an equality predicate. Arbitrary cross or theta joins are not supported.

```
SELECT *
FROM Orders
LEFT JOIN Product
ON Orders.product_id = Product.id
```

```
SELECT *
FROM Orders
RIGHT JOIN Product
ON Orders.product_id = Product.id
```

```
SELECT *
FROM Orders
FULL OUTER JOIN Product
ON Orders.product_id = Product.id
```

Interval Joins

Returns a simple Cartesian product restricted by the join condition and a time constraint. An interval join requires at least one equi-join predicate and a join condition that bounds the time on both sides. Two appropriate range predicates can define such a condition (<, <=, >=, >), a BETWEEN predicate, or a single equality predicate that compares time attributes of the same type (i.e., processing time or event time) of both input tables.

```
SELECT *
FROM Orders o, Shipments s
WHERE o.id = s.order_id
AND o.order_time BETWEEN s.ship_time - INTERVAL '4' HOUR
AND s.ship_time
```

The following predicates are examples of valid interval join conditions:

- ltime = rtime
- ltime >= rtime AND ltime < rtime + INTERVAL '10' MINUTE
- ltime BETWEEN rtime - INTERVAL '10' SECOND AND rtime + INTERVAL '5' SECOND

Temporal Joins Overview

Temporal Table = table that evolves over time (dynamic table)

- Can store *history* (changelog) or *latest snapshot* (dimension table)

Types of Temporal Joins:

- **Event-Time Temporal Join** → matches data using event timestamps
- **Processing-Time Temporal Join** → uses the system's processing time

SQL Standard Syntax:

```
SELECT ...
FROM t1
JOIN t2 FOR SYSTEM_TIME AS
OF t1.rowtime
ON t1.key = t2.key
```

Keyed Join (Shuffle Join)

- Both datasets are **partitioned by key** across the cluster.
- Records with the same key end up on the same node.
- Good when **both datasets are large.**
- **Tradeoff:** Shuffling large amounts of data is **expensive.**

When to Use

- Both datasets are large.
- Keys are evenly distributed.

Broadcast Join

- One dataset is **small** enough to fit in memory.
- It is **replicated (broadcast)** to all nodes.
- Large dataset is scanned locally and joined with the broadcasted dataset.
- **Tradeoff:** Works only if the smaller dataset fits in memory.

When to Use

- One dataset is **very small**.
- Need **fast lookups** without shuffle.

Comparison

Feature	Keyed Join	Broadcast Join
Dataset Size	Large-Large	Large-Small
Data Movement	Heavy shuffle	Small broadcast
Memory Requirement	Moderate	Small dataset in memory
Performance	Slower for big data	Faster when conditions fit

Handling Skewed Joins

Data Skew Problem:

- When certain keys appear disproportionately often, they create **hotspots.**
- Leads to bottlenecks, uneven workload distribution, and degraded performance.

Flink Solutions:

- **Key Salting:** Append random “salt” values to hot keys to spread load across partitions.
- **Repartitioning:** Redistribute data to balance operator load.
- **Load-Aware Scheduling:** Some frameworks optimize by monitoring operator load.

ANY SUGGESTIONS?

DATACOUCH
CREATING INFINITE POSSIBILITIES

DSAI
Data Science and AI Academy

Piramal Finance


THANK YOU