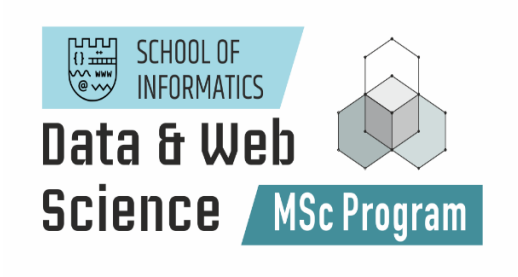



# Distributed Data Processing

Υποχρεωτική Ατομική Προγραμματιστική εργασία

## Technical Report

Ιπποκράτης Κοτσάνης
ΑΜ: 131

### ΘΕΜΑ

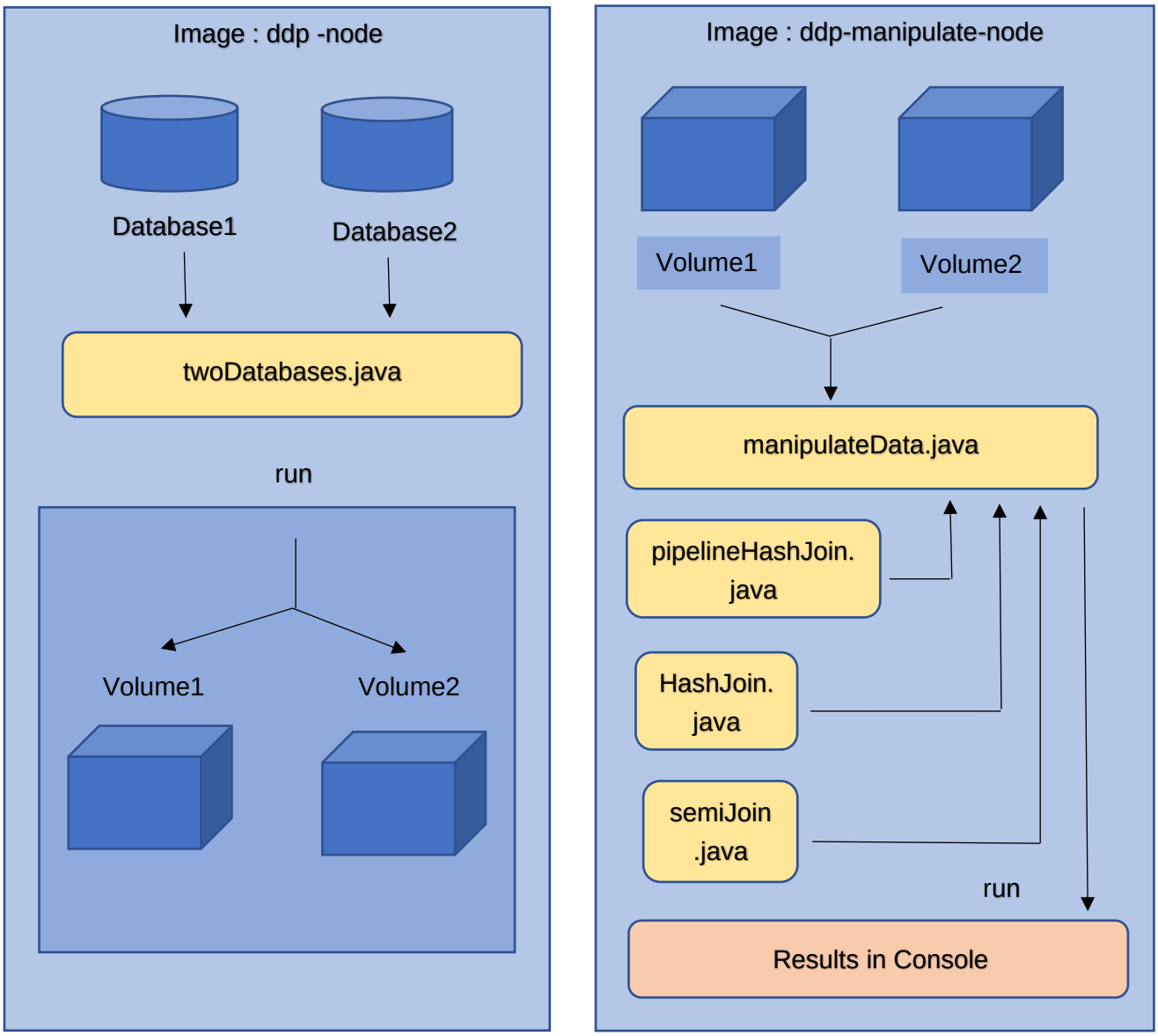
Να δημιουργηθεί ένα πρόγραμμα (κατά προτίμηση σε Java ή Scala), το οποίο να έχει τις παρακάτω λειτουργίες:

- Να γεμίζει με τυχαίες εγγραφές δύο πίνακες σε δύο διαφορετικές βάσεις δεδομένων τύπου Redis ή Ignite ή SQLite. Οι πίνακες να συνδέονται βάσει συνθήκης ισότητας και στα πεδία να υπάρχει πεδίο (τεχνητής) χρονοσφραγίδας.

- Να υλοποιεί, υποβάλλοντας SQL ερωτήματα στις δύο βάσεις δεδομένων τους παρακάτω αλγόριθμους σύνδεσης εξωτερικά από τις βάσεις δεδομένων: pipelined hash join, semi-join και προαιρετικά (θα έχει bonus) intersection bloom filter join. Σε όλες τις συνδέσεις, εκτός από την ισότητα στο πεδίο σύνδεσης, να ελέγχεται οι χρονοσφραγίδες να μην απέχουν πάνω από ένα όριο.

Θεσσαλονίκη, 2023

➢ Σχεδιάγραμμα υλοποίησης της εργασίας:




➢ Επεξήγηση Image 1:

Η εργασία υλοποιήθηκε σε **Java** και χρησιμοποιήθηκε η βιβλιοθήκη **jdbc** της Java προκειμένου να μπορέσουμε να εισάγουμε τα connectionString των δύο βάσεων δεδομένων.

1. Dockerfile:

Στο docker file χρησιμοποιούμε την OpenJDK και εγκαθιστούμε την βιβλιοθήκη jdbc. Επίσης έχουμε εισάγει δύο Volumes προκειμένου να αποθηκεύσουμε τις δύο βάσεις δεδομένων τις οποίες έχουμε γεμίσει με δεδομένα.

2. Επεξήγηση αρχείου twoDatabases.java:

Στο αρχείο αυτό ο κώδικας δημιουργεί και εκτελεί μια εφαρμογή Java που αλληλεπιδρά με δύο βάσεις δεδομένων SQLite. Αρχικά, η εφαρμογή συνδέεται στις δύο βάσεις δεδομένων ("database1.db" και "database2.db") χρησιμοποιώντας τον JDBC driver και το URL της κάθε μίας. Στη συνέχεια, δημιουργούνται τέσσερις πίνακες ("table1" και "table2") σε κάθε μία από τις δύο βάσεις δεδομένων, αν δεν υπάρχουν ήδη.

Οι πίνακες αδειάζουν, δηλαδή διαγράφονται όλες οι εγγραφές από αυτούς, για να σιγουρευτούμε ότι δεν υπάρχει καμία εγγραφή.

Χρησιμοποιώντας έναν γεννήτορα τυχαίων αριθμών (Random()), δημιουργούνται 50 εγγραφές με τυχαίες τιμές για τα πεδία "record_id", "value" και "timestamp".

Οι εγγραφές τοποθετούνται στους πίνακες με συγκεκριμένους κανόνες:

- 20 εγγραφές προστίθενται στον "table1" της πρώτης βάσης δεδομένων.
- 30 εγγραφές προστίθενται στον "table2" της πρώτης βάσης δεδομένων.
- 40 εγγραφές προστίθενται στον "table1" της δεύτερης βάσης δεδομένων.
- 50 εγγραφές προστίθενται στον "table2" της δεύτερης βάσης δεδομένων.

Τέλος, οι συνδέσεις με τις βάσεις δεδομένων κλείνουν και εμφανίζεται το μήνυμα "Program was run successfully!" στην κονσόλα.

➢ Επεξήγηση Image 2:

Στο image 2 έχουμε υλοποιήσει τέσσερις εφαρμογές όπου περιέχουν τις κλάσεις manipulateData, η οποία είναι και η κύρια κλάση και εκτελεί την main() μέθοδο, την κλάση HashJoin που κάνει extend την κλάση manipulateData και περιλαμβάνει μεθόδους για την υλοποίηση του HashJoin, την κλάση pipelineHashJoin, η οποία επίσης κάνει extend την κλάση manipulateData και περιλαμβάνει μεθόδους για την υλοποίηση του pipelineHashJoin και τέλος την κλάση semiJoin, η οποία επίσης κάνει extend την κλάση manipulateData και περιλαμβάνει μεθόδους για την υλοποίηση του semiJoin.

1. manipulateData.java, HashJoin.java, pipelineHashJoin.java, semiJoin.java:

Η εφαρμογή manipulateData.java συνδέεται σε δύο βάσεις δεδομένων SQLite ("database1.db" και "database2.db") και εκτελεί διάφορες λειτουργίες συνένωσης δεδομένων. Συγκεκριμένα, υλοποιεί τους αλγορίθμους Hash Join, Pipeline Hash Join και Semi Join.

Ο κώδικας αρχικά συνδέεται με τις δύο βάσεις δεδομένων, δημιουργεί ένα Map με κλειδιά τα ονόματα των πινάκων και τιμές λίστες που περιέχουν τα δεδομένα των

πινάκων από κάθε βάση δεδομένων. Έπειτα, εκτυπώνει τα δεδομένα αυτά στην κονσόλα.

Στη συνέχεια, εφαρμόζονται οι αλγόριθμοι συνένωσης δεδομένων:

- **Hash Join:** Υλοποιείται σε δύο εκδοχές, μια "lazy" και μια "eager". Στην "lazy" εκδοχή, οι έλεγχοι του timestamp γίνονται κατά την σύνδεση και όχι εκ των προτέρων. Επίσης γίνεται hash του μικρού πίνακα και γίνονται έλεγχοι με τον μεγάλο πίνακα και το αντίστροφο σε ξεχωριστή μέθοδο. Στην "eager" εκδοχή οι έλεγχοι του timestamp γίνονται εξαρχής. Επίσης, γίνεται hash του μικρού πίνακα και γίνονται έλεγχοι με τον μεγαλύτερο πίνακα και το αντίστροφο σε ξεχωριστή μέθοδο. Το αποτέλεσμα είναι οι εγγραφές που ικανοποιούν τη συνθήκη του hash join και του timestamp. Η υλοποίηση του hash join έγινε για εξάσκηση!

- **Pipeline Hash Join**: Η διαφορά με τον Hash Join είναι ότι δεν έχει ακολουθιακά τις 2 φάσεις (build & process), αλλά χρησιμοποιεί pipeline κάνοντας hash και τους 2 πίνακες. Ο αλγόριθμος Pipeline Hash Join υλοποιείται σε δύο εκδοχές, μια "lazy" και μια "eager". Στην εκδοχή "lazy", δημιουργείται ένα hash table για κάθε πίνακα και γίνεται έλεγχος για τις εγγραφές που έχουν την ίδια τιμή και ικανοποιούν το χρονικό περιθώριο του timestamp. Στην εκδοχή "eager", δημιουργείται ένα hash table για κάθε πίνακα, αλλά πρώτα γίνεται έλεγχος για τη συνθήκη της χρονικής σφραγίδας (timestamp) και έπειτα πραγματοποιείται η συνένωση. Πιο συγκεκριμένα:

1. Η μέθοδος performPipelineHashJoinLazy υλοποιεί το Pipeline Hash Join με Lazy «σκεπτικό». Χρησιμοποιεί μία λίστα με όνομα joinedDataLazy για να αποθηκεύσει τα συζευγμένα αποτελέσματα.
2. Αρχικά, αρχικοποιούνται δύο κατακερματισμένοι πίνακες (hash tables) για τα δεδομένα των πινάκων 1 και 2. Οι πίνακες αυτοί αναπαρίστανται με χρήση των Map hashTable1 και hashTable2, αντίστοιχα. Ο κατακερματισμένος πίνακας αποτελείται από λίστες συμβολοσειρών, με κλειδί την τιμή του κατακερματισμού και τιμή τις αντίστοιχες εγγραφές του πίνακα.
3. Τα δεδομένα των πινάκων 1 και 2 προστίθενται σε έναν κοινό πίνακα με όνομα mergeTable με τη σειρά που εμφανίζονται στις αρχικές λίστες. Ο πίνακας αυτός ανακατεύεται με χρήση ενός τυχαίου αριθμού (seed) για να αλλάξει η σειρά των εγγραφών. Αυτό αποσκοπεί για να προσομοιώσουμε ένα streaming περιβάλλον και να εκμεταλλευτούμε το πλεονέκτημα του pipeline hash join, στο οποίο οι πλειάδες κάθε φορά μπορούν να έρχονται από οποιονδήποτε πίνακα.
4. Σε αυτό το σημείο, ξεκινά η lazy υλοποίηση. Για κάθε (εισερχόμενη) εγγραφή στον mergeTable, ελέγχεται αν είναι εγγραφή του πίνακα 1 (table1Data) ή εγγραφή του πίνακα 2 (table2Data). Αν είναι εγγραφή του πίνακα 1 ελέγχουμε αν το hash table του πίνακα 1 (hashTable1) έχει ως κλειδί το value της εγγραφής. Αν δεν το έχει τότε εισάγουμε το value ως κλειδί και τιμή την εγγραφή, ενώ στην περίπτωση που υπάρχει ήδη το value της εγγραφής ως κλειδί, απλά

προσθέτουμε την εγγραφή ως μια νέα τιμή στο hashTable1Data. Στην συνέχεια θα αναζητήσουμε όλες τις εγγραφές του hashTable2 που έχουν κλειδί το value της εγγραφής που εξετάζουμε και θα τις εισάγουμε στην λίστα matchingRows2. Τέλος διατρέχουμε την λίστα των matching rows και ελέγχουμε αν ικανοποιείται η συνθήκη του timestamp δηλαδή να μην απέχουν πάνω από 10 δευτερόλεπτα. Αν ικανοποιείται εισάγουμε τις joined εγγραφές στην λίστα αποτελεσμάτων, την joinedDataLazy. Με το ίδιο ακριβώς σκεπτικό γίνεται η υλοποίηση αν ανήκει η εγγραφή που έρχεται στον πίνακα 2 (table2Data).

1. Η μέθοδος performPipelineHashJoinEager υλοποιεί τον αλγόριθμο Pipeline Hash Join με Eager «σκεπτικό». Η λειτουργία της είναι αρκετά παρόμοια με τη μέθοδο performPipelineHashJoinLazy, με τη διαφορά ότι εδώ γίνεται έλεγχος χρονικής σφραγίδας πριν από την προσθήκη των εγγραφών στους κατακερματισμένους πίνακες hashTable1 και hashTable2. Αυτός ο έλεγχος εξασφαλίζει ότι οι εγγραφές που δεν πληρούν το κριτήριο χρονικής διαφοράς δεν προστίθενται στους κατακερματισμένους πίνακες.
2. Αρχικά, αρχικοποιούνται δύο κατακερματισμένοι πίνακες (hash tables) για τα δεδομένα των πινάκων 1 και 2. Οι πίνακες αυτοί αναπαρίστανται με χρήση των Map hashTable1 και hashTable2, αντίστοιχα. Ο κατακερματισμένος πίνακας αποτελείται από λίστες συμβολοσειρών, με κλειδί την τιμή του κατακερματισμού και τιμή τις αντίστοιχες εγγραφές του πίνακα.
3. Τα δεδομένα των πινάκων 1 και 2 προστίθενται σε έναν κοινό πίνακα με όνομα mergeTable με τη σειρά που εμφανίζονται στις αρχικές λίστες. Ο πίνακας αυτός ανακατεύεται με χρήση ενός τυχαίου αριθμού (seed) για να αλλάξει η σειρά των εγγραφών. Αυτό αποσκοπεί για να προσομοιώσουμε ένα streaming περιβάλλον και να εκμεταλλευτούμε το πλεονέκτημα του pipeline hash join, στο οποίο οι πλειάδες κάθε φορά μπορούν να έρχονται από οποιονδήποτε πίνακα.
4. Σε αυτό το σημείο, ξεκινά η eager υλοποίηση. Για κάθε (εισερχόμενη) εγγραφή στον mergeTable, ελέγχεται αν είναι εγγραφή του πίνακα 1 (table1Data) ή εγγραφή του πίνακα 2 (table2Data). Αν είναι εγγραφή του πίνακα 1, διατρέχουμε όλες τις εγγραφές του πίνακα 2 (table2Data) και ελέγχουμε αν ικανοποιείται ο χρονικός περιορισμός των 10 δευτερολέπτων στο timestamp. Αν ικανοποιείται ελέγχουμε αν το hash table του πίνακα 1 (hashTable1) έχει ως κλειδί το value της εγγραφής. Αν δεν το έχει τότε εισάγουμε το value ως κλειδί και τιμή την εγγραφή, ενώ στην περίπτωση που υπάρχει ήδη το value της εγγραφής ως κλειδί, απλά προσθέτουμε την εγγραφή ως μια νέα τιμή στο hashTable1Data. Στην συνέχεια θα αναζητήσουμε όλες τις εγγραφές του hashTable2 που έχουν κλειδί το value της εγγραφής που εξετάζουμε και θα τις εισάγουμε στην λίστα matchingRows2 και αντίστοιχα όλες τις εγγραφές του hashTable1 που έχουν κλειδί το value της εγγραφής που εξετάζουμε και θα τις εισάγουμε στην λίστα matchingRows1 . Τέλος διατρέχουμε τις λίστες των matching rows και εισάγουμε τις joined εγγραφές στην λίστα αποτελεσμάτων, την joinedDataEager. Με το ίδιο ακριβώς σκεπτικό γίνεται η υλοποίηση αν ανήκει η εγγραφή που έρχεται στον πίνακα 2 (table2Data).
5. Σε αυτό το σημείο να σημειωθεί ότι επειδή τα hash tables είναι άδεια όσο αυξάνονται οι προσπελάσεις άρα και τα δεδομένα μέσα στα hash tables,

προκύπτουν περιττές προσπελάσεις και διπλότυπα, πράγμα που μας δείχνει ότι το eager σκεπτικό δεν είναι optimum. Για τον λόγο αυτό έχουμε αλλάξει την λίστα joinedDataEager σε Set για να αποφύγουμε τα διπλότυπα και να αξιολογήσουμε τα αποτελέσματα της υλοποίησης πιο εύκολα.

- **Semi Join**: Υλοποιείται επίσης σε δύο εκδοχές, μια "lazy" και μια "eager". Αυτός ο αλγόριθμος επιστρέφει μόνο τις εγγραφές από τον πρώτο πίνακα (table1) που υπάρχουν και στον δεύτερο πίνακα (table2). Στην "lazy" εκδοχή, γίνεται έλεγχος για κάθε εγγραφή του πρώτου πίνακα με τον δεύτερο πίνακα, ελέγχοντας κατά την σύνδεση το timestamp. Στην "eager" εκδοχή, πρώτα γίνεται ο έλεγχος του timestamp και μετά έλεγχος για κάθε εγγραφή του πρώτου πίνακα με τον δεύτερο πίνακα. Πιο συγκεκριμένα:

1. Η μέθοδος performTable1DataSemiJoinTable2Data υλοποιεί το Semi Join με Lazy «σκεπτικό». Χρησιμοποιεί μία λίστα με όνομα semiJoinedTable1Data για να αποθηκεύσει τα semi joined αποτελέσματα του table1Data.
2. Αρχικά, αρχικοποιούνται μία βοηθητική λίστα η ArrayListTable2 και ένα set που ονομάζεται Unique Values.
3. Στην συνέχεια θέλουμε να πάρουμε την προβολή του table2Data. Επομένως διατρέχουμε όλες τις εγγραφές του table2Data και για κάθε εγγραφή κρατάμε την τιμή του value (value2). Αν το set uniqueValues δεν περιέχει το value το προσθέτει και προσθέτει και την εγγραφή στην βοηθητική λίστα. Ο σκοπός του uniqueValues είναι απαλοιφή διπλοτύπων στο value, ενώ ο σκοπός της βοηθητικής λίστας είναι να κρατήσουμε όλη την μοναδική εγγραφή και όχι μόνο το value της μιας και θα πρέπει να γίνει έλεγχος του timestamp στην συνέχεια με τον άλλο πίνακα.
4. Σε αυτό το σημείο, ξεκινά η lazy υλοποίηση. Για κάθε μοναδική στο value εγγραφή του πίνακα table2Data ελέγχουμε όλες τις εγγραφές του πίνακα 1 (table1Data) αν έχουν κοινό κλειδί. Αν έχουν και ικανοποιούν τον περιορισμό του timestamp στα 10 δευτερόλεπτα προσθέτουμε την εγγραφή του table1Data στην λίστα semiJoinedTable1Data.

1. Η μέθοδος performTable1DataSemiJoinTable2DataTimestampFirst υλοποιεί το Semi Join με eager «σκεπτικό». Χρησιμοποιεί μία λίστα με όνομα semiJoinedTable1DataTimestampFirst για να αποθηκεύσει τα semi joined αποτελέσματα του table1Data.
2. Αρχικά, αρχικοποιούνται μία βοηθητική λίστα, η ArrayListTable2 και ένα set που ονομάζεται Unique Values και 2 λίστες την table1DataTimestampChecked και την table2DataTimestampChecked.
3. Στην συνέχεια θέλουμε να πάρουμε την προβολή του table2Data. Επομένως διατρέχουμε όλες τις εγγραφές του table2Data και για κάθε εγγραφή κρατάμε την τιμή του value (value2). Αν το set uniqueValues δεν περιέχει το value το προσθέτει και προσθέτει και την εγγραφή στην βοηθητική λίστα. Ο σκοπός του uniqueValues είναι απαλοιφή διπλοτύπων στο value, ενώ ο σκοπός της

βοηθητικής λίστας να κρατήσουμε όλη την μοναδική εγγραφή και όχι μόνο το value της μιας και θα πρέπει να γίνει έλεγχος του timestamp στην συνέχεια με τον άλλο πίνακα.

4. Σε αυτό το σημείο, ξεκινά η eager υλοποίηση. Για κάθε εγγραφή στον πίνακα 1 (table1Data) και για κάθε μοναδική στο value εγγραφή του πίνακα table2Data ελέγχουμε αν πληρείται το κριτήριο του timestamp. Αν ναι εισάγουμε τις εγγραφές του πίνακα 1 στην λίστα table1DataTimestampChecked και του πίνακα 2 στην λίστα table2DataTimestampChecked. Τέλος διατρέχουμε τις εγγραφές των νέων παραπάνω λιστών και ελέγχουμε αν έχουν κοινό κλειδί. Αν έχουν, προσθέτουμε την εγγραφή του table1Data στην λίστα semiJoinedTable1Data.

Στο τέλος, κλείνουν οι συνδέσεις στις βάσεις δεδομένων.

- Αξιολόγηση αποτελεσμάτων - συμπεράσματα:

<u>Ακολουθούν screenshots με τα δεδομένα των tables:</u>

```
 => => naming to docker.io/library/ddp-manipulate-node
PS C:\Users\ippok\Documents\DockerDir\TwoDatabasesDDP2> docker run -it -v database1_
olume:/volumes/database2 ddp-manipulate-node
Data of Table1 - Volume1:
Record ID: record0, Value: 32, Timestamp: 1685804993560
Record ID: record1, Value: 31, Timestamp: 1685804993678
Record ID: record2, Value: 52, Timestamp: 1685804993775
Record ID: record3, Value: 93, Timestamp: 1685804993869
Record ID: record4, Value: 16, Timestamp: 1685804993978
Record ID: record5, Value: 65, Timestamp: 1685804994080
Record ID: record6, Value: 76, Timestamp: 1685804994180
Record ID: record7, Value: 92, Timestamp: 1685804994281
Record ID: record8, Value: 11, Timestamp: 1685804994370
Record ID: record9, Value: 22, Timestamp: 1685804994479
Record ID: record10, Value: 1, Timestamp: 1685804994568
Record ID: record11, Value: 97, Timestamp: 1685804994653
Record ID: record12, Value: 85, Timestamp: 1685804994738
Record ID: record13, Value: 15, Timestamp: 1685804994847
Record ID: record14, Value: 44, Timestamp: 1685804994945
Record ID: record15, Value: 47, Timestamp: 1685804995046
Record ID: record16, Value: 32, Timestamp: 1685804995145
Record ID: record17, Value: 43, Timestamp: 1685804995248
Record ID: record18, Value: 85, Timestamp: 1685804995334
Record ID: record19, Value: 19, Timestamp: 1685804995431
```

```
Data of Table1 - Volume2:
Record ID: record0, Value: 62, Timestamp: 1685804993560
Record ID: record1, Value: 90, Timestamp: 1685804993678
Record ID: record2, Value: 97, Timestamp: 1685804993775
Record ID: record3, Value: 26, Timestamp: 1685804993869
Record ID: record4, Value: 51, Timestamp: 1685804993978
Record ID: record5, Value: 66, Timestamp: 1685804994080
Record ID: record6, Value: 66, Timestamp: 1685804994180
Record ID: record7, Value: 38, Timestamp: 1685804994281
Record ID: record8, Value: 8, Timestamp: 1685804994370
Record ID: record9, Value: 23, Timestamp: 1685804994479
Record ID: record10, Value: 44, Timestamp: 1685804994568
Record ID: record11, Value: 72, Timestamp: 1685804994653
Record ID: record12, Value: 77, Timestamp: 1685804994738
Record ID: record13, Value: 73, Timestamp: 1685804994847
Record ID: record14, Value: 67, Timestamp: 1685804994945
Record ID: record15, Value: 107, Timestamp: 1685804995046
Record ID: record16, Value: 60, Timestamp: 1685804995145
Record ID: record17, Value: 43, Timestamp: 1685804995248
Record ID: record18, Value: 80, Timestamp: 1685804995334
Record ID: record19, Value: 43, Timestamp: 1685804995431
Record ID: record20, Value: 70, Timestamp: 1685804995528
Record ID: record21, Value: 95, Timestamp: 1685804995604
Record ID: record22, Value: 52, Timestamp: 1685804995676
Record ID: record23, Value: 75, Timestamp: 1685804995749
Record ID: record24, Value: 68, Timestamp: 1685804995828
Record ID: record25, Value: 60, Timestamp: 1685804995884
Record ID: record26, Value: 10, Timestamp: 1685804995955
Record ID: record27, Value: 11, Timestamp: 1685804996031
Record ID: record28, Value: 83, Timestamp: 1685804996098
Record ID: record29, Value: 90, Timestamp: 1685804996158
Record ID: record30, Value: 72, Timestamp: 1685804996218
Record ID: record31, Value: 97, Timestamp: 1685804996266
Record ID: record32, Value: 92, Timestamp: 1685804996318
Record ID: record33, Value: 89, Timestamp: 1685804996356
Record ID: record34, Value: 50, Timestamp: 1685804996419
Record ID: record35, Value: 98, Timestamp: 1685804996470
Record ID: record36, Value: 29, Timestamp: 1685804996521
Record ID: record37, Value: 67, Timestamp: 1685804996570
Record ID: record38, Value: 25, Timestamp: 1685804996616
Record ID: record39, Value: 83, Timestamp: 1685804996660
```

```
Data of Table2 - Volume1:
Record ID: record0, Value: 71, Timestamp: 1685804993560
Record ID: record1, Value: 12, Timestamp: 1685804993678
Record ID: record2, Value: 8, Timestamp: 1685804993775
Record ID: record3, Value: 79, Timestamp: 1685804993869
Record ID: record4, Value: 12, Timestamp: 1685804993978
Record ID: record5, Value: 29, Timestamp: 1685804994080
Record ID: record6, Value: 41, Timestamp: 1685804994180
Record ID: record7, Value: 87, Timestamp: 1685804994281
Record ID: record8, Value: 27, Timestamp: 1685804994370
Record ID: record9, Value: 69, Timestamp: 1685804994479
Record ID: record10, Value: 6, Timestamp: 1685804994568
Record ID: record11, Value: 16, Timestamp: 1685804994653
Record ID: record12, Value: 50, Timestamp: 1685804994738
Record ID: record13, Value: 32, Timestamp: 1685804994847
Record ID: record14, Value: 25, Timestamp: 1685804994945
Record ID: record15, Value: 45, Timestamp: 1685804995046
Record ID: record16, Value: 9, Timestamp: 1685804995145
Record ID: record17, Value: 55, Timestamp: 1685804995248
Record ID: record18, Value: 64, Timestamp: 1685804995334
Record ID: record19, Value: 80, Timestamp: 1685804995431
Record ID: record20, Value: 3, Timestamp: 1685804995528
Record ID: record21, Value: 81, Timestamp: 1685804995604
Record ID: record22, Value: 62, Timestamp: 1685804995676
Record ID: record23, Value: 60, Timestamp: 1685804995749
Record ID: record24, Value: 4, Timestamp: 1685804995828
Record ID: record25, Value: 27, Timestamp: 1685804995884
Record ID: record26, Value: 78, Timestamp: 1685804995955
Record ID: record27, Value: 84, Timestamp: 1685804996031
Record ID: record28, Value: 33, Timestamp: 1685804996098
Record ID: record29, Value: 7, Timestamp: 1685804996158
```

```
Data of Table2 - Volume2:
Record ID: record0, Value: 67, Timestamp: 1685804993560
Record ID: record1, Value: 43, Timestamp: 1685804993678
Record ID: record2, Value: 0, Timestamp: 1685804993775
Record ID: record3, Value: 47, Timestamp: 1685804993869
Record ID: record4, Value: 27, Timestamp: 1685804993978
Record ID: record5, Value: 74, Timestamp: 1685804994080
Record ID: record6, Value: 6, Timestamp: 1685804994180
Record ID: record7, Value: 93, Timestamp: 1685804994281
Record ID: record8, Value: 9, Timestamp: 1685804994370
Record ID: record9, Value: 50, Timestamp: 1685804994479
Record ID: record10, Value: 36, Timestamp: 1685804994568
Record ID: record11, Value: 43, Timestamp: 1685804994653
Record ID: record12, Value: 55, Timestamp: 1685804994738
Record ID: record13, Value: 52, Timestamp: 1685804994847
Record ID: record14, Value: 77, Timestamp: 1685804994945
Record ID: record15, Value: 31, Timestamp: 1685804995046
Record ID: record16, Value: 4, Timestamp: 1685804995145
Record ID: record17, Value: 75, Timestamp: 1685804995248
Record ID: record18, Value: 24, Timestamp: 1685804995334
Record ID: record19, Value: 61, Timestamp: 1685804995431
Record ID: record20, Value: 87, Timestamp: 1685804995528
Record ID: record21, Value: 8, Timestamp: 1685804995604
Record ID: record22, Value: 12, Timestamp: 1685804995676
Record ID: record23, Value: 26, Timestamp: 1685804995749
Record ID: record24, Value: 36, Timestamp: 1685804995828
Record ID: record25, Value: 9, Timestamp: 1685804995884
Record ID: record26, Value: 55, Timestamp: 1685804995955
Record ID: record27, Value: 44, Timestamp: 1685804996031
Record ID: record28, Value: 89, Timestamp: 1685804996098
Record ID: record29, Value: 6, Timestamp: 1685804996158
Record ID: record30, Value: 50, Timestamp: 1685804996218
Record ID: record31, Value: 2, Timestamp: 1685804996266
Record ID: record32, Value: 53, Timestamp: 1685804996318
Record ID: record33, Value: 35, Timestamp: 1685804996356
Record ID: record34, Value: 33, Timestamp: 1685804996419
Record ID: record35, Value: 85, Timestamp: 1685804996470
Record ID: record36, Value: 40, Timestamp: 1685804996521
Record ID: record37, Value: 16, Timestamp: 1685804996570
Record ID: record38, Value: 22, Timestamp: 1685804996616
Record ID: record39, Value: 54, Timestamp: 1685804996660
Record ID: record40, Value: 84, Timestamp: 1685804996709
Record ID: record41, Value: 24, Timestamp: 1685804996729
Record ID: record42, Value: 53, Timestamp: 1685804996754
Record ID: record43, Value: 30, Timestamp: 1685804996780
Record ID: record44, Value: 10, Timestamp: 1685804996807
Record ID: record45, Value: 88, Timestamp: 1685804996830
Record ID: record46, Value: 84, Timestamp: 1685804996852
Record ID: record47, Value: 60, Timestamp: 1685804996876
Record ID: record48, Value: 58, Timestamp: 1685804996902
Record ID: record49, Value: 38, Timestamp: 1685804996930
```

Αποτελέσματα Hash Join :

```
Execution Time: 10 milliseconds
Hash Join - Joined Data - Small table Hash - LAZY:
Record ID: record17, Value: 43, Timestamp: 1685804995248, Record ID: record1, Value: 43, Timestamp: 1685804993678
Record ID: record15, Value: 47, Timestamp: 1685804995046, Record ID: record3, Value: 47, Timestamp: 1685804993869
Record ID: record3, Value: 93, Timestamp: 1685804993869, Record ID: record7, Value: 93, Timestamp: 1685804994281
Record ID: record17, Value: 43, Timestamp: 1685804995248, Record ID: record11, Value: 43, Timestamp: 1685804994653
Record ID: record2, Value: 52, Timestamp: 1685804993775, Record ID: record13, Value: 52, Timestamp: 1685804994847
Record ID: record1, Value: 31, Timestamp: 1685804993678, Record ID: record15, Value: 31, Timestamp: 1685804995046
Record ID: record14, Value: 44, Timestamp: 1685804994945, Record ID: record27, Value: 44, Timestamp: 1685804996031
Record ID: record12, Value: 85, Timestamp: 1685804994738, Record ID: record35, Value: 85, Timestamp: 1685804996470
Record ID: record18, Value: 85, Timestamp: 1685804995334, Record ID: record35, Value: 85, Timestamp: 1685804996470
Record ID: record4, Value: 16, Timestamp: 1685804993978, Record ID: record37, Value: 16, Timestamp: 1685804996570
Record ID: record9, Value: 22, Timestamp: 1685804994479, Record ID: record38, Value: 22, Timestamp: 1685804996616

Execution Time: 1 milliseconds
Hash Join - Joined Data - Large table Hash - LAZY:
Record ID: record15, Value: 31, Timestamp: 1685804995046, Record ID: record1, Value: 31, Timestamp: 1685804993678
Record ID: record13, Value: 52, Timestamp: 1685804994847, Record ID: record2, Value: 52, Timestamp: 1685804993775
Record ID: record7, Value: 93, Timestamp: 1685804994281, Record ID: record3, Value: 93, Timestamp: 1685804993869
Record ID: record37, Value: 16, Timestamp: 1685804996570, Record ID: record4, Value: 16, Timestamp: 1685804993978
Record ID: record38, Value: 22, Timestamp: 1685804996616, Record ID: record9, Value: 22, Timestamp: 1685804994479
Record ID: record35, Value: 85, Timestamp: 1685804996470, Record ID: record12, Value: 85, Timestamp: 1685804994738
Record ID: record27, Value: 44, Timestamp: 1685804996031, Record ID: record14, Value: 44, Timestamp: 1685804994945
Record ID: record3, Value: 47, Timestamp: 1685804993869, Record ID: record15, Value: 47, Timestamp: 1685804995046
Record ID: record1, Value: 43, Timestamp: 1685804993678, Record ID: record17, Value: 43, Timestamp: 1685804995248
Record ID: record11, Value: 43, Timestamp: 1685804994653, Record ID: record17, Value: 43, Timestamp: 1685804995248
Record ID: record35, Value: 85, Timestamp: 1685804996470, Record ID: record18, Value: 85, Timestamp: 1685804995334

Execution Time: 11 milliseconds
Hash Join - Joined Data - Small table Hash - First do Timestamp checks - EAGER:
Record ID: record17, Value: 43, Timestamp: 1685804995248, Record ID: record1, Value: 43, Timestamp: 1685804993678
Record ID: record15, Value: 47, Timestamp: 1685804995046, Record ID: record3, Value: 47, Timestamp: 1685804993869
Record ID: record3, Value: 93, Timestamp: 1685804993869, Record ID: record7, Value: 93, Timestamp: 1685804994281
Record ID: record17, Value: 43, Timestamp: 1685804995248, Record ID: record11, Value: 43, Timestamp: 1685804994653
Record ID: record2, Value: 52, Timestamp: 1685804993775, Record ID: record13, Value: 52, Timestamp: 1685804994847
Record ID: record1, Value: 31, Timestamp: 1685804993678, Record ID: record15, Value: 31, Timestamp: 1685804995046
Record ID: record14, Value: 44, Timestamp: 1685804994945, Record ID: record27, Value: 44, Timestamp: 1685804996031
Record ID: record12, Value: 85, Timestamp: 1685804994738, Record ID: record35, Value: 85, Timestamp: 1685804996470
Record ID: record18, Value: 85, Timestamp: 1685804995334, Record ID: record35, Value: 85, Timestamp: 1685804996470
Record ID: record4, Value: 16, Timestamp: 1685804993978, Record ID: record37, Value: 16, Timestamp: 1685804996570
Record ID: record9, Value: 22, Timestamp: 1685804994479, Record ID: record38, Value: 22, Timestamp: 1685804996616

Execution Time: 5 milliseconds
Hash Join - Joined Data - Large table Hash - First do Timestamp checks - EAGER:
Record ID: record15, Value: 31, Timestamp: 1685804995046, Record ID: record1, Value: 31, Timestamp: 1685804993678
Record ID: record13, Value: 52, Timestamp: 1685804994847, Record ID: record2, Value: 52, Timestamp: 1685804993775
Record ID: record7, Value: 93, Timestamp: 1685804994281, Record ID: record3, Value: 93, Timestamp: 1685804993869
Record ID: record37, Value: 16, Timestamp: 1685804996570, Record ID: record4, Value: 16, Timestamp: 1685804993978
Record ID: record38, Value: 22, Timestamp: 1685804996616, Record ID: record9, Value: 22, Timestamp: 1685804994479
Record ID: record35, Value: 85, Timestamp: 1685804996470, Record ID: record12, Value: 85, Timestamp: 1685804994738
Record ID: record27, Value: 44, Timestamp: 1685804996031, Record ID: record14, Value: 44, Timestamp: 1685804994945
Record ID: record3, Value: 47, Timestamp: 1685804993869, Record ID: record15, Value: 47, Timestamp: 1685804995046
Record ID: record1, Value: 43, Timestamp: 1685804993678, Record ID: record17, Value: 43, Timestamp: 1685804995248
Record ID: record11, Value: 43, Timestamp: 1685804994653, Record ID: record17, Value: 43, Timestamp: 1685804995248
Record ID: record35, Value: 85, Timestamp: 1685804996470, Record ID: record18, Value: 85, Timestamp: 1685804995334
```

Στο hash join παρατηρούμε ότι σε γενικές γραμμές η Lazy υλοποίηση είναι πιο καλή από άποψη performance, αλλά και optimistic σκεπτικού. Στο Lazy έχουμε 10 < 11 και 1 < 5 milliseconds.

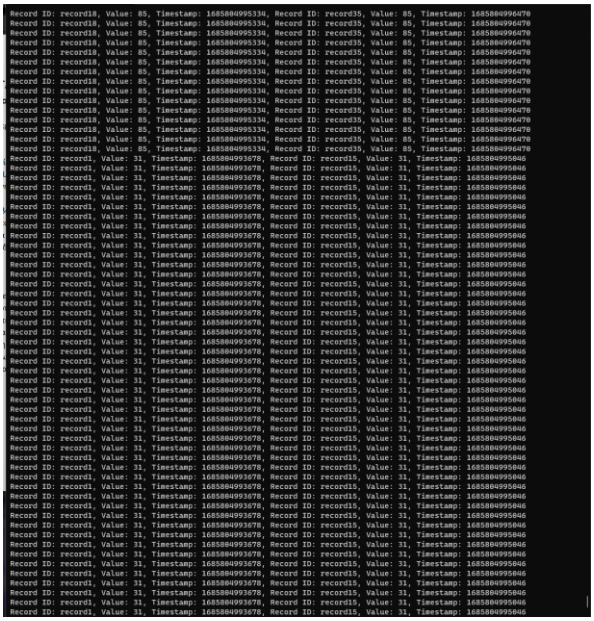
Παρατηρούμε ότι είτε στο lazy είτε στο eager όταν κάνουμε hash τον μεγάλο πίνακα (με τις περισσότερες εγγραφές) το execution time είναι μικρότερο από το να κάναμε hash τον μικρό πίνακα, διότι οι προσπελάσεις είναι λιγότερες, δηλαδή παίρνουμε μία μία τις εγγραφές του μικρού πίνακα και ελέγχουμε την σύνδεση στο hash table του μεγάλου πίνακα.

Αποτελέσματα Pipeline Hash Join :

```
Execution Time: 2 milliseconds
Pipeline Hash Join - Joined Data - LAZY:
Record ID: record15, Value: 47, Timestamp: 1685804995046, Record ID: record3, Value: 47, Timestamp: 1685804993869
Record ID: record3, Value: 93, Timestamp: 1685804993869, Record ID: record7, Value: 93, Timestamp: 1685804994281
Record ID: record4, Value: 16, Timestamp: 1685804993978, Record ID: record37, Value: 16, Timestamp: 1685804996570
Record ID: record2, Value: 52, Timestamp: 1685804993775, Record ID: record13, Value: 52, Timestamp: 1685804994847
Record ID: record9, Value: 22, Timestamp: 1685804994479, Record ID: record38, Value: 22, Timestamp: 1685804996616
Record ID: record14, Value: 44, Timestamp: 1685804994945, Record ID: record27, Value: 44, Timestamp: 1685804996031
Record ID: record12, Value: 85, Timestamp: 1685804994738, Record ID: record35, Value: 85, Timestamp: 1685804996470
Record ID: record17, Value: 43, Timestamp: 1685804995248, Record ID: record11, Value: 43, Timestamp: 1685804994653
Record ID: record17, Value: 43, Timestamp: 1685804995248, Record ID: record1, Value: 43, Timestamp: 1685804993678
Record ID: record18, Value: 85, Timestamp: 1685804995334, Record ID: record35, Value: 85, Timestamp: 1685804996470
Record ID: record1, Value: 31, Timestamp: 1685804993678, Record ID: record15, Value: 31, Timestamp: 1685804995046

Execution Time: 18 milliseconds
Pipeline Hash Join - Joined Data - EAGER:
Record ID: record12, Value: 85, Timestamp: 1685804994738, Record ID: record35, Value: 85, Timestamp: 1685804996470
Record ID: record4, Value: 16, Timestamp: 1685804993978, Record ID: record37, Value: 16, Timestamp: 1685804996570
Record ID: record14, Value: 44, Timestamp: 1685804994945, Record ID: record27, Value: 44, Timestamp: 1685804996031
Record ID: record15, Value: 47, Timestamp: 1685804995046, Record ID: record3, Value: 47, Timestamp: 1685804993869
Record ID: record17, Value: 43, Timestamp: 1685804995248, Record ID: record1, Value: 43, Timestamp: 1685804993678
Record ID: record3, Value: 93, Timestamp: 1685804993869, Record ID: record7, Value: 93, Timestamp: 1685804994281
Record ID: record1, Value: 31, Timestamp: 1685804993678, Record ID: record15, Value: 31, Timestamp: 1685804995046
Record ID: record2, Value: 52, Timestamp: 1685804993775, Record ID: record13, Value: 52, Timestamp: 1685804994847
Record ID: record9, Value: 22, Timestamp: 1685804994479, Record ID: record38, Value: 22, Timestamp: 1685804996616
Record ID: record17, Value: 43, Timestamp: 1685804995248, Record ID: record11, Value: 43, Timestamp: 1685804994653
Record ID: record18, Value: 85, Timestamp: 1685804995334, Record ID: record35, Value: 85, Timestamp: 1685804996470
```

Όμοια με το hash join, έτσι και στο pipeline hash join, η διαφορά στο performance μεταξύ lazy και eager είναι εμφανής. Όπως συζητήσαμε και στο μάθημα προτιμάμε να κάνουμε τους ελέγχους χρονοσφραγίδας κατά την σύνδεση και όχι εκ των προτέρων, μιας και τέτοιου είδους optimistic σκεπτικά επιφέρουν καλύτερα αποτελέσματα και προτιμώνται. Επίσης στην eager υλοποίηση παρατηρήθηκε ότι οι προσπελάσεις είναι πάρα πολλές και έχουμε πολλά διπλότυπα στα αποτελέσματα, τα οποία γονατίζουν το σύστημα. Στην παραπάνω εικόνα έχουμε χρησιμοποιήσει set, αντί για λίστα για να αποφύγουμε τα διπλότυπα και τις πολλές προσπελάσεις. Σε περίπτωση που είχαμε λίστα στα αποτελέσματα μας θα εμφανιζόταν στην κονσόλα κάτι τέτοιο:



Αποτελέσματα Semi Join :

```
Execution Time: 2 milliseconds
Semi Join - Semi Joined Data - Table1 - LAZY:
Record ID: record17, Value: 43, Timestamp: 1685804995248
Record ID: record15, Value: 47, Timestamp: 1685804995046
Record ID: record3, Value: 93, Timestamp: 1685804993869
Record ID: record2, Value: 52, Timestamp: 1685804993775
Record ID: record1, Value: 31, Timestamp: 1685804993678
Record ID: record14, Value: 44, Timestamp: 1685804994945
Record ID: record12, Value: 85, Timestamp: 1685804994738
Record ID: record18, Value: 85, Timestamp: 1685804995334
Record ID: record4, Value: 16, Timestamp: 1685804993978
Record ID: record9, Value: 22, Timestamp: 1685804994479

Execution Time: 3 milliseconds
Semi Join - Semi Joined Data - Table1 - Check Timestamp first - EAGER:
Record ID: record17, Value: 43, Timestamp: 1685804995248
Record ID: record15, Value: 47, Timestamp: 1685804995046
Record ID: record3, Value: 93, Timestamp: 1685804993869
Record ID: record2, Value: 52, Timestamp: 1685804993775
Record ID: record1, Value: 31, Timestamp: 1685804993678
Record ID: record14, Value: 44, Timestamp: 1685804994945
Record ID: record12, Value: 85, Timestamp: 1685804994738
Record ID: record18, Value: 85, Timestamp: 1685804995334
Record ID: record4, Value: 16, Timestamp: 1685804993978
Record ID: record9, Value: 22, Timestamp: 1685804994479
```

Όμοια και στο semi join η διαφορά στο performance μεταξύ lazy και eager είναι εμφανής αλλά όχι και τόσο μεγάλη στο συγκεκριμένο παράδειγμα που τα δεδομένα δεν είναι πολλά.

Για να έχει νόημα το semi join μεταξύ δύο πινάκων table1Data semiJoin table2Data, θα πρέπει το όρισμα table1Data να λαμβάνει τον μικρότερο πίνακα (με τις λιγότερες εγγραφές) και το όρισμα table2Data τον μεγαλύτερο πίνακα. Στο παραπάνω screenshot η συνθήκη αυτή ικανοποιείται και έχει νόημα να εκτελεστεί το semi join για να φιλτραριστεί το table1Data.

Γενικά ένα semi join για να έχει νόημα και να είναι αποδοτικό θα πρέπει να ικανοποιεί την παρακάτω συνθήκη:

$$size(\Pi_A(S)) + size(R \ltimes_A S)) < size(R)$$

Όπου R είναι table1Data και S είναι table2Data.

**Σε κάθε περίπτωση προτιμάμε lazy υλοποίηση δηλαδή την ώρα που γίνεται η σύνδεση να κάνουμε τον έλεγχο του timestamp, ώστε να μην χάνουμε χρόνο συνεχώς να ελέγχουμε περιοδικά τα hash tables.**

Σε όλες τις υλοποιήσεις το timestamp που χρησιμοποιήθηκε είναι 10 seconds.

Παρόλα αυτά όταν αλλάζω το χρονικό παράθυρο από 10 seconds σε 2 ή σε 1 seconds, χάνονται εγγραφές στην lazy υλοποίηση συγκριτικά με την eager στα αποτελέσματα. Δεν μπορώ να ερμηνεύσω αυτό γιατί συμβαίνει.. αλλά θα χαρώ πολύ να το συζητήσουμε από κοντά!

Timestamp = 1 second, πχ στο semi join:

```
Execution Time: 1 milliseconds
Semi Join - Semi Joined Data - Table1 - LAZY:
Record ID: record3, Value: 93, Timestamp: 1685804993869

Execution Time: 3 milliseconds
Semi Join - Semi Joined Data - Table1 - Check Timestamp first - EAGER:
Record ID: record17, Value: 43, Timestamp: 1685804995248
Record ID: record15, Value: 47, Timestamp: 1685804995046
Record ID: record3, Value: 93, Timestamp: 1685804993869
Record ID: record2, Value: 52, Timestamp: 1685804993775
Record ID: record1, Value: 31, Timestamp: 1685804993678
Record ID: record14, Value: 44, Timestamp: 1685804994945
```