# PARTITION ALGORITHM APPROACH FOR MINING FREQUENT PATTERN

## Mining Frequent Patterns using Partition Algorithm

A partition concept has been proposed to increase the execution speed with minimum cost. For each itemsets, that is one itemset, 2-itemsets, 3-itemsets etc., a separate partition will be created during data insertion in to the table. Initially a set of frequent 1-itemsets is found by scanning the databases and get the numbers of occurrences of each item from the partition of those particular items using the pointer, the items satisfying the minimum support count will be included in the frequent 1-itemsets $L_1$. Like Apriori algorithm $L_1$ is used to find $L_2$, the set of $L_2$ is used to find $L_3$, and so on, until no more frequent k-itemsets can be found.

To find, $L_k$, it is not necessary to scan the full database; it is enough to search the count of each data itemsets from its partition. Initially to generate frequent itemsets, an important property called the Apriori property used for reducing the search space. Join and Prune are two-steps to find the frequent itemsets. In join, Lk is find from a set of candidate k-item sets $C_k$ which is generated by joining $Lk_{-1}$ with itself.

In pruning, to find the count of each candidate in $C_k$, the partition of each itemsets will be checked and the count which is not less than minimum support count are frequent and belongs to $L_k$. To reduce the size of $C_k$, the apriori property is used. The performance of partition algorithm for finding the frequent data items is efficient when compared to other existing Algorithms.

**Table : Transaction Database for Partition Algorithm**

| TID | List of Items |
|---|---|
| T1 | I1, I2, I4 |
| T2 | I2, I4 |
| T3 | I3, I4 |
| T4 | I1, I2, I5 |
| T5 | I1, I4 |
| T6 | I1, I3, I5 |
| T7 | I1, I2 ,I3, I5 |
| T8 | I1, I2, I3 |

Consider the Database D, with seven transactions, here partitioning has been used to store and retrieve the data from the database. Different types of partitioning techniques are available, in this thesis; range partitioning has been used to increase the performance when mining the frequent patterns in the database.

Tabl shows a transaction database for partition algorithm and it contains 8 transactions. In this table, transaction T1 contains I1,I2,I3 and transaction T2 contains I2, I4 and so on..

From the candidate 1 itemset $C_1$ generate the frequency 1 itemset $L_1$ by calculating the no of occurrences of the data items directly from the partition instead of scanning the whole database. Here the minimum support count taken is 2. Candidate 1 itemset which is satisfying the minimum support count will be include in $L_1$. The following figure. shows, the generation of candidate 1 itemsets and frequent 1 itemsets.

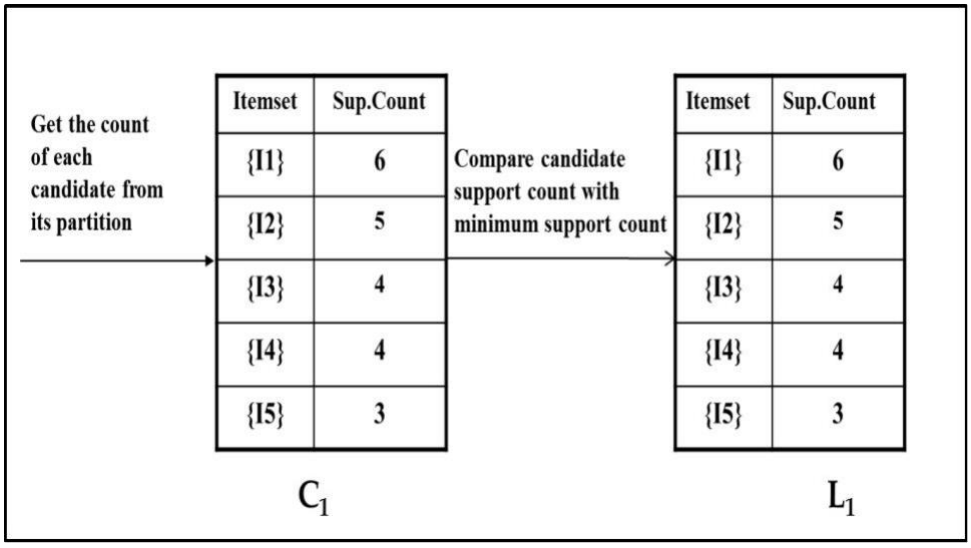


**Figure Generation of Frequent 1- Itemset using Partition Algorithm**

The candidate 2 itemsets are generated by joining $L_1$ by itself and check whether the subset of the frequent itemsets are also frequent, since in $L_1$ all the items have been included, there will be no pruning.

For calculating the support count, instead of scanning the whole database, it is enough to get the count from the appropriate partition. The candidate 2 itemsets which is satisfying the minimum support count will be included in the frequent 2 itemsets $L_2$. The following figure shows the generation of frequent 2 itemsets using partition. Finally 8 frequent 2 itemsets have been generated, and this has been used to generate the candidate 2 itemsets.

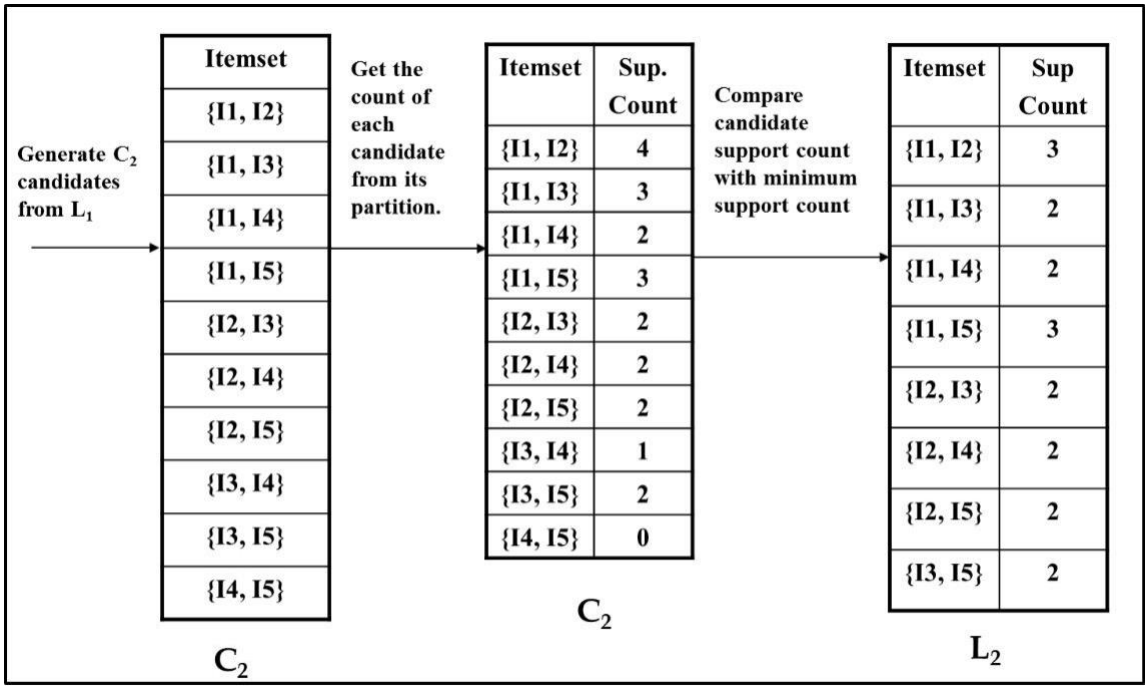


**Figure Generation of Frequent 2- Itemset using Partition Algorithm**

The candidate 3 itemsets are generated by joining L2 by itself and find whether the subset of the frequent itemsets are also frequent, only the itemsets {I1,I2,I4},{I1,I2,I5},{I1,I2,I3} and {I1,I3,I5} have been included for the next step that is considered as candidate 3 itemsets since whose subset is also a frequent itemset.

The remaining itemsets have been removed because whose subset is not frequent one. For calculating the support count, instead of scanning the whole database, it is enough to get the count from the appropriate partition. The candidate 3 itemsets which is satisfying the minimum support count will be included in the frequent 3 itemsets L3.

The following figure shows the generation of frequent 3 itemsets using partition. Finally 3 frequent 3 itemsets have been generated, and this has been used to generate the candidate 3 itemsets.

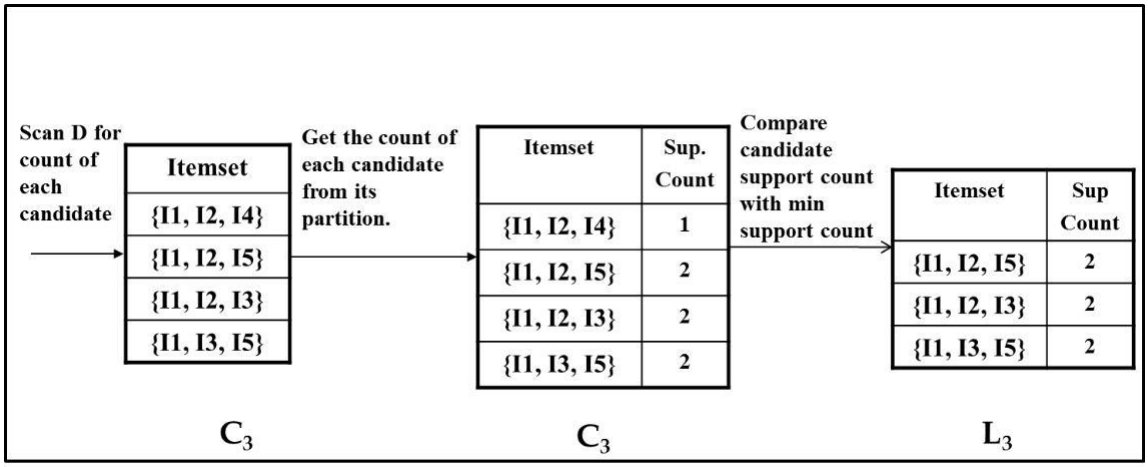


**Figure Generation of Frequent 3 - Itemset using Partition Algorithm**

The candidate 4 itemsets are generated by joining L3 by itself and check whether the subset of the frequent itemsets are also frequent, only the itemsets {I1,I2,I3,I5}have been included for the next step that is considered as candidate 4 itemsets since whose subset is also a frequent itemset.

For calculating the support count, directly get the count from the partition, instead of scanning the whole database. Since the support count is 1, which is not satisfying the minimum support, it is pruned and L4 = ¢ and algorithm terminates. With the frequent 3 itemsets, association rule is applied which shows the association between the data items.

## Implementation and Result

The partition algorithm has been implemented using PL/SQL in Oracle Database and then compared with Apriori algorithm. Table 6.2 shows the performance evaluation between Apriori and Partition algorithm. The algorithms are compared with different sets of record like 1000, 2500, 5000 etc.

**Table Performance Evaluation of Apriori and Partition**

| ALGORITHMS | TOTAL NO OF RECORDS | | | | | |
|---|---|---|---|---|---|---|
| | 1000 | 2500 | 5000 | 10000 | 20000 | 30000 |
| APRIORI | 8 min | 20 min | 42 min | 86min | 178 min | 264min |
| PARTITION | 30 sec | 1.5 min | 3 min | 6 min | 13 min | 19 min |

When the no of records is 1000, the Apriori algorithm takes 8 minutes and partition algorithm takes 30 secs to find the frequent data items sets. When the total no of records is 2500, the partition algorithm has taken 1.5 min and Apriori algorithm taken 20 min and when the total no of records is 5000, Apriori algorithm taken 42 minutes and partition algorithm takes 3 minutes.

In this way, the partition algorithm has better performance over Apriori algorithm.

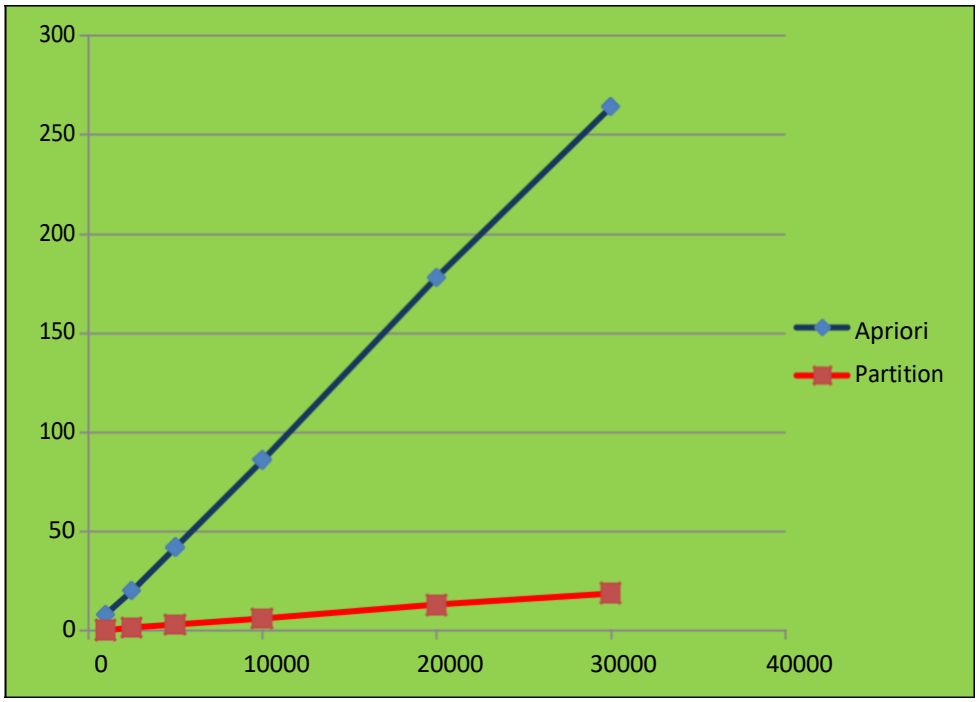


**Figure  Performance Evaluation Graph of Partition Algorithm**

Figure shows the generation of frequent itemset and association rule generation graph and by using the table , graph have been plotted for Apriori and partition algorithms. In this graph, total no of records has taken in the x-axis, and the execution time has taken as y-axis. Graphical representation shows, partition algorithm has efficient performance over the Apriori algorithm.