Phase 4: Lightning Fast and Space Efficient Inequality Joins with selectivity estimation and Multiple predicates

Ira A. Fulton Schools of Engineering, Arizona State University, Tempe

Group 25	Group members	ASU ID
1	Vishnu Vasanth Radja	1210378036
2	Ravi Nihalani	1210448145
3	Azhaku Sakthi Vel M	1210299529
4	Madhulahari Illuri	1209311047
5	Manogna Vennamaneni	1209373643

1. Abstract

Inequality joins, which join relational tables on inequality conditions, are used in various applications. In this implementation, we implement fast inequality join algorithm by implementing selectivity estimation using sampling followed by Join for multiple predicates. For selectivity of an inequality join, we are performing Histogram Sampling, Uniform sampling, Random sampling and average Random sampling. The sampling size considered for performing multi predicate joins is 1% after running various experiments. We have implemented a centralized version of selectivity and join algorithm on top of minibase. We have compared the four selectivity approaches with respect to the time it takes with IE Join Algorithm and the experiments clearly show that this implementation is more scalable and several orders of magnitude faster.

2. Introduction

This report is an implementation of Inequality joins for multiple predicates. Inequality joins, which join relational tables on inequality conditions, are used in various applications. In this implementation, we implement fast inequality join algorithms by implementing selectivity estimation followed by joins for multiple predicates. For selectivity of an inequality join, we are performing Histogram Sampling, Uniform sampling, Random sampling and average Random sampling. The sampling size considered is 1%, 2%, 5% and 10% for all three sampling methods. The sampling method for the given data set which takes the least amount of time is the method to go ahead with before the multiple predicates are run using the IE Join Algorithm for join in the last assignment. We have implemented a centralized version of selectivity and join algorithm on top of minijava. We have compared the four selectivity approaches with respect to the time it takes and

also the joining algorithm IE Join and the experiments clearly show that this implementation is more scalable and several orders of magnitude faster.

3. Implementation Details

- **Task 1: Selectivity Estimation**
    - **Purpose:** Implemented Histogram, Uniform, Random and Random Average sampling over the given multiple dual predicate join relations to compute the estimate.
    - **Comparison:** Used different sampling size or percentage and measure how good is the estimate comparing all types of Sampling methods. Here we calculate the Time Taken and tuples count for the 4 sampling methods used with different sampling sizes. Then we conclude the best sampling method and sampling size to go ahead with Task 2.
    - **Sampling Size:** We used 1%, 2%, 5% and 10% of tables sizes for Histogram, Uniform, Random and Random average sampling techniques.
    - **Objective** - To evaluate whether Histogram, Uniform, Random or Random Average Sampling to be considered. Also to conclude how many runs of sampling should be run. The end result is to come up with order of dual predicate inequality joins which gives the joins output in less time.
    - **Order of Inequality Joins:** The dual predicate join that gives the less number of tuples are considered to execute first and then in ascending order of number of tuples each dual predicate join gives, we execute the **IE join iteratively** to optimize the given query in less number of computations and Time Taken.
    - **Design Choices:** Based on our experiments, we have considered 1% sampling size and 2 runs of random average runs to calculate the order of dual predicates for Task 2.We used pipelining technique to do dual predicate multiple inequality join in task 2. Histogram gave better sampling results and we have also added experiments for that.
    - **Data Structures:** Array-Lists and HashMap have been used extensively. Data from only specified column numbers are stored separately in arrays for easy access. As most of the operations required were searching, inserting, deleting and sorting elements. In most of the cases, binary search has been used to optimize the initialization process of IE join. For implementing Histograms we've used Array List and Collections for sorting.
    - **Known Issues/ Limitations/ Assumptions Made:**
        - The Diagonality must be maintained between the given predicates. The program tries to find an optimized order of the predicates such that the resultant order also maintain the diagonality. If no such scenario is met where the order is optimized and is diagonalized, then the input order of queries is passed on to the next phase of the program. Hence for the program to not crash and to output the count of the tuples, the input order of the predicates must be at least diagonalized, if not optimized.
        - Using the table names given in the query, the program automatically extracts the names of the files containing relations/ tables and expects them to be present in the same folder (preferable inside '*res*' folder) where the query file is present.

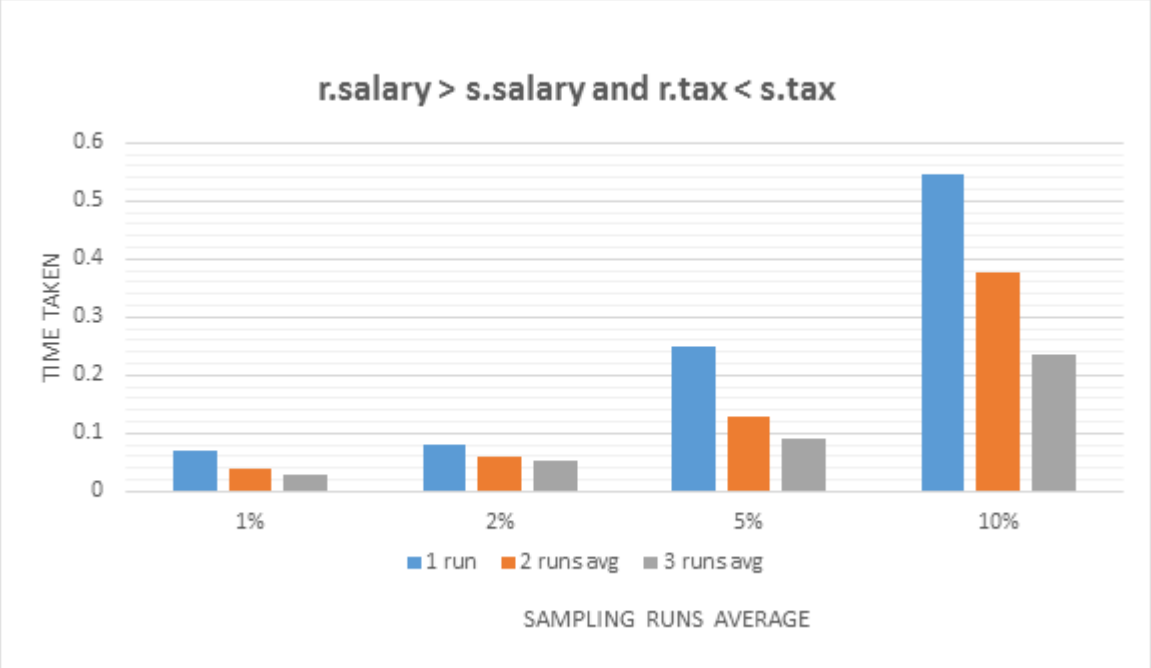
        - The columns of the query must be of Integer type so that arithmetic operations can be performed on them, else it will result into "**Error: Integer Column not Provided**".
        - 1<= Op1 <= 4, 1<= Op2 <= 4. Current implementation does not support OR operation between predicates.
        - The Program outputs the count of the number of the tuples finally.
    - **Details on Optimization:** We have considered histogram for sampling optimizations in our experiments. The histogram considers the data distribution across the table and hence the samples selected for the selectivity are the best representatives for the entire table.

RANDOM SAMPLING: Taking the first 1% of records/rows from the table in random rather than taking them in sequential order is called Random Sampling. The samples taken in random order are further considered for sampling using IE join for each dual predicate Join. The results of the experiment are formulated below.

- **Example Query taken:**
  - select count(*) From F1 r, F2 s, F3 t, F4 u,F5 v
    Where r.salary > s.salary and r.tax < s.tax (1)
    AND s.start < t.end AND s.end > t.start (2)
    AND t.salary > v.salary AND t.tax < v.tax (3)
    AND v.start < w.end AND v.end > w.start (4)

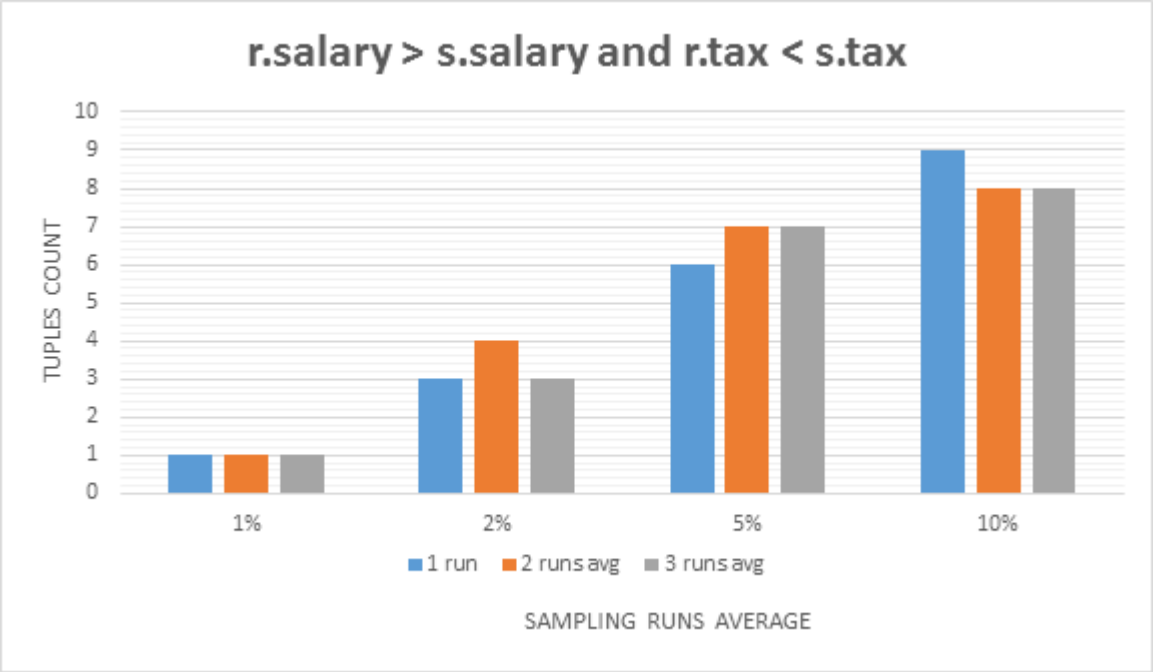
- **EXPERIMENTS FOR RANDOM SAMPLING :**
  - (**Sample percentage vs Time taken in seconds**)
    - Dual Predicate Join considered for below table:
      - **r.salary > s.salary and r.tax < s.tax**

TIME TAKEN (in seconds)	**1 run**	**2 runs avg**	**3 runs avg**
1%	0.07	0.04	0.03
2%	0.079	0.058	0.053
5%	0.249	0.128	0.091
10%	0.546	0.378	0.235



- **(Sample Percentage vs Number of Tuples)**

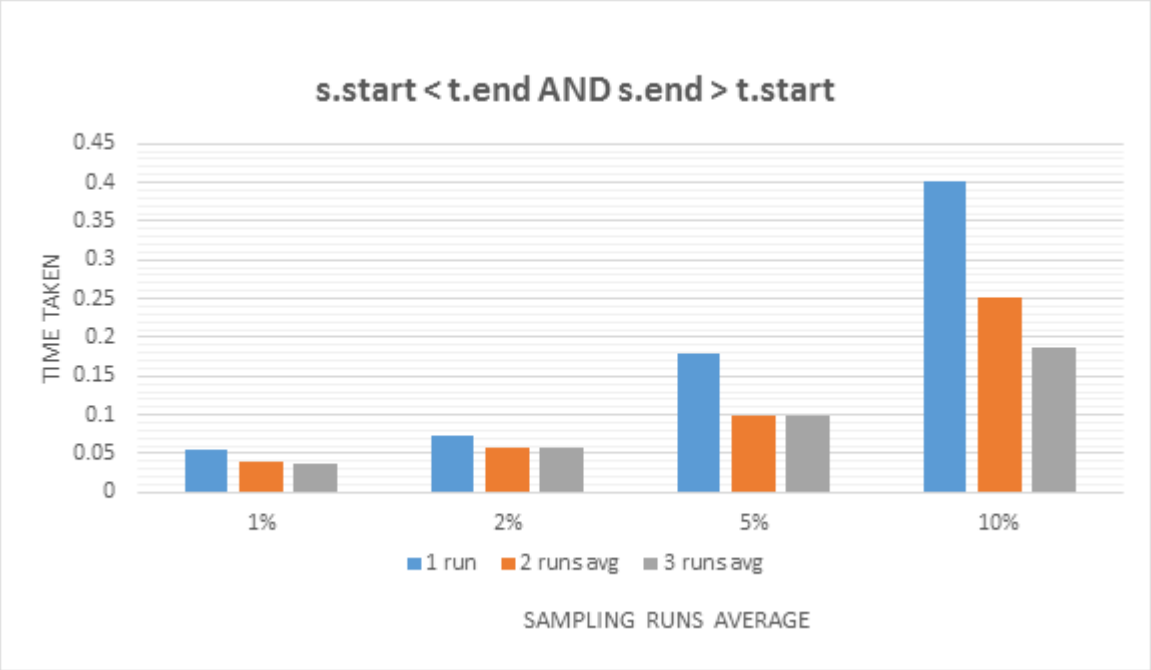
TUPLES COUNT	1 run	2 runs avg,	3 runs avg
1%	1	1	1
2%	3	4	3
5%	6	7	7
10%	9	8	8



- **s.start < t.end AND s.end > t.start**

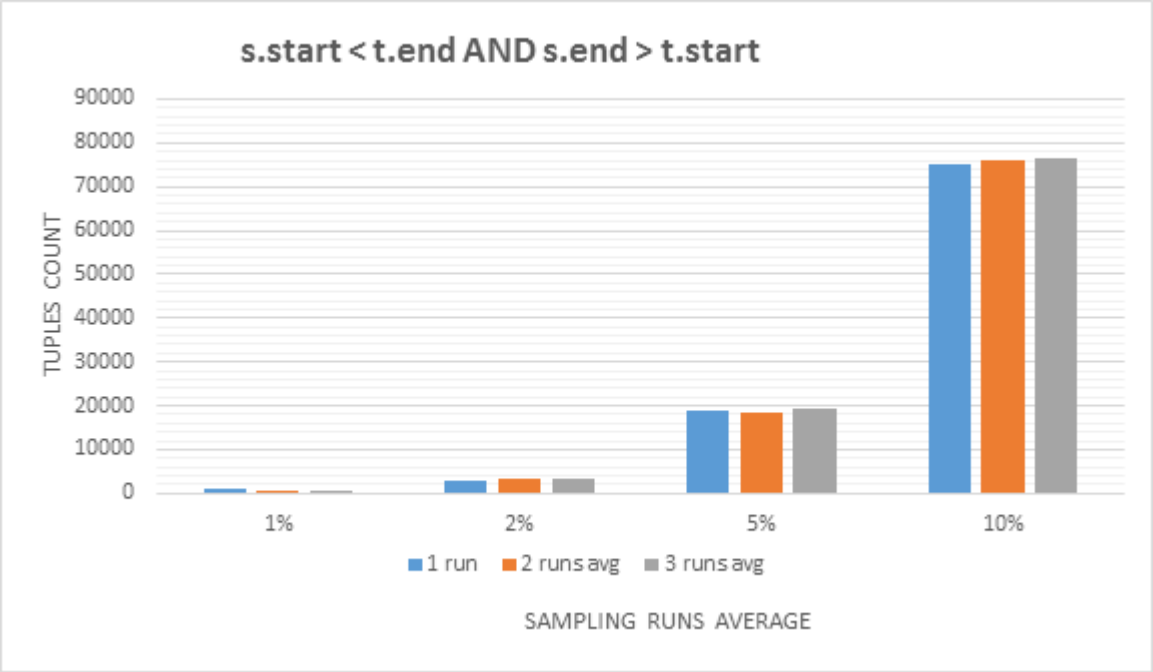
Sample percentage vs Time taken

TIME TAKEN	1 run	2 runs average	3 runs average
1%	0.056	0.039	0.036
2%	0.073	0.058	0.057
5%	0.18	0.1	0.1
10%	0.401	0.252	0.187



Sample Percentage vs Number of Tuples

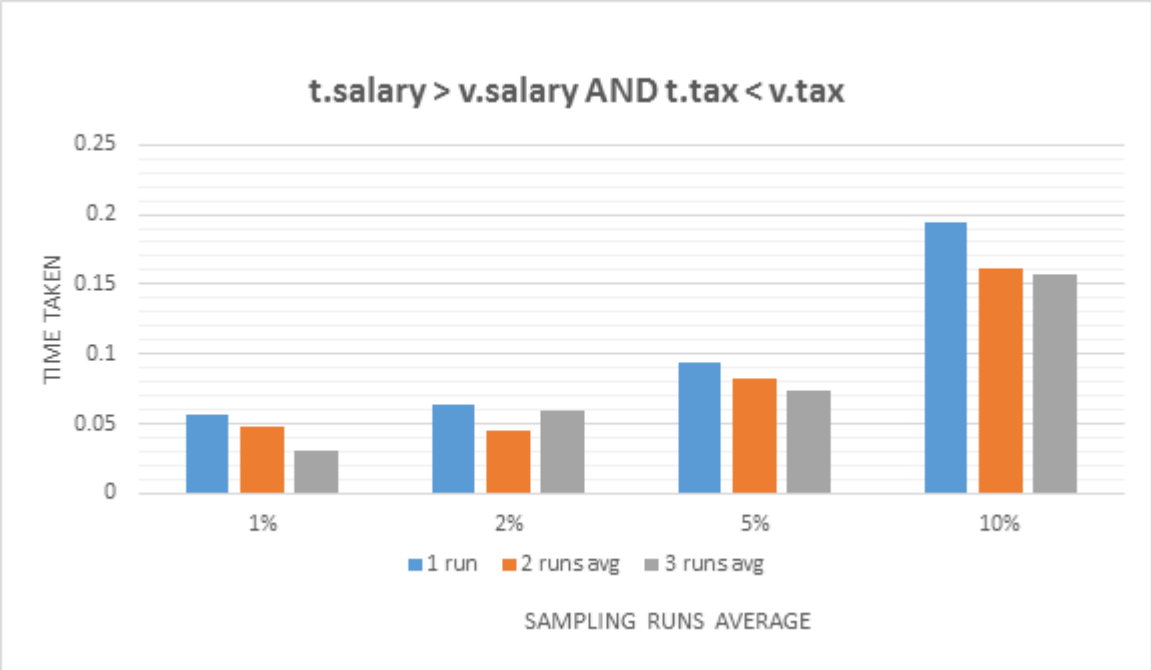
TUPLES COUNT	1 run	2 runs average	3 runs average
1%	855	798	732
2%	3067	3313	3207
5%	18679	18237	19394
10%	75299	75844	76497



- **t.salary > v.salary AND t.tax < v.tax**

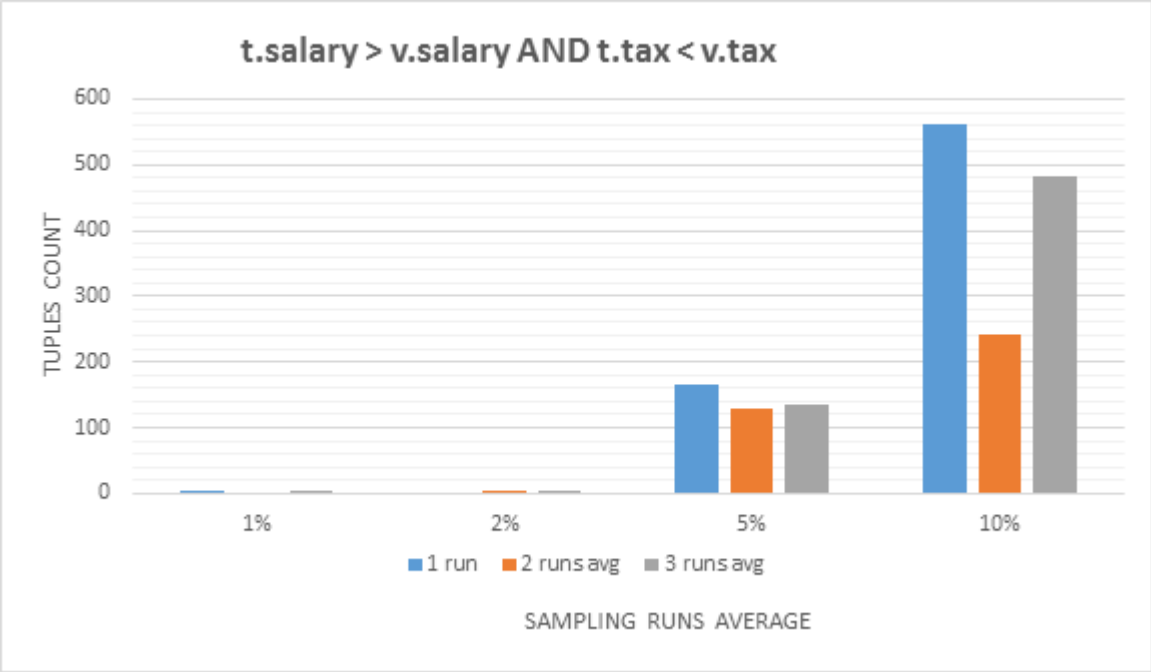
Sample percentage vs Time taken

TIME TAKEN	1 run	2 runs average	3 runs average
1%	0.057	0.048	0.031
2%	0.063	0.045	0.06
5%	0.094	0.082	0.073
10%	0.195	0.161	0.157



Sample percentage vs Number of Tuples

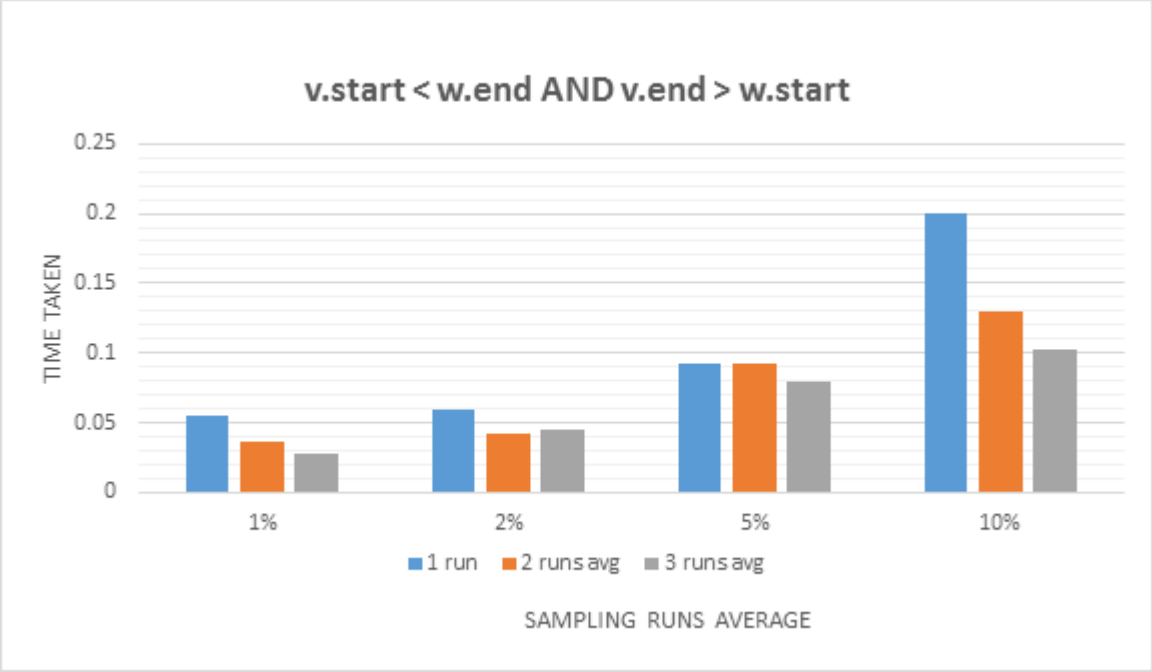
TUPLES COUNT	1 run	2 runs average	3 runs average
1%	1	0	1
2%	0	1	1
5%	164	128	136
10%	561	241	482



- **v.start < w.end AND v.end > w.start**

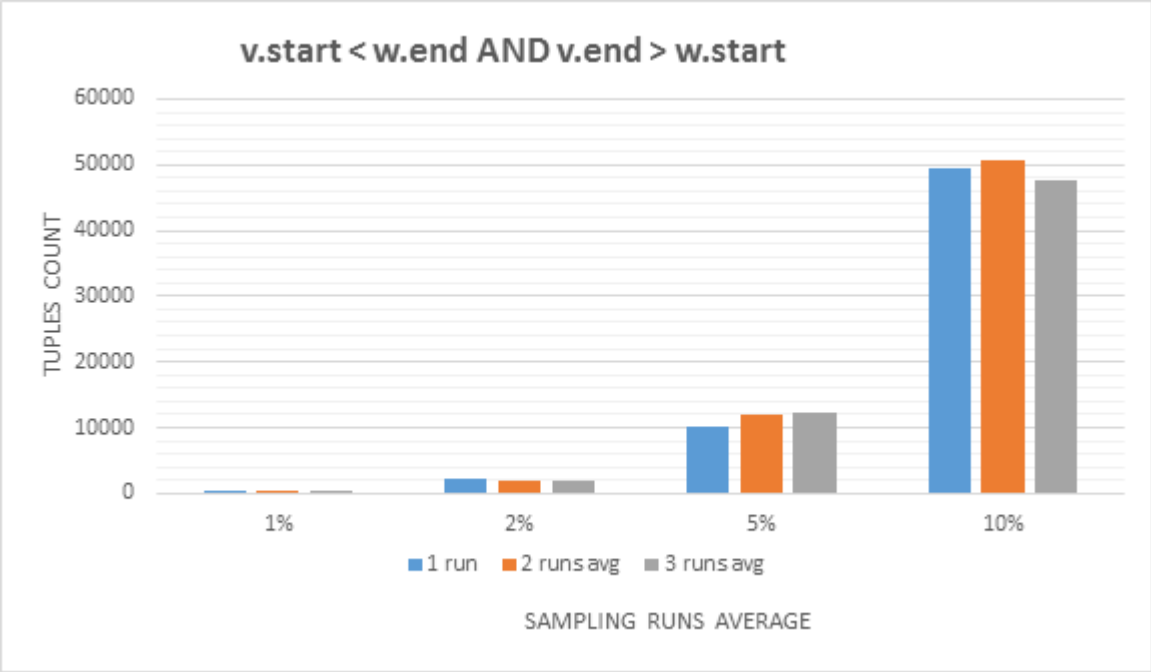
Sample percentage vs Time taken

TIME TAKEN	1 run	2 runs average	3 runs average
1%	0.055	0.037	0.028
2%	0.060	0.0416	0.0443
5%	0.093	0.092	0.08
10%	0.2	0.130	0.102



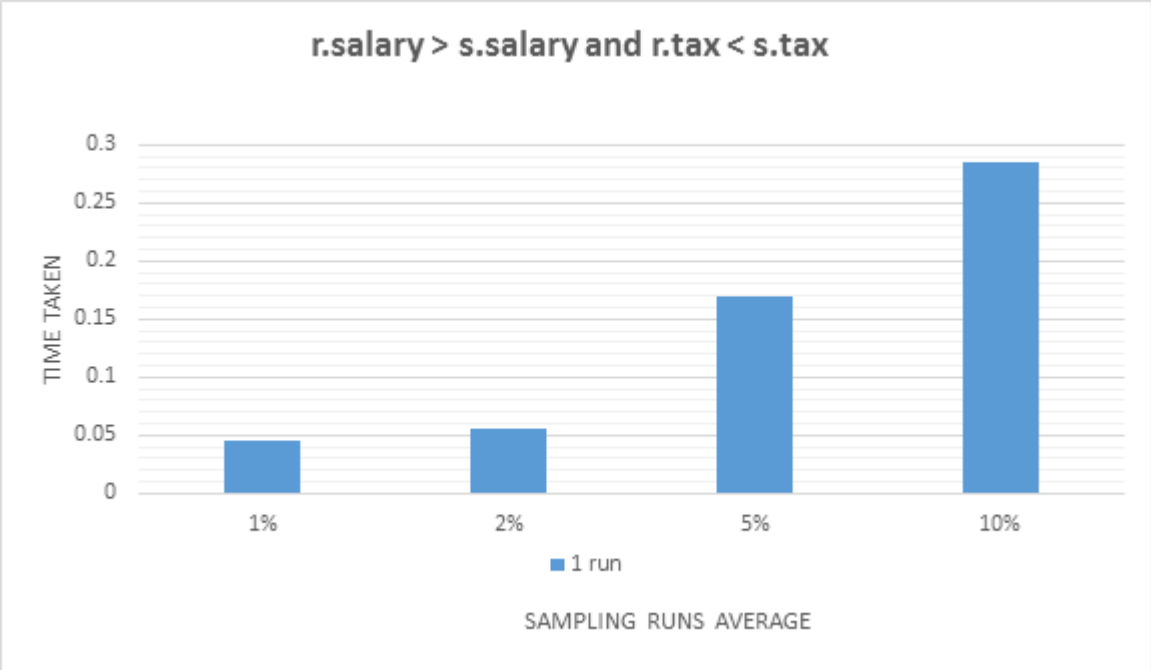
Sample percentage vs Number of Tuples

TUPLES COUNT	1 run	2 runs average	3 runs average
1%	513	547	502
2%	2132	2033	1785
5%	10296	11874	12398
10%	49484	50639	47551



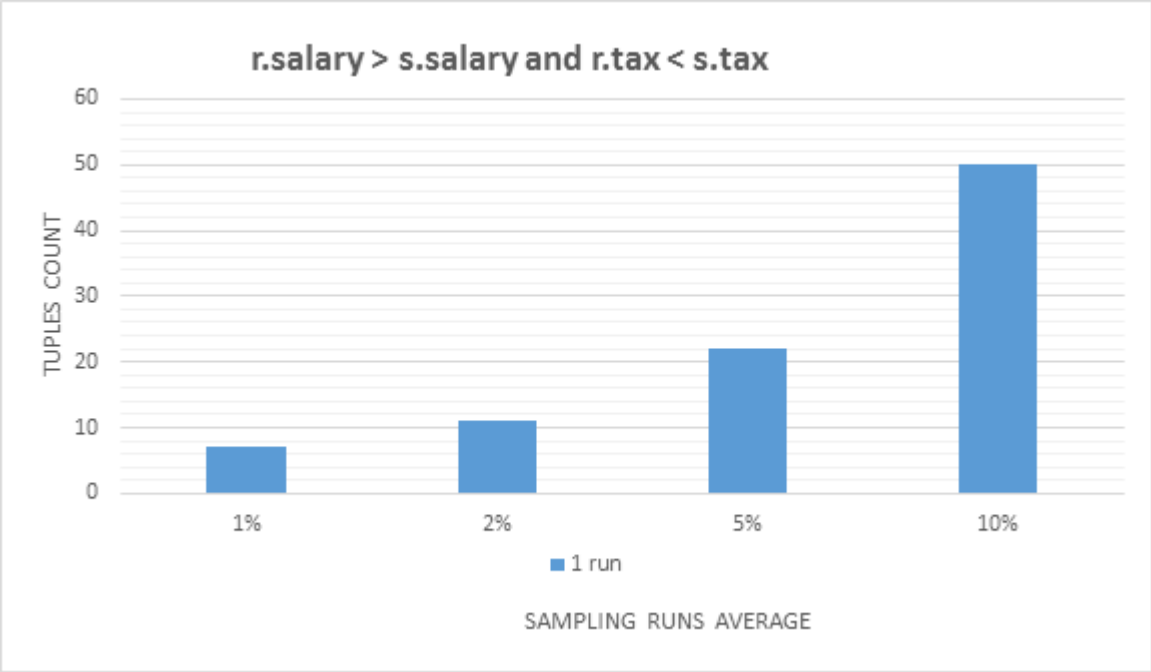
- **UNIFORM SAMPLING:** Taking 1% of records in a sequential order and perform sampling. Then, predicates are rearranged based on the average number of samples resulted.
  - **SAMPLING PERCENTAGE vs TIME TAKEN and TUPLES COUNT**: For average runs correlation table for Dual predicate Inequality Join:
  - **<u>EXPERIMENTS FOR UNIFORM SAMPLING</u>:**
    - **Example Query taken:**
      - select count(*) From F1 r, F2 s, F3 t, F4 u,F5 v
        Where r.salary < r.salary and r.tax < s.tax (1)
        AND s.start < t.end AND s.end > t.start (2)
        AND t.salary > v.salary AND t.tax < v.tax (3)
        AND v.start < w.end AND v.end > w.start (4)
    - **Example table considered** :
      - **r.salary > s.salary and r.tax < s.tax**
        Sample percentage vs Time taken

TIME TAKEN	1 run
1%	0.046
2%	0.056
5%	0.169
10%	0.285



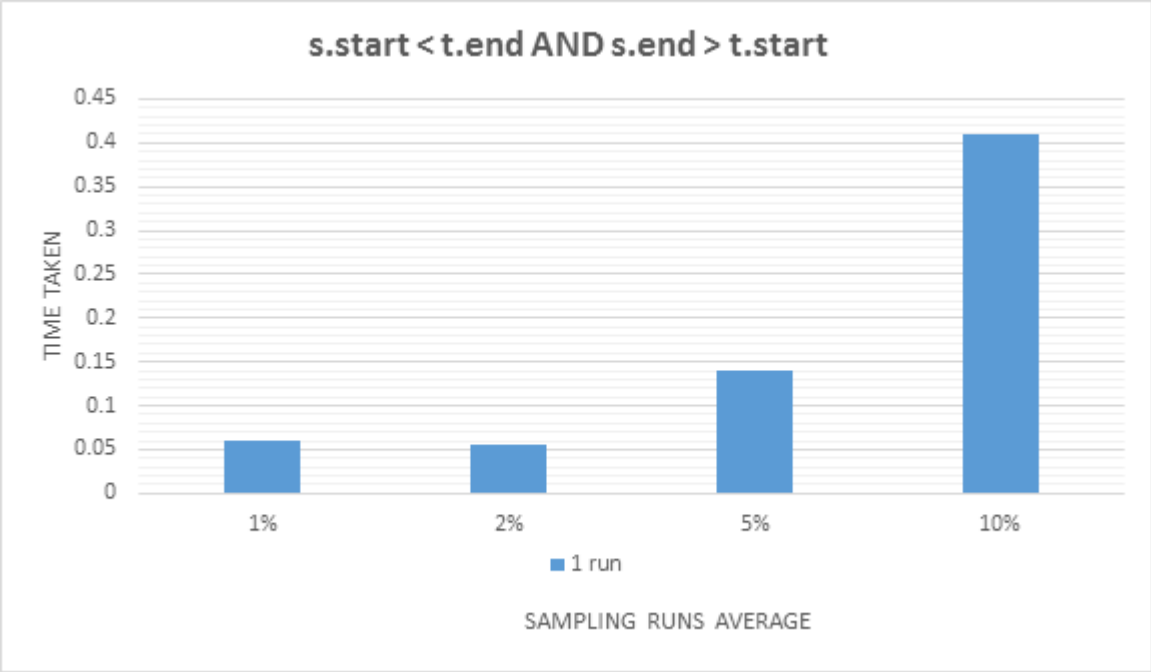
Sample percentage vs Number of Tuples

TUPLES COUNT	1 run
1%	7
2%	11
5%	22
10%	50



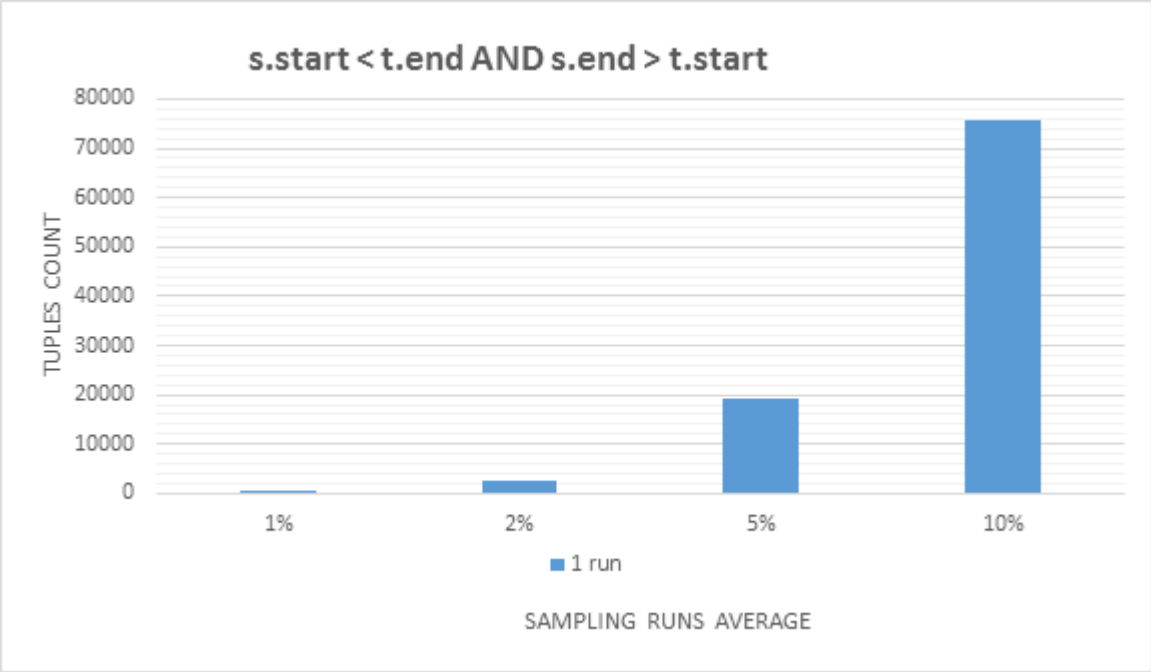
- **s.start < t.end AND s.end > t.start**
  Sample percentage vs Time taken

TIME TAKEN	1 run
1%	0.059
2%	0.055
5%	0.139
10%	0.41



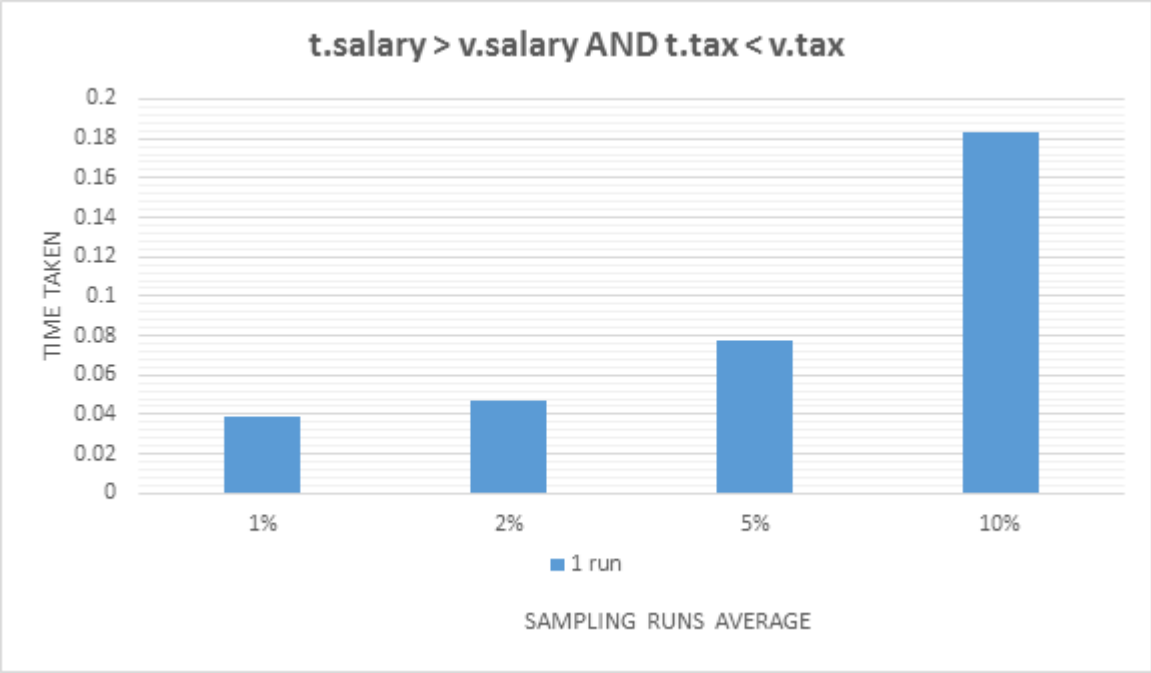
Sample percentage vs Number of Tuples

TUPLES COUNT	1 run
1%	680
2%	2754
5%	19052
10%	75815



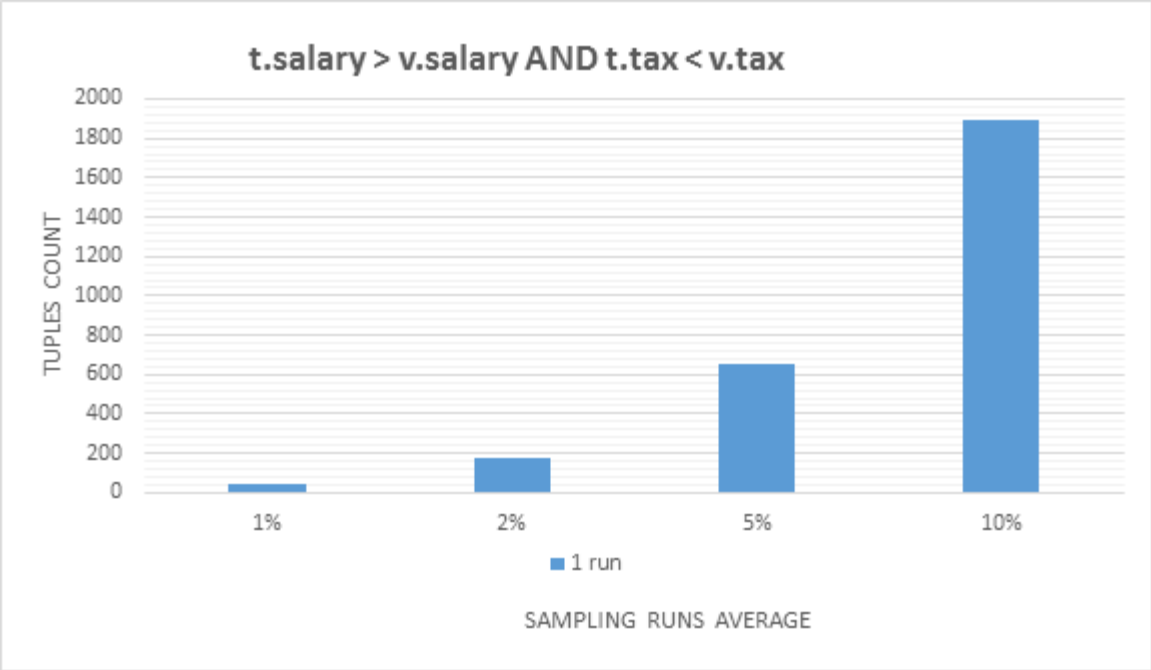
- **t.salary > v.salary AND t.tax < v.tax**
  Sample percentage vs Time taken

TIME TAKEN	1 run
1%	0.039
2%	0.047
5%	0.078
10%	0.183



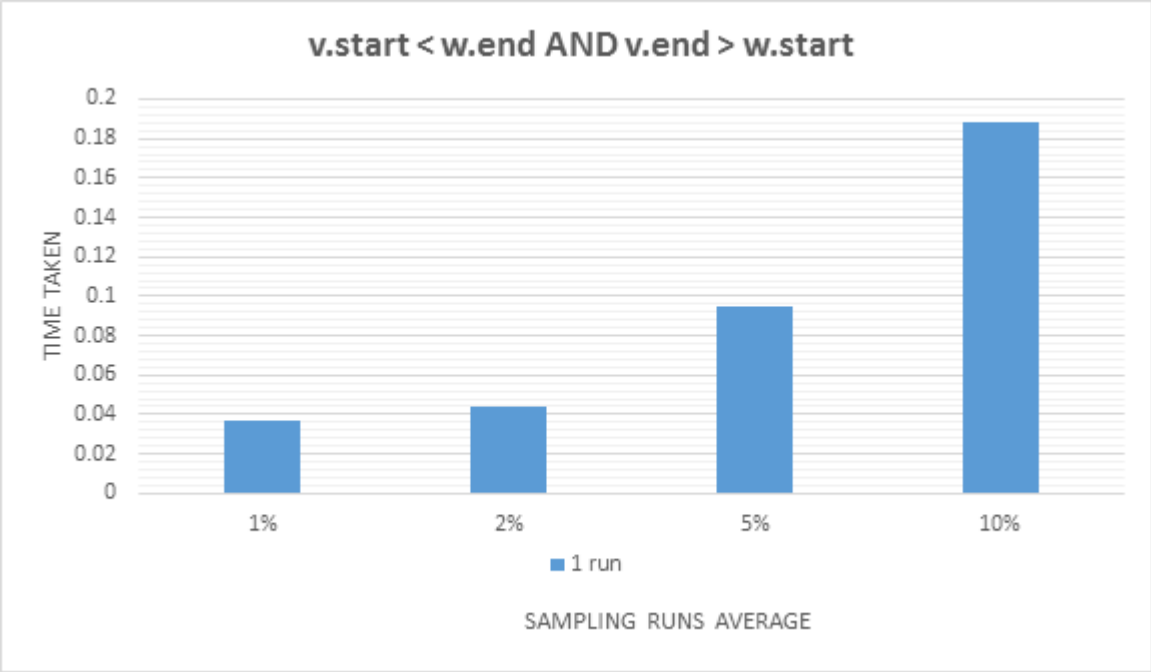
Sample percentage vs Number of Tuples

TUPLES COUNT	1 run
1%	41
2%	172
5%	657
10%	1895



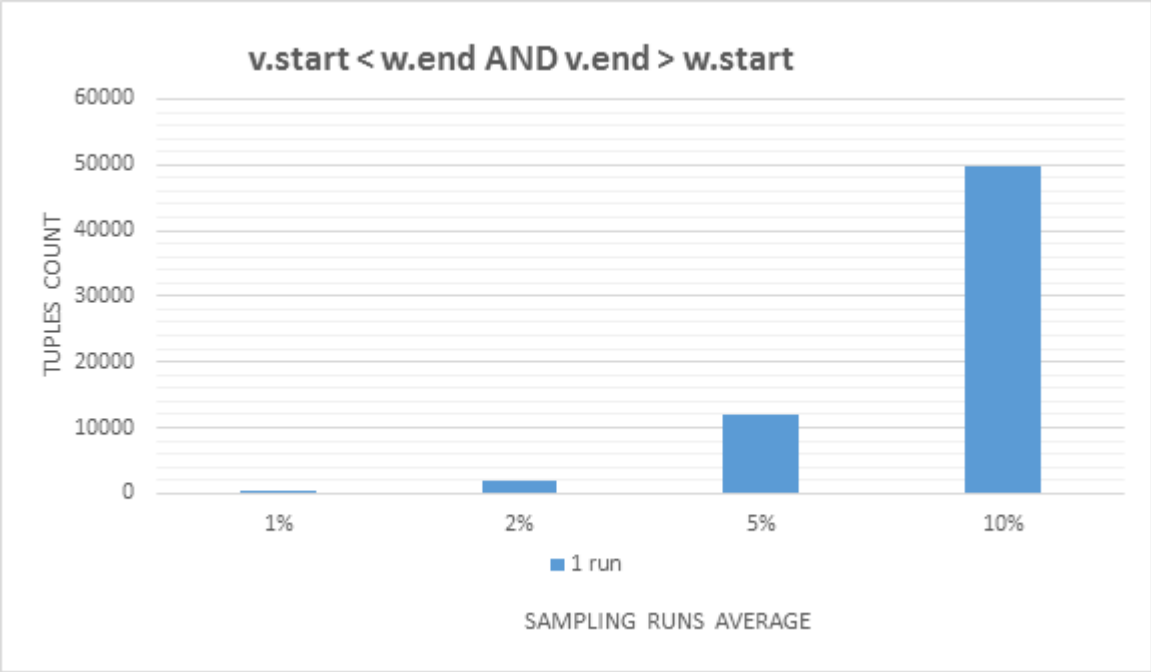
- **v.start < w.end AND v.end > w.start**
  Sample percentage vs Time taken

TIME TAKEN	1 run
1%	0.037
2%	0.044
5%	0.095
10%	0.188



Sample percentage vs Number of Tuples

TUPLES COUNT	1 run
1%	511
2%	1875
5%	11912
10%	49629



HISTOGRAM SAMPLING:

Historical Sampling is a technique in which we are selecting sample tuples from a table only after considering the distribution of the data in it. Suppose that we have 5k rows in a table, if the lowest integer value is 57 and the highest is 100. Our histogram we generate steps ranging between the lowest and highest scores and count the number of results within each step to get a sense of score distribution in the table. Let's say we generate five consecutive steps of similar range sizes, the steps and ranges would be 50 to 60, 60 to 70, 70 to 80, 80 to 90, and 90 to 100, lower bound excluded and upper bound included. The histogram would contain the steps and the number of results that fall within each step.

After generating sets having tuples in their specific ranges, we select tuples evenly from all the sets. In totality 1% of the total number of tuples in the table are considered for sampling but by using Histogram technique the tuples considered are well distributed resulting into the best and optimized order of the predicates which will run the query in the fasted possible way.

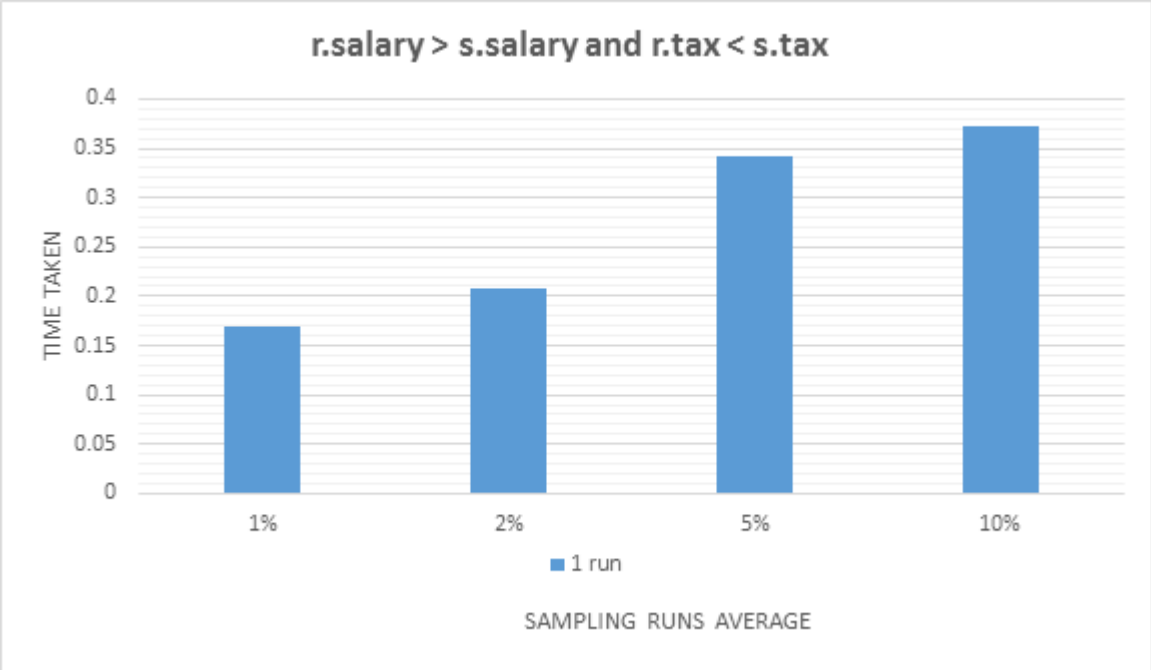
- **Example Query taken**:
  - select count(*) From F1 r, F2 s, F3 t, F4 u,F5 v
    Where r.salary < r.salary and r.tax < s.tax (1)
    AND s.start < t.end AND s.end > t.start (2)
    AND t.salary > v.salary AND t.tax < v.tax (3)
    AND v.start < w.end AND v.end > w.start (4)

EXPERIMENTS FOR HISTOGRAM:

- **Example table considered** :
  - **r.salary > s.salary and r.tax < s.tax**

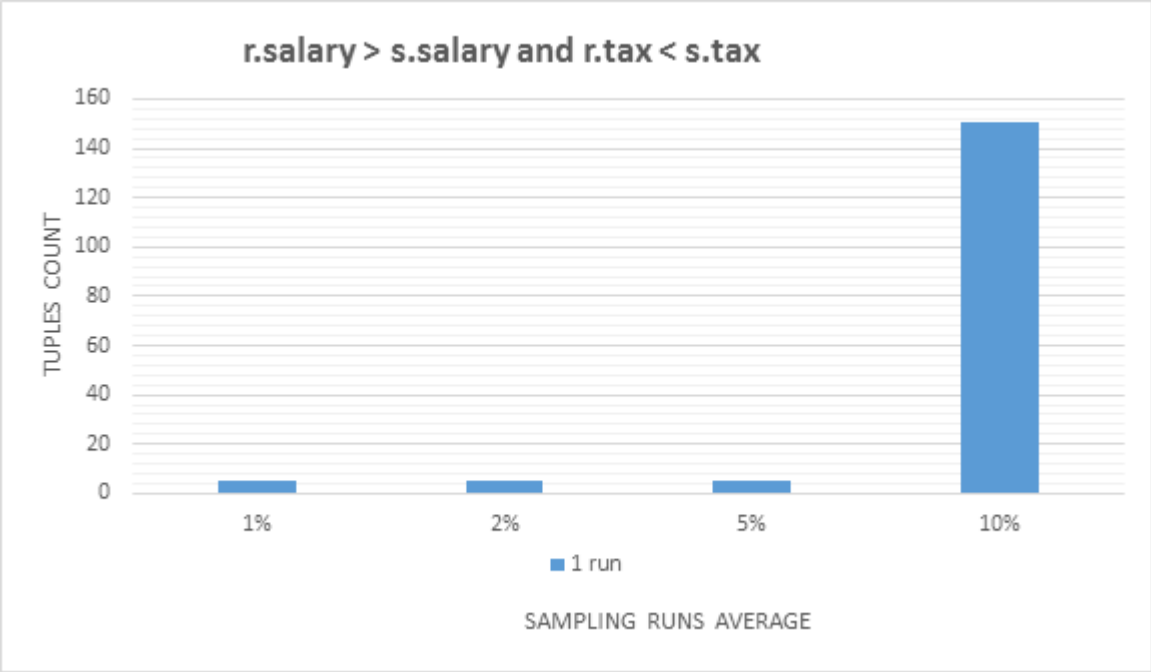
Sample percentage vs Time taken

TIME TAKEN	1 run
1%	0.17
2%	0.208
5%	0.342
10%	0.373



Sample percentage vs Number of Tuples

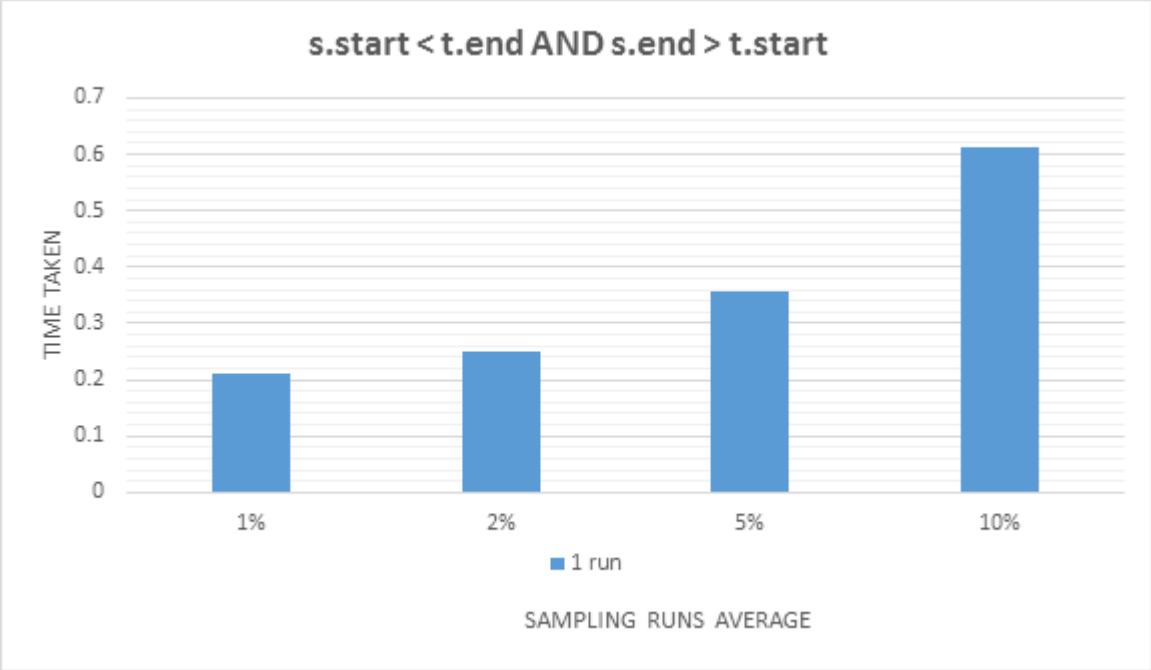
TUPLES COUNT	1 run
1%	5
2%	5
5%	5
10%	151



- **s.start < t.end AND s.end > t.start**

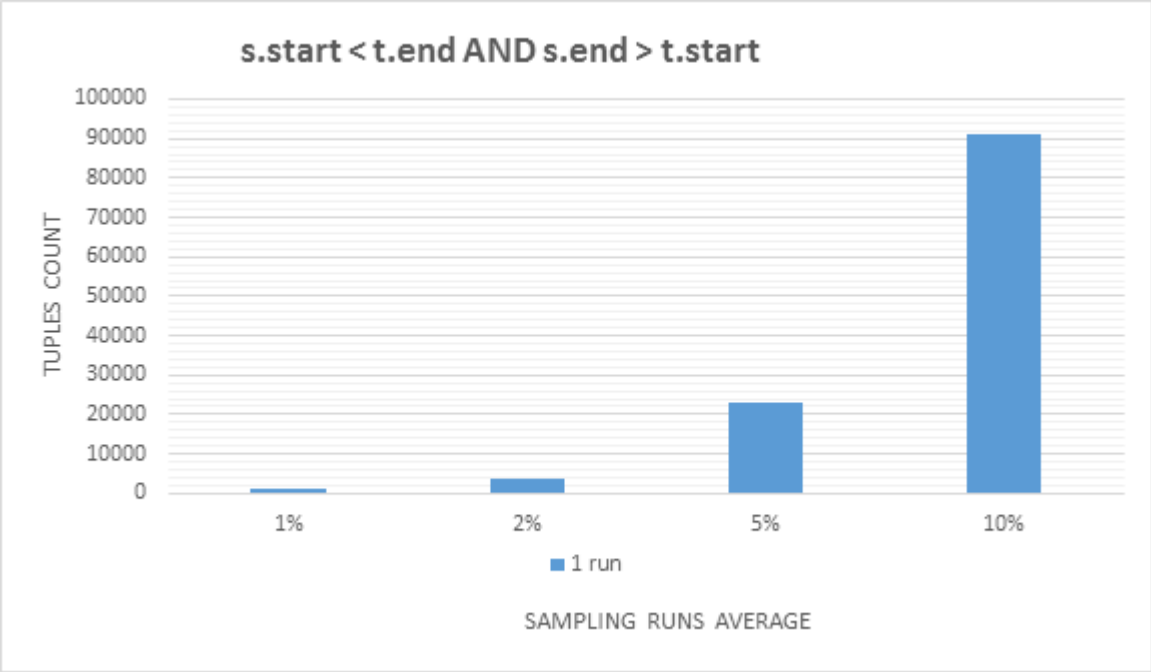
Sample percentage vs Time taken

TIME TAKEN	1 run
1%	0.21
2%	0.25
5%	0.355
10%	0.611



Sample percentage vs Number of Tuples

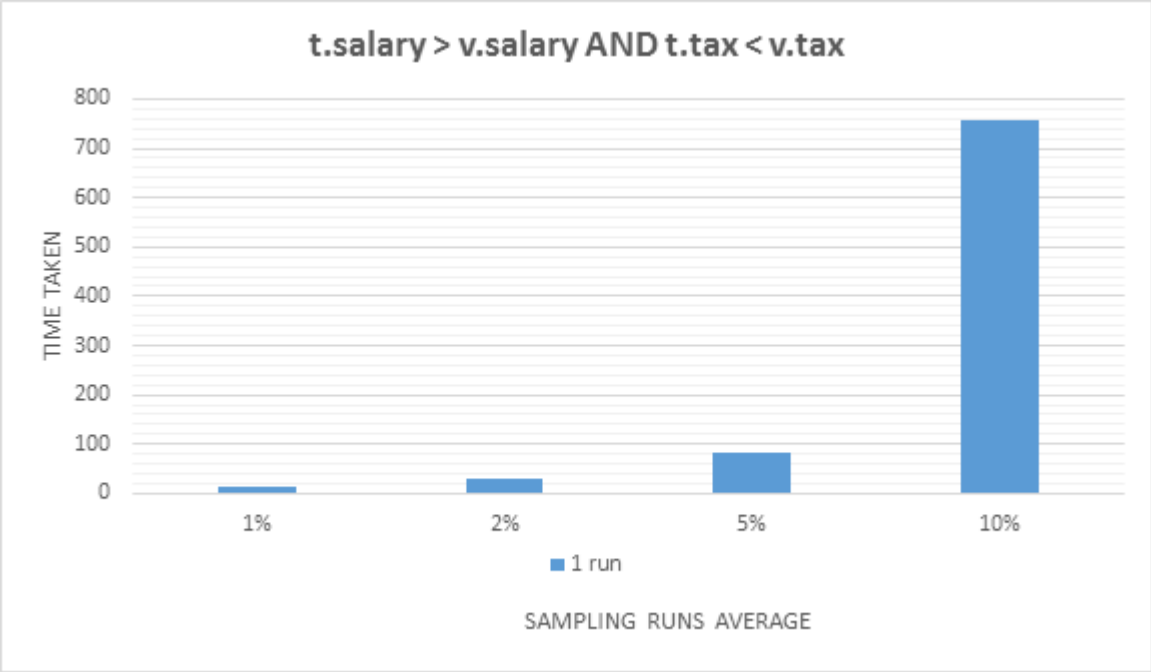
TUPLES COUNT	1 run
1%	927
2%	3664
5%	22790
10%	91163



- **t.salary > v.salary AND t.tax < v.tax**

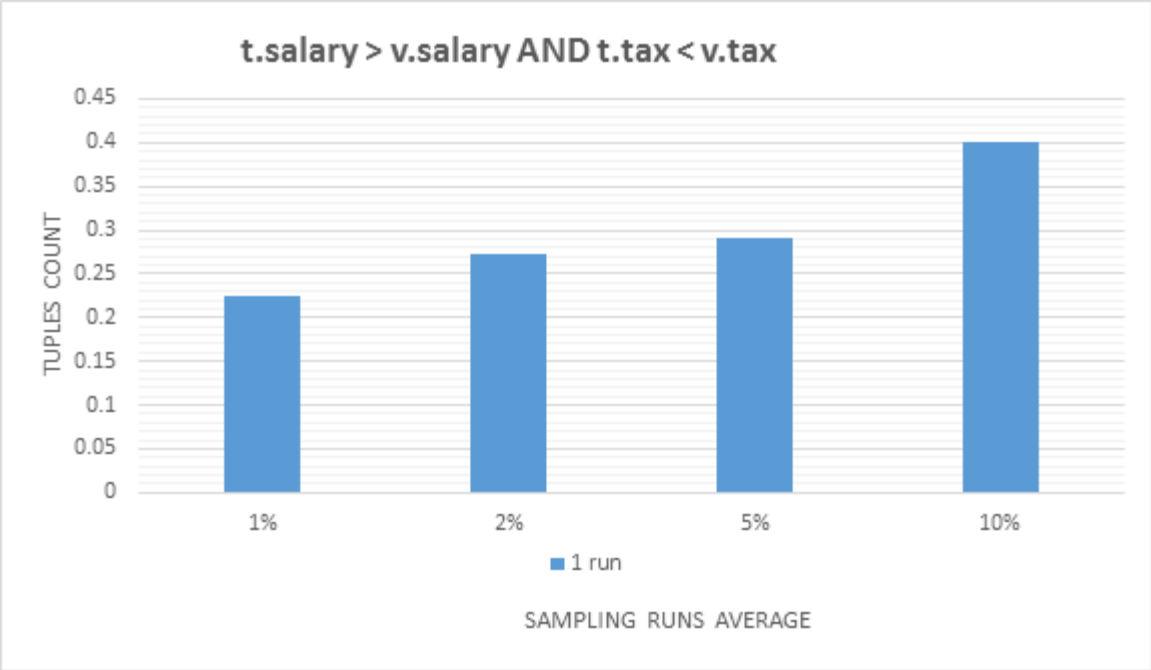
Sample percentage vs Time taken

TIME TAKEN	1 run
1%	14
2%	29
5%	84
10%	758



Sample percentage vs Number of Tuples

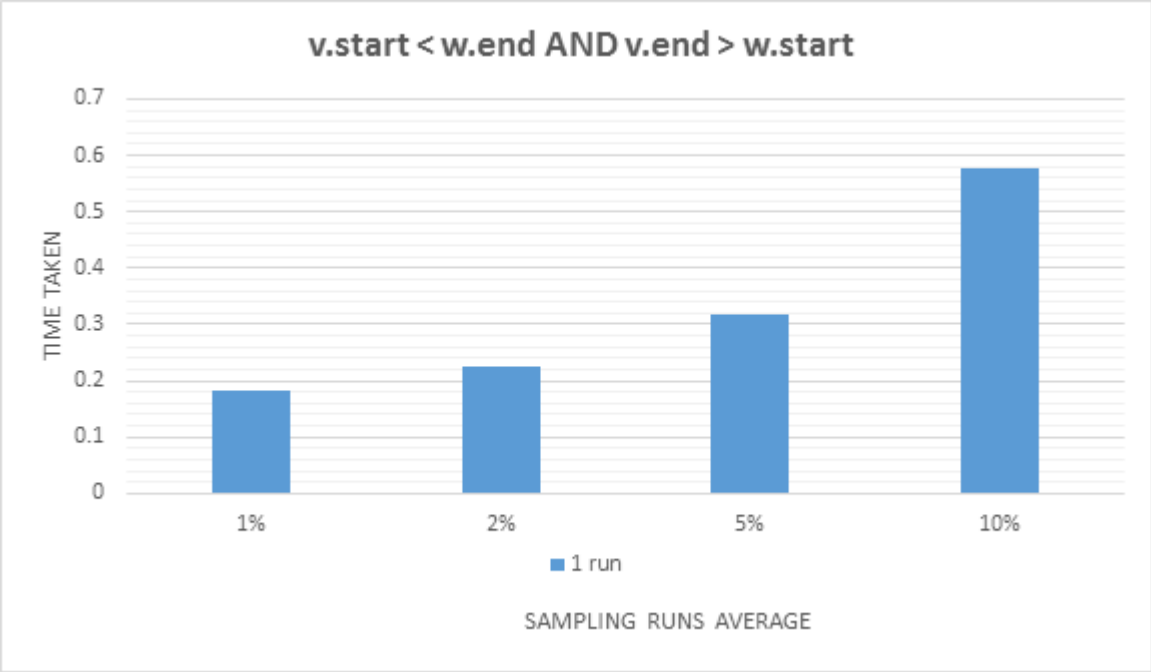
TUPLES COUNT	1 run
1%	0.224
2%	0.273
5%	0.29
10%	0.40



- **v.start < w.end AND v.end > w.start**

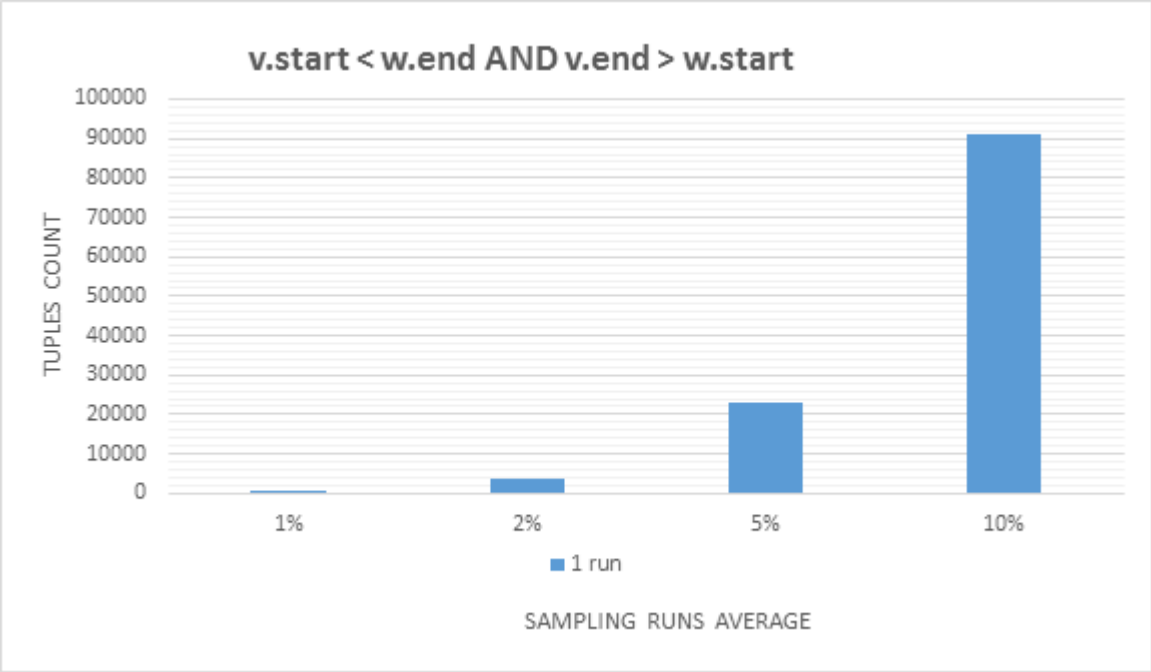
Sample percentage vs Time taken

TIME TAKEN	1 run
1%	0.182
2%	0.224
5%	0.318
10%	0.576



Sample percentage vs Number of Tuples

TUPLES COUNT	1 run
1%	911
2%	3662
5%	22818
10%	90939



TASK 2: JOIN OPTIMIZATION WITH MULTIPLE PREDICATES

- **Purpose**- Determine which two dual predicates order should be joined first thus minimizing the execution time of the given multi predicate query.
- **Pre-processing** (identifying the right ordering.) - First estimate the selectivity of all pair combinations of the join predicates in the query (by using the method from Task 1) which is done above.
- **Second Step** - The remaining predicates will be evaluated on the result of the join.
- **Multi-way inequality Join**- Implemented this with series of two-way inequality joins formed as a left-deep plan.
- **Greedy algorithm** - Number of Joins combinations is large, a greedy algorithm should be used to improve the performance.
    - A simple greedy approach would choose the order of the two-way joins based on their estimated selectivity, i.e., the two-way joins with the higher selectivity are pushed down in the plan.
- **Handling Intermediate Results** -
    - The approach is to support the pipelining of the operators.

- **EXPERIMENTS:**
  - **Experiment 1: Considering 4 dual predicates**
    - **Input Query:**
      - SELECT count(*)
      - FROM F1 r, F2 s, F3 t, F4 v, F5 w
      - WHERE r.salary > s.salary AND r.tax < s.tax
      - AND s.start < t.end AND s.end > t.start
      - AND t.salary > v.salary AND t.tax < v.tax
      - AND v.start < w.end AND v.end > w.start

    - **Optimized order or dual predicates got from Task 1 Sampling:**

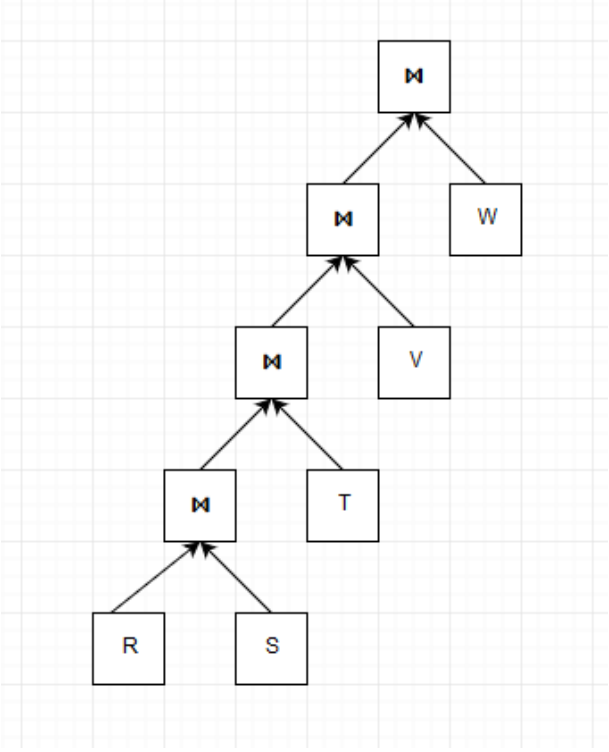
      Order by sorting number of tuples returned after sampling:

      r.salary > s.salary AND r.tax < s.tax
      t.salary > v.salary AND t.tax < v.tax
      v.start < w.end AND v.end > w.start
      s.start < t.end AND s.end > t.start

      But as you can see here, there is no diagonality between 1st and 2nd dual predicates.
      After doing diagonality check, we arrive at this order of dual predicates, which will be used for Join.

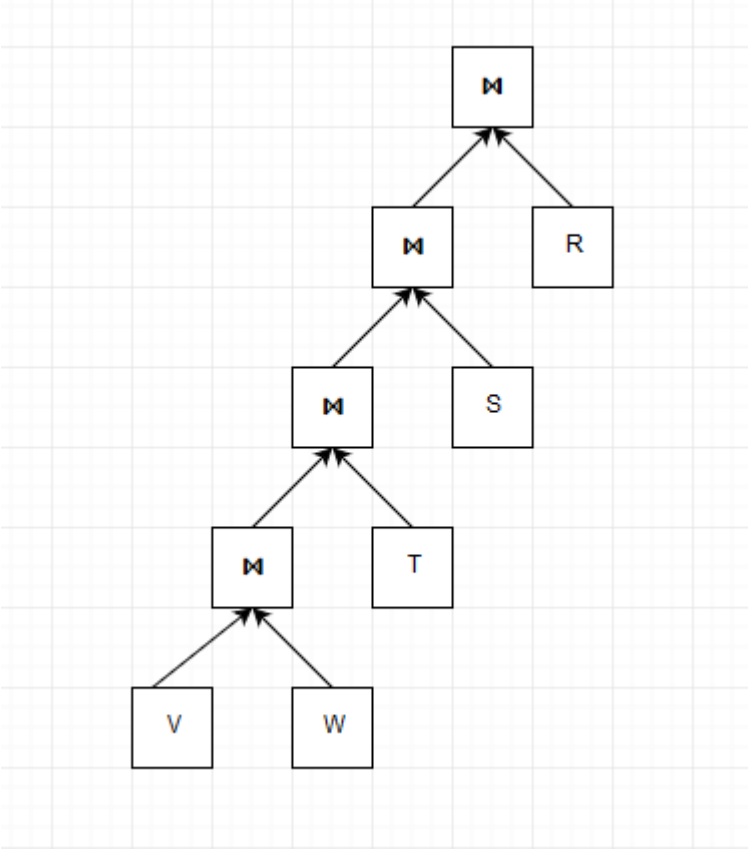
    - **After doing diagonality check (final order of predicates):**

      r.salary > s.salary AND r.tax < s.tax
      s.start < t.end AND s.end > t.start
      t.salary > v.salary AND t.tax < v.tax
      v.start < w.end AND v.end > w.start



- **Random ordering of query:**

  v.start < w.end AND v.end > w.start
  t.salary > v.salary AND t.tax < v.tax
  s.start < t.end AND s.end > t.start
  r.salary > s.salary AND r.tax < s.tax



- **Sampling Techniques Used:**

  - When histogram is used for sampling:

    Time taken for sampling: 0.383
    Overall time for Join: 14.55
    No. of tuples returned: 832667

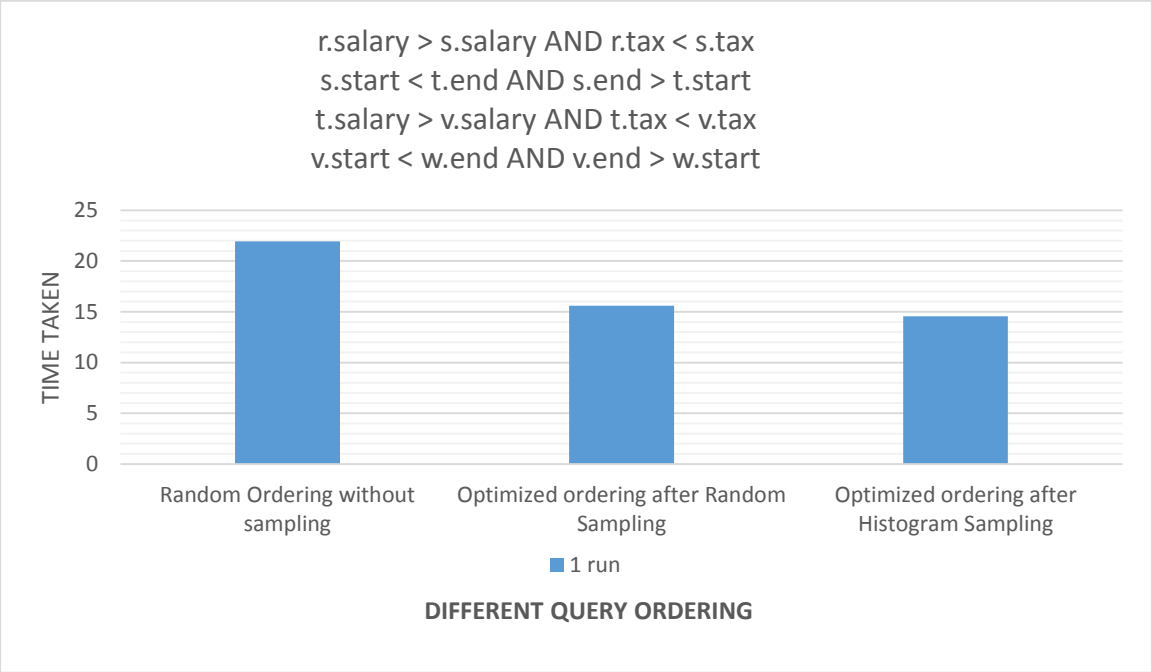
  - When random sampling of 1%:

    Time taken for sampling: 0.118
    Overall time for Join: 15.602
    No. of tuples returned: 832667

  - When Uniform sampling of 1%:

    Time taken for sampling: 0.074
    Overall time for Join: 15.30
    No. of tuples returned: 832667

Query type	Time taken
Query Without Sampling	21.93
Optimized ordering after Random Sampling	15.60
Optimized ordering after Histogram Sampling	14.55



Optimized Query takes more time than running an Input Query in this particular experiment is because, with input query we just pass the query directly to IEJoin, whereas for the optimized query we sort the ordering of dual predicates as per the count of tuples returned and the check for diagonality of dual predicates and re-order them if not, which requires time for computation.

- # EXPERIMENT 2: CONSIDERING 3 DUAL PREDICATES
  - **Input Query:**
    - SELECT count(*)
    - FROM F1 r, F2 s, F3 t, F4 v, F5 w
    - WHERE t.start > w.start AND t.id2 < w.id2
    - AND w.end < v.dept AND w.id2 < v.id2
    - AND s.start < t.end AND s.end > t.start

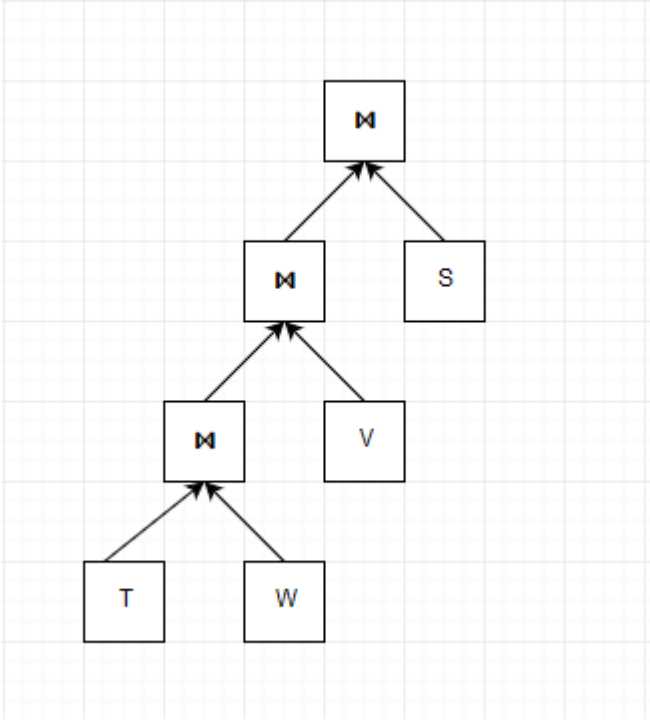
    - **Optimized order or dual predicates got from Task 1 Sampling:**

      Order by sorting number of tuples returned after sampling:

      t.start > w.start AND t.id2 < w.id2
      w.end < v.dept AND w.id2 < v.id2
      s.start < t.end AND s.end > t.startpoda

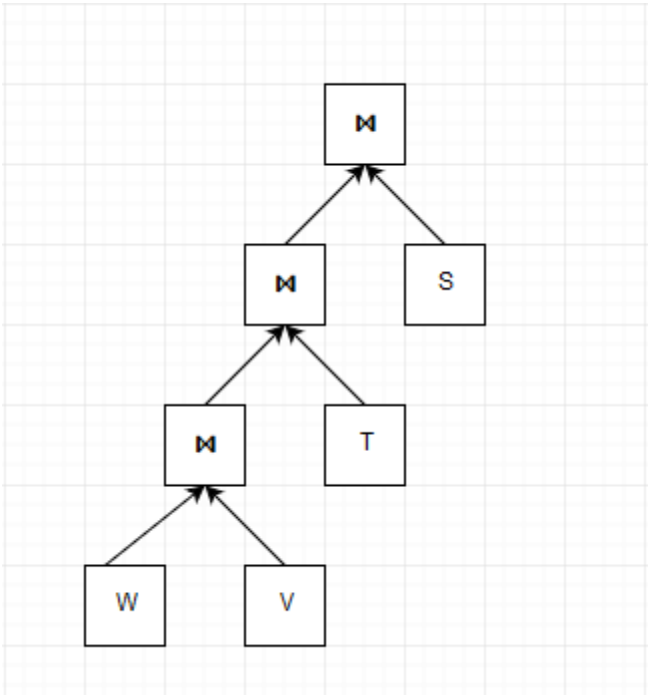
      After doing diagonality check (final order of predicates):

      t.start > w.start AND t.id2 < w.id2
      w.end < v.dept AND w.id2 < v.id2
      s.start < t.end AND s.end > t.start



Random ordering of the query:

```
SELECT count(*)
FROM F1 r, F2 s, F3 t, F4 v, F5 w
AND w.end < v.dept AND w.id2 < v.id2
WHERE t.start > w.start AND t.id2 < w.id2
AND s.start < t.end AND s.end > t.start
```



- **Sampling Technique Used:**
    - When histogram is used for sampling:

The order of dual predicates :

```
t.start > w.start AND t.id2 < w.id2
s.start < t.end AND s.end > t.start
w.end < v.dept AND w.id2 < v.id2
```

Time taken for sampling: 0.306
Overall time for Join: 14.6
No. of tuples returned: 448596

- When random sampling of 1% :

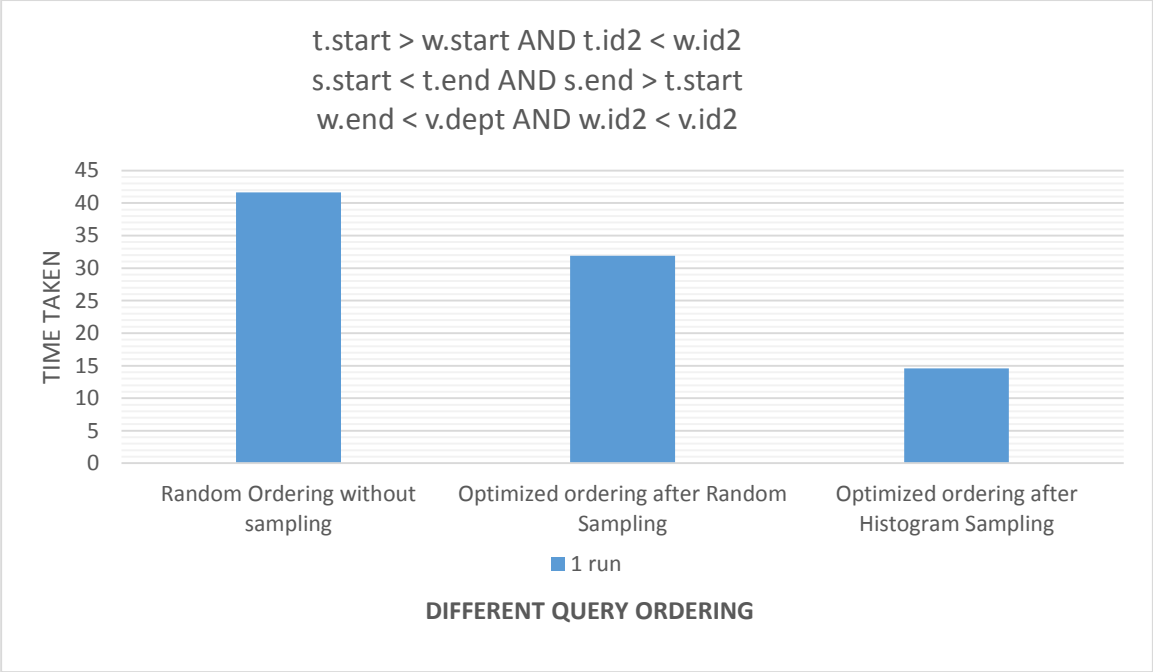
Time taken for sampling: 0.091
Overall time for Join: 31.92
No. of tuples returned: 448596

- When Uniform sampling of 1%:

Time taken for sampling: 0.063

Overall time for Join: 30.012
No. of tuples returned: 448596

Query ordering	**Time taken**
Random Ordering without sampling	41.64
Optimized ordering after Random Sampling	31.92
Optimized ordering after Histogram Sampling	14.575



- **Conclusion:**
  - We have successfully implemented Selectivity Estimation with 4 different sampling techniques followed by accomplishing Join Optimization with Multiple Predicates.
  - The four different techniques for Sampling implemented include Uniform sampling with 1% of samples, Random 1% sampling, Random 1% sampling with average of multiple runs and also Histograms. Histograms on an average gave us the best results eventually.
  - In this report, we have detailed the various experiments we conducted to prove that the technique of sampling for selectivity helps in improving the time taken for join results to a great extent. We have conducted separate experiments for Task 1 to compare the time and quality of results generated for all 4 sampling techniques. In the end we have conducted 2 experiments for Task 2 , one experiment with 4 dual predicates and other experiment with 3 dual predicates.
  - After all the experiments, histogram sampling takes more time for sampling, it gives a better dual predicates order by which time take for the entire query is reduced by around 50% when compared to Random or Uniform Sampling.

- Between Uniform and Random Sampling, Uniform Sampling takes less time for sampling but the difference between time taken for Uniform Sampling and Random Sampling is negligible. Taking into consideration the tradeoff and advantage of considering the random sampling for better ordering result, we have chosen random sampling with 2 runs.
- The various experiments detailed in this report clearly concludes that this implementation is more scalable and several orders of magnitude faster compared to simple IE Join Algorithm implemented in the previous phase.