DBSys : Database Management System Implementation

Due: January 9th, 2019, 11.59pm

INTRODUCTION

In the previous phase of the project you implemented different page replacement policies to support buffer management. For this phase of the project, you will experiment with a join operation. A join operation combines related tuples from same or different relations on different attributes schemes.

In particular, you will implement a novel operator for inequality joins and deliver a comparative analysis of different join implementations. Inequality joins have the following operators in the conjuncts of the join condition: $<$ (less than), $\leq$ (less than or equal), $\geq$ (greater than or equal) and $>$ (greater than).

TASK 1

Write tests to use nested-loop join (NLJ) operation to join two tuples for the given inequality condition. If necessary, extend the code to support the task. You can start with simple relations as the ones in the test, but also add support to load the provided (larger) relations.

Task 1a - Test/extend the existing NLJ with single predicate inequality joins.

Task 1b - Test/extend NLJ with two predicates inequality join.

TASK 2

Write a program to implement an inequality join operation using sorting, permutation array, and bit array. Refer to the following paper for this task:

Khayyat Z. et al. "Fast and scalable inequality joins" VLDB Journal, 2017. You can ignore incremental and distributed versions of the join.

Task 2a - Implement *single predicate* self join operation.

Task 2b - Extend the single predicate inequality self join (*Task 2a*) to two predicates in the condition of the inequality **self join**.

Task 2c - Implement two predicates inequality join.

Task 2d - Optimize the inequality joins created in *Task 2a - 2c* to improve the bit-array scan operation. You can use the indexing method in the paper, or any other optimization (explain the rationale for your choice in the documentation).

It is recommended that you implement the join operations in the order they are listed. In general, single predicate joins are easier to understand and evaluate/debug. Notice that a decision that you have to make is about HOW to implement the join: following the Iterator interface or a blocking operation.

INPUT FORMAT

In the archive QueriesData.zip, you are given a set of text files named query_name.txt, where X is the query number.

Single Predicate Query:

LINE 1: $Rel1\_col\#$ $Rel2\_col\#$
LINE 2: $Rel1$ $Rel2$
LINE 3: $Rel1\_col\#$ op_1 $Rel2\_col\#$

Two Predicates Query:

LINE 1: $Rel1\_col\#$ $Rel2\_col\#$
LINE 2: $Rel1$ $Rel2$
LINE 3: $Rel1\_col\#$ op_1 $Rel2\_col\#$
LINE 4: AND/OR
LINE 5: $Rel1\_col\#$ op_2 $Rel2\_col\#$

where,

$Rel1$ and $Rel2$ are the names of the relations being joined, $col\#$ is the column number from left to right in the relation, op_X is an integer value that denotes an inequality operator (1 for $<$, 2 for $\leq$, 3 for $\geq$ and 4 for $>$). The queries above contain the words LINE 1..LINE 5 only for the purpose of referencing.

For the test data, in archive QueriesData.zip you are given relation_name.txt where Line1 will be the header and each of the following lines denotes a tuple in the relation. For example,

attrString,attrInteger,attrInteger,attrInteger
John,0,344345,232423
Kevin,1,543434,353353

It is recommend to verify the correctness of the implemented algorithms. A straightforward strategy is to first test with very small instances in the input relation (up to a dozen of tuples) to manually verify the output and check corner cases. For larger instances, a natural test is to compare the output of the Nested Loop vs the output of Self Join and Inequality Join (depending on the kind of query).

Furthermore, it is recommended that you implement each task as an independent program such that no two join operations are dependent on each other for their successful completion. Also, make sure that your program is not file name dependent or sample dependent. Your implementation must be generic and is able to run with different databases (with different attribute schema) and different queries.

DELIVERABLES

This is a **group** submission.

- You will create three directories: Code, Output and Report. In Code directory, you will copy your code. In output, you will submit the the query files(query_name.txt) that you ran and the resulting joined tuples(Query_Name_Task_Number.txt). In report directory you will put your report.

- Make sure to have a "README"-like section in the report with instruction to run your code.
- Report must comprehensively discuss different join operations and clearly present their comparative evaluation and your interpretation of the evaluation. Examples of the experiments you can run are plots with increasing number of tuples (x axis) and the execution time of the algorithms implemented in Minibase (y axis).

Please ZIP your submission, name it as Group_X, where X is your group number, and upload it in Moodle under the new assignment link (IELJoin).

Getting started

Suggested steps to get started
Joins:

- Read the slides and the book for nested loop join and iterator interface.
- Read the Minibase documentation online including:
  - `http://research.cs.wisc.edu/coral/mini_doc/intro/single_user.html` (focus on iterator)
  - `http://research.cs.wisc.edu/coral/mini_doc/joins/joins.html`
  - `http://research.cs.wisc.edu/coral/mini_doc/planner/iterator.html`
  - `http://research.cs.wisc.edu/coral/mini_doc/joins/interface/joins.h` (focus on nested loop)
- Run again the join tests, focusing on the nested loop example in Query 2.
- Look at the NestedLoopsJoins.java file in iterator folder. This will lead you to more files, such as PredEval.java. Don't forget that you have the documentation for these Java files.
- Once you have a clear mind about the NLJ data processing and assembling of condition expressions, you can start looking at how to use it for inequality joins.
- Condition expression is the core of nested-loop join parameters, in which you will have operator, type and symbol of two operands and the next iterator. The class Operator has various values representing different operators, such as Greater Than or Less Than or Equal to. Here if where we can set the operator according to the predicate to be implemented. Later on, for two predicates joins, you can set each condition expression separately as above and combine them as an array.

Loading data:

- Data comes from text files R.txt, S.txt and Q.txt. An empty minibase database should be created and each input data file inserted one record at a time into the database. The heap file can be used for this purpose. The initial size of the database should be set to accommodate the entire dataset (you know the number of rows and the types, so it is a simple operation considering 1024 bytes per page).