CSI 3030 Project Guidelines

Objective

- Implement a new symmetric hash join query operator replacing the current hash join implementation.
- Modifications to be done in Optimizer and Executor component

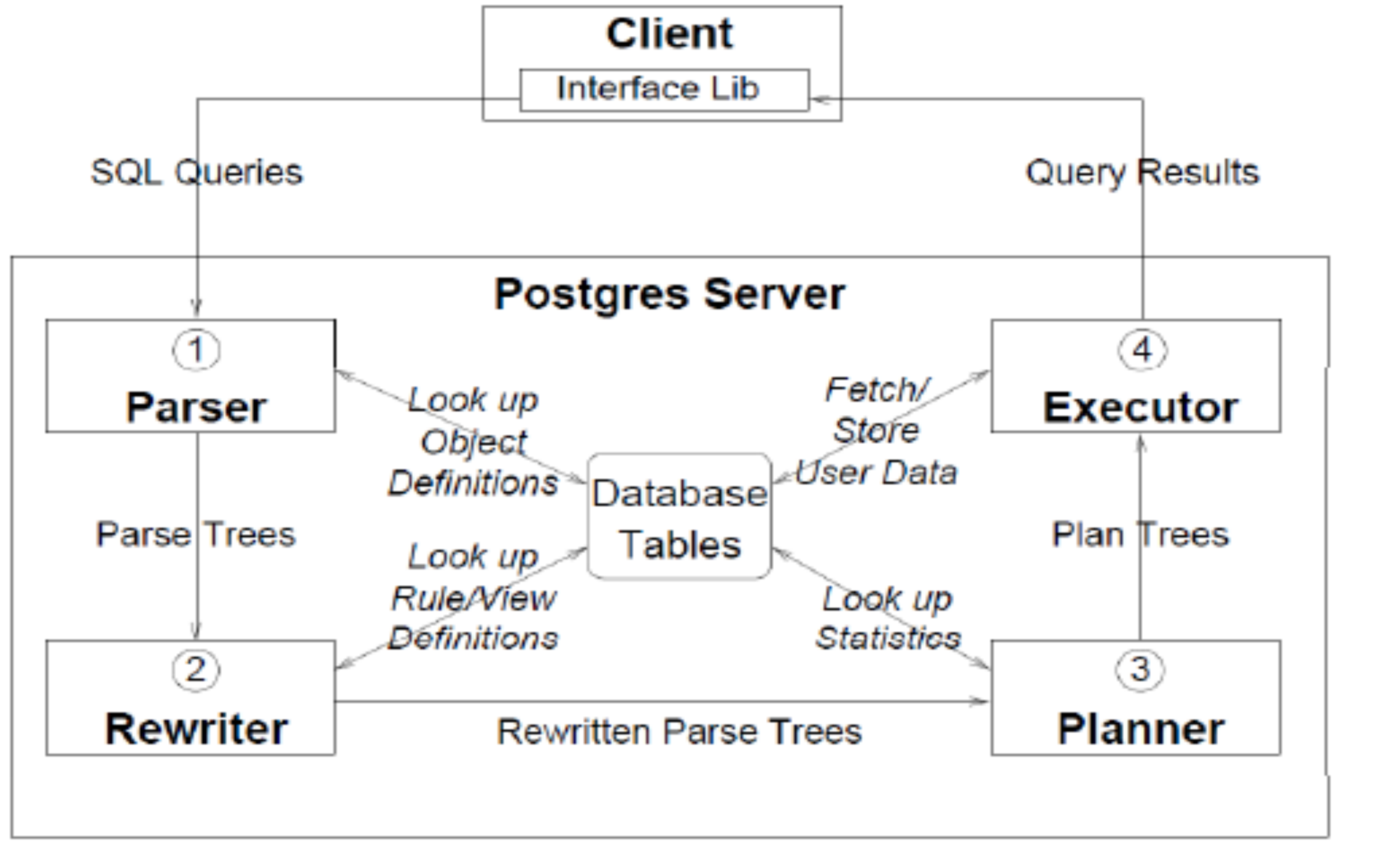
What you need to know

- Understand what a hash join is.
- How is it implemented in POSTGRESQL.
- Need to know what all files to be modified.
- Understand POSTGRESQL backend architecture.

What you are to implement

- Symmetric Hash Join

PotgreSQL Backend



Reference: Tom Lane, A Tour of PostgreSQL Internals

A Tour of PostgreSQL Internals https://www.postgresql.org/files/developer/tour.pdf

More reading to do

- Understand how Optimizer and Executor works
- Refer to src/backend/optimizer/README
- Refer to src/backend/executor/README

Relevant Files

- Src/backend/executor
  - nodeHashJoin.c: This file implements the actual processing of the hash join operator.
  - nodeHash.c: This file is responsible for creating and maintaining a hash table.

*You will be given a list of methods to be modified for this Project (***slide 19 onwards***).

Relevant Files

- src/backend/optimizer/plan/
  - createplan.c: This file contains the code that creates a hash join node in the query plan.
- src/include/nodes/
  - execnodes.h: This file contains the structure HashJoinState that maintains the state of the hash join during execution.

Deliverables

- All of the **relevant files** listed in the two previous slides to zipped and submitted
- Send the zipped file to any one of the TA's emails. Copying the other two **lab** TAs' (emails available here https://github.com/ferna11i/3130-DBMS)
- Insert comments in the parts you have modified. Also add in any necessary explanations.
- Need a test query which would execute your code as per the requirement. Test Query is available here (schema.txt, query.txt, output.txt) (https://github.com/ferna11i/3130-DBMS/tree/master/Project)
- All comments in code to be preceded by 'CSI3130:'
- **Project Deadline: Monday December 9 at 6:00 pm at the latest.**

Additional Help

- http://doxygen.postgresql.org/: Source code browser
- ELOG in POSTGRESQL

Hash Join & Symmetric Hash Join explained

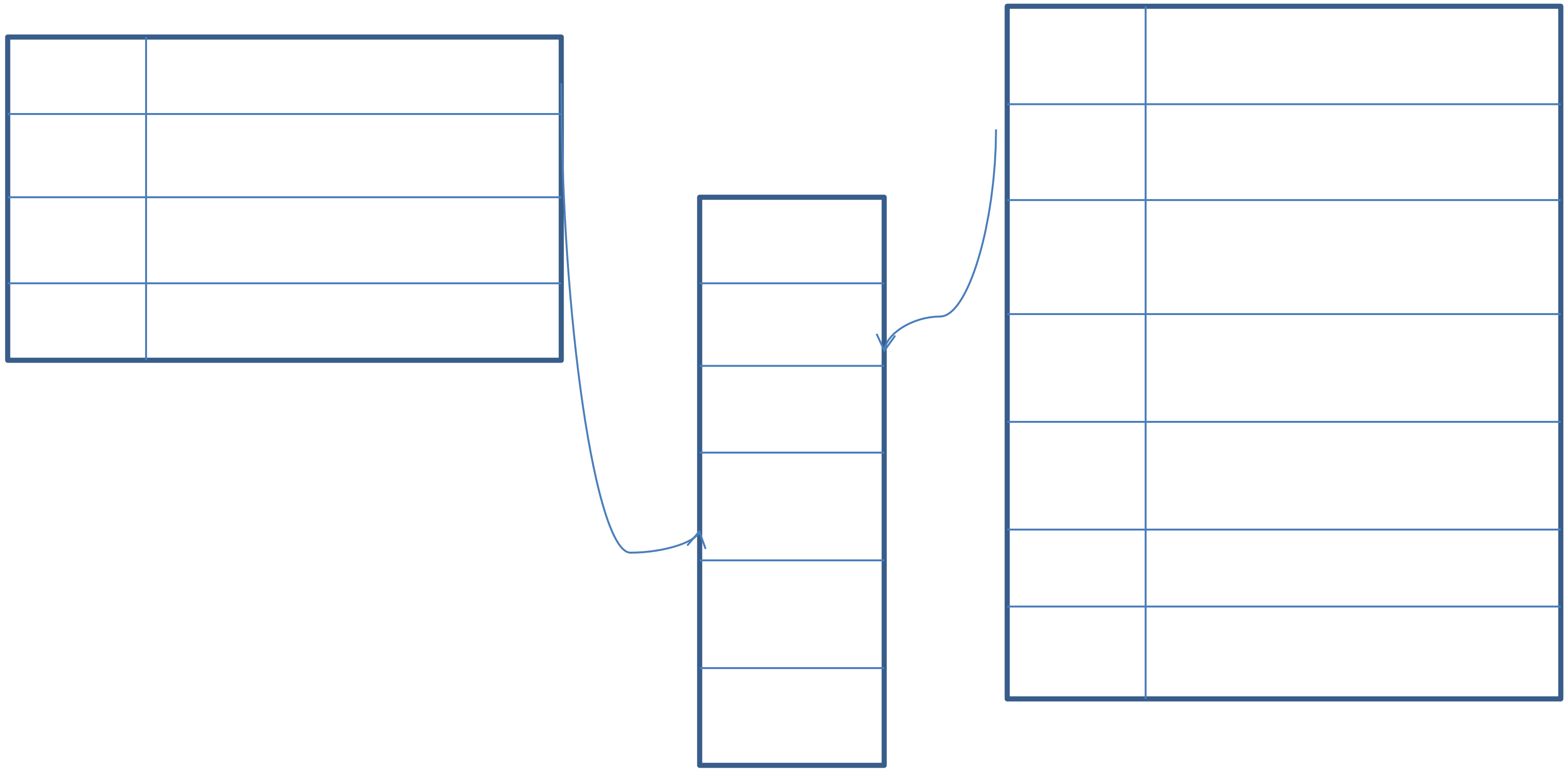
What is a JOIN

- Between tables or self
- Query accessing multiple rows of the same or different tables
- Order of execution : Sequential

Hash Join

- Loads candidate records from one side of the join into a hash table.
- Probe for each record from other side of the join
- Purpose of the Hash Table - Indexing

Hash Join



Facts about hash joins

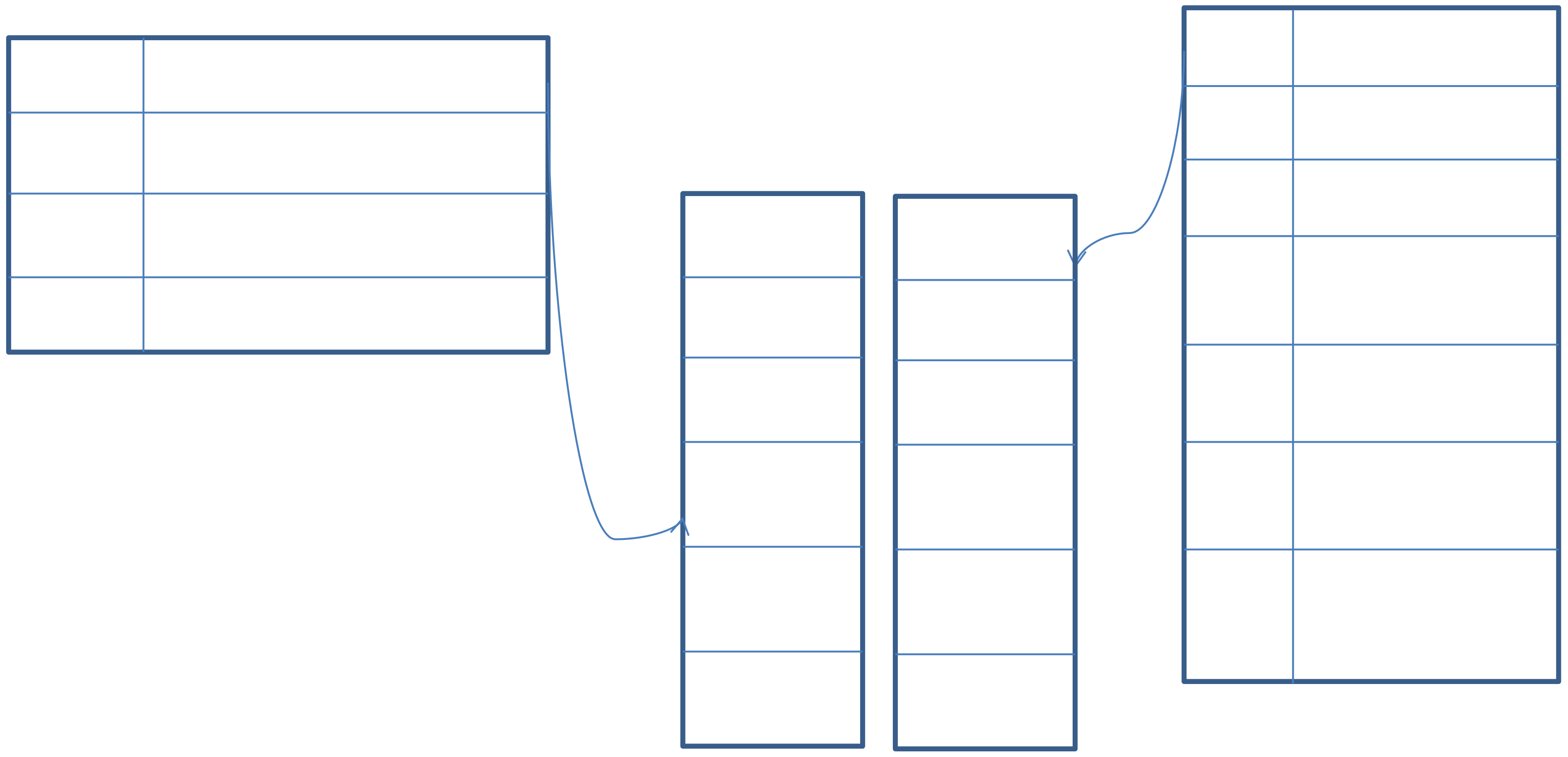
- Hash joins do not need indexes on predicates
- Reduce the hash table size to improve the performance
- Cannot perform joins that have range conditions in the join predicates

Drawbacks with Hash Join

- Bottleneck in query execution pipeline
- Sufficient memory required to store inner relation.
- Hybrid Hash Join!!!

Symmetric Hash Join

- Maintains two hash tables
- Two Hash functions



Other requirements to be considered

- Should run until it gives one output tuple each time
- State should be saved – should include detail on which tuple is running using state node.

Further Project Details

Disabling use of Multiple Batches

- Present implementation
  - Hybrid hash join
  - Tuples from inner / outer are considered in batches

Changes in nodeHash.c

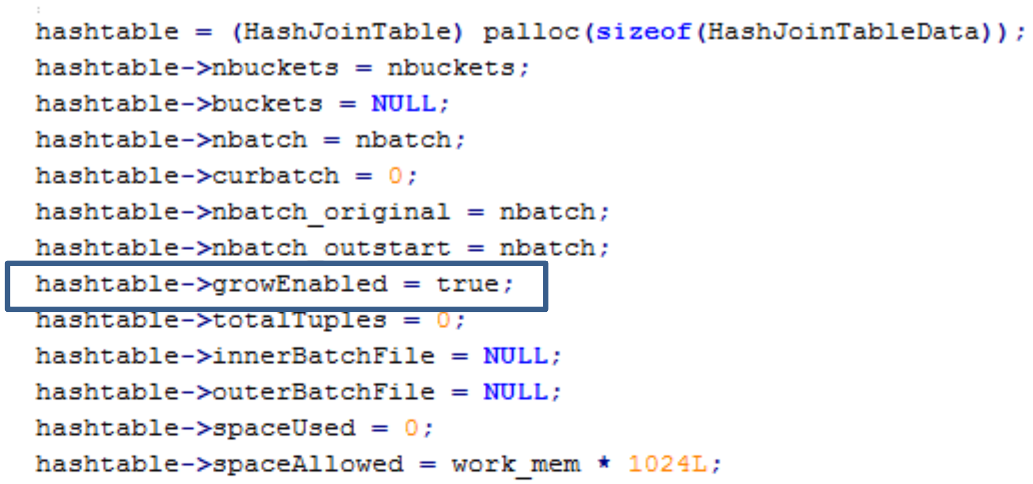
- Modify ExecHash
- Disable Batches
- Implement ExecScanHashBucket for both Inner and Outer

ExecHash

- Remove the present error message
- It should have the same code as MultiExecHash

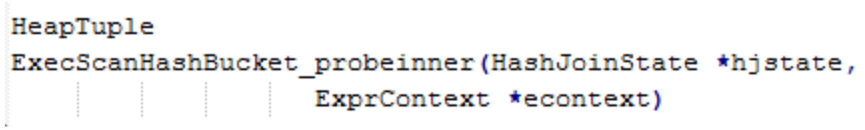
Where to disable?

- nodeHash.c
  - ExecHashTableCreate



```
hashtable = (HashJoinTable) palloc(sizeof(HashJoinTableData));
hashtable->nbuckets = nbuckets;
hashtable->buckets = NULL;
hashtable->nbatch = nbatch;
hashtable->curbatch = 0;
hashtable->nbatch_original = nbatch;
hashtable->nbatch_outstart = nbatch;
hashtable->growEnabled = true;
hashtable->totalTuples = 0;
hashtable->innerBatchFile = NULL;
hashtable->outerBatchFile = NULL;
hashtable->spaceUsed = 0;
hashtable->spaceAllowed = work_mem * 1024L;
```

Changes to scanhashbucket



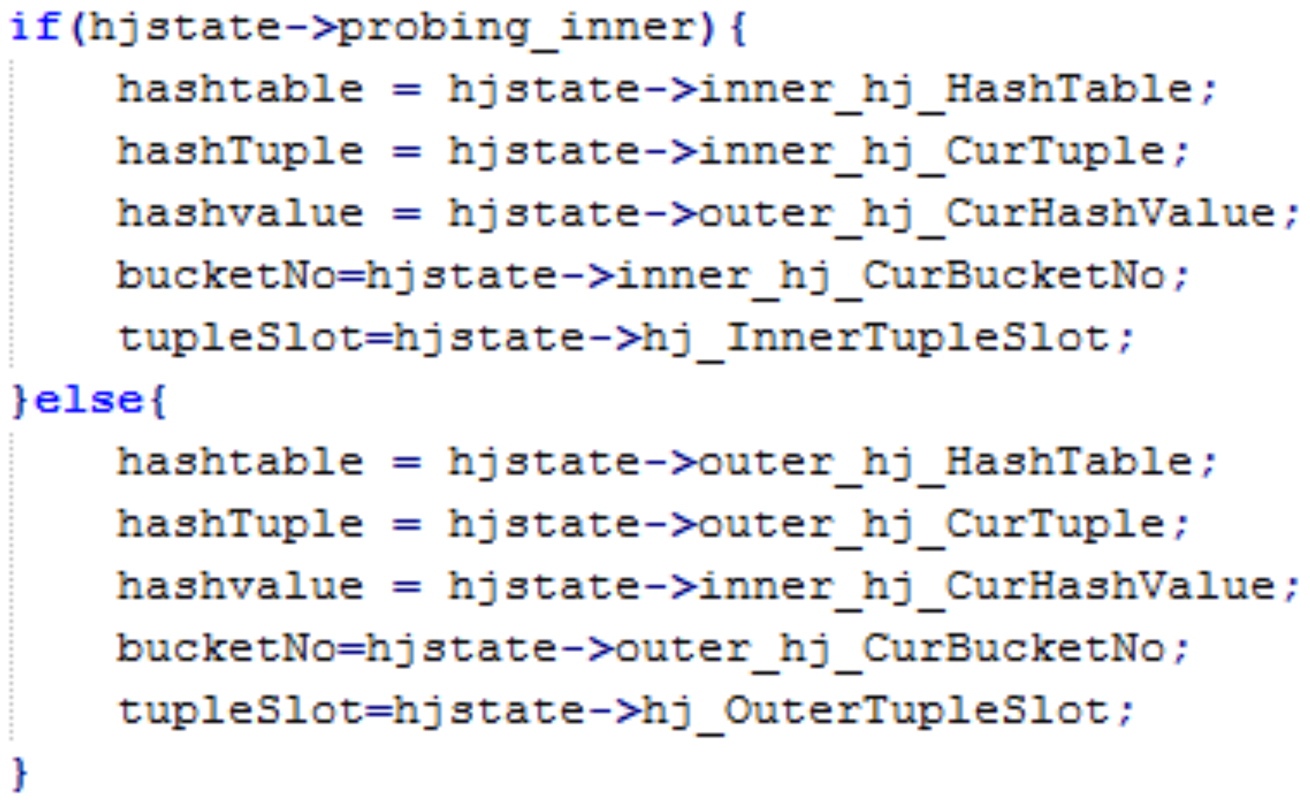
```
HeapTuple
ExecScanHashBucket_probeinner(HashJoinState *hjstate,
                              ExprContext *econtext)
```

- Present implementation
  - Two functions : one for probing inner and other for outer
  - Returns HeapTuple

Changes to scanhashbucket

- Need only one method for scannning both hash tables
- Return HashTuple instead of HeapTuple
- Check for type of hash table in the method : whether its inner or outer.

Sample Implementation Code



```
if(hjstate->probing_inner){
    hashtable = hjstate->inner_hj_HashTable;
    hashTuple = hjstate->inner_hj_CurTuple;
    hashvalue = hjstate->outer_hj_CurHashValue;
    bucketNo=hjstate->inner_hj_CurBucketNo;
    tupleSlot=hjstate->hj_InnerTupleSlot;
}else{
    hashtable = hjstate->outer_hj_HashTable;
    hashTuple = hjstate->outer_hj_CurTuple;
    hashvalue = hjstate->inner_hj_CurHashValue;
    bucketNo=hjstate->outer_hj_CurBucketNo;
    tupleSlot=hjstate->hj_OuterTupleSlot;
}
```

Deliverables

- All of the relevant files listed to be submitted
- Send in the Zipped file to Any of the TA's emails. Copying the other TAs (emails available here https://github.com/ferna11i/3130-DBMS)
- Insert comments in the parts you have modified. Also add in any necessary explanations.
- Need a test query which would execute your code as per the requirement. Test Query is available here (schema.txt, query.txt, output.txt) (https://github.com/ferna11i/3130-DBMS/tree/master/Project)
- All comments in code to be preceded by 'CSI3130:'
- **Project Deadline: Monday December 9 at 6:00 pm at the latest.**