CSI 3130 - Project Guidelines

Created	@October 17, 2022 2:50 PM
Class	CSI 3130
Type	Lab

Objective:

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

Requirements:

- Understand what a hash join is.
- How is it implemented in POSTGRESQL.
- Need to know what all files to be modified.
- Understand the POSTGRESQL backend architecture.
- Understand how Optimizer and Executor works in PostgreSQL
  - Refer to src/backend/optimizer/README and src/backend/executor/README

Files to be modified:

If you have Linux installed on a VM, you can access these files where you downloaded the postgreSQL source code. (Eg. in your downloads directory).

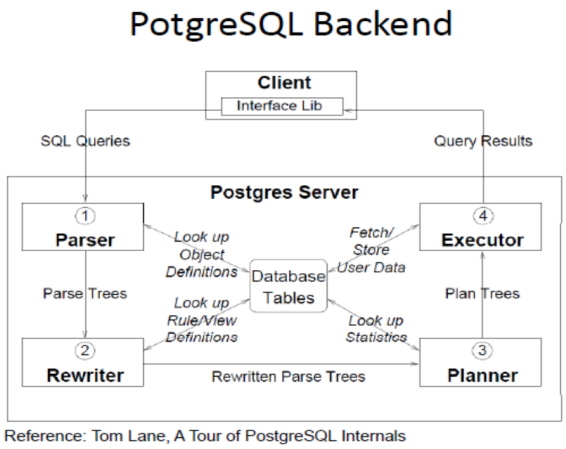
If you are using WSL, the source code files can be found here:

> C:\Users**<Windows user>**\AppData\Local\Packages\CanonicalGroupLimited.UbuntuonWindows_79rhkp1fndgsc\Loca
>
> rootfs\home\postgres

- 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.
- 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.

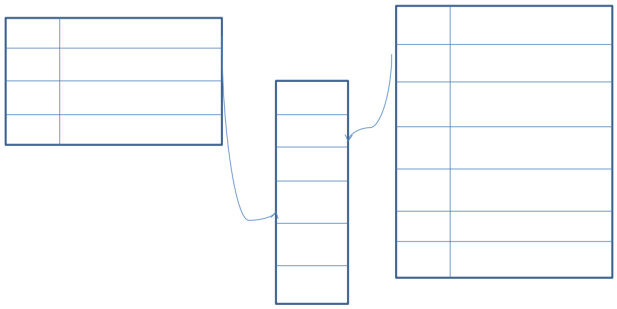
PostgreSQL Backend architecture:

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



Hash Join

- Loads candidate records from one side of the join into a hash table.
- Probe the hash table for each record in from other side of the join.
- Hash joins only works where the join condition is an equality condition.



Drawback:

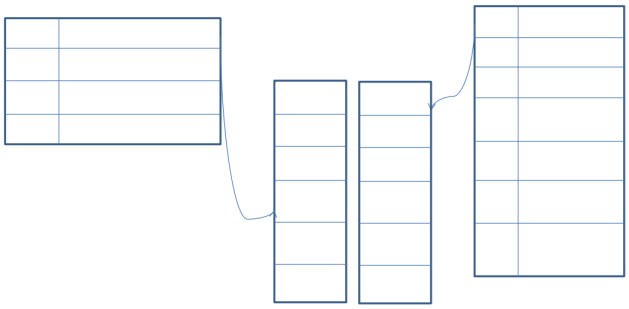
- Sufficient memory required to store inner relation.

PostgreSQL's solution - Hybrid Hash Join

- Divides the tuples of the two relations into multiple partitions, or batches.
- Only one partition of the inner relation's tuples is kept in main memory at any time.
- The remaining batches reside on disk.
- If an outer tuple belongs to another partition, it is written to a temporary file and processed later.

Symmetric Hash Join

- Maintains two hash tables with different hash functions.



Algorithm:

1. Read a tuple from the inner relation.
2. Hash it using the inner relation's hash function and insert it into the inner relation's hash table.
3. Use this tuple to probe the outer relation's hash table by using the outer relation's hash function.
4. When no more matches are found, take a tuple from the outer relation.
5. Hash it using the outer relation's hash function and insert it into the outer relation's hash table.
6. Use this tuple to probe the inner relation's hash table by using the inner relation's hash function.
7. Repeat steps 1 - 6 until one of the relations is exhausted. (For example, inner realtion)
8. At that point, the remaining tuples of the outer realtion are hashed using the inner relation's hash function and probes the inner relation's hash table. (There is no need to hash and store the tuples in the outer relations hash table anymore, since there are no inner relation tuples to probe it.)

Other requirements:

- Should run until it gives one output tuple for each pull call.
- State should be saved – should include details on which tuple is running using state node.
- Refer to the "SHJ-project-description.pdf" for more details.

Deliverables:

- The following files should be submitted : nodeHashjoin.c, nodeHash.c, execnodes.h and createplan.c. Although you do not have to change other files, you may need to explore additional files to understand how certain procedures and structures are implemented.
- You can submit these files by sending them to the TA by email in a ZIPed file. To use the zip command, make sure that all the required files are in the current directory.
  - Email your project to **Abhisht Makarand Joshi** ajosh053@uottawa.ca .
  - Keep **Vikas Gogia** vgogi074@uottawa.ca and **Daniel Lobo** daniel.lobo@uottawa.ca in CC.
- Make sure to clearly emphasize parts of the files that you have modified by inserting comments before and after changes. Also, include any necessary comments to describe how modifications are done or how you implemented new features. All comments should be preceded by 'CSI3530:' or 'CSI3130' depending on your course.