**Deep Bodra**

**5801 1841**

**DBI Week 5(2/3 - 2/7)**

**Detailed explanation of the topics covered in class**

**2/4/2020**

1. Introduction
   1. Relational Algebra is a way of representing queries of the databases using operators called relational operators
   2. The relational operators perform operations on the relations and yield another relation as an output
2. Relational Operators
   1. Unary Operators

They operate on single relation

* + 1. σ (Sigma) (Selection/Filter)
       1. σc= <t | t ∈R ∧ c(t) = True>
       2. It selects tuples from a relation that satisfy the condition c
       3. The condition c is a pure function of the Relation R
    2. π (Pi) (Projection)
       1. πA, B, .. = <t | t ∈R ∧ t’ =f(t)>
       2. It applies the function f(t) on each tuple
       3. This function may select a subset of the attributes of R
       4. This function may also concatenate attributes of a relation
    3. δ (Delta) (Duplicate elimination)
       1. It removes a tuple t1 if there is any other tuple t2 such that they have all the attributes same
    4. Ɣ (Gamma) (Groupby)
       1. ƔGroup by attributes, Aggregate
       2. The operator will first group the tuples that have same values for the list of group by attributes
       3. It will then apply the aggregate function like SUM, COUNT, COUNT DISTINCT, AVG, MIN, MAX, etc
  1. Binary Operators
     1. R ⋂ S (Set Intersection)
        1. It yields a relation which has tuples in both R and S
        2. It has implicit duplicate elimination
     2. R ⋃ S (Set Union)
        1. It yields a relation which has tuples either in R or S
        2. It has implicit duplicate elimination
     3. R\S (Set Difference)
        1. It yields a relation which has tuples in R but not in S
        2. It has implicit duplicate elimination
     4. R⋂BS (Bag Intersection)
        1. It performs intersection in a little different way
        2. If R has a tuple t, m times and S has the same tuple n times, then the same tuple will appear in the output relation min(m,n) times
     5. R ⋃B S (Bag Union)
        1. It is same as concatenating the tuples of the relations R and S
        2. The implementation of this operator is trivial. We just iterate the relation R and when we are done then switch to the relation S
     6. R \B S (Bag Difference)
        1. It performs difference in a little different way
        2. If R has a tuple t, m times and S has the same tuple n times, then the same tuple will appear in the output relation max(0, m-n) times
     7. R X S (Cartesian Product)
        1. R X S = < toa | t∈R ∧ a ∈ S> where o = concatenate
        2. Basically it concatenates each tuple of R with every other tuple of S
     8. R ⋈ S(Join)
        1. The output of this operator is same as σc (R X S)
        2. Equi join: Join such that R.A = R.B (some attribute A of R matches some attribute B of S)
        3. Natural Join: Join without any conditions. In this case the condition is implicit which is “Join on attributes that have same attribute names”
        4. Left Outer Join: R ⋈ S U**B** (R not joined and NULL padding for the unknown attributes)
        5. Right Outer join can be defined in the same way
  2. Iterator Model
     1. It is a abstract data type with 3 functions Init(), GetNext() and Close()
     2. It enables us to use work with the operators independently and without having to worry about the implementation details
     3. Init(): set up the data structure that will be required for the implementation
     4. GetNext(): return the next record. It may be as simple as getting the next element of the array or as hard as getting the next leaf of a tree
     5. Close(): free up the resources/memory
  3. Pull Architecture
     1. A query can be represented using operators and relations in the form of a tree
     2. The operator at the top calls GetNext() on the operators at the lower level when tuples are needed
     3. This is like pulling the tuples and hence the name
  4. Single and Double Buffering
     1. Buffers can be added to the pull architecture at each level to improve the performance
     2. But this is good only on the 1990 hardware
     3. Disadvantages for modern hardware
        1. Bad cache behaviour
        2. No parallelism
        3. No peak performance from SSD’s
  5. Batch Processing
     1. Instead of processing single tuples we can process tuples in batches
  6. Pipe
     1. We can run each operator in a different thread
     2. The pipes then can be used as an interface between these threads
     3. This pipe is thread safe
     4. Threads at the lower level can push into the pipe when the pipe has space and those at the upper level can remove from the pipe only when there are tuples in the pipe
     5. This still does not help us gain peak performance even if we run each operator on a different core because of the non uniformity in the load distribution

**2/6/2020**

* 1. Advantages of pipe
     1. Parallelism
     2. Smoothens tuple flow
  2. Distributed Database
     1. Since the machines with more cores were expensive, the idea of using multiple machines with 1-2 cores in a distributed fashion came into play
     2. Each machine had a hard drive and a CPU and was connected to the other machines over the network.
     3. Each machine ran a Postgres instance
     4. This idea was used by GreenPlum, AsterData, RedShift, etc
     5. Data Sharding
        1. The data was divided into chunks and distributed to various servers
        2. Drawback: TCP/IP overhead while moving data required for query execution
     6. Load Balancing
        1. Each machine does different amount of computation
        2. A time consuming query on a single machine would make the overall result slower
  3. Correct Approach: Datapath (continued)
     1. Keep all the cores busy even if the query is as simple as SELECT
     2. A chunk with around 1-2M tuples is the unit of execution
     3. |Core1| |Core2| ........ |Core 128|

| Chunk1 |

| Chunk2 |

| Chunk3 | (Knob)

.

.

.

* + 1. It makes use of a Push architecture
    2. The chunks are prepared beforehand without them being requested explicitly
    3. But the chunks are not scheduled on a core beforehand
    4. The chunk will be given to a core only when it is ready to take it i.e. when it has finished its previous execution
    5. The knob is used to control the flow of chunks
    6. Sometimes the chunks are dropped when the CPU cannot cope up with the incoming chunk rate
    7. Adaptive Execution: The core on which an operator should run on is not decided beforehand. The decision is made in real time based on the utilisation and other factors.

--THANK YOU--