Exercise 1. Query execution

Assume we have the following relational schema:

Customer(Cid, Name)  
Order(Oid, Customer, Volume)

There are 1000 Customer tuples and 100000 Order tuples. Size of each tuple is 100 bytes.

Additionally, assume we have the following query, that asks for the total volume of orders of a Customer called “Alex”:

SELECT sum(o.Volume) FROM Customer c, Order o   
 WHERE c.Cid = o.Customer AND c.Name = “Alex”;

1. Translate this SQL query to a relational algebra expression. (Hint: you may use the sum function.)

Sum(π o.Volume(σ c.Name = “Alex”(× (ρ c Customer,  ρ o Order) c.Cid = o.Customer)))

1. Explain how you would implement each operator, i.e. use a keyword that determines which algorithm you would use to implement an operator. (E.g. 2-Phase External Sort)

Так как покупателей 1000, а заказов 100000, имеет смысл сначала обработать таблицу покупателей с помощью Table Access (так как у нас ничего не известно про индексы, то можно сделать фуллскан)

Alexes = Fullscan (Customer) Name = “Alex”, который вернет нам индексы Алексов,

Затем надо сделать Join таблиц по индексу покупателя, можно для этого воспользоваться например Nested Loop с вторым аргументом – покупателями, так как их меньше

Res = NestedLoop(Order, Alexes) Alexes.Cid = Order.Customer

И затем подсчитать их с помощью суммы

Answer = sum(Res.Volume)

1. For each operator, give the amount of main memory that you would allocate. Why? How much memory do you need to process the whole query?

По написанному в прошлом дз в принципе можно сделать вывод что минимальное количество памяти для обработки каждого запроса отдельно и всех целиком – 1-2 страницы, тогда обработка будет медленной и неэффективной, но это возможно. В более благоприятном случае хотелось бы таблицу минимум по размеру полученного после первого фуллскана результата + 1, то есть по числу покупателей – Алексов + 1 для хотя бы одной записи заказов. Максимум это 1001 запись, каждая из которых составляет 100 байт, то есть 100100 байт.

При этом этой памяти достаточно как для каждой из операций, так и для запроса целиком, потому что в фуллскане мы используем память на <= 1000 записей, сохраним туда найденные записи и далее можем проходить по таблице заказов, читая по одной записи.

Exercise 2  Join methods

Assume we have the following Query:

SELECT \* FROM R, S, T   
 WHERE R.rid = S.sid AND S.sid = T.tid AND T.tid = R.rid

1. Give 3 different query plans for this query (with different join methods combinations and ordering).

Так как нам нужно сделать джойн трех таблиц, есть как минимум три способа это сделать, различающиеся тем, в какой момент какие таблицы мы попарно джойним. Индексов у нас нет, какой-либо специфичной информации о таблицах тоже, так что будем использовать простой Table Access через Full Scan и в качестве джойна разные функции, например Nested Loop, Hash Join и Sort Merge Join

* 1 план:
  + S = FullScan(S)
  + R = FullScan(R)
  + RS = NestedLoop(R, S) R.rid = S.sid
  + T = Fullscan(T)
  + TRS = NestedLoop(T, RS) T.tid = RS.rsid
* 2 план:
  + S = FullScan(S)
  + T = FullScan(T)
  + TS = NestedLoop(T, S) T.tid = S.sid
  + R = FullScan(R)
  + RTS = HashJoin(R, TS) R.rid = TS.tsid
* 3 план:
  + S = FullScan(T)
  + R = FullScan(R)
  + RT = NestedLoop(R, T) R.rid = T.tid
  + S = FullScan(S)
  + SRT = SortMergeJoin(S, RT) S.sid = RT.rtid

Тот или иной вариант может быть более оптимальным в зависимости от размеров таблиц, так как хотелось бы сначала обрабатывать менее объемные таблицы, которые возможно уже вернут меньше результатов, так что не придется обрабатывать лишнюю информацию из больших таблиц.

1. For each given plan in the previous part, specify the size of each table so that each plan would be optimal.

Для первого плана оптимальным размером таблиц будет то, что S по размеру меньше всех, а T – больше всех, таким образом мы в самом низу дерева исполнения обработаем самую компактную таблицу и не будем тащить наверх большое количество ненужных результатов, то есть S <= R <= T.

Во втором случае, так как таблица Т вторая в NestedLoop, она должна быть самая маленькая, далее в HashJoin второй аргумент тоже в идеале меньше первого, то есть TS меньше R, так что T <= S <= R.

В третьем плане все так же, вначале мы хотим обработать самую маленькую таблицу Т, и в самый последний момент – S, то есть T <= R <= S.

1. Take one of the plans from the previous part and assume none of the tables fit in main memory, i.e. memory is at most half the size of the smallest table. Under these conditions, how do you allocate buffers? What will your page replacement policy be?

Можно рассмотреть первый план и нехватку памяти. Если у нас совсем ничего не помещается, в принципе можно обрабатывать записи попарно. Так, вначале считывается одна запись из S и одна из R, они сравниваются, если не подходят под условие джойна – считывается следующая запись из R, и так далее. Когда мы находим подходящие друг другу по айди записи, мы считываем запись из таблицы T и проверяем ее соответствие, если она не подходит – читаем следующую из T.

Таким образом при вложенных циклах в теории достаточно ячеек памяти для хранения 3 записей, из которых реже всего стоит замещать первую, а чаще всего – последнюю, так что вероятно стратегия замещения MRU.