Implementation on Postgresql of previously submitted work:

Table1 code:

Create table Final1(

A1 serial,

A2 Int);

Create table B1 (

B11 int) inherits (Final1);

Create table Table1(

B22 int) inherits(B1)

Create table B2\_2000(

Check (B22<2000)

) inherits(Table1);

Create table B2\_4000(

Check (B22<4000)

) inherits(Table1);

Create table B2\_6000(

Check (B22<6000)

) inherits(Table1);

Create table B2\_8000(

Check (B22<8000)

) inherits(Table1);

Create table B2\_10000(

Check (B22<10000)

) inherits(Table1);

Create table B2\_12000(

Check (B22<12000)

) inherits(Table1);

Create table B2\_14000(

Check (B22<14000)

) inherits(Table1);

Create table B2\_16000(

Check (B22<16000)

) inherits(Table1);

Create table B2\_18000(

Check (B22<18000)

) inherits(Table1);

Create table B2\_20000(

Check (B22<20000)

) inherits(Table1);

Do $$

Begin

For i in 1..1000 Loop

For j in 1..10000 Loop

if (j<=2000) Then

Insert Into B2\_2000 values(1,i,j);

ElSIF (j<=4000) Then

Insert Into B2\_4000 values(1,i,j);

ElSIF (j<=6000) Then

Insert Into B2\_6000 values(1,i,j);

ElSIF (j<=8000) Then

Insert Into B2\_8000 values(1,i,j);

ElSIF (j<=10000) Then

Insert Into B2\_10000 values(1,i,j);

ElSIF (j<=12000) Then

Insert Into B2\_12000 values(1,i,j);

ElSIF (j<=14000) Then

Insert Into B2\_14000 values(1,i,j);

ElSIF (j<=16000) Then

Insert Into B2\_16000 values(1,i,j);

ElSIF (j<=18000) Then

Insert Into B2\_18000 values(1,i,j);

ElSIF (j<=20000) Then

Insert Into B2\_20000 values(1,i,j);

ELSE

Raise exception 'Course name not exist';

End if;

End Loop;

End Loop;

End; $$

Commit;

Note:

1. I used column A1 as serial. It will play a role of saving all the data sequentially.
2. I chose the data and created partition in a way that partition cannot exceed the block limit of 8KB per page.
3. I similarly created table 2.
4. I could also consider column B1 and B2 as nested table by creating create type (as in object- oriented database) but Postgresql plus provides this functionality. It is not possible in Postgresql as per my knowledge.
5. TOAST is also not beneficial because our aim is perform JOIN between to tables with the minimum physical storage access.

Reason 1 for inefficient of TOAST:

Before performing the operation, I observed the storage structure of postgreSQL. PostgreSQL uses a fixed page size (commonly 8 kB) and does not allow tuples to span multiple pages. Therefore, it is not possible to store very large field values directly. When a row is attempted to be stored that exceeds this size, TOAST basically breaks up the data of large columns into smaller "pieces" and stores them into a TOAST table. Each table has its own associated (unique) TOAST table, which may or may not ever end up being used, depending on the size of rows inserted. The mechanism is accomplished by splitting up the large column entry into 2KB bytes and storing them as chunks in the TOAST tables. It then stores the length and a pointer to the TOAST entry back where the column is normally stored. Because of how the pointer system is implemented, most TOAST' able column types are limited to a max size of 1GB, but as our array can have 1B integer values too. It can exceed 1GB limit. So, TOAST may not be a very helpful option in this case.

Reason 2 for inefficient of TOAST: Suppose I used TOAST than to performing join operation I need to access each row and I need to follow each pointer and do the same until I finish whole table. This could lead me reading unnecessary data as I required to follow all the pointers. In our case column B’s could itself be more than 8 kb so TOAST will cluster it. AS a result, reading a single row itself is very expensive. It could be useless if it is not a part of a join operation. On the other hand table inheritance and partition gives us flexibility to divide the table into part as per our need and a size of less than 8 kb.

Moreover, by using single column tables for Column B’s will help us to not to cover multiple block for saving a single row and also reducing unnecessary access of secondary storage.

I have less values in Table 1 compare to Table 2 so to optimize the join operation, I considered outer table as Table 1.

To fix the order of join operation :

**Set Join\_collapse\_limit= 1**

Now, I created index on column B1 and performed partition wise JOIN operation, Example is as follows:

Create unique index B2\_indx1 on Table1(B22).

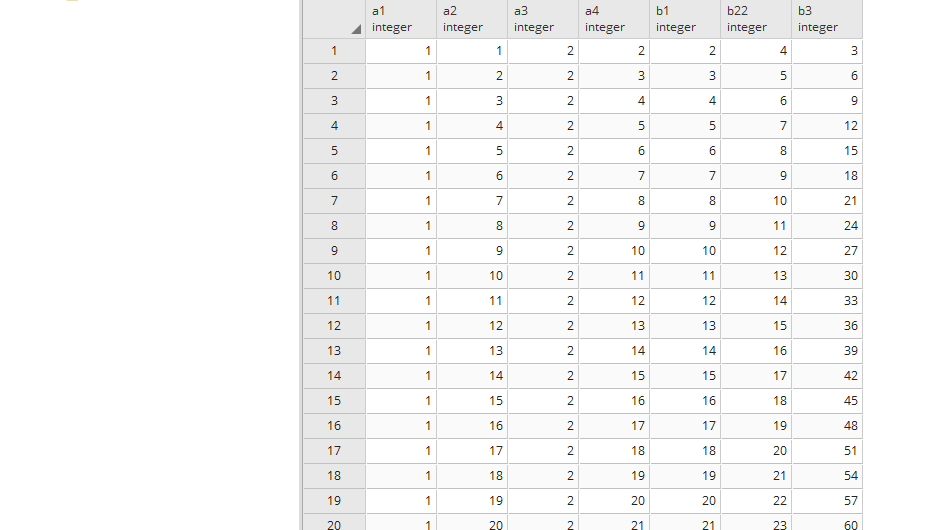
Create unique index B2\_indx2 on Table2(B22).

This index only applies to parent table. It does not work for the child tables.

Now, Perform JOIN operation partition-wise: Example:

Select \* from Table1 , Table2 where Table1.B22=Table2.B22 and table1.b22 between 0 and 2000;

Result:



It is just a part of a result.

The same thing can be perform on remaining partitions as per the requirement.

Thank you!