Requirements for Improvements in PostgreSQL Database Partitioning

Simon Riggs Major Developer, PostgreSQL

1 Partitioning Requirements

Requirements are organised into three categories: User, Plans and Technical

1.1 Basic User Requirements

Requirements prefixed R

R1 We need a way to cheaply and easily route both INSERTs and COPYs to correct tables, **without** additional DDL, also to allow partitioning to be usable for OLTP by reducing overhead.

Prio: **High**

R2 Oracle-style partitioning syntax on CREATE TABLE

Prio: **High**

R3 Allow STABLE functions such as CURRENT_DATE can be used for partition elimination. Requires evaluation of partitioning constraints at execution time.

Prio: **High**

R4 Support high numbers of partitions without significant disruption of planning or execution timing, or other negative server effects.

Prio: **High** up to 1000

Prio: Med up to 10000

R5 Automated partitioning, so that we can say things like "one new partition each day" and have the system sort out the details.

This is an Oracle 11g feature known as Interval Partitioning.

Prio: **High**

R6 Synchronous Scan support. Scans of all partitions should begin their scan at whatever point an already existing all-partition scan is in progress. This would extend synchronous scanning feature in PostgreSQL 8.3 to work with partitioning.

Prio: Med

simon@2ndquadrant.com

1.2 Additional Plan Support

Requirements prefixed P

There are a number of places where using partitioning causes performance regressions, plus some places where additional optimization is possible. Each item is discussed further in the Technical Details chapter.

P1 Allow outside-in joins from Dimension to Fact tables.

Prio: **High**

P2 Allow merge-join push-down so that multiple tables with matching partitioning definitions can be easily and cheaply joined.

This is enabled by an Oracle 11g feature known as REF partitioning that allows a table to be partitioned in the same way as a table to which it is related by a Foreign Key check. Merge joins of partitioned tables are supported by Oracle 11g and SQLServer 2005.

Prio: **High**

P3 Support LIMIT and push-down, to ensure we pass down the information that we need only some of the result rows.

Prio: Med

P4 Support intelligent ordering, so that indexes can be used for ordering when the plan allows that. i.e. ORDER BY push down.

Prio: Med

P5 Support min(), max() optimisations

Which requires the support of push-down for ORDER BY ... LIMIT 1

Prio: Med

P6 Sorted partition access

Prio: Med

1.3 Technical Requirements

Requirements prefixed T

T1 Use physical tlist in all possible places in executor

Prio: **High**

T2 Consequence of R4 we must aim to significantly reduce planning time for partitioning, so it is closer to O(logN) rather than O(N)+ or worse.

Prio: High

T3 Ability to partition by a column that is *not* stored on every row, to save space.

Prio: Low

T4 ANALYZE ... CASCADE so that all sub-tables are analyzed. Why do we need this, when we have autovacuum?

Prio: Low

2 Technical Details

2.1 P1: Outside-In Joins

```
SELECT sum(F.measure)
FROM Dimension1 D1, Dimension2 D2, Fact F
WHERE
   D1.Dim1Key = F.Dim1Key
AND D1.Dim2Key = F.Dim2Key
AND D1.Dim1Label = 'value'
AND D2.Dim2Label = 'value'
```

specific example:

```
SELECT sum(F.sales)
FROM TimeDimension T, StoreDimension S, Fact F
WHERE
   T.TimeKey = F.TimeKey
AND S.StoreKey = F.StoreKey
AND T.FinancialQuarter = 4
AND T.FinancialYear = 2006
AND S.Region = 'South'
```

Without partitioning this causes an Nested Loops join from the dimension tables into the indexes on the fact table. This then create an IndexScan or a BitmapIndexScan. This plan should be the same speed with partitioning, whereas it is currently significantly slower.

simon@2ndquadrant.com

2.2 P2: Partition aware Merge Joins

Support partition-aware plans for joins between tables with similar partitioning schemes. (This will become important later for supporting parallel query.)

Currently we do:

```
MergeJoin
|--Sort
   |--Append
      |--SeqScan T1_P1
      |--SeqScan T1_P2
|--Sort
   |--Append
      |--SeqScan T2_P1
      |--SeqScan T2_P2
```

though would like to use this plan

```
Append
|--MergeJoin
   |--Sort
      |--SeqScan T1_P1
   |--Sort
      |--SeqScan T2_P1
|--MergeJoin
   |--Sort
      |--SeqScan T1_P2
   |--Sort
      |--SeqScan T2_P2
```

This plan should be faster with partitioning when we have a restriction on one or both of the partitioned tables.

2.3 P3: Pushdown LIMIT plans

Currently we use this plan

```
Limit
|--Append
   |--SeqScan
   |--SeqScan...
```

though would like to use this plan

```
Append
|--Limit
   |--SeqScan
|--Limit
   |--SeqScan
```

More complex example, currently we do

Limit
```
|--Sort
   |--Append
      |--SeqScan
      |--SeqScan...
```

though would like to use this plan

```
Append
|--Limit
   |--Sort
      |--SeqScan
|--Limit
   |--Sort
      |--SeqScan
```

2.4 Physical Tlist support

Currently, Append node disallows using physical tlists which effects the rest of the plan since we must re-palloc all rows, taking time and memory. Significantly reduce performance for large queries.

see optimizer/plan/createplan.c: use_physical_tlist()

2.4.1 Parent table removal

Currently the parent table is not removed from the plan (ever), so all plans contain at least two tables. This means we always use an Append plan/node.

We would benefit from recognising that the parent table does not have any rows in it, so it can be excluded. Thus many queries reduce to just a single partition, which would then not require an Append node, which would then be memory optimised.

2.4.2 Alter APPEND to support physical tlists

It may be possible to alter Append nodes to support physical tlists in situations where we can prove that the partitions are mutually exclusive.

2.5 Sorted Partition Access

Data partitioned by month, with query like this:

```
select *
from partitionedtable
where /search-clause/
order by month desc
limit 100
```

Currently we do this plan

```
Limit
|--Sort
   |--Append
      |--SeqScan Anytable
      |--SeqScan Anytable
      |--SeqScan Anytable
```

Best plan would be an Append node that has a sorted list of child nodes beneath it, but that no other Sort is required.

e.g.

```
Limit
|--Append
   |--SeqScan Latest-Month
   |--SeqScan Latest-Month-1
   |--SeqScan Latest-Month-2
```

2.6 Issues with numbers of partitions

- Relcache overflow from too many tables
- No index on pg_inherits catalog table