**CS631 Project**

# Table partitioning

*by*

Sushmita Bhattacharya (133050022)
Shubham Saxena (133050039)
Ruhi Sharma (133050027)


**Department of Computer Science and Engineering**

**Indian Institute of Technology Bombay**

# 1 Problem Statement

Many databases support table partitioning, where rows are partitioned based on some condition and stored separately. Partitioning can provide tremendous benefit to a wide variety of applications by improving performance, manage- ability, and availability. PostgreSQL supports basic table partitioning. It does not have direct syntax for this, but provides a way to implement it using table inheritance.

The goal of the project is to provide syntactic support for Range partitioning along the lines of Oracle or SQL Server. Following are the intended tasks.

1. **Partition Creation**
   Allows tables to be partitioned internally based on partitioning attribute specified.
2. **Insertion,Deletion and Updation of Tuples**
   - Allow tuples to be handled directly for insertions or deletions or up- dations when partition name is specified.
   - Implement triggers to re-target the correct partition when table name is specified.
3. **Tracking Metadata of Partitioned Tables**
   Create new relation to keep track of on what conditions table is partitioned, what are the subtables, and range of partitioned attribute of each subtables.
4. **Creation Of Indices**
   When indices are created on the main table, automatically create the same index on subtables.

# 2 Syntax

This project provide syntactic support for partitioning along the lines of Oracle or SQL Server. Partitioning is specified in query by using keywords PARTITION BY RANGEP and then mentioning partitioning attribute. Then List of partition names along with their ranges is specified using the following syntax.

Create table<Main Table Name>(<attribute_list>)
PARTITION BY RANGEP(<partitioning attribute>)
(PARTITION<partition1_name>VALUES LESS THAN (<upper_bound1>),
PARTITION<partition2_name>VALUES LESS THAN (<upper_bound2>),
PARTITION<partition3_name>VALUES LESS THAN (<upper_bound3>),
..., DEFAULT);

# 3 Data Types Supported for Partitioning Attribute

Data types supported for partitioned attribute:

1. Integer

- Int8
- Int4
- Int2

2. Date

## 4 Constraints Considered

Various constraints we considered while implementation:

1. Table can be partitioned only while creating, partition table syntax is not supported with alter table.
2. There can be only one partitioning attribute per table.
3. Only range partitioning is supported.
4. Partitioning attribute can be of Integer or Date type.

## 5 Code Changes

All paths to source files are relative to postgresql directory postgresql-9.2.4. Changes done in the following files:

1. src/backend/parser/gram.y
   This file is changed to add grammar for partitioning
2. src/include/parser/kwlist.h
   This file is changed to add various keywords like RANGEP, LESS, THAN, PARTITION.
3. src/include/nodes/parsenodes.h
   This file is changed to add nodes(struct) like PARTIONINFO, CONSTRAINTP.
4. src/include/nodes/nodes.h
   This file is changed to add type for various nodes added in parsenodes.h .
5. src/include/commands/tablecmds.h
   This is changed for tablecmds.c
6. src/backend/commands/tablecmds.c
   This file is changed to implement insertion,deletion using triggers.
7. src/backend/nodes/copyfuncs.c
   This file is changed because we have changed parsenodes.h
8. src/backend/nodes/equalfuncs.c
   This file is changed because we have changed parsenodes.h
9. src/backend/parser/parse_utilcmd.c
   This file is changed to set the values in context of create statement.

10. src/backend/tcop/utility.c
    This file is changed to check if there is any partition info in the table and if it is so then implement all the changes required for it in parse_utilcmd.c

# 6 Implementation Details

**Algorithm**

```
if(table->Partitioninfo!=NULL)
{
    Create main_table
    Create pg_partition table if not exist
    for each partition in partinfo
    {
        Insert tuple(parent_oid,partition_oid,partition_column,partition_column_type
                                        ,min_value,max_value) into pg_partition table.
        Create partition table with name <main_table>_<partition_name>using inheritance.
        Add table constraints to the partition_table to define the allowed key values in each
        partition.
    }
    Add trigger on main_table for insertion using add_trigger function
}



  add_trigger()
{
    if(partition_attribute_value is in range of any partition)
        insert the tuple into the corresponding partition
    else
        insert the tuple into the default partition if default partition is specified during
        the partitioning
}
```

# 7 Future Scope

In our implementation, we assumed that the partitioning must be done during table creation. An extension can be done so that this restriction is removed and partitioning can be done during table alteration also. We have assumed only single partitioning attribute. As an extension, we can remove this restriction by having any number as partitioning attributes. Further an extension can be done to support all data types for partitioning attribute.

## 8 Conclusion

We have implemented the code to create partitions along the lines of Oracle during table creation. The code is also supporting selection, insertion, and deletion in partitioned table.