How to Move Rows Across Partitions in Postgres Plus(R)

A Postgres Evaluation Quick Tutorial From EnterpriseDB

November 29, 2009

EnterpriseDB Corporation, 235 Littleton Road, Westford, MA 01866, USA
T +1 978 589 5700 **F** +1 978 589 5701 **E** info@enterprisedb.com **www**.enterprisedb.com

Introduction

Table partitions use simple rules driven by data values in table rows to spread the rows among the partitions. This increases query and maintenance efficiency. However, once you have partitioned a table it is sometimes necessary to automatically move rows between the partitions following updates to data values that change which partition the row should reside in. This Quick Tutorial shows how to automatically move data across partitions and maintain partition keys at the same time.

This EnterpriseDB Quick Tutorial will help you get started with the Postgres Plus Standard Server or Postgres Plus Advanced Server database products in a Linux, Windows or Mac environment. It is assumed that you have already downloaded and installed Postgres Plus Standard Server or Postgres Plus Advanced Server on your desktop or laptop computer.

This Quick Tutorial is designed to help you expedite your Technical Evaluation of Postgres Plus Standard Server or Postgres Plus Advanced Server. For more informational assets on conducting your evaluation of Postgres Plus, visit the self-service web site, Postgres Plus Open Source Adoption center.

In this Quick Tutorial you will learn how to do the following:

- enable row movement in a partitioned table
- enter and change test data to test row movement

Before stepping through this tutorial, you should be familiar with the process of setting up a partitioned table. If you are not familiar with the steps involved in setting up a partitioned table, please see the Tutorial, '*How to Create a Table Partition in Postgres Plus*', available at:

http://www.enterprisedb.com/learning/all_platforms.do.

Usage Note: While the examples in this tutorial are demonstrated in a Windows environment, the steps are the same for the Linux and Mac environments. You will notice slight variations between the operating systems; there are differences in the tools used (e.g. terminal windows and text editors), the use of forward slashes vs. back slashes in path specifications, and the installation directory locations.

Feature Description

What is Row Movement?

Partitioning divides large tables into smaller physical portions to improve query

performance, simplify table loading and unloading operations, and conserve money by keeping seldom-used data on less-expensive or slower storage media. The data stored in a partitioned table is *logically* a part of a parent table, but is *physically* stored in one or more child tables.

Partitioned tables are created with rules, triggers or a combination of both; the rules and triggers divide the data between the different partitions according to the constraints defined by the rules and triggers. When you partition a table, you define a set of rules that tell the Postgres Plus server to route each row into an appropriate partition.

For example, if you are partitioning a sales history table, you may partition the table by the date at which each transaction is recorded. The transaction date is known as the *partition key*. Since the partitioning rules control where each row resides (based on the partition key), any change to a partition key value may logically move a row from one partition to another. Row movement is the process of *automatically* moving such a row from its original partition to the new partition when the partition key changes.

Another example of using row movement to maintain data integrity in a partitioned table would be if you were using a database to manage an employee table; if the table is partitioned based on the department number, and an employee moves to a new department, row movement allows the data to migrate between partitions.

If row movement is not enabled, and you try to re-assign an employee to a new department, you would likely encounter an error such as:

```
ERROR:  new row for relation "employees_part1" violates check constraint
"employees_part1_dept_check"
```

If you don't enable row movement, you won't be able to update the partition key; the error message prevents you from orphaning a row in the wrong partition.

The following steps walk you through how to set up support for row movement and updateable partition keys.

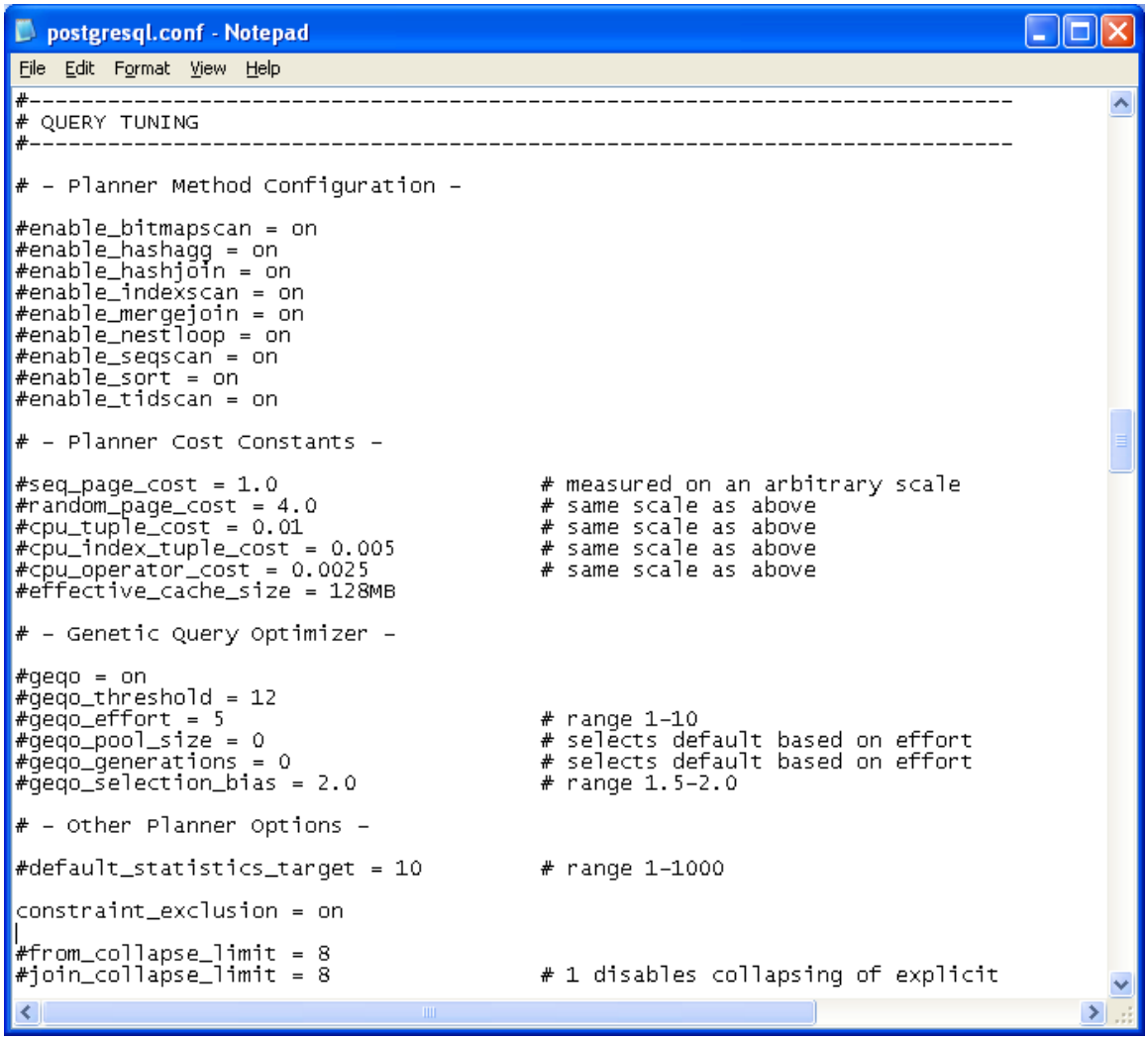
Enabling Row Movement in a Partitioned Table

If you've reviewed the tutorial, '*How to Partition a Table in Postgres Plus*', you're already familiar with the first few steps in this tutorial. For this tutorial, we are creating a master table named `employees`, and then partitioning the data stored in that table into two partitions based on the value of the `dept` column. The following example uses triggers to create the partitioned table.

Step 1 (IMPORTANT): Enable `constraint_exclusion`.

Since Postgres Plus uses the concept of constraint exclusion to enable partition boundary checking, it is critical that you set the following parameter in the `postgresql.conf`

file. You can open the `postgresql.conf` file (shown below) from the `Start` menu by navigating to the `Postgres Plus` menu, and choosing `Expert Configuration` and then `Edit postgresql.conf`. When the `postgresql.conf` file opens, scroll down to the `QUERY TUNING` section and remove the pound sign (`#`) from in front of the `constraint_exclusion` entry. Set the `constraint_exclusion` parameter to `on`, and save the `postgresql.conf` file before exiting.



After setting this parameter, you must signal the server to reload the configuration file using the `pg_ctl` utility. Open a terminal window, and navigate to:

```
C:\PostgresPlus\8.3R2AS\dbserver\bin
```

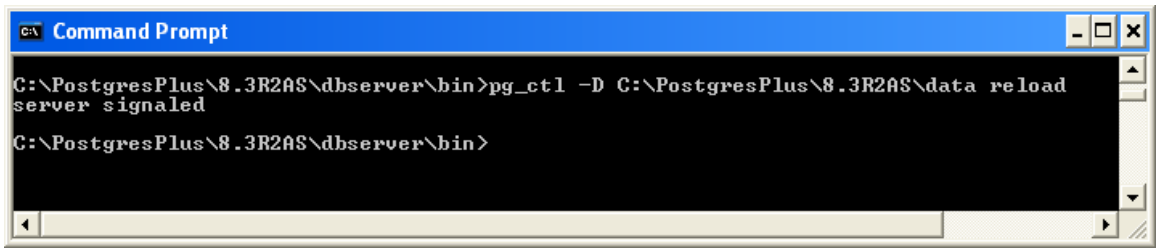
Enter the following command (as shown in the Terminal window):

```
pg_ctl -D <datadir> reload
```

<datadir> is the full path to your data directory.

By default, the data for a standard installation of Postgres Plus Advanced Server is stored in `C:\PostgresPlus\8.3R2AS\data`. The location of the data directory may vary depending on your version of Postgres Plus and the installation options chosen at install time.

The following screenshot shows the process of reloading the configuration file in Windows:



Step 2: Create the parent table:

```
CREATE TABLE employees(dept INT, name VARCHAR(10));
```

Step 3: Create the child (partitioned) tables:

```
CREATE TABLE employees_part1 (CHECK (dept = 10)) INHERITS
(employees);

CREATE TABLE employees_part2 (CHECK (dept = 20)) INHERITS
(employees);
```

Step 4: Create the trigger:

This step creates the trigger that Postgres Plus uses when adding a record to the parent table. Instead of inserting the data directly into the parent table, the trigger diverts data into the appropriate partitioned table based on the value of the `dept` column. If the value of `dept` is `10`, the new record is added to `employees_part1`; if the value is equal to `20`, the new record is added to `employees_part2`.

```
CREATE OR REPLACE FUNCTION employees_insert_trigger()
RETURNS TRIGGER AS $$
BEGIN
    IF ( NEW.DEPT = 10 )  THEN
```



```
        INSERT INTO employees_part1 VALUES(NEW.*);
    ELSIF ( NEW.DEPT = 20 ) THEN
        INSERT INTO employees_part2 VALUES(NEW.*);
    ELSE
        RAISE EXCEPTION 'Organization out of range.  Fix
the employees_insert_trigger() function!';
    END IF;
    RETURN NULL;
END;
$$
LANGUAGE plpgsql;

CREATE TRIGGER insert_employees
    BEFORE INSERT ON employees
    FOR EACH ROW
    EXECUTE PROCEDURE employees_insert_trigger();
```

Step 5: Add sample data to the new table:

```
INSERT INTO employees VALUES(10, 'Craig');
INSERT INTO employees VALUES(10, 'Mike');
INSERT INTO employees VALUES(10, 'Michelle');
INSERT INTO employees VALUES(10, 'Joe');
INSERT INTO employees VALUES(10, 'Scott');
INSERT INTO employees VALUES(20, 'Roger');
INSERT INTO employees VALUES(20, 'Fred');
INSERT INTO employees VALUES(20, 'Sam');
INSERT INTO employees VALUES(20, 'Sonny');
INSERT INTO employees VALUES(20, 'Chris');
```

At this point, the partitioned tables have been created and populated with data. If the value of the `dept` column for any of the rows entered in the table changes, Postgres Plus will report an error.

Step 6: Enable row movement to allow data within the `employees` table to move from one child partition to another:

First, add a trigger to move data stored in table partition `employees_part1`:

```
CREATE OR REPLACE FUNCTION employees_part1_update_trigger()
RETURNS TRIGGER AS $$
BEGIN
    IF ( NEW.DEPT != OLD.DEPT ) THEN
     DELETE FROM employees_part1
       WHERE OLD.dept=dept AND OLD.name=name;

     INSERT INTO employees values(NEW.*);

    END IF;
    RETURN NULL;
```



```
END;
$$
LANGUAGE plpgsql;

CREATE TRIGGER update_employees_part1
    BEFORE UPDATE ON employees_part1
    FOR EACH ROW
    EXECUTE PROCEDURE employees_part1_update_trigger();
```

Then, add a trigger to move data stored in table partition `employees_part2`:

```
CREATE OR REPLACE FUNCTION employees_part2_update_trigger()
RETURNS TRIGGER AS $$
BEGIN
    IF ( NEW.DEPT != OLD.DEPT ) THEN
      DELETE FROM employees_part2
        WHERE OLD.dept=dept AND OLD.name=name;

      INSERT INTO employees values(NEW.*);

    END IF;
    RETURN NULL;
END;
$$
LANGUAGE plpgsql;

CREATE TRIGGER update_employees_part2
    BEFORE UPDATE ON employees_part2
    FOR EACH ROW
    EXECUTE PROCEDURE employees_part2_update_trigger();
```

You can demonstrate that the triggers are performing as expected by executing the following series of `UPDATE` and `SELECT` statements at the `psql` command line. The `psql` command line is available through the `Start` menu by navigating to the `Postgres Plus` menu, and selecting `SQL Shell (psql)`. First, review the content of the two partitions; notice that `Roger` appears in `employees_part2` because he is assigned to department `20`:

```
postgres=# SELECT * FROM employees_part1;
dept |   name
-----+----------
  10 | Mike
  10 | Michelle
  10 | Joe
  10 | Scott
(4 rows)
```



```
postgres=# SELECT * FROM employees_part2;
dept | name
-----+-------
  20 | Roger
  20 | Fred
  20 | Sam
  20 | Sonny
  20 | Chris
  20 | Craig
(6 rows)
```

Now, transfer `Roger` from department `20` to department `10`:

```
postgres=# UPDATE employees SET dept=10 WHERE name='Roger';
UPDATE 0
```

A `SELECT` statement shows that `Roger` has moved from `employees_part2` to `employees_part1`:

```
postgres=# SELECT * FROM employees_part2;
dept | name
-----+-------
  20 | Fred
  20 | Sam
  20 | Sonny
  20 | Chris
  20 | Craig
(5 rows)

postgres=# SELECT * FROM employees_part1;
dept |   name
-----+----------
  10 | Mike
  10 | Michelle
  10 | Joe
  10 | Scott
  10 | Roger
(5 rows)
```

Conclusion

In this Tutorial, we demonstrated how to enable automatic row movement and updateable partition keys for partitioned tables in Postgres Plus.

If you haven't checked out EnterpriseDB's Evaluation Guide, please do so; it may help you move onto the next step in your Postgres Plus evaluation. The guide can be accessed

at:

http://www.enterprisedb.com/learning/documentation.do

You should now be able to proceed with a technical evaluation of Postgres Plus.

The following resources should help you move on with this step:

- Postgres Plus Technical Evaluation Guide
- Postgres Plus Getting Started resources
- Postgres Plus Quick Tutorials
- Postgres Plus User Forums
- Postgres Plus Documentation
- Postgres Plus Webinars

