Partitioning PostgreSQL

Edwin Grubbs
edwin@grubbs.org
http://mysql.meetup.com/284/

- PostgreSQL can only optimize SELECT queries when using Range partitioning or List partitioning.
- Partitioning a table is normally only worthwhile when the size of the table exceeds physical memory.
- PostgreSQL Partitioning Docs
  - http://www.postgresql.org/docs/8.3/interactive/ddl-partitioning.html
- PostgreSQL Tablespaces Docs
  - http://www.postgresql.org/docs/8.3/interactive/manage-ag-tablespaces.html
- PL/Proxy stored procedure language makes it easy to call stored procedures on other databases servers over the network, and it provides the ability to partition your data amongst remote servers (sharding) by hashing field values.
  - https://developer.skype.com/SkypeGarage/DbProjects/PlProxy
  - http://kaiv.wordpress.com/2007/07/27/postgresql-cluster-partitioning-with-plproxy-part-i/

Implementing Partitioning

1. Create a master table from which all the child tables will inherit.

```
CREATE TABLE shipment (
    id SERIAL PRIMARY KEY,
    address TEXT NOT NULL,
    shipping_date TIMESTAMP NOT NULL);
```

2. Create child tables to serve as each partition of the master table using table constraints to define the allowed key values in each partition.

```
CREATE TABLE shipment_part_2008 (
    CHECK (shipping_date >= DATE '2008-01-01'
           AND shipping_date < DATE '2009-01-01')
) INHERITS (shipment);
CREATE TABLE shipment_part_pre2008 (
    CHECK (shipping_date < DATE '2008-01-01')
) INHERITS (shipment);
```

3. Create an index on the key column(s) for each partition.

```
CREATE INDEX shipping_date_2008 ON shipment_part_2008 (shipping_date);
CREATE INDEX shipping_date_pre2008 ON shipment_part_pre2008 (shipping_date);
```

4. Ensure that the constraint exclusion configuration parameter is enabled in postgresql.conf so that queries will be optimized for partitioning (child tables will not be searched for values they can't contain).

```
constraint_exclusion = on
```

Implementing Partitioning (continued)

5. Optionally, define a trigger or rule to redirect data inserted into the master table to the appropriate partition. An update trigger is not necessary.

```
CREATE OR REPLACE FUNCTION shipment_insert()
RETURNS TRIGGER AS $$
BEGIN
    IF (NEW.shipping_date >= DATE '2008-01-01'
            AND NEW.shipping_date < DATE '2009-01-01') THEN
        INSERT INTO shipment_part_2008 VALUES (NEW.*);
    ELSIF (NEW.shipping_date < DATE '2008-01-01') THEN
        INSERT INTO shipment_part_pre2008 VALUES (NEW.*);
    ELSE
        RAISE EXCEPTION 'Date out of range.  Fix the shipment_insert()
   function!';
    END IF;
    RETURN NULL;
END;
$$
LANGUAGE plpgsql;

CREATE TRIGGER shipment_insert_trigger
BEFORE INSERT ON shipment
FOR EACH ROW EXECUTE PROCEDURE shipment_insert();
```

Using Partitioned Tables

```
INSERT INTO shipment (address, shipping_date) VALUES ('Alaska', '2008-08-08');
INSERT INTO shipment (address, shipping_date) VALUES ('Texas', '2007-07-07');
UPDATE shipment SET address = 'Dakota' WHERE address = 'Alaska';

SELECT * FROM ONLY shipment;
 id | address | shipping_date
----+---------+---------------
(0 rows)

SELECT * FROM shipment;
 id | address |    shipping_date
----+---------+---------------------
  1 | Dakota  | 2008-08-08 00:00:00
  2 | Texas   | 2007-07-07 00:00:00
(2 rows)

SELECT * FROM shipment_part_2008;
 id | address |    shipping_date
----+---------+---------------------
  1 | Dakota  | 2008-08-08 00:00:00
(1 row)

SELECT * FROM shipment_part_pre2008;
 id | address |    shipping_date
----+---------+---------------------
  2 | Texas   | 2007-07-07 00:00:00
(1 row)
```

Verify Query Optimization

```
EXPLAIN SELECT * FROM shipment WHERE shipping_date > '2008-02-01';
                                              QUERY PLAN
---------------------------------------------------------------------------------------------------------
 Result  (cost=0.00..45.43 rows=734 width=44)
   ->  Append  (cost=0.00..45.43 rows=734 width=44)
         ->  Seq Scan on shipment  (cost=0.00..23.75 rows=367 width=44)
               Filter: (shipping_date > '2008-02-01 00:00:00'::timestamp without time zone)
         ->  Bitmap Heap Scan on shipment_part_2008 shipment  (cost=7.09..21.68 rows=367 width=44)
               Recheck Cond: (shipping_date > '2008-02-01 00:00:00'::timestamp without time zone)
               ->  Bitmap Index Scan on shipping_date_2008  (cost=0.00..7.00 rows=367 width=0)
                     Index Cond: (shipping_date > '2008-02-01 00:00:00'::timestamp without time zone)
(8 rows)

EXPLAIN SELECT * FROM shipment WHERE shipping_date > '2007-02-01';
                                              QUERY PLAN
---------------------------------------------------------------------------------------------------------
 Result  (cost=0.00..67.11 rows=1101 width=44)
   ->  Append  (cost=0.00..67.11 rows=1101 width=44)
         ->  Seq Scan on shipment  (cost=0.00..23.75 rows=367 width=44)
               Filter: (shipping_date > '2007-02-01 00:00:00'::timestamp without time zone)
         ->  Bitmap Heap Scan on shipment_part_2008 shipment  (cost=7.09..21.68 rows=367 width=44)
               Recheck Cond: (shipping_date > '2007-02-01 00:00:00'::timestamp without time zone)
               ->  Bitmap Index Scan on shipping_date_2008  (cost=0.00..7.00 rows=367 width=0)
                     Index Cond: (shipping_date > '2007-02-01 00:00:00'::timestamp without time zone)
         ->  Bitmap Heap Scan on shipment_part_pre2008 shipment  (cost=7.09..21.68 rows=367 width=44)
               Recheck Cond: (shipping_date > '2007-02-01 00:00:00'::timestamp without time zone)
               ->  Bitmap Index Scan on shipping_date_pre2008  (cost=0.00..7.00 rows=367 width=0)
                     Index Cond: (shipping_date > '2007-02-01 00:00:00'::timestamp without time zone)
(12 rows)
```

Query Planning Problem

Constraint exclusion only works when the query's WHERE clause contains constants. The planner analyzes the query before values from parameters (in prepared statements) or stored procedures are substituted in the query. For the same reason, "stable" functions such as CURRENT_DATE must be avoided.
The now() function in the query below definitely returns a value after 2008-01-01, but it ignores the constraint and still searches in shipment_part_pre2008.
You can always query a child table directly instead of the parent table to avoid depending on the optimizer.

```
EXPLAIN SELECT * FROM shipment WHERE shipping_date > now();
                                              QUERY PLAN
------------------------------------------------------------------------------------------------------
 Result  (cost=0.00..71.70 rows=1101 width=44)
   ->  Append  (cost=0.00..71.70 rows=1101 width=44)
         ->  Seq Scan on shipment  (cost=0.00..26.50 rows=367 width=44)
               Filter: (shipping_date > now())
         ->  Bitmap Heap Scan on shipment_part_2008 shipment  (cost=7.10..22.60 rows=367 width=44)
               Recheck Cond: (shipping_date > now())
               ->  Bitmap Index Scan on shipping_date_2008  (cost=0.00..7.01 rows=367 width=0)
                     Index Cond: (shipping_date > now())
         ->  Bitmap Heap Scan on shipment_part_pre2008 shipment  (cost=7.10..22.60 rows=367 width=44)
               Recheck Cond: (shipping_date > now())
               ->  Bitmap Index Scan on shipping_date_pre2008  (cost=0.00..7.01 rows=367 width=0)
                     Index Cond: (shipping_date > now())
(12 rows)
```

Table Inheritance

- Child tables *do* inherit:
  - NOT NULL constraints (even if part of a primary key constraint)
  - Table constraints (i.e. CHECK() constraints)
  - Column default values (even if part of a primary key constraint)
- Child tables *do not* inherit:
  - Indexes (even if part of a primary key constraint)
  - Foreign key constraints
  - Permissions
  - Ownership
- Inherited changes to the parent table are propagated to the children.
- You can't rename inherited columns on child tables. You must rename the column on the parent table.
- You can change the NOT NULL constraint and the default value for inherited columns on a child table, but changing these items on the parent will still propagate to the children.
- You can use ALTER TABLE to enable or disable inheritance on child tables.

Tablespaces

Tablespaces make it easy to specify a different location on the filesystem to create new tables and indexes.

For testing purposes, you can mount a ramdisk under Linux using tmpfs. Surprisingly, the tmpfs partition actually didn't improve performance.

```
mkdir /mnt/dbspace2/
mount -t tmpfs -o size=100M,noatime tmpfs /mnt/dbspace2/
mkdir /mnt/dbspace2/postgresql
mkdir /mnt/dbspace2/postgresql/data
chown postgres:postgres /mnt/dbspace2/postgresql/data
chmod 0700 /mnt/dbspace2/postgresql/data
```

You can change the tablespace for all new tables and indexes being created by setting this variable.

```
SET default_tablespace = space1;
```

Or you can specify the tablespace in the CREATE TABLE or CREATE INDEX statement.

```
CREATE TABLESPACE dbspace2 LOCATION '/mnt/dbspace2/postgresql/data';

CREATE TABLE shipment_part_2008 (
    CHECK (shipping_date >= DATE '2008-01-01' AND shipping_date < DATE '2009-01-01')
) INHERITS (shipment) TABLESPACE dbspace2;

CREATE INDEX shipping_date_2008 ON shipment_part_2008 (shipping_date) TABLESPACE dbspace2;
```