Major Features: Postgres 9.5

BRUCE MOMJIAN

POSTGRESQL is an open-source, full-featured relational database.
This presentation gives an overview of the Postgres 9.5 release.

Last updated: December, 2015

9.5 Feature Outline

1. INSERT ... ON CONFLICT, also known as "UPSERT"
2. Block-Range Indexes (BRIN) which enable compact indexing of very large tables
3. Analytic operations GROUPING SETS, CUBE, and ROLLUP
4. Row-Level Security (RLS)
5. In-memory sorting and hashing performance improvements
6. Multi-core and large memory scalability improvements
7. Automated management of the number of WAL files
8. Additional JSONB data manipulation functions and operators
9. Enhancements to Foreign Data Wrappers
10. Allow Indexed PostGIS LIMIT distance calculations without CTEs

To be released in 2015, full item list at
http://www.postgresql.org/docs/devel/static/release-9-5.html

1. INSERT ... ON CONFLICT

- Turns a conflicting INSERT into an UPDATE
- Works for VALUES and SELECT as a row source
- Handles concurrent operations without errors
- Is row-oriented, unlike MERGE, which is batch-oriented
- Does not have the problems associated with the UPSERT/MERGE implementations of other vendors (http://www.pgcon.org/2014/schedule/attachments/327_upsert_weird.pdf)

INSERT ... ON CONFLICT Example

```
CREATE TABLE ins_update_test (x INTEGER PRIMARY KEY);

INSERT INTO ins_update_test VALUES (1);

INSERT INTO ins_update_test VALUES (1);
ERROR:  duplicate key value violates unique constraint
"ins_update_test_pkey"
DETAIL:  Key (x)=(1) already exists.
```

INSERT ... ON CONFLICT Example

```
INSERT INTO ins_update_test VALUES (1)
    ON CONFLICT DO NOTHING;
INSERT 0 0

INSERT INTO ins_update_test VALUES (1)
    ON CONFLICT (x) DO UPDATE SET x = 2;
INSERT 0 1

SELECT * FROM ins_update_test;
 x
---
 2
```

INSERT … ON CONFLICT … EXCLUDED Example

```
CREATE TABLE customer (cust_id INTEGER PRIMARY KEY, name TEXT);

INSERT INTO customer VALUES (100, 'Big customer');

INSERT INTO customer VALUES (100, 'Non-paying customer');
ERROR:  duplicate key value violates unique constraint
"customer_pkey"
DETAIL:  Key (cust_id)=(100) already exists.

INSERT INTO customer VALUES (100, 'Non-paying customer')
    ON CONFLICT (cust_id) DO UPDATE SET name = EXCLUDED.name;

SELECT * FROM customer;
 cust_id |        name
---------+---------------------
     100 | Non-paying customer
```

INSERT ... ON CONFLICT with SELECT

```
CREATE TABLE merge (x INTEGER PRIMARY KEY);

INSERT INTO merge VALUES (1), (3), (5);

INSERT INTO merge SELECT * FROM generate_series(1, 5);
ERROR:  duplicate key value violates unique constraint
"merge_pkey"
DETAIL:  Key (x)=(1) already exists
```

INSERT … ON CONFLICT with SELECT

```
INSERT INTO merge SELECT * FROM generate_series(1, 5)
    ON CONFLICT DO NOTHING;

SELECT * FROM merge;
 x
---
 1
 3
 5
 2
 4
```

INSERT ... ON CONFLICT ... UPDATE with SELECT

```
CREATE TABLE merge2 (x INTEGER PRIMARY KEY, status TEXT);

INSERT INTO merge2 VALUES (1, 'old'), (3, 'old'), (5, 'old');

INSERT INTO merge2 SELECT *, 'new' FROM generate_series(2, 5)
    ON CONFLICT (x) DO UPDATE SET status = 'conflict';

SELECT * FROM merge2;
 x |  status
---+----------
 1 | old
 2 | new
 3 | conflict
 4 | new
 5 | conflict
```

2. Block-Range Indexes (BRIN)

- Tiny indexes designed for large tables
- Minimum/maximum values stored for a range of blocks (default 1MB, 128 8k pages)
- Allows skipping large sections of the table that cannot contain matching values
- Ideally for naturally-ordered tables, e.g. insert-only tables are chronologically ordered
- Index is 0.003% the size of the heap
- Indexes are inexpensive to update
- Index every column at little cost
- Slower lookups than btree

Block-Range Indexes (BRIN) Example

```
CREATE TABLE brin_example AS
SELECT generate_series(1,100000000) AS id;

CREATE INDEX btree_index ON brin_example(id);
CREATE INDEX brin_index ON brin_example USING brin(id);

SELECT relname, pg_size_pretty(pg_relation_size(oid))
FROM pg_class
WHERE relname LIKE 'brin_%' OR relname = 'btree_index'
ORDER BY relname;
   relname     | pg_size_pretty
---------------+----------------
 brin_example  | 3457 MB
 btree_index   | 2142 MB
 brin_index    | 104 kB
```

http://michael.otacoo.com/postgresql-2/postgres-9-5-feature-highlight-

3. Analytic Operations GROUPING SETS, CUBE, and ROLLUP

- Allows specification of multiple GROUP BY combinations in a single query
- Avoids the need for UNION ALL and recomputation
- Empty fields are left NULL

Employee Table

```
SELECT * FROM employee ORDER BY name;
 name  | office | department
-------+--------+------------
 Jill  | PHL    | Marketing
 Lilly | SFO    | Sales
 Mark  | PHL    | Marketing
 Nancy | PHL    | Sales
 Sam   | SFO    | Sales
 Tim   | PHL    | Shipping
```

Group By Example

```
SELECT office, COUNT(*)
FROM employee
GROUP BY office;
 office | count
--------+-------
 SFO    |     2
 PHL    |     4

SELECT department, COUNT(*)
FROM employee
GROUP BY department;
 department | count
------------+-------
 Marketing  |     2
 Shipping   |     1
 Sales      |     3
```

Group By with Union All

```
SELECT office, NULL as department, COUNT(*)
FROM employee
GROUP BY office
UNION ALL
SELECT NULL as office, department, COUNT(*)
FROM employee
GROUP BY department
ORDER BY 1;
 office | department | count
--------+------------+-------
 PHL    |            |     4
 SFO    |            |     2
        | Marketing  |     2
        | Shipping   |     1
        | Sales      |     3
```

GROUPING SETS Example

```
SELECT office, department, COUNT(*)
FROM employee
GROUP BY GROUPING SETS (office, department)
ORDER BY office, department;
 office | department | count
--------+------------+-------
 PHL    |            |     4
 SFO    |            |     2
        | Marketing  |     2
        | Sales      |     3
        | Shipping   |     1
```

ROLLUP Example

```
SELECT office, department, COUNT(*)
FROM employee
GROUP BY ROLLUP (office, department)
ORDER BY office, department;
 office | department | count
--------+------------+-------
 PHL    | Marketing  |     2
 PHL    | Sales      |     1
 PHL    | Shipping   |     1
 PHL    |            |     4
 SFO    | Sales      |     2
 SFO    |            |     2
        |            |     6
```

CUBE Example

```
SELECT office, department, COUNT(*)
FROM employee
GROUP BY CUBE (office, department)
ORDER BY office, department;
 office | department | count
--------+------------+-------
 PHL    | Marketing  |     2
 PHL    | Sales      |     1
 PHL    | Shipping   |     1
 PHL    |            |     4
 SFO    | Sales      |     2
 SFO    |            |     2
        | Marketing  |     2
        | Sales      |     3
        | Shipping   |     1
        |            |     6
```

Grouping Sets Equivalent of Cube

```
SELECT office, department, COUNT(*)
FROM employee
GROUP BY GROUPING SETS
    ((office, department), office, department, ())
ORDER BY office, department;
 office | department | count
--------+------------+-------
 PHL    | Marketing  |     2
 PHL    | Sales      |     1
 PHL    | Shipping   |     1
 PHL    |            |     4
 SFO    | Sales      |     2
 SFO    |            |     2
        | Marketing  |     2
        | Sales      |     3
        | Shipping   |     1
        |            |     6
```

4. Row-Level Security (RLS)

- Allows SELECT, INSERT, UPDATE, OR DELETE permission control over existing rows with USING expression
- Also INSERT or UPDATE control over added and modified rows with CHECK expression
- Expressions can contain checks for the current user, subqueries, time comparisons, and function calls
- Enabled with GUC row_security, CREATE POLICY, and ALTER TABLE … ENABLE ROW LEVEL SECURITY

Row-Level Security Example
Table Setup

```
SHOW row_security;
 row_security
--------------
 on

CREATE TABLE orders (id INTEGER, product TEXT,
                     entered_by TEXT);

ALTER TABLE orders ENABLE ROW LEVEL SECURITY;

CREATE POLICY orders_control ON orders FOR ALL TO PUBLIC
USING (entered_by = CURRENT_USER);

GRANT ALL ON TABLE orders TO PUBLIC;
```

Row-Level Security Example
User Setup

```
CREATE USER emp1;

CREATE USER emp2;

SET SESSION AUTHORIZATION emp1;

INSERT INTO orders VALUES (101, 'fuse', CURRENT_USER);

SET SESSION AUTHORIZATION emp2;

INSERT INTO orders VALUES (102, 'bolt', CURRENT_USER);
```

Row-Level Security Example Testing

```
SET SESSION AUTHORIZATION postgres;

SELECT * FROM orders;
 id  | product | entered_by
-----+---------+------------
 101 | fuse    | emp1
 102 | bolt    | emp2
```

Row-Level Security Example Testing

```
SET SESSION AUTHORIZATION emp1;

SELECT * FROM orders;
 id  | product | entered_by
-----+---------+------------
 101 | fuse    | emp1

SET SESSION AUTHORIZATION emp2;

SELECT * FROM orders;
 id  | product | entered_by
-----+---------+------------
 102 | bolt    | emp2
```

5. In-Memory Sorting and Hashing Performance Improvements

- Allow VARCHAR(), TEXT and NUMERIC() to use the abbreviated sorting optimization
- Use `memcmp()` as quick string equality checks before collation comparisons
- Decrease the average number of hash entries per bucket from 10 to 1
- Pre-allocate the maximum number of hash buckets in cases where we are likely to use multiple `work_mem`-sized batches
- Allow CREATE INDEX, REINDEX, and CLUSTER to use inlined sorting
- Allow use of 128-bit accumulators for aggregate computations

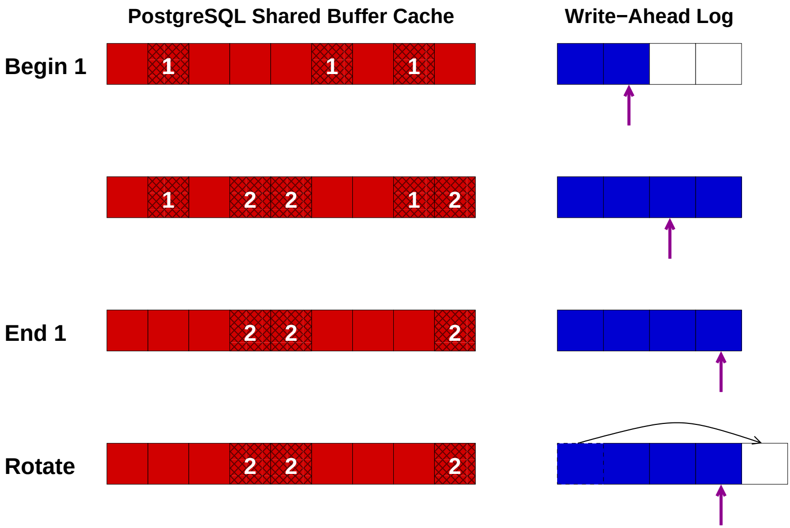
6. Multi-Core and Large Memory Scalability Improvements

- Improve concurrency of shared buffer replacement
- Reduce the number of page locks and pins during index scans
- Make backend local tracking of buffer pins memory efficient
- Improve lock scalability on multi-socket systems
- Increase the number of shared buffer mapping hash table entries from 16 to 128
- Allow searching for a free shared buffer to use minimal locking
- Force buffer descriptors to be CPU-cache aligned (128 bytes)
- Reduce btree page pinning

7. Automated Management of the Number of WAL Files

- New GUC variables `min_wal_size` and `max_wal_size` control the minimum and maximum size of the `pg_xlog` directory
- Previously `checkpoint_segments` controlled only the maximum directory size (previously WAL files were not removed)
- Size specified in bytes, not segment files
- Allows use of additional WAL files only when needed

Management of WAL Files



8. Additional JSONB Data Manipulation Functions and Operators

- Add `jsonb_set()`, which allows replacement of or addition to JSONB documents
- Allow removal of JSONB documents using the subtraction operator
- Allow merging of JSONB documents using the concatenation (`||` operator)
- Add function to remove null values from documents

9. Enhancements to Foreign Data Wrappers

- Add IMPORT FOREIGN SCHEMA to create a local table matching the schema of a foreign table
- Allow foreign tables to be part of inheritance trees
- Allow CHECK constraints on foreign tables
- Add infrastructure for foreign table join pushdown

10. Allow Indexed PostGIS LIMIT Distance Calculations without CTEs

- Nearest neighbor searches allow index lookups to return the closest matches, e.g. return the 10 nearest points to a given point
- Only the bounding boxes of two-dimensional objects are indexed, e.g. polygon, circle, line
- Previously LIMIT could not combine bounding box index lookups with accurate calculations
- Now LIMIT bounding box index filtering can recheck using accurate distance calculations
- Workaround was to use a CTE with a 10x limit, then an outer query to do accurate distance calculations

Pre-9.5 LIMIT Distance Example

```
WITH index_query AS (
    SELECT st_distance(geom,
             'SRID=3005;POINT(1011102 450541)') AS distance,
           parcel_id, address
    FROM parcels
    ORDER BY geom <-> 'SRID=3005;POINT(1011102 450541)'
    LIMIT 100
)
SELECT *
FROM index_query
ORDER BY distance
LIMIT 10;
```

http://boundlessgeo.com/2011/09/indexed-nearest-neighbour-search-in-postgis/

http://shisaa.jp/postset/postgis-postgresqls-spatial-partner-part-3.html

9.5 LIMIT Distance Example

```
SELECT st_distance(geom,
         'SRID=3005;POINT(1011102 450541)') AS distance,
       parcel_id, address
FROM parcels
ORDER BY geom <-> 'SRID=3005;POINT(1011102 450541)'
LIMIT 10
```

http://www.postgresonline.com/journal/archives/350-PostGIS-2.2-leveraging-power-of-PostgreSQL-9.5.html

http://postgis.net/docs/manual-dev/geometry_distance_knn.html

Conclusion

http://momjian.us/presentations

https://www.flickr.com/photos/thevlue/