Major Features: Postgres 11

Bruce Momjian

PostgreSQL is an open-source, full-featured relational database. This presentation gives an overview of the Postgres 11 release.

 http://momjian.us/presentations

Last updated: November, 2018

Postgres 11 Feature Outline

1. Partitioning improvements
2. Parallelism improvements
3. Stored procedures with transaction control
4. Executor-stage compilation
5. Prevent table rewrite for ALTER TABLE … ADD COLUMN with non-NULL default
6. Finer-grained access control
7. Write-ahead log (WAL) improvements
8. Allow 'quit' and 'exit' to exit *psql*
9. Miscellaneous

Full item list at https://www.postgresql.org/docs/devel/static/release-11.html

1. Partitioning Improvements

- Partitioning syntax added in Postgres 10
  - simplified administration
- Postgres 11
  - faster partition pruning during optimization
  - executor-level partition pruning, e.g., for joins
  - hash partitioning
  - move updated rows to new partitions
  - allow a default partition for non-matching rows
  - allow unique/primary indexes when the partition key is included, and allow foreign keys to reference them
  - more items

Hash Partitioning Example

```
-- hash partition
CREATE TABLE part_test (x int, y text) PARTITION BY hash (x);


-- create child partitions
CREATE TABLE part_test_0 PARTITION OF part_test FOR VALUES
                         WITH (MODULUS 4, REMAINDER 0);
CREATE TABLE part_test_1 PARTITION OF part_test FOR VALUES
                         WITH (MODULUS 4, REMAINDER 1);
CREATE TABLE part_test_2 PARTITION OF part_test FOR VALUES
                         WITH (MODULUS 4, REMAINDER 2);
CREATE TABLE part_test_3 PARTITION OF part_test FOR VALUES
                         WITH (MODULUS 4, REMAINDER 3);
```

Partitioning Row Migration

```
-- insert 1k rows
INSERT INTO part_test SELECT generate_series(0, 999), 'old';


-- What partition contains row zero?
SELECT relname
FROM pg_class
WHERE oid = (SELECT tableoid FROM part_test WHERE x = 0);
   relname
-------------
 part_test_0


-- change row zero to row 1003
UPDATE part_test SET x = 1003, y = 'new' WHERE x = 0;


--What partition contains row 1003?  Values are hashed twice.
SELECT relname
FROM pg_class
WHERE oid = (SELECT tableoid FROM part_test WHERE x = 1003);
   relname
-------------
 part_test_1
```

Partitioning Row Distribution

```
-- How are the rows distributed?
SELECT name, y, COUNT(*)
FROM part_test, LATERAL (
SELECT relname
FROM pg_class
WHERE pg_class.oid = part_test.tableoid) AS table_name (name)
GROUP BY name, y
ORDER BY 1, 2;
    name     |  y   | count
-------------+-----+-------
 part_test_0 | old |   258
 part_test_1 | new |     1
 part_test_1 | old |   234
 part_test_2 | old |   276
 part_test_3 | old |   231
```

Range Partitioning Example

```
-- range partition
CREATE TABLE part_test2 (instant TIMESTAMP WITH TIME ZONE, description TEXT)
                        PARTITION BY RANGE (instant);

CREATE TABLE part_test2_2017 PARTITION OF part_test2 FOR VALUES
                             FROM ('2017-01-01') TO ('2018-01-01');
CREATE TABLE part_test2_2018 PARTITION OF part_test2 FOR VALUES
                             FROM ('2018-01-01') TO ('2019-01-01');
-- create default partition
CREATE TABLE part_test2_default PARTITION OF part_test2 DEFAULT;

-- add primary key to parent table
ALTER TABLE part_test2 ADD PRIMARY KEY (instant);
```

Default Partition

```
-- insert two years of rows
INSERT INTO part_test2
    SELECT generate_series('2017-01-01'::timestamptz,
                           '2018-12-31', '1 day'), 'rain';


-- insert rows outside of the defined range
INSERT INTO part_test2 VALUES ('2019-02-20', 'snow');


SELECT name, COUNT(*)
FROM part_test2, LATERAL (
SELECT relname
FROM pg_class
WHERE pg_class.oid = part_test2.tableoid) AS table_name (name)
GROUP BY name
ORDER BY 1;
        name         | count
---------------------+-------
 part_test2_2017     |   365
 part_test2_2018     |   365
 part_test2_default  |     1
```

2. Parallelism Improvements

- Parallel btree index builds
- Parallel hash joins
- Parallelize individual SELECTs in UNION
- Seven more items

3. Stored Procedures with Transaction Control

- Similar to functions
  - allows transaction commit and abort blocks inside procedures
  - returns no values
  - commands prohibited in transaction blocks are still prohibited in procedures, e.g., VACUUM
  - inner transactions cannot be committed independently of outer transactions, i.e., no autonomous transactions
- Supported languages
  - PL/pgSQL
  - PL/Perl
  - PL/Python
  - PL/Tcl
  - SPI

Stored Procedure Example

```
CREATE TABLE system (status text NOT NULL);
-- no more than one row in the table
CREATE UNIQUE INDEX ON system ((true));

CREATE TABLE customer (name TEXT, sales_monthly_total NUMERIC(10,2));
CREATE TABLE employee (name TEXT, sales_monthly_total NUMERIC(10,2));
```

Stored Procedure Example

```
CREATE PROCEDURE end_of_month_processing() AS $$
BEGIN
     INSERT INTO system VALUES ('end-of-month processing')
        ON CONFLICT ((true)) DO UPDATE SET status = excluded.status;
    -- allow all sessions to see the new status
    COMMIT;


    UPDATE customer SET sales_monthly_total = 0;
    UPDATE employee SET sales_monthly_total = 0;
    INSERT INTO system VALUES ('normal operation')
       ON CONFLICT ((true)) DO UPDATE SET STATUS = excluded.status;
    -- allow all sessions to see the new status
    COMMIT;


    -- inform managers only after complete
    PERFORM email_managers('end-of-month processing complete');
END
$$ LANGUAGE plpgsql;


CALL end_of_month_processing();
```

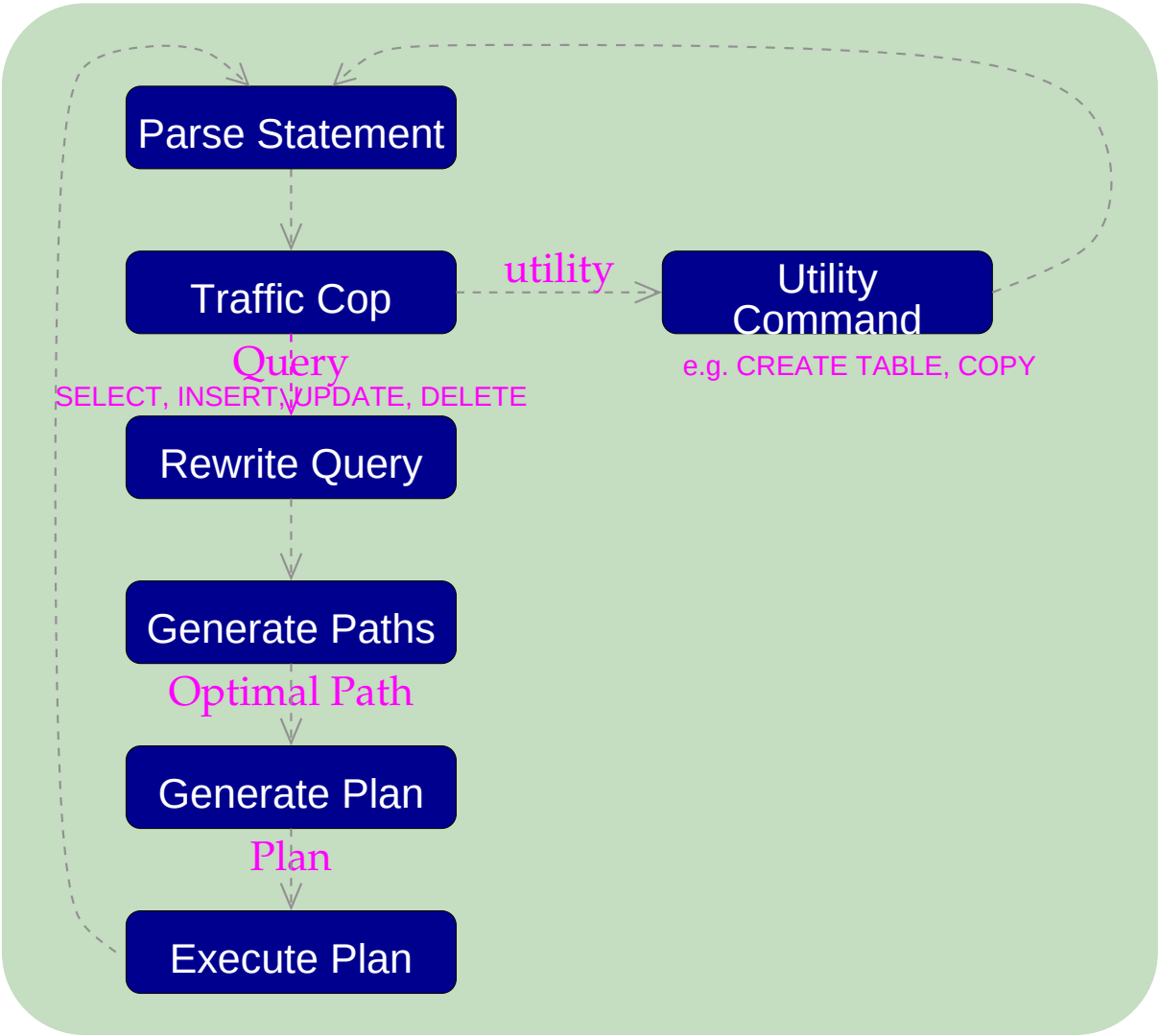
Stored Procedure Example

```
CREATE TABLE web_session (data JSONB, last_active TIMESTAMP WITH TIME ZONE);


-- add five web sessions
INSERT INTO web_session
        SELECT '{"abc": 1}', CURRENT_TIMESTAMP
        FROM generate_series(1, 5);
```

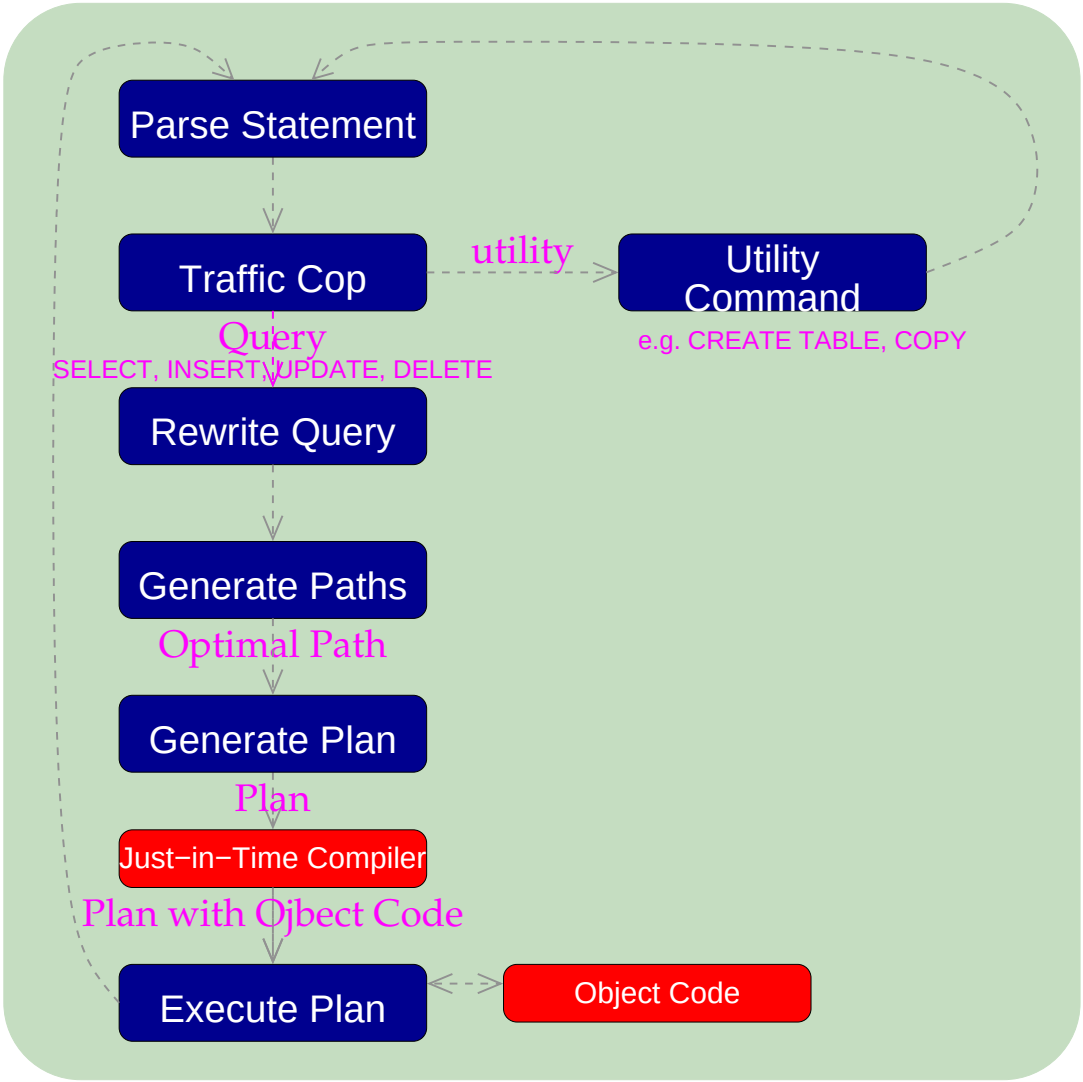
Stored Procedure Example

```
CREATE PROCEDURE expire_web_sessions(min_expire INTERVAL) AS $$
DECLARE
    rows INTEGER;
BEGIN
    WHILE TRUE LOOP
        -- clock_timestamp() is updated on every loop
        DELETE FROM web_session WHERE last_active < clock_timestamp()-min_expire;
        GET DIAGNOSTICS rows = ROW_COUNT;
        COMMIT;

        RAISE NOTICE '% rows deleted', rows;

        -- check at half of expiration time
        PERFORM pg_sleep(EXTRACT(EPOCH FROM min_expire) / 2);
    END LOOP;
END
$$ LANGUAGE plpgsql;

CALL expire_web_sessions('15 minutes');
NOTICE:  0 rows deleted
NOTICE:  0 rows deleted
NOTICE:  5 rows deleted
NOTICE:  0 rows deleted
```

4. Executor-Stage Compilation



Executor-Stage Compilation



5. Prevent Table Rewrite For ALTER TABLE … ADD COLUMN with Non-NULL Default

```
-- Postgres 10
CREATE TABLE alter_test (id SERIAL, name TEXT);
INSERT INTO alter_test (name) SELECT repeat('x', 100);
SELECT relfilenode FROM pg_class WHERE relname = 'alter_test';
 relfilenode
-------------
       16439


ALTER TABLE alter_test ADD COLUMN col1 INTEGER;


SELECT relfilenode FROM pg_class WHERE relname = 'alter_test';
 relfilenode
-------------
       16439


ALTER TABLE alter_test ADD COLUMN col2 INTEGER DEFAULT 1;


SELECT relfilenode FROM pg_class WHERE relname = 'alter_test';
 relfilenode
-------------
       16447
```

Prevent Table Rewrite For ALTER TABLE ... ADD COLUMN with Non-NULL Default

```
-- Postgres 11
CREATE TABLE alter_test (id SERIAL, name TEXT);
INSERT INTO alter_test (name) SELECT repeat('x', 100);
SELECT relfilenode FROM pg_class WHERE relname = 'alter_test';
 relfilenode
-------------
       16388


ALTER TABLE alter_test ADD COLUMN col1 INTEGER;


SELECT relfilenode FROM pg_class WHERE relname = 'alter_test';
 relfilenode
-------------
       16388


ALTER TABLE alter_test ADD COLUMN col2 INTEGER DEFAULT 1;


SELECT relfilenode FROM pg_class WHERE relname = 'alter_test';
 relfilenode
-------------
       16388
```

6. Finer-Grained Access Control

- Superuser-only file system access now controlled by role membership
  - pg_read_server_files
  - pg_write_server_files
  - pg_execute_server_program
- Superuser-only access to file system functions now controlled by function execute permissions
  - pg_ls_dir()
  - pg_read_file()
  - pg_read_binary_file()
  - pg_stat_file()
- Superuser-only import/export of large objects now controlled by function execute permissions
  - lo_import()
  - lo_export()

7. Write-Ahead Log (WAL) Improvements

- Allow WAL file size to be specified via *initdb* or *pg_resetwal*
  - default still 16MB
  - larger files simplify WAL archiving for active clusters
- Halve number of WAL files kept in *pg_wal*
- Zero trailing bytes during forced WAL switch

8. Allow 'quit' and 'exit' to Exit *Psql*

```
$ psql test
psql (11beta3)
Type "help" for help.

test=> quit

$ psql test
psql (11beta3)
Type "help" for help.

test=> exit

$ psql test
psql (11beta3)
Type "help" for help.

test=> SELECT
test-> quit
Use \q to quit.
test-> \q
```

Allow 'quit' and 'exit' to Exit *Psql*

```
$ psql test
psql (11beta3)
Type "help" for help.

test=>      quit
test->      exit
test-> \q

$ psql test
psql (11beta3)
Type "help" for help.

test=> SELECT '
test'> \q
Use control-D to quit.
test'> ^D
```

9. Miscellaneous

- Allow unique indexes to contain non-unique columns via INCLUDE
  - additional columns can be used for index lookups or index-only scans
- Window function improvements
- Add extensions to convert JSONB data to/from PL/Perl and PL/Python
- Add support for large pages on Windows
- Allow *pg_prewarm* to restore previous shared buffer contents
- Allow a password prompt for TLS private key access
- Sharding advances

Conclusion

http://momjian.us/presentations

https://www.flickr.com/photos/8113246@N02/