What's new in pg11

Presented to: Orlando Code Camp

Dave Cramer
March 2019

Dave Cramer

- Work for Crunchy Data
- Major contributor to the PostgreSQL project
- Maintainer of the PostgreSQL JDBC driver
- Working with PostgreSQL since 1999

Crunchy Data

Leading provider of trusted open source PostgreSQL technology, support and training.

Powering Innovation With The World's Most Advanced Open Source Database

CRUNCHY Enterprise PostgreSQL

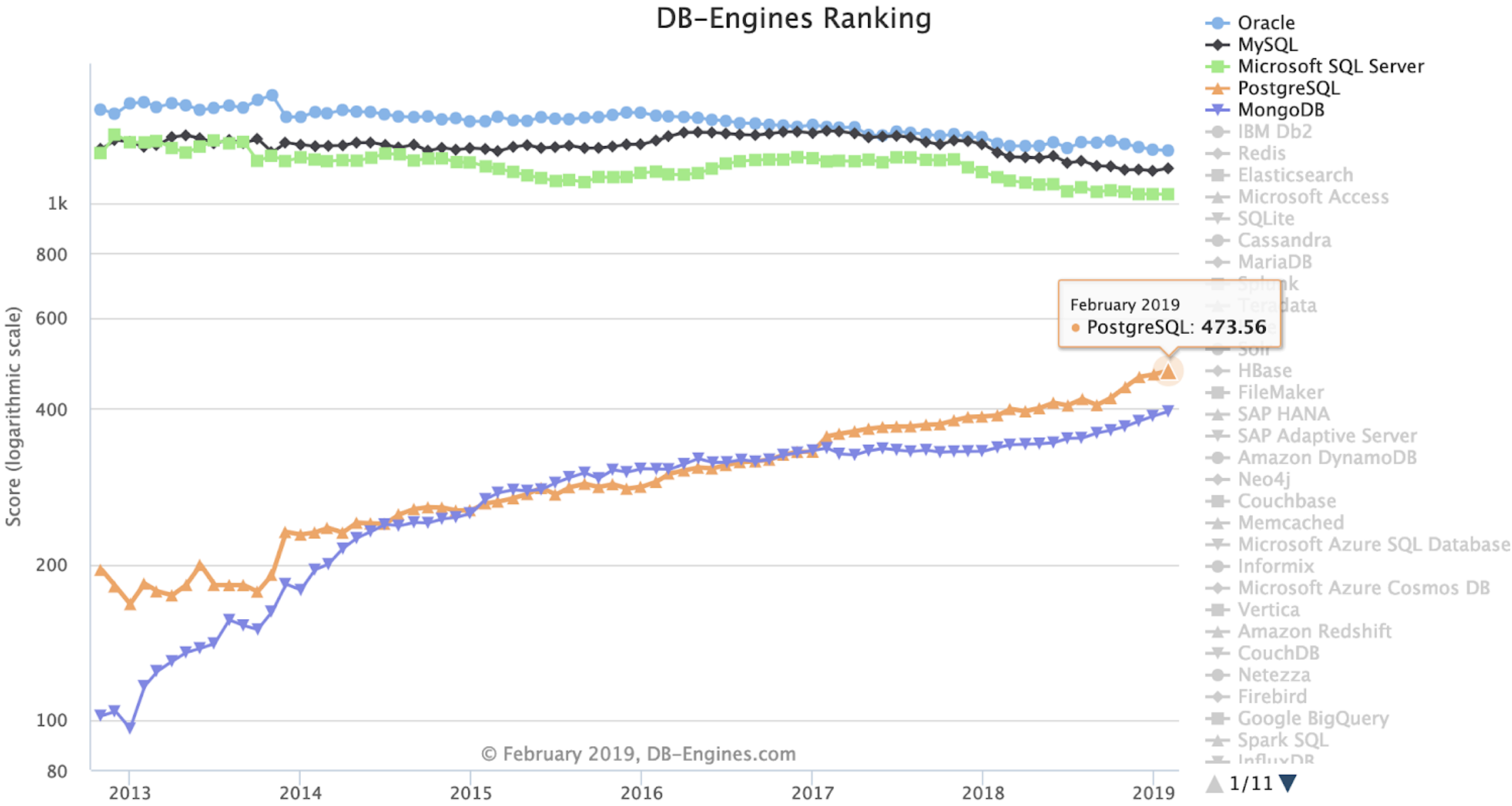
What is PostgreSQL

- Open Source Object-Relational Database System
- Excellent support for Standard Query Language (SQL)
- Very closely adheres to the ANSI-SQL standard
- Fully ACID Compliant

What is PostgreSQL

- "BSD" style license
  - Use, copy, modify, and distribute for any purpose
  - No requirement to contribute changes back

- Many commercial forks exist (they typically can't keep up)
- Entirely written in C


DB-Engines Ranking
Score (logarithmic scale)
1k
800
600
400
200
100
80
2013
2014
2015
2016
2017
2018
2019
© February 2019, DB-Engines.com
February 2019
PostgreSQL: 473.56
Oracle
MySQL
Microsoft SQL Server
PostgreSQL
MongoDB
IBM Db2
Redis
Elasticsearch
Microsoft Access
SQLite
Cassandra
MariaDB
HBase
FileMaker
SAP HANA
SAP Adaptive Server
Amazon DynamoDB
Neo4j
Couchbase
Memcached
Microsoft Azure SQL Database
Informix
Microsoft Azure Cosmos DB
Vertica
Amazon Redshift
CouchDB
Netezza
Firebird
Google BigQuery
Spark SQL
1/11

History of PostgreSQL

- Began by Michael Stonebraker at the University of California Berkley
- "Postgres" name chosen as "after Ingres"
- Ingres developed from 1977 to 1985
- Postgres started in 1986 and under active development ever since
- Query language changed from "POSTQUEL" to SQL in 1995
- Released as open source under the name Postgres95
- Moved out of academia in 1996
- PostgreSQL officially began with version 6.0 in 1997

PostgreSQL Community

- Collaborators world-wide contribute to PostgreSQL
- Development is primarily through mailing lists
- Tracking of patches through "Commitfests"
- Various other technologies used for communication (IRC, Slack, Hangouts, etc)
- Different levels of contributors
  - Core team
  - Committers
  - Major Contributors
  - Contributors

PostgreSQL Core Team

Formally charged with coordinating release activities, handling confidential communication, deciding on policy, managing permissions (commit access), discipline, handling difficult decisions.

- Essentially Steering committee
- Generally avoids technical decision making
- Core team members:
  - Tom Lane
  - Peter Eisentraut
  - Magnus Hagander
  - Bruce Momjian
  - Dave Page

PostgreSQL Committers

Individuals with access to commit code directly to PostgreSQL. All changes to PostgreSQL go through one of the committers. Generally expected to have at least two committers involved in larger changes / patches.

- Responsible for committed code
- Currently 22 Committers
- Chosen by the Core committee

Major Contributors

Long-term contributors who have developed major features or have contributed to the PostgreSQL project in a significant way.

- Currently approximately 40 (around 20 non-committers)
- Minimum of 3-4 years of strong contributions typically
- Major vs. Regular contributors discussed at PGCon Dev Meeting

Regular Contributors

Regular contributors to PostgreSQL through submitting patches, providing patch reviews, or supporting the project in other ways.

- Currently about 50
- Individuals categorized as either Major Contributor or Contributor
- No overlap between the two sets
- Contributors + Major Contributors + Committers + Core = 100ish
- Many more individuals involved, reviewers, translators, etc.

Release Process

- Commitfest 1
- Commitfest 2
- Commitfest 3
- Commitfest 4
- Beta Stabilization Period
- Beta Release
- GA

Release Process

• Commitfest 1	• Sept 2017
• Commitfest 2	• November 2017
• Commitfest 3	• January 2018
• Commitfest 4	• March 2018
• Beta Stabilization Period	
• Beta Release	• Sept 20th 2018
• GA	• Oct 2018

What is this commitfest?

- All new features or bug patches are sent to a mailing list
- Strangely we don’t have a bug tracker, bugs are tracked in mailing lists
- Commitfest is actually an application which tracks
  - Issue
  - Mailing list messages about the issues and proposed fix
  - Patch to fix
  - Discussion about how to proceed
  - Reviewers
  - Commits

PostgreSQL 11

Released October 18, 2018
The theme of this release is performance, larger data sets and ease of use
Next major release will be fall 2019 PostgreSQL 12

Interesting Statistics

- 220 Commits
- 313 Contributors
- 3150 files changed

PostgreSQL development

- Initial features sometimes incomplete
- This lays the ground work or infrastructure
- Sometimes you need a partially complete feature to continue discussion
- The partially complete features allows people to “see” what the next step is (or isn’t)

Partitioning

- v10 added declarative partitioning, note you have to create the partitions manually.
- Before v10 partitioning was done through inheritance and rules
  - Very manual process
  - Very prone to error
- We did have constraint exclusion though
  - Partitions were removed if the table had a check constraint

Partition by range Example

```
CREATE TABLE MEASUREMENT(
        CITY_ID INT NOT NULL,
        LOGDATE DATE NOT NULL,
        PEAKTEMP INT,
        UNITSALES INT)
      PARTITION BY RANGE(LOGDATE);
```

Partition by range Example

- CREATE TABLE measurement_y2006m02

  PARTITION OF measurement
  FOR VALUES FROM ('2006-02-01') TO ('2006-03-01');

- CREATE TABLE measurement_y2006m03

  PARTITION OF measurement
  FOR VALUES FROM ('2006-03-01') TO ('2006-04-01');

Explain

```
explain select * from measurement where logdate='2006-03-01';
QUERY PLAN
------------------------------------------------------------------------------
 Append  (cost=0.00..33.12 rows=9 width=16)
   ->  Seq Scan on measurement_y2006m03  (cost=0.00..33.12 rows=9
width=16)
         Filter: (logdate = '2006-03-01'::date)
```

More explain

```
explain select * from measurement where logdate='2006-04-01';
                        QUERY PLAN
-----------------------------------------------------
 Result  (cost=0.00..0.00 rows=0 width=16)
   One-Time Filter: false
```

Partition by List Example

- Partition by list
- create table cities(

  city_id bigserial not null,

  name text not null,

  population bigint)

  partition by list (left(lower(name), 1));
- And a sub partition
  - create table cities_ab partition of cities for values in ('a','b');

explain

```
explain select * from cities where name='boo';
                        QUERY PLAN
-----------------------------------------------------------
 Append  (cost=0.00..23.38 rows=5 width=48)
   ->  Seq Scan on cities_ab  (cost=0.00..23.38 rows=5 width=48)
         Filter: (name = 'boo'::text)
```

Finally what's new in pg11

- Default partition
- Hash partitioning
- Primary keys on partitions
- Update row movement
- Faster pruning
- Runtime pruning

Default partition

We can now create default partitions in the event that the partition is not created

create table cities_partdef partition of cities default;

This provides a place for rows to go that do not have a partition (yet)

Hash Partitions

We now have the ability to create a table partitioned using a hash function

```
CREATE TABLE ORDERS(
    ORDER_ID BIGINT NOT NULL,
    CUST_ID BIGINT NOT NULL,
    STATUS TEXT)
      PARTITION BY HASH (ORDER_ID);
```

Hash Partitions

```
create table orders_p1 partition of orders for values with (modulus 4,
remainder 0);

create table orders_p1 partition of orders for values with (modulus 4,
remainder 2);

create table orders_p1 partition of orders for values with (modulus 4,
remainder 3);

create table orders_p1 partition of orders for values with (modulus 4,
remainder 4);
```

this is essentially partitioned randomly.

Primary keys on partitioned tables

Previous versions did not support primary key on partitioned tables

support for primary key, foreign key indexes and triggers on partitioned tables

Update Moves Rows

Before version 11 updates to data which changed the partition key did not move the rows to the correct partition

update statements will now move affected rows to the correct partition

Faster Pruning

- Pruning partitions is no longer uses an exhaustive linear search
  - Pruning partitions uses the partition definition instead of constraints
  - For list and range partitions a binary search is used
  - For hash partitions a hashing function finds the partition

Runtime Pruning

- We now have the ability to use parameters in prepared statements to do partition pruning
- The first stage of pruning is described above
- For prepared statements where the query parameters are only known at run time a second pruning phase is required

DBA Features

- Parallel CREATE INDEX
  - 2x-4x faster
- ALTER TABLE ADD COL non-null DEFAULT
- VACUUM skip index scan
- Validate checksums on base backup

Parallelism

Prerequisites for Parallel Queries

- max_parallel_workers_per_gather > 0
- dynamic_shared_memory_type != none
- in general the query must be deterministic.
- functions must be marked PARALLEL SAFE (ie they must be guaranteed to return)
- nested parallel queries are not allowed
- transaction isolation level not serializable

Parallel Scans (all in 10)

sequence - the tables blocks will be divided by cooperating processes. They are handed out one at a time so still sequential

bitmap heap - leader is chosen and the leader scans the index(es) and builds a bitmap of blocks that need to be scanned. These are distributed to the cooperating processes. So the heap scan is parallel, but the index scan is not.

index or index-only scan, supported only for b-tree indexes. Processes take turns reading from the index. Results are returned sorted in order.

Parallel Joins

nested loop join - inner side is non parallel, then the outer tuples are looked up with cooperating processes.

merge join - inner side is non parallel, same as above outer tuples are looked up with cooperating processes

hash join (new in pg11) without parallel the inner side is executed in full by every cooperating process. With parallel the work is divided

JIT compilation

This is experimental and requires you to build PostgreSQL “with-llvm”

At this point only some of the SQL code will be JIT compiled.

Substantial startup time (JIT) so useful for long running queries

PostgreSQL 12 this will be turned on by default

JIT planner simple

- Perform JIT if query cost > jit_above_cost
- Optimize if query cost > jit_optimize_above_cost
- Inline if query_cost > jit_inline_cost

SQL stored procedures

Now have real stored procedures

transactions inside procedures

new CREATE PROCEDURE command

executed with CALL procedure_name()

Live demo. What could go wrong ?

Channel binding for SCRAM auth

as of pg10 we now have SCRAM authentication.

SCRAM is a protocol which enables password authentication in a way that is not susceptible to man in the middle, or replay attacks.

The protocol is beyond scope for this talk, but briefly sends a number of messages which are all hashed using common hashing algorithms

pg11 increases the security by adding channel binding. This feature enables the client and server to be confident that there is no man in the middle

much, much more

https://www.postgresql.org/docs/devel/static/release-11.html

Conclusions

What's really new is that many more people are using it and migrating to it

much, much more

https://www.postgresql.org/docs/devel/static/release-11.html

Mastering PostgreSQL https://masteringpostgresql.com/

You can use the code **orlandocodecamp** for a 15% discount for the first 20 people.

Also check out https://pgloader.io/ written by the author.

THANK YOU!

Dave Cramer
dave.cramer@crunchydata.ca