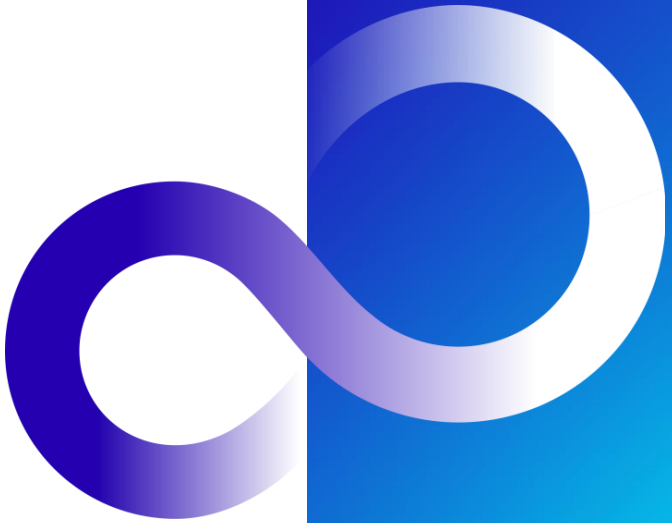
PostgreSQL 14 and beyond

Amit Kapila

PostgreSQL Committer and Major Contributor

FUJITSU



Self-introduction

FUJITSU

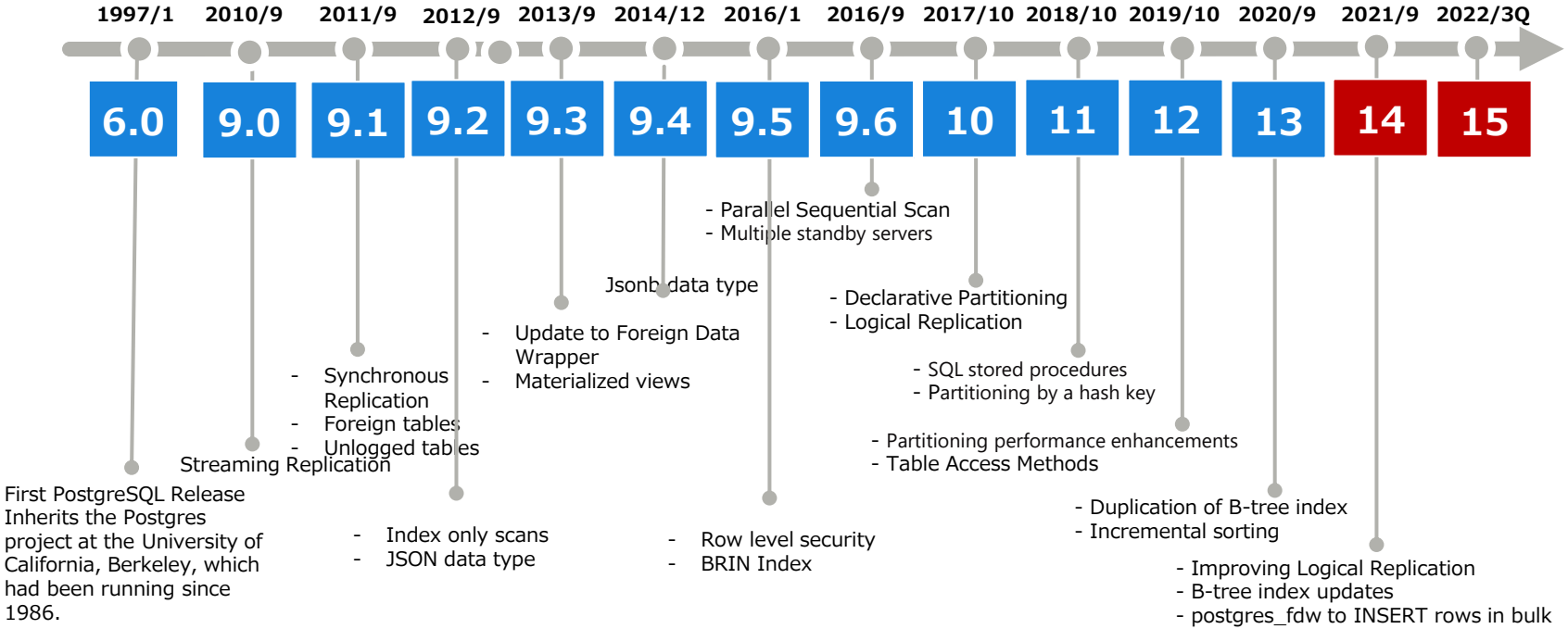
- Key milestones in my career
  - Exploring the world of databases at Oracle
  - Started working as an Open Source Developer in PostgreSQL community
  - Contributing to the PostgreSQL community as a committer
  - Leading PostgreSQL Development teams

- My specialized field of Contributions
  - Logical Replication, Parallel Query, Hash Indexes, Performance/Scalability Improvements

Evolution of the OSS database "PostgreSQL”

FUJITSU

- Ongoing version upgrades once a year
- Enhanced support for large volume data in recent years



Contents

FUJITSU

- Key features in PostgreSQL-14
- Performance improvements in PostgreSQL-14
- PostgreSQL-15 and beyond

- Disclaimer:
  - The PostgreSQL-15 and beyond is based on what I could see being proposed in community at this stage.

FUJITSU

Key features in PostgreSQL-14

Logical Replication Improvements(1)

FUJITSU

- Allow logical replication of in-progress transactions. This helps in reducing apply lag for large in-progress transactions but can be improved further. For more details about this features, you can read blog.

- Allow decoding of prepared transactions. This can help to build a two-phase commit for multi-master solution and help in reducing apply lag. Note that currently the subscriber-side work is yet to be done but out-of-core solutions can use it for their output plugins.

- Improve the performance of logical decoding of transactions containing DDLs. It has been observed that decoding of a transaction containing truncation of a table with 1000 partitions would be finished in 1s whereas before PostgreSQL-14 it used to take 4-5 minutes.

- Allow logical replication to transfer data in binary format. This is generally faster, if slightly less robust.

Logical Replication Improvements(2)

FUJITSU

- Allow multiple transactions during table sync in logical replication. This will (a) avoid copying the entire table again if error happens in synchronization phase, (b) avoid the risk of exceeding the CID limit, (c) remove the need to hold WAL till the entire synchronization is finished.

- The ALTER SUBSCRIPTION makes it easier to add/remove publications using the new ADD/DROP PUBLICATION syntax.

- Add system view pg_stat_replication_slots to report replication slot activity. This helps user to monitor the spill or stream activity. This also helps user to monitor the total bytes decoded via a particular slot.

- Read more about these improvements in blog.

SQL Features(1)

FUJITSU

- CREATE FUNCTION and CREATE PROCEDURE statements for language SQL with a function body conforms to the SQL standard and is portable to other implementations. Instead of the PostgreSQL-specific AS $$ string literal $$ syntax, this allows writing out the SQL statements making up the body unquoted, example:

```
CREATE PROCEDURE insert_data(a integer, b integer)  LANGUAGE SQL
    BEGIN ATOMIC
    INSERT INTO tbl VALUES (a);
    INSERT INTO tbl VALUES (b);
    END;
```

- Allow procedures to have OUT parameters. This will ease the migration.

- Add OR REPLACE for CREATE TRIGGER. This allows pre-existing triggers to be conditionally replaced. This will also ease the migration.

SQL Features(2)

FUJITSU

- ALTER TABLE ... DETACH PARTITION ... CONCURRENTLY
  - Allow a partition be detached from its partitioned table without blocking concurrent queries, by running in two transactions.
  - Because it runs in two transactions, it cannot be used in a transaction block.
  - In case the second transaction is cancelled, or a crash occurs, there's ALTER TABLE .. DETACH PARTITION .. FINALIZE, which executes the final steps.

- Allow TRUNCATE to operate on foreign tables. The postgres_fdw module also now supports this.

SQL Features(3)

FUJITSU

- Improved Subscripting. Allow extensions and built-in data types to implement subscripting. For example, now jsonb can use subscripting in below way:

Earlier:

```
SELECT jsonb_column->'key' FROM table;
UPDATE table SET jsonb_column =  jsonb_set(jsonb_column, '{"key"}', '"value"');
```

PostgreSQL-14:

```
SELECT jsonb_column['key'] FROM table;
UPDATE table SET jsonb_column['key'] = '"value"';
```

- Clearly, it would be much better for Updates.

SQL Features(4)

FUJITSU

- Add support for multirange data types. These are like range data types, but they allow the specification of multiple, ordered, non-overlapping ranges. All existing range types now also support multirange versions.

Earlier:

```
SELECT daterange(CURRENT_DATE, CURRENT_DATE + 1);
      daterange
------------------------
 [2021-07-27,2021-07-28)
```

PostgreSQL 14:

```
SELECT datemultirange( daterange(CURRENT_DATE, CURRENT_DATE + 2),
daterange(CURRENT_DATE + 5, CURRENT_DATE + 8));
                  datemultirange
 ------------------------------------------------
{[2021-07-27,2021-07-29),[2021-08-01,2021-08-04)}
```

SQL Features(5)

FUJITSU

- ECPG now supports a DECLARE STATEMENT construct.
  - Allow an ECPG SQL identifier to be linked to a specific connection.
  - When the identifier is used by dynamic SQL statements, these SQLs are executed by using the associated connection. This is done via DECLARE ... STATEMENT. For example,

```
EXEC SQL BEGIN DECLARE SECTION;
char dbname[128];
char *dym_sql = "SELECT current_database()";
EXEC SQL END DECLARE SECTION;

int main()
{
EXEC SQL CONNECT TO postgres AS con1;
EXEC SQL CONNECT TO testdb AS con2;
EXEC SQL AT con1 DECLARE stmt STATEMENT;
EXEC SQL PREPARE stmt FROM :dym_sql;
EXEC SQL EXECUTE stmt INTO :dbname;
printf("%s¥n", dbname);
EXEC SQL DISCONNECT ALL;
return 0;
}
```

Data Corruption

FUJITSU

- Allow amcheck to also check heap pages. Previously it only checked B-Tree index pages.

- Add command-line utility pg_amcheck to simplify running contrib/amcheck operations on many relations. It has a wide variety of options for choosing which relations to check and which checks to perform, and it can run checks in parallel if you want.

- Add a new functionality pg_surgery which allows changes to row visibility information. This is useful for correcting database corruption. OTOH, if misused, then this can easily corrupt a database that was not previously corrupted, or further corrupt an already-corrupted database, or to destroy data.

Indexing(1)

FUJITSU

- Allow some GiST indexes to be built by presorting the data. Presorting happens automatically and allows for faster index creation and smaller indexes. It is currently supported only for point datatypes.

- Allow BRIN indexes to record multiple min/max values per range.
  - This is useful if there are groups of values in each page range.
  - It allows more efficient handling of outlier values.
  - Its possible to specify the number of values kept for each page range, either as a single point or an interval boundary.

```
CREATE TABLE t (a int);
CREATE INDEX ON t  USING brin (a int4_minmax_multi_ops(values_per_range=16));
```

- Allow BRIN indexes to use bloom filters. This allows BRIN indexes to be used effectively with data that is not physically localized in the heap.

Indexing(2)

FUJITSU

- Allow SP-GiST to use INCLUDE'd columns. This will allow more index-only-scans for SP-GIST indexes.

- Allow REINDEX to process all child tables or indexes of a partitioned relation.

- Allow REINDEX to change the tablespace of the new index. This is done by specifying a TABLESPACE clause. A --tablespace option was also added to reindexdb to control this.

Extended Statistics

FUJITSU

- Allow extended statistics on expressions. A simple example may look like this:

```
CREATE TABLE t (a int);
CREATE STATISTICS s ON mod(a,10), mod(a,20) FROM t;
ANALYZE t;
```

The collected statistics are useful e.g. to estimate queries with those expressions in WHERE or GROUP BY clauses:

```
SELECT * FROM t WHERE mod(a,10) = 0 AND mod(a,20) = 0;
SELECT 1 FROM t GROUP BY mod(a,10), mod(a,20);
```

This feature will allow better query plans when expressions are used in queries.

- Increase the number of places extended statistics can be used for OR clause estimation.

Vacuuming(1)

FUJITSU

- Allow vacuum to skip index vacuuming when the number of removable index entries is insignificant. This reduces the time of vacuum due to skipping index scans in such cases.

- Allow vacuum operations to be more aggressive if the table is near xid or multixact wraparound. This is controlled by vacuum_failsafe_age and vacuum_multixact_failsafe_age. It is expected that this mechanism will almost always trigger within an autovacuum to prevent wraparound, long after the autovacuum began.

Vacuuming(2)

FUJITSU

- Speed up vacuuming of databases with many relations by using existing stats if possible. For example, when there are 20,000 tables and 10 autovacuum workers are running, the benchmark showed that the change improved the performance of autovacuum more than three times.

- Allow vacuum to eagerly add newly deleted btree pages to the free space map to allow them to reused. Previously vacuum could only place preexisting deleted pages in the free space map. This will inturn reduce the need to allocate new pages for btree index which will control its size.

- Allow vacuum to reclaim space used by unused trailing heap line pointers. This avoids line pointer bloat with certain workloads, particularly those involving continual range DELETEs and bulk INSERTs against the same table.

- Allow vacuum to be more aggressive in removing dead rows during Create Index Concurrently and Reindex Concurrently operations.

FUJITSU

Performance Improvements in PostgreSQL-14

Performance Improvements(1)

FUJITSU

- Improve the speed of computing MVCC visibility snapshots on systems with many CPUs and high session counts. This also improves performance when there are many idle sessions. The is ~2x gain for very large number of connections for read-only queries. See blog for more details.

- Improve the performance of updates/deletes on partitioned tables when only a few partitions are affected.
  - This also allows updates/deletes on partitioned tables to use execution-time partition pruning.
  - For inherited UPDATE/DELETE, instead of generating a separate subplan for each target relation, we now generate a single subplan that is just exactly like a SELECT's plan, then stick ModifyTable on top of that.

Performance Improvements(2)

FUJITSU

- Allow a query referencing multiple foreign tables to perform foreign table scans in parallel.
  - Currently, the only node type that can be run concurrently is a ForeignScan that is an immediate child of such an Append.
  - In the case where such ForeignScans access data on different remote servers, this would run those ForeignScans concurrently, and overlap the remote operations to be performed simultaneously, so it'll improve the performance especially when the operations involve time-consuming ones.
  - postgres_fdw supports this type of scan if async_capable is set.

- Add ability to use LZ4 compression on TOAST data. Read more about this feature in blog.
  - This can be set at the column level, or set as a default via server setting default_toast_compression.
  - The server must be compiled with --with-lz4 to support this feature; the default is still pglz.
  - The LZ4 gets better compression than PGLZ with less CPU usage.
  - Some of tests shown the speed is improved by more than 2 times and the size is slightly bigger after using lz4.
  - I suggest to test this with the production data before using any one of the methods.

Performance Improvements(3)

FUJITSU

- Allow btree index additions to remove expired index entries to prevent page splits.
  - This is particularly helpful for reducing index bloat on tables whose indexed columns are frequently updated.
  - This mechanism attempts to delete whatever duplicates happen to be present on the page when the duplicates are suspected to be caused by version churn from successive UPDATEs.
  - Changing the value of only one column covered by one index during an UPDATE always necessitates a new set of index tuples — one for each and every index on the table. Note in particular that this includes indexes that were not “logically modified” by the UPDATE. This optimization is helpful for such indexes.

- Implement pipeline mode in libpq.
  - This allows multiple queries to be sent and only wait for completion when a specific synchronization message is sent.
  - It increases client application complexity, and extra caution is required to prevent client/server deadlocks, but pipeline mode can offer considerable performance improvements, in exchange for increased memory usage from leaving state around longer.
  - Pipeline mode is most useful when the server is distant, i.e., network latency "ping time" is high, and also when many small operations are being performed in rapid succession.

Performance Improvements(4)

FUJITSU

- Add executor method to cache results from the inner-side of nested loop joins.
  - This is useful if only a small percentage of rows is checked on the inner side and is controlled by GUC enable_memoize.
  - The benefits of using a parameterized nested loop with a result cache increase when there are fewer distinct values being looked up and the number of lookups of each value is large.

- Extend the FDW API (and postgres_fdw) to allow batching inserts into foreign tables.
  - The idea is that if the FDW supports batching, and batching is requested, accumulate rows and insert them in batches. Otherwise use the per-row inserts.
  - The batching is usually much more efficient than inserting individual rows, due to high latency for each round-trip to the foreign server.

- Speed up queries having expr IN (const-1, const-2, etc.) clause. This is achieved by replacing current linear search with hash table look up for such statements.

Performance Improvements(5)

FUJITSU

- Speed up truncation, drop, or abort of create table operations on small tables during recovery on clusters with a large number of shared buffers.
  - This improves the performance by more than 100 times in many cases when several small tables (tested with 1000 relations) are truncated and where the server is configured with a large value of shared buffers (greater than equal to 100GB).

- Improve speed of recovery, standby apply, and vacuum for large updates. The performance improvement comes from optimizing page compaction algorithm which we need to use after large updates.

- Improve the I/O performance of parallel sequential scans. This was done by allocating blocks in groups to parallel workers.

FUJITSU

PostgreSQL-14 complete feature list

FUJITSU

PostgreSQL-15 and beyond

PostgreSQL-15 and beyond(1)

FUJITSU

- Various improvements in Logical Replication
  - Subscriber-side 2PC support
  - Allow publications for Schema.
  - Some options/tools to allow conflict resolution.
  - Replication of sequences.
  - Row-level filters which will facilitate sharding of data
  - Column-level filtering
  - Improve network bandwidth by not sending empty transactions
  - Enable logical replication from standby
- Server-side compression in back up technology.
- Improve automatic switchover/failover technology.

PostgreSQL-15 and beyond(2)

FUJITSU

- Various improvements in Hash Indexes
  - Allow unique indexes
  - Allow multi-column indexes
- Shared memory stats collector.
  - more reliable as we don’t need to communicate over udp protocol
  - performant, due to lesser reads and writes
- Parallel Writes.
  - Parallel Inserts
  - Parallel Copy From …
  - Improvements in Parallel Query

PostgreSQL-15 and beyond(3)

FUJITSU

- Asynchronous I/O. This will allow to prefetch data and improve the speed of the system.
- Direct I/O: This will bypass the OS cache and lead to better performance in some cases.
- 2PC via FDW. This is to further advance the PostgreSQL based sharding solution.
- Improvements in vacuum technology by using performance data structure.
- Improvements in partitioning technology.
- Global Temporary Table: Better management of temporary table and ease of migration.
- Incremental maintenance of materialized views

Thank you