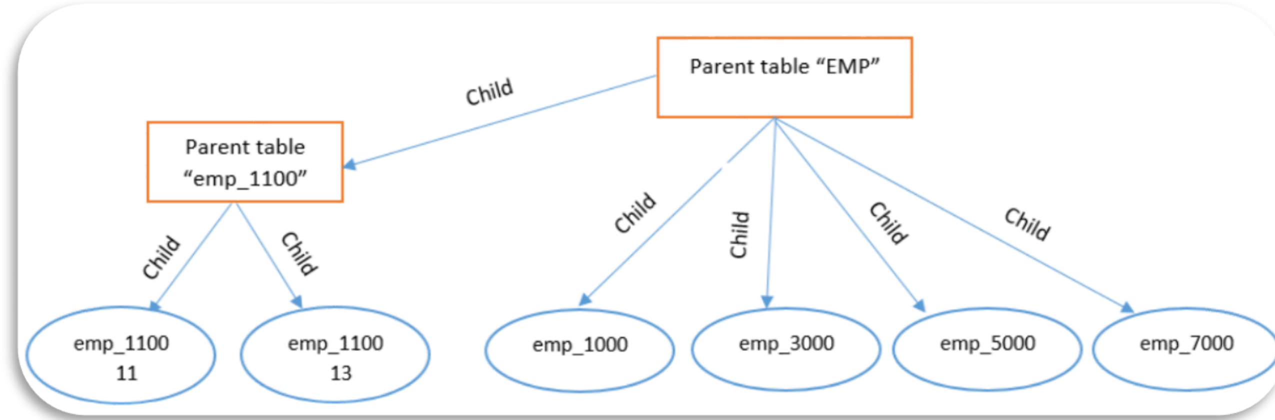
FUJITSU
shaping tomorrow with you
"Divide and Rule"
Partitioning in PostgreSQL11
Rajni Baliyan

Partitioning?

- Subdivide a parent table into a number of smaller child tables/partitions

https://www.elephind.com



https://www.americanantiquarian.org/recent-acquisitions-newspaper-department

© 2018 Fujitsu Australia Software Technology

Agenda

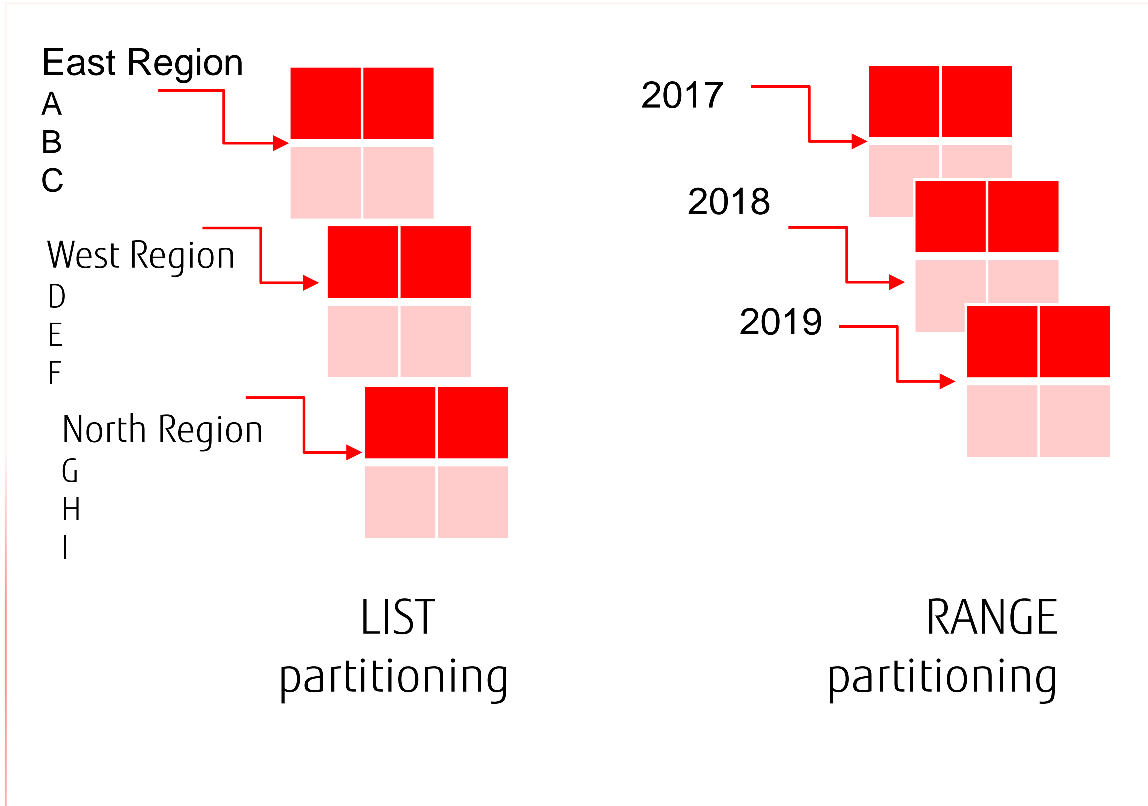
FUJITSU

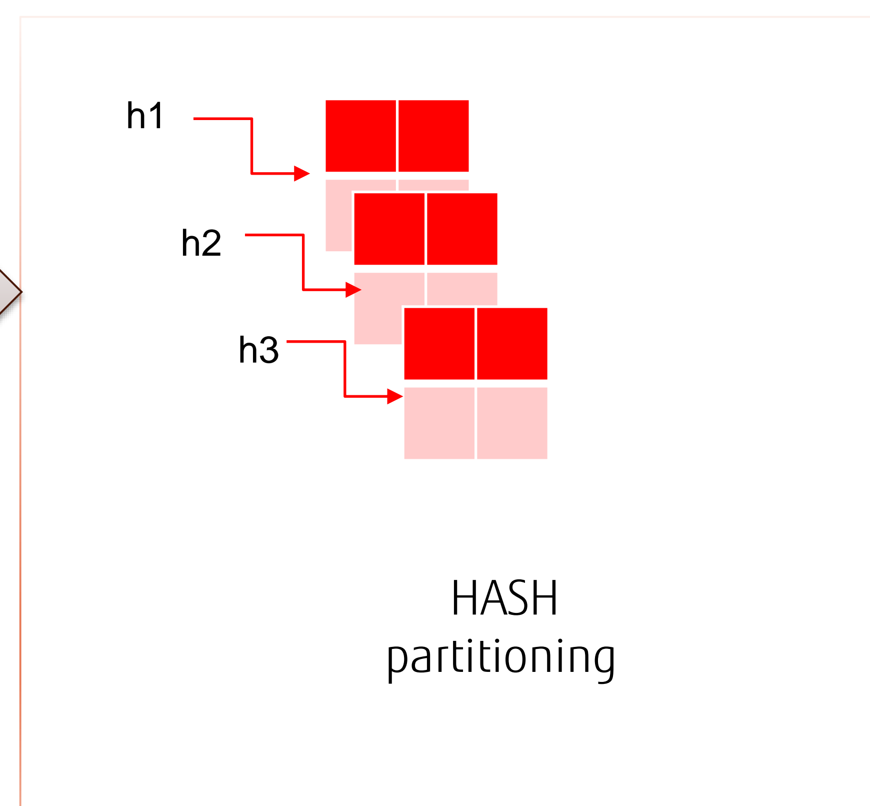
- Partitioning in PostgreSQL
- Partitioning benefits
- When to partition?
- Partitioning exceptions
- When not to use partitioning?
- Partitioning limitations in PG10
- Partitioning Improvements in PG11
- Useful commands
- Limitations in PG11
- What’s next? PG12…
- PG native partitioning vs pg_partman
- Questions

© 2018 Fujitsu Australia Software Technology

Partitioning in PostgreSQL

- Inheritance partitioning prior to PG10
- Declarative from PG10





© 2018 Fujitsu Australia Software Technology

Partitioning benefits

FUJITSU

- Easy maintenance of big tables.
- Address performance issue because of data growth over the period of time.
- Improves query performance.
- Address I/O performance issues by keeping partitions on different tablespaces.
- Address storage issues- partitions can spin across multiple tablespaces and disk file systems.
- Transparent to application.
- Best suited for applications where data grow is enormous and only recent data is required – IoT etc.

© 2018 Fujitsu Australia Software Technology

Partitioning benefits

FUJITSU

FAST

- Queries will access only relevant partitions.
- Reporting queries access most or all of the data in an entire partition
- Better I/O

FLEXIBLE

- Easy maintenance- adding and removing of partition is easy.
- Archiving of historic data.
- Easy backup and restore of partition.
- Add new table as partition of existing partitioned table.

CHEAPER

- Maintenance of INDEXES

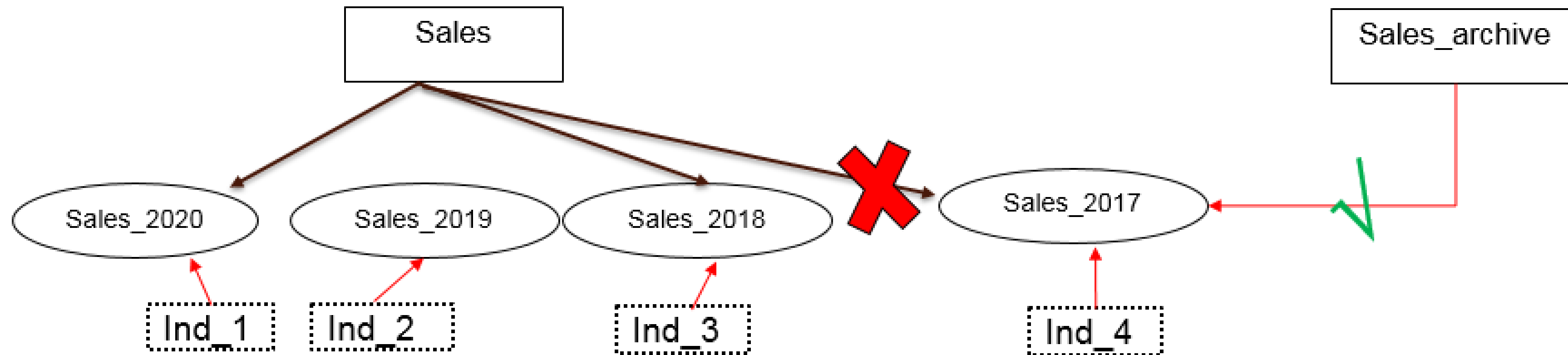
ATTENTION!!!

identifying right partitioning type and partitioning key

© 2018 Fujitsu Australia Software Technology

When to partition?

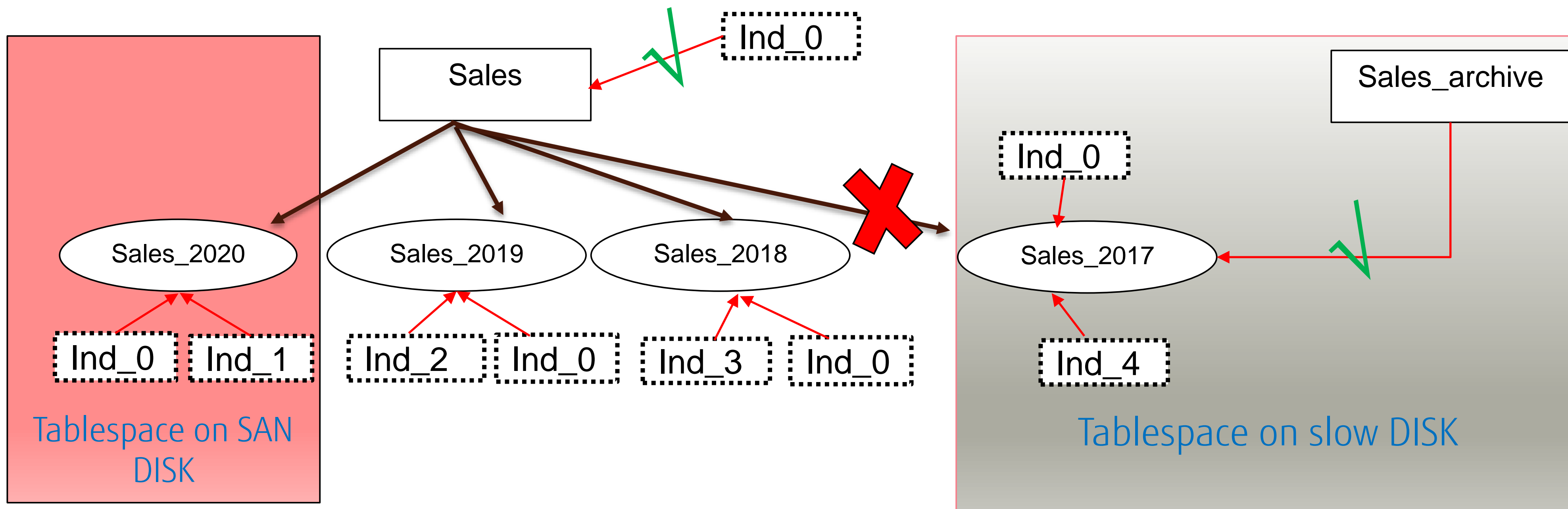
- Some suggestions ,when to partition-
- Table size is very big.
- Data archiving is the requirement.



© 2018 Fujitsu Australia Software Technology

When to partition?

- Better I/O- when content of the table needs to be distributed across different types of Storage devices to achieve better I/O



© 2018 Fujitsu Australia Software Technology

Partitioning exceptions

FUJITSU

- CHECK and NOT NULL constraints-
  - inherit by partitions from partitioned tables.
  - CHECK constraints marked “NO INHERIT” are not allowed to be created on partitioned tables.
- ONLY to add or drop a constraint-
  - supported on partitioned table as long as there are no partitions.
  - ONLY will result in an error as adding or dropping constraints on only the partitioned table, when partitions exist, is not supported
- TRUNCATE ONLY on a partitioned table will always return an error.
- Columns should be same in partitioned table and partitions.

© 2018 Fujitsu Australia Software Technology

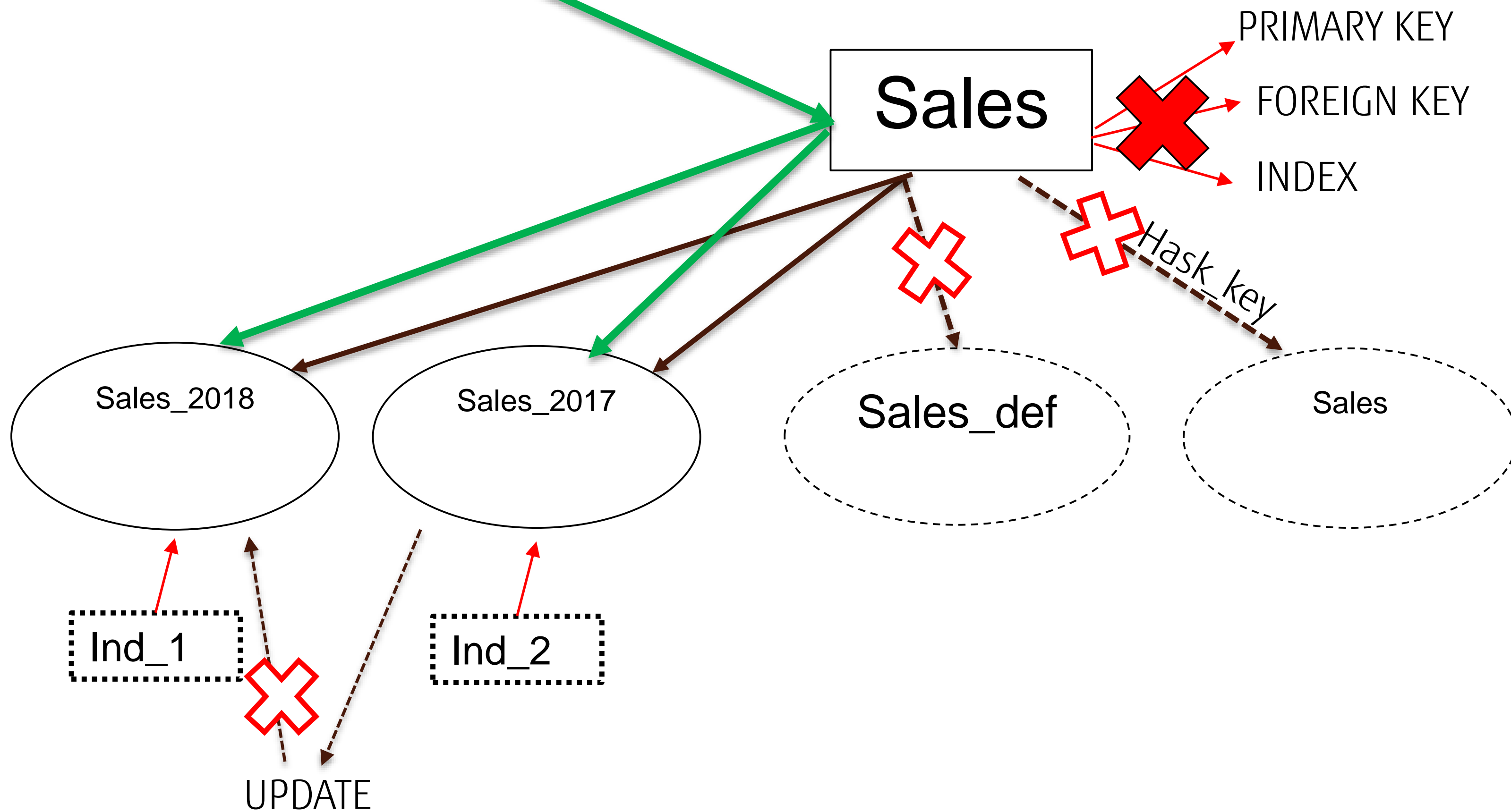
When not to use partitioning?

FUJITSU

- Table size is not too big.
- No performance issues faced.
- Application is mostly read/write intensive.
- If partitioning need matches the exceptions.
- Partitioned key is not in WHERE clause of the query.
- When INDEX manageability is the known problem.

© 2018 Fujitsu Australia Software Technology

Partitioning limitations in PG10



© 2018 Fujitsu Australia Software Technology

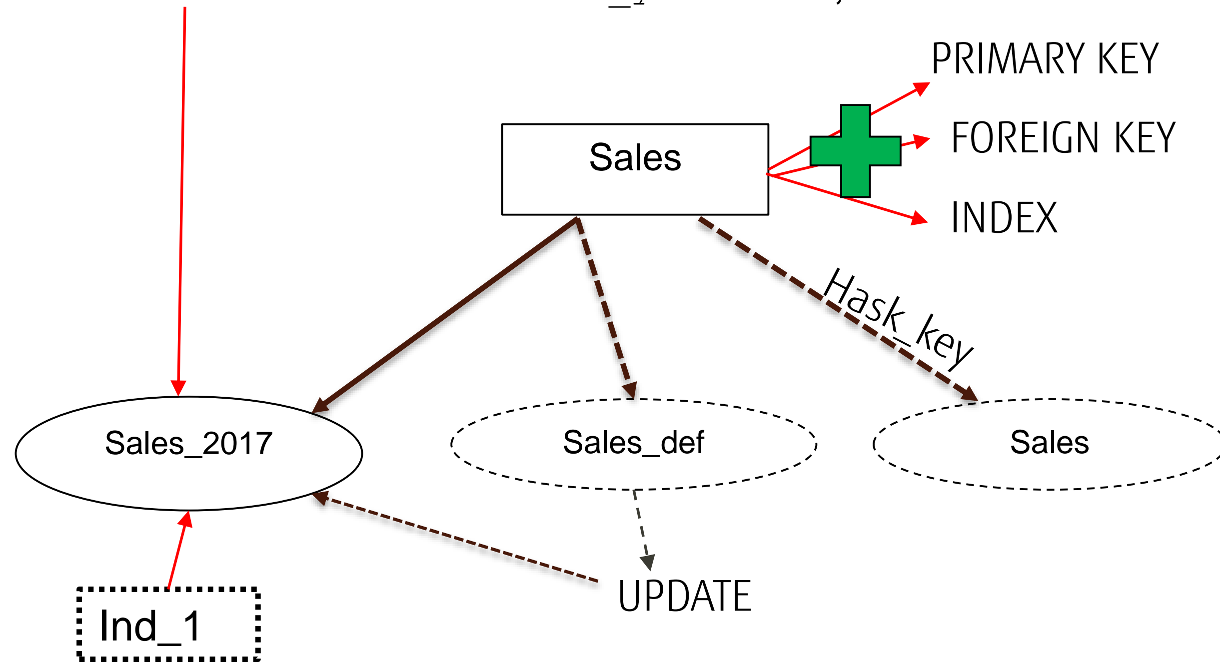
Partitioning limitations in PG10

FUJITSU

- No INDEX, PRIMARY key, UNIQUE constraint, or exclusion constraint spanning all partitions automatically.
- No support for HASH partitioning.
- No default partition.
- FOREIGN keys referencing partitioned tables are not supported
- No pruning of partitions during query. Results in poor performance.
- No row movement across partitions when doing UPDATE.
- Error while using the ON CONFLICT clause with partitioned tables will cause an error
- Trigger based rules.

© 2018 Fujitsu Australia Software Technology

Partitioning Improvements in PG11



© 2018 Fujitsu Australia Software Technology

Partitioning Improvements in PG11

FUJITSU

- Support for PRIMARY KEY, FOREIGN KEY, indexes, and triggers on partitioned tables.
- Parent INDEX automatically applicable to partitioned tables.
- Allow a DEFAULT partition for non-matching rows.
- Partition by a hash key- hash partitioning.
- Row movement across partitions on UPDATE
- Improve SELECT performance through enhanced partition elimination strategies during query planning and execution
- No more trigger functions to be created.

In general, partitions now have most of the capabilities of ordinary tables.

© 2018 Fujitsu Australia Software Technology

Let's start with partitioning- RANGE

FUJITSU

- Continuous data distribution based on predicted range of values.
- Create table "emp" and five partition by "RANGE" and insert some rows.
  - CREATE TABLE emp (emp_id int, emp_name text, joining_date date not null) PARTITION BY RANGE (emp_id);
    - CREATE TABLE emp_1000 PARTITION OF emp FOR VALUES FROM (1000) TO (3000);
    - CREATE TABLE emp_3000 PARTITION OF emp FOR VALUES FROM (3000) TO (5000);
  - INSERT INTO EMP (emp_id,emp_name,joining_date) VALUES (1001,'AA','2016-09-30');
  - INSERT INTO EMP (emp_id,emp_name,joining_date) VALUES (3501,'AAB','2017-07-5');

```
demo112=# \d+ emp
                                        Table "public.emp"
    Column    |  Type   | Collation | Nullable | Default | Storage  | Stats target | Description
--------------+---------+-----------+----------+---------+----------+--------------+-------------
 emp_id       | integer |           |          |         | plain    |              |
 emp_name     | text    |           |          |         | extended |              |
 joining_date | date    |           | not null |         | plain    |              |
Partition key: RANGE (emp_id)
Partitions: emp_1000 FOR VALUES FROM (1000) TO (3000),
            emp_3000 FOR VALUES FROM (3000) TO (5000),
            emp_5000 FOR VALUES FROM (5000) TO (7000),
            emp_7000 FOR VALUES FROM (7000) TO (9000),
            emp_9000 FOR VALUES FROM (9000) TO (11000)
```


© 2018 Fujitsu Australia Software Technology

Let’s start with partitioning- RANGE

FUJITSU

- Sub-partitioning- adding table with partitions to previously created partitioned table.
- Create parent table-

```
CREATE TABLE emp_1100(LIKE EMP) PARTITION BY RANGE(EMP_ID);
```

- Create two partitions-

```
CREATE TABLE emp_1100_11 PARTITION OF emp_1100 FOR VALUES FROM (11000) TO
(13000);
CREATE TABLE emp_1100_13 PARTITION OF emp_1100 FOR VALUES FROM (13000) TO
(15000);
```


© 2018 Fujitsu Australia Software Technology

Let's start with partitioning- RANGE

FUJITSU

- In PG11-

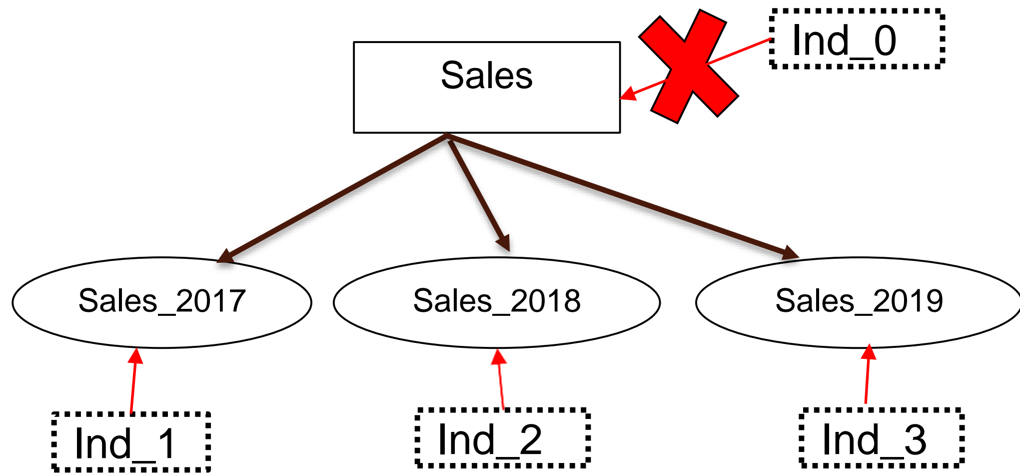
```
demo11=# \d+ emp
                                          Table "public.emp"
    Column     |  Type   | Collation | Nullable | Default | Storage  | Stats target | Description
---------------+---------+-----------+----------+---------+----------+--------------+-------------
 emp_id        | integer |           | not null |         | plain    |              |
 emp_name      | text    |           |          |         | extended |              |
 joining_date  | date    |           | not null |         | plain    |              |
Partition key: RANGE (emp_id)
Indexes:
    "emp_pkey" PRIMARY KEY, btree (emp_id)
Partitions: emp_1000 FOR VALUES FROM (1000) TO (3000),
            emp_1100 FOR VALUES FROM (11000) TO (15000), PARTITIONED,
            emp_3000 FOR VALUES FROM (3000) TO (5000),
            emp_5000 FOR VALUES FROM (5000) TO (7000),
            emp_7000 FOR VALUES FROM (7000) TO (9000),
            emp_9000 FOR VALUES FROM (9000) TO (11000)
```

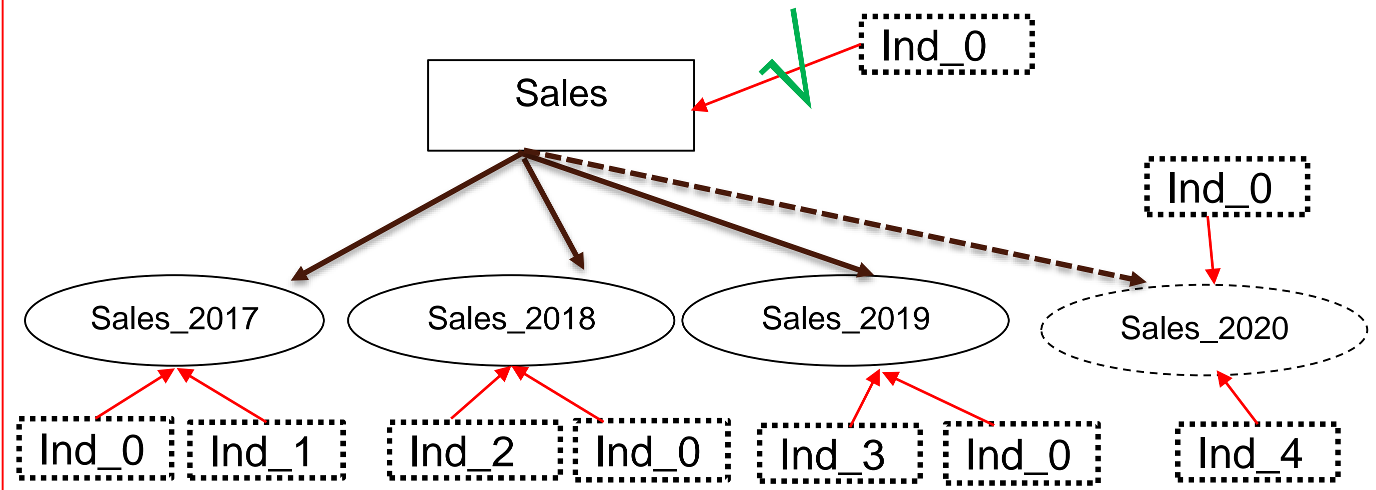
- PG10-

```
demo10=# \d+ emp
                                          Table "public.emp"
    Column     |  Type   | Collation | Nullable | Default | Storage  | Stats target | Description
---------------+---------+-----------+----------+---------+----------+--------------+-------------
 emp_id        | integer |           |          |         | plain    |              |
 emp_name      | text    |           |          |         | extended |              |
 joining_date  | date    |           | not null |         | plain    |              |
Partition key: RANGE (emp_id)
Partitions: emp_1000 FOR VALUES FROM (1000) TO (3000),
            emp_1100 FOR VALUES FROM (11000) TO (15000),
            emp_3000 FOR VALUES FROM (3000) TO (5000),
            emp_5000 FOR VALUES FROM (5000) TO (7000),
            emp_7000 FOR VALUES FROM (7000) TO (9000),
            emp_9000 FOR VALUES FROM (9000) TO (11000)
```


© 2018 Fujitsu Australia Software Technology

PG11- Automatic Index Creation





© 2018 Fujitsu Australia Software Technology

PG11- Automatic Index Creation

FUJITSU

- PG10-
  - Manual on each partition :

  - fails on parent table :
- PG11-
  - Query performance will improve as it has to smaller data set having partition key in WHERE Clause.
  - CREATE INDEX now also possible on parent.

  - Cascade to each existing and new partitions
    - Attach Index- If same INDEX already exist.
    - Create Index- if no INDEX .

```
demo10=# create index ind on measurement(peaktemp);
ERROR:  cannot create index on partitioned table "measurement"
```

```
demo10=# create index ind on measurement_y2017 (peaktemp);
CREATE INDEX
```

```
demo11=# CREATE INDEX ind_parent on emp (joining_date);
CREATE INDEX
```

```
demo11=# \d emp_1100
                    Table "public.emp_1100"
    Column     |  Type   | Collation | Nullable | Default
---------------+---------+-----------+----------+---------
 emp_id        | integer |           | not null |
 emp_name      | text    |           |          |
 joining_date  | date    |           | not null |
Partition of: emp FOR VALUES FROM (11000) TO (15000)
Partition key: RANGE (emp_id)
Indexes:
    "emp_1100_pkey" PRIMARY KEY, btree (emp_id)
    "emp_1100_joining_date_idx" btree (joining_date)
    "ind_child" btree (emp_name)
Number of partitions: 2 (Use \d+ to list them.)
```


© 2018 Fujitsu Australia Software Technology

PG11- Foreign Key Support

- PG10- No column as FOREIGN KEY in partitioned table

```
demo10=#  CREATE TABLE depart(dep_id numeric PRIMARY KEY) ;
CREATE TABLE
demo10=# \d+ depart
                                 Table "public.depart"
 Column |  Type   | Collation | Nullable | Default | Storage | Stats target | Description
--------+---------+-----------+----------+---------+---------+--------------+-------------
 dep_id | numeric |           | not null |         | main    |              |
Indexes:
    "depart_pkey" PRIMARY KEY, btree (dep_id)

demo10=# CREATE TABLE dep_head( head_id numeric REFERENCES depart(dep_id)) PARTITION BY RANGE (head_id);
ERROR:  foreign key constraints are not supported on partitioned tables
LINE 1: CREATE TABLE dep_head( head_id numeric REFERENCES depart(dep...
                                                ^
```

- PG11: FOREIGN KEYs are allowed. But no FK reference to the partitioned master table.

```
demo11=#  CREATE TABLE depart(dep_id numeric PRIMARY KEY) ;
CREATE TABLE
demo11=# \d+ depart
                                 Table "public.depart"
 Column |  Type   | Collation | Nullable | Default | Storage | Stats target | Description
--------+---------+-----------+----------+---------+---------+--------------+-------------
 dep_id | numeric |           | not null |         | main    |              |
Indexes:
    "depart_pkey" PRIMARY KEY, btree (dep_id)

demo11=# CREATE TABLE dep_head( head_id numeric REFERENCES depart(dep_id)) PARTITION BY RANGE (head_id);
CREATE TABLE
demo11=# \d+ dep_head
                                 Table "public.dep_head"
 Column  |  Type   | Collation | Nullable | Default | Storage | Stats target | Description
---------+---------+-----------+----------+---------+---------+--------------+-------------
 head_id | numeric |           |          |         | main    |              |
Partition key: RANGE (head_id)
Foreign-key constraints:
    "dep_head_head_id_fkey" FOREIGN KEY (head_id) REFERENCES depart(dep_id)
Number of partitions: 0
```


© 2018 Fujitsu Australia Software Technology

PG11- Row migration on UPDATE

- UPDATE statements can move a row across partition boundaries.
- This occurs when the update happens to affect a column that participates in defining the boundaries.
- Frequently doing so might defeat the purpose of partitioning.
- Not available in PG10. Give error-

```
demo10=#  UPDATE emp SET emp_id =9100 WHERE joining_date='2014-02-12';
ERROR:  new row for relation "emp_1000" violates partition constraint
DETAIL:  Failing row contains (9100, AAL, 2014-02-12).
```

- In PG11- moves rows across partitions.

```
demo11=#  SELECT relname FROM pg_class WHERE oid = (SELECT tableoid FROM emp  where joining_date= '2011-06-10');
 relname
----------
 emp_9000
(1 row)

demo11=# UPDATE emp SET emp_id =1000 WHERE joining_date='2011-06-10';
UPDATE 1
demo11=#  SELECT relname FROM pg_class WHERE oid = (SELECT tableoid FROM emp  where joining_date= '2011-06-10');
 relname
----------
 emp_1000
(1 row)
```


© 2018 Fujitsu Australia Software Technology

PG11- Hash partitioning

FUJITSU

- Hash partitioning is a method to separate out information in a randomized way rather than putting the data in the form of groups unlike RANGE partitioning.
- Divide rows (more or less) equally into multiple partitions
- Much useful for data ware house kind of application.
- Partitioning is based on module and remainder.
- An INSERT statement that does not match the hash value will fail when storing tuples directly on a partition.

```
demo11=# \d+ hash_key
                                      Table "public.hash_key"
 Column |  Type   | Collation | Nullable | Default | Storage  | Stats target | Description
--------+---------+-----------+----------+---------+----------+--------------+-------------
 id     | numeric |           |          |         | main     |              |
 name   | text    |           |          |         | extended |              |
Partition key: HASH (id)
Partitions: hash_key_1 FOR VALUES WITH (modulus 4, remainder 1),
            hash_key_2 FOR VALUES WITH (modulus 4, remainder 2),
            hash_key_3 FOR VALUES WITH (modulus 4, remainder 3)

demo11=#
demo11=#
demo11=#
demo11=# INSERT INTO hash_key SELECT generate_series(0, 99999), 'A';
ERROR:  no partition of relation "hash_key" found for row
DETAIL:  Partition key of the failing row contains (id) = (4).
```


© 2018 Fujitsu Australia Software Technology

PG11- Hash partitioning

SYNTAX:

CREATE TABLE <table_name> (col1 numeric, col2 text) **PARTITION BY hash** (hash_key);

CREATE TABLE part1_name PARTITION OF <table_name> FOR VALUES WITH (**MODULUS** 4, **REMAINDER** 0);

```
demo11=# CREATE TABLE hash_key (id numeric, name text) PARTITION BY hash (id);
CREATE TABLE
demo11=# CREATE TABLE hash_key_0 PARTITION OF hash_key FOR VALUES WITH (MODULUS 4, REMAINDER 0);
CREATE TABLE hash_key_1 PARTITION OF hash_key FOR VALUES WITH (MODULUS 4, REMAINDER 1);
CREATE TABLE hash_key_2 PARTITION OF hash_key FOR VALUES WITH (MODULUS 4, REMAINDER 2);
CREATE TABLE hash_key_3 PARTITION OF hash_key FOR VALUES WITH (MODULUS 4, REMAINDER 3);

CREATE TABLE
```

Note:

- MODULUS is number of partitions, and REMAINDER is number, 0 or more, but less than MODULUS
  - MODULUS clause value > REMAINDER clause
  - Number of partitions >= MODULUS value, else Insert error

© 2018 Fujitsu Australia Software Technology

PG11- Hash partitioning

FUJITSU

- Advantages of HASH partitioning over RANGE partitioning.
  - Not aware beforehand how much data will map into a given range.
  - The sizes of range partitions would differ quite substantially or would be difficult to balance manually
  - Avoid data skew in partitions.
  - Performance features such as parallel DML, partition pruning, and partition-wise joins are important
  - Maximize I/O throughput.
  - Partition pruning and partition-wise joins on a partitioning key are important.

© 2018 Fujitsu Australia Software Technology

PG11- DEFAULT partition

FUJITSU

- Default partition should exist prior to insert rows else result in error.
- With default partition- rows can be inserted outside of the defined range.
- It can be called as "catch all" partition
- Syntax:
  - `CREATE TABLE emp_default PARTITION OF emp DEFAULT;`
  - `INSERT INTO EMP (emp_id,emp_name,joining_date) VALUES (940305,'AAS','2014-03-11');`

```
demo11=# CREATE TABLE emp_default PARTITION OF emp DEFAULT;
CREATE TABLE
demo11=# \d+ emp
                                    Table "public.emp"
    Column     |  Type   | Collation | Nullable | Default | Storage  | Stats target | Description
---------------+---------+-----------+----------+---------+----------+--------------+-------------
 emp_id        | integer |           | not null |         | plain    |              |
 emp_name      | text    |           |          |         | extended |              |
 joining_date  | date    |           | not null |         | plain    |              |
Partition key: RANGE (emp_id)
Indexes:
    "emp_pkey" PRIMARY KEY, btree (emp_id)
Partitions: emp_1000 FOR VALUES FROM (1000) TO (3000),
            emp_1100 FOR VALUES FROM (11000) TO (15000), PARTITIONED,
            emp_3000 FOR VALUES FROM (3000) TO (5000),
            emp_5000 FOR VALUES FROM (5000) TO (7000),
            emp_9000 FOR VALUES FROM (9000) TO (11000),
            emp_default DEFAULT

demo11=# INSERT INTO EMP (emp_id,emp_name,joining_date) VALUES (630303,'AAR','2014-09-23');
INSERT 0 1
```


© 2018 Fujitsu Australia Software Technology

PG11- DEFAULT partition

FUJITSU

- Trying to create partition for inserted row- will fail

```
demo11=# INSERT INTO EMP (emp_id,emp_name,joining_date) VALUES (940305,'AAS','2014-03-11');
INSERT 0 1
demo11=#
demo11=# CREATE TABLE emp_1100_90 PARTITION OF emp FOR VALUES FROM (800000) TO (950000);
ERROR:  updated partition constraint for default partition "emp_default" would be violated by some row
```

- Create new partition for inserted rows –
  - Detach default→ Create new→ Move data→ → Attach new → Reattach default
    - `ALTER TABLE emp  DETACH PARTITION emp_default;`
    - `CREATE TABLE emp_def_pr1 (like emp);`
    - `INSERT INTO emp_def_pr1 (SELECT * FROM emp_default);`

```
demo11=# ALTER TABLE emp  DETACH PARTITION emp_default;
ALTER TABLE
demo11=# CREATE TABLE emp_def_pr1 (like emp);
CREATE TABLE
demo11=# INSERT INTO emp_def_pr1 (SELECT * FROM emp_default);
INSERT 0 1
demo11=# SELECT * FROM emp_def_pr1;
 emp_id | emp_name | joining_date
--------+----------+--------------
 940305 | AAS      | 2014-03-11
(1 row)
```


© 2018 Fujitsu Australia Software Technology

PG11- DEFAULT partition

FUJITSU

- Attach newly created partition to the parent ‘emp’ table.
  - ALTER TABLE emp ATTACH PARTITION emp_def_pr1 FOR VALUES FROM (800000) TO (1000000);

```
demo11=# ALTER TABLE emp ATTACH PARTITION emp_def_pr1 FOR VALUES FROM (800000) TO (1000000);
ALTER TABLE
```

- New data inserted for defined bounds will move to new partition.
  - INSERT INTO EMP (emp_id,emp_name,joining_date) VALUES (890305,'AAT','2018-07-21');
  - SELECT * FROM emp;

```
demo11=# INSERT INTO EMP (emp_id,emp_name,joining_date) VALUES (890305,'AAT','2018-07-21');
INSERT 0 1
demo11=# SELECT * FROM emp_def_pr1;
 emp_id | emp_name | joining_date
--------+----------+--------------
 940305 | AAS      | 2014-03-11
 890305 | AAT      | 2018-07-21
(2 rows)
```


© 2018 Fujitsu Australia Software Technology

PG11- DEFAULT partition

FUJITSU

- Re-attach default partition for future use.
  - ALTER TABLE emp ATTACH PARTITION emp_default DEFAULT;
  - \d+ emp
  - Test: INSERT INTO EMP (emp_id,emp_name,joining_date) VALUES (890305,'AAT','2018-07-21');

```
demo11=# ALTER TABLE emp  ATTACH PARTITION emp_default DEFAULT;
ALTER TABLE
demo11=# \d+ emp
                                       Table "public.emp"
    Column    |  Type   | Collation | Nullable | Default | Storage  | Stats target | Description
--------------+---------+-----------+----------+---------+----------+--------------+-------------
 emp_id       | integer |           | not null |         | plain    |              |
 emp_name     | text    |           |          |         | extended |              |
 joining_date | date    |           | not null |         | plain    |              |
Partition key: RANGE (emp_id)
Indexes:
    "emp_pkey" PRIMARY KEY, btree (emp_id)
Partitions: emp_1000 FOR VALUES FROM (1000) TO (3000),
            emp_1100 FOR VALUES FROM (11000) TO (15000), PARTITIONED,
            emp_3000 FOR VALUES FROM (3000) TO (5000),
            emp_5000 FOR VALUES FROM (5000) TO (7000),
            emp_9000 FOR VALUES FROM (9000) TO (11000),
            emp_def_pr1 FOR VALUES FROM (800000) TO (1000000),
            emp_default DEFAULT

demo11=# INSERT INTO EMP (emp_id,emp_name,joining_date) VALUES (1940305,'AAU','2014-03-11');
INSERT 0 1
demo11=# SELECT * FROM emp_default;
 emp_id  | emp_name | joining_date
---------+----------+--------------
 1940305 | AAU      | 2014-03-11
(1 row)
```


© 2018 Fujitsu Australia Software Technology

PG11- Partition pruning

FUJITSU

- Partition pruning is a query optimization technique that improves performance for declaratively partitioned tables.

- Significantly cheaper plan when enabled.

- This is possible by using parameter "`enable_partition_pruning`"
  - Can be set at session level-
    - `SET enable_partition_pruning=on;    -- on' by default`

© 2018 Fujitsu Australia Software Technology

PG11- Partition pruning

FUJITSU

- With partition pruning enabled, the planner will examine the definition of each partition and will include partitions meeting the query's WHERE clause.
  - When `enable_partition_pruning=on`

```
demo11=# set  enable_partition_pruning=on;
SET
demo11=#  EXPLAIN SELECT count(*) FROM measurement WHERE logdate >= DATE '2018-01-01';
                                            QUERY PLAN
---------------------------------------------------------------------------------------------------
 Aggregate  (cost=35.09..35.10 rows=1 width=8)
   ->  Append  (cost=9.42..33.38 rows=682 width=0)
         ->  Bitmap Heap Scan on measurement_y2018  (cost=9.42..27.92 rows=680 width=0)
               Recheck Cond: (logdate >= '2018-01-01'::date)
               ->  Bitmap Index Scan on measurement_y2018_pkey  (cost=0.00..9.25 rows=680 width=0)
                     Index Cond: (logdate >= '2018-01-01'::date)
         ->  Seq Scan on measurement_y2019_01  (cost=0.00..1.01 rows=1 width=0)
               Filter: (logdate >= '2018-01-01'::date)
         ->  Seq Scan on measurement_y2019_02  (cost=0.00..1.04 rows=1 width=0)
               Filter: (logdate >= '2018-01-01'::date)
(10 rows)
```


© 2018 Fujitsu Australia Software Technology

PG11- Partition pruning

FUJITSU

- Without partition pruning, same query will scan each partition.
  - When `enable_partition_pruning=off`

```
demo11=# show enable_partition_pruning;
 enable_partition_pruning
--------------------------
 off
(1 row)

demo11=#  EXPLAIN SELECT count(*) FROM measurement WHERE logdate >= DATE '2018-01-01';
                                                QUERY PLAN
------------------------------------------------------------------------------------------------------
 Aggregate  (cost=47.43..47.44 rows=1 width=8)
   ->  Append  (cost=0.00..45.71 rows=686 width=0)
         ->  Seq Scan on measurement_y2016  (cost=0.00..1.02 rows=1 width=0)
               Filter: (logdate >= '2018-01-01'::date)
         ->  Bitmap Heap Scan on measurement_y2017  (cost=4.18..11.28 rows=3 width=0)
               Recheck Cond: (logdate >= '2018-01-01'::date)
               ->  Bitmap Index Scan on measurement_y2017_pkey  (cost=0.00..4.17 rows=3 width=0)
                     Index Cond: (logdate >= '2018-01-01'::date)
         ->  Bitmap Heap Scan on measurement_y2018  (cost=9.42..27.92 rows=680 width=0)
               Recheck Cond: (logdate >= '2018-01-01'::date)
               ->  Bitmap Index Scan on measurement_y2018_pkey  (cost=0.00..9.25 rows=680 width=0)
                     Index Cond: (logdate >= '2018-01-01'::date)
         ->  Seq Scan on measurement_y2019_01  (cost=0.00..1.01 rows=1 width=0)
               Filter: (logdate >= '2018-01-01'::date)
         ->  Seq Scan on measurement_y2019_02  (cost=0.00..1.04 rows=1 width=0)
               Filter: (logdate >= '2018-01-01'::date)
(16 rows)
```


© 2018 Fujitsu Australia Software Technology

Useful commands

- Check table and partition description
  - `\d+ name`
- DETACH an existing partition.
  - `ALTER TABLE name DETACH PARTITION partition_name;`
- ATTACH a new partition.
  - `ALTER TABLE name  ATTACH PARTITION partition_name { FOR VALUES partition_bound_spec | DEFAULT };`
- How rows are distributed?
  - `SELECT * FROM emp, LATERAL (SELECT relname FROM pg_class WHERE pg_class.oid = emp.tableoid) AS table_name (emp) GROUP BY emp, emp_id, emp_name;`
- Which partition contains row?
  - `SELECT relname FROM pg_class WHERE oid = (SELECT tableoid FROM <tablename> where <condition>);`

© 2018 Fujitsu Australia Software Technology

Limitations in PG11

FUJITSU

- There is no way to create an exclusion constraint spanning all partitions- possible on individual partitions.
- FOREIGN keys referencing partitioned tables are not supported.
- When an UPDATE causes a row to move from one partition to another, there is a chance that another concurrent UPDATE or DELETE misses this row.
- BEFORE ROW triggers, if necessary, must be defined on individual partitions, not the partitioned table.
- Mixing temporary and permanent relations in the same partition tree is not allowed.

© 2018 Fujitsu Australia Software Technology

What's next? PG12...

FUJITSU

- There's still more to do here in the future.
- At the moment execution-time pruning only performs pruning of Append nodes. May be in future we can have pruning for MergeAppend or for ModifyTable nodes (UPDATE/DELETE)
- Enhancement in partition pruning performance and views.
- Never-the-less what we have for PG11 is a significant improvement over PG10!

© 2018 Fujitsu Australia Software Technology

PG native partitioning vs pg_partman

FUJITSU

Feature	9.6	PG 10	PG 11	pg_partman
Declarative partitioning	⊗	√	√	√
INSERT- auto routing	⊗	√	√	√* using triggers
UPDATE –auto routing	⊗	⊗	√	⊗
Foreign Key	⊗	⊗	√*	⊗
Unique Indexes	⊗	⊗	√	
Default partitioning	⊗	⊗	√	√* on parent table
Hash partitioning	⊗	⊗	√	√
Partition level aggregation /joins	⊗	⊗	√* constraint should match both sides	⊗
Partition pruning	⊗	⊗	√	⊗
Trigger based	√	⊗	⊗	√
Automatic child creation	⊗	⊗	⊗	√* using triggers
Automatic privileges transfer to new and existing child	⊗	⊗	√	√* using separate function

© 2018 Fujitsu Australia Software Technology