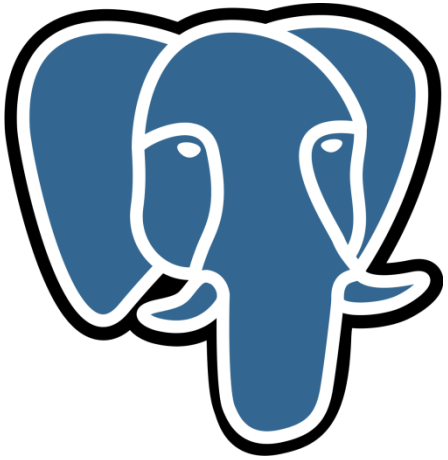
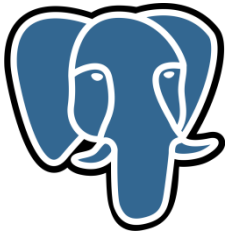


What's new in PostgreSQL 11?

@ COSCUP x GNOME.Asia x openSUSE.Asia 2018

Taiwan PostgreSQL User Group

林宗禧

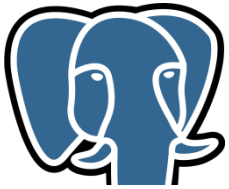


Outline

- About Me
- DB-Engines Ranking
- PostgreSQL 11 新特性
- PostgreSQL Global
- 台灣PostgreSQL使用者社群

About Me

- 我是林宗禧 (José Lin)
  - 2012 在此地畢業
  - 研究BIM+Cloud(Hadoop/HBase)
- 工作在研究分散式資料庫系統
  - 熟PG與MongoDB
  - 接觸多種NoSQL/NewSQL
  - 去中心化分散式架構 (2016-)
- 成為PostgreSQL愛好者
  - 2013年起拜訪國內PG愛好者、認識日本JPUG
  - 2017年起著手PG社群...直至今日



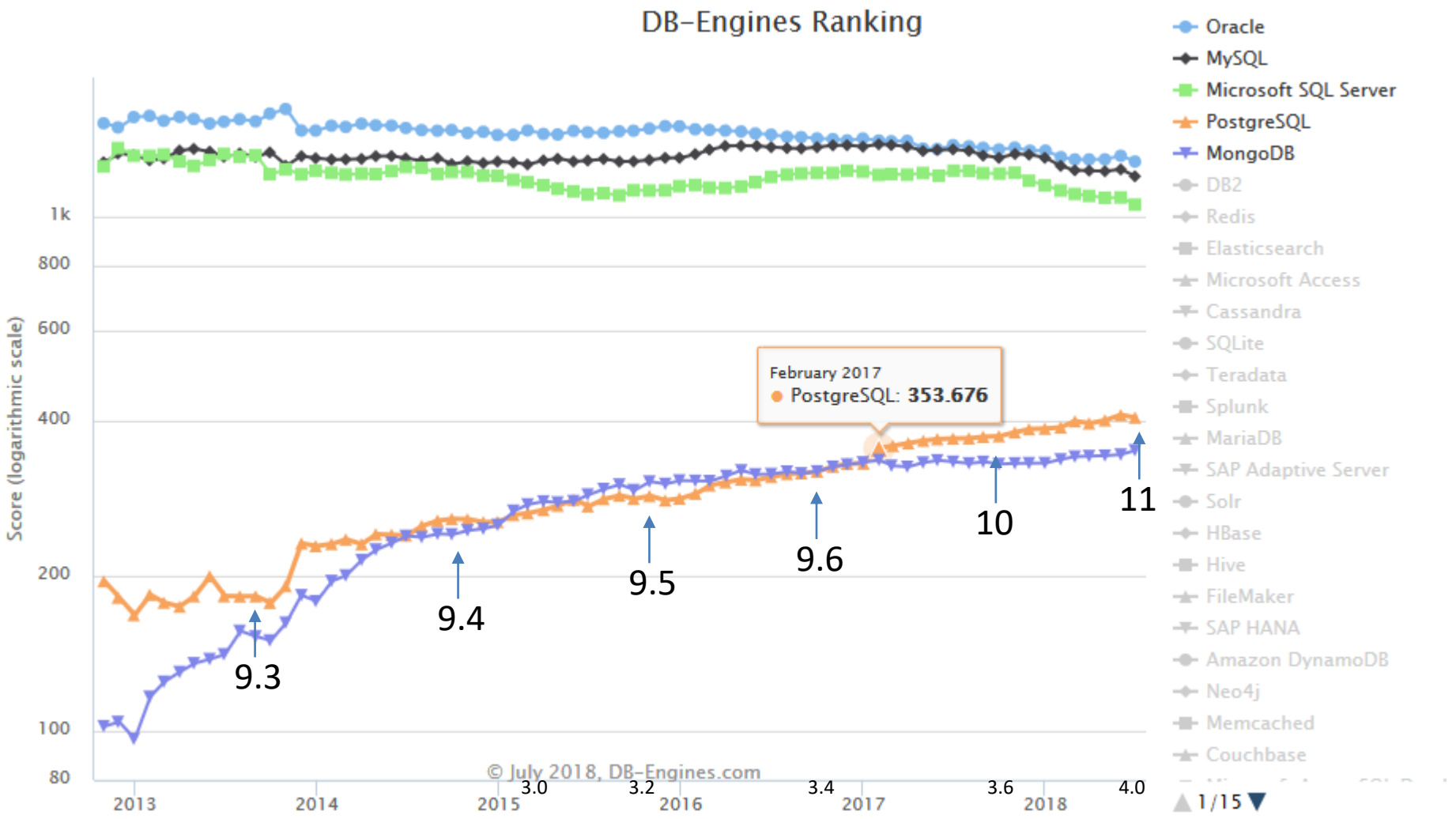
DB-Engines Ranking

343 systems in ranking, July 2018

Rank Jul 2018	Rank Jun 2018	Rank Jul 2017	DBMS	Database Model	Score Jul 2018	Score Jun 2018	Score Jul 2017
1.	1.	1.	Oracle	Relational DBMS	1277.79	-33.47	-97.09
2.	2.	2.	MySQL	Relational DBMS	1196.07	-37.62	-153.04
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1053.41	-34.32	-172.59
4.	4.	4.	PostgreSQL	Relational DBMS	405.81	-4.86	+36.37
5.	5.	5.	MongoDB	Document store	350.33	+6.54	+17.56
6.	6.	6.	DB2	Relational DBMS	186.20	+0.56	-5.05
7.	7.	↑ 9.	Redis	Key-value store	139.91	+3.61	+18.40
8.	8.	↑ 10.	Elasticsearch	Search engine	136.22	+5.18	+20.25
9.	9.	↓ 7.	Microsoft Access	Relational DBMS	132.58	+1.59	+6.45
10.	10.	↓ 8.	Cassandra	Wide column store	121.06	+1.84	-3.07
11.	11.	11.	SQLite	Relational DBMS	115.28	+1.02	+1.41
12.	12.	12.	Teradata	Relational DBMS	78.22	+2.45	-0.14
13.	↑ 14.	↑ 16.	Splunk	Search engine	69.24	+3.46	+8.94
14.	↓ 13.	↑ 18.	MariaDB	Relational DBMS	67.51	+1.67	+13.15

1. Number of mentions of the system (Google, Bing and Yandex...)
2. General interest in the system. (Google Trends)
3. Frequency of technical discussions about the system. (IT-related Q&A sites: Stack Overflow /DBA Stack Exchange)
4. Number of job offers, in which the system is mentioned. (Indeed and Simply Hired)
5. Number of profiles in professional networks, in which the system is mentioned. (LinkedIn and Upwork)
6. Relevance in social networks. (number of Twitter tweets)

DB-Engines Ranking



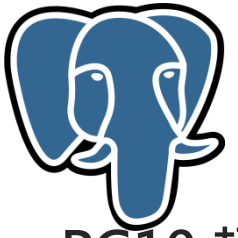
The DBMS of the Year 2017

DBMS of the Year: PostgreSQL (2018.1.3)

- While in our last year's popularity ranking PostgreSQL already ran in on place 3, 2017 was an even better year for PostgreSQL. With a total gain of **55.81 scoring points (+17%)** and improving its score in each of the single monthly rankings of 2017, it outperformed all other systems in 2017.

2017 : PostgreSQL 10 新特性

1. **資料表分割強化 (Table Partitioning)**
2. **邏輯複製(Logical Replication)**
3. **平行查詢強化 (Parallel Queries)**
4. **FDW強化(Additional FDW Push-Down)**
5. **多節點同步寫入 (Quorum Commit)**
6. **ID欄位功能(Identity columns)**
7. **安全認證提升(SCRAM-SHA-256 Authentication)**
8. 多欄位關聯(Multi-column Correlation Statistics)
9. 全文檢索支持 JSON 和 JSONB
10. 新增 pg_hba_file_rules 項目
11. 新增 pg_stat_activity 監控項目
12. 新增 pg_sequence 系統表
13. Row層級的安全政策(Row-level security)
14. Schema 預設權限(Default permissions on schemas)



PostgreSQL 10/11 新特性

PG10 超過100多項更新

1. **原生資料表分割 (Native Partition Table)**
2. **邏輯複製(Logical Replication)**
3. **平行查詢強化 (Parallel Queries)**
4. FDW強化(Additional FDW Push-Down)
5. 多節點同步寫入 (Quorum Commit)
6. ID欄位功能(Identity columns)
7. 安全認證提升(SCRAM-SHA-256 Authentication)
8. 多欄位關聯(Multi-column Correlation Statistics)
9. 全文檢索支持 JSON 和 JSONB
10. 新增 pg_hba_file_rules 項目
11. 新增 pg_stat_activity 監控項目
12. 新增 pg_sequence 系統表
13. Row層級的安全政策(Row-level security)
14. Schema 預設權限(Default permissions on schemas)

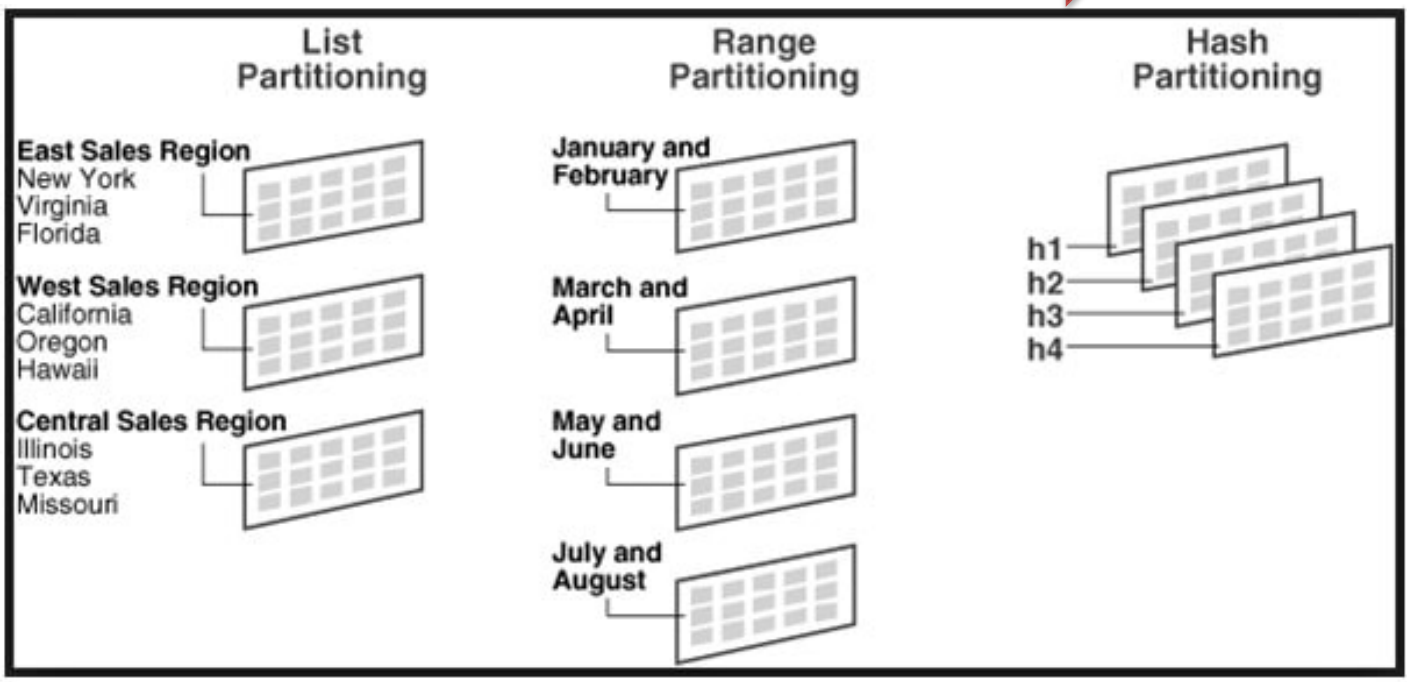
PG11 超過160多項更新

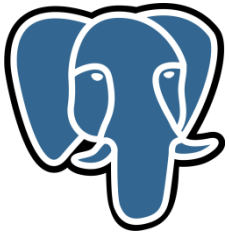
1. **資料表分割強化 (Table Partitioning)**
   Hash Partition / INSERT ON CONFLICT…
2. **平行查詢強化 (Parallel Queries)**
   Parallel Hash/Append…
3. **邏輯複製強化 (Logical Replication)**
   TRUNCATE…
4. 架構面 (Architecture)
   ROLE / LLVM / LDAP…
5. SQL語句 (SQL Statement)
   LOCK Table / STATISTICS of function index
6. PL/pgSQL (PL/pgSQL language)
   PROCEDURE object…
7. 參數設定 (Configuration parameters)
8. 常用指令 (Utilities / Commands)
9. 預設模組 (Contrib Modules)

資料表分割強化 (Table Partitioning)

1-1. Hash Partition

PG 10 PG 10 ADD PG 11



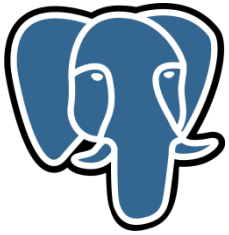


資料表分割強化 (Table Partitioning)

1-1. Hash Partition

```
postgres=> CREATE TABLE hash1 (c1 NUMERIC, c2 VARCHAR(10)) PARTITION BY
HASH(c1) ;
```

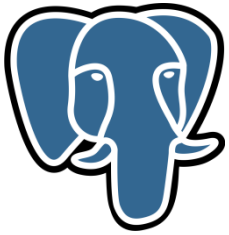
```
postgres=> CREATE TABLE hash1a PARTITION OF hash1 FOR VALUES
        WITH (MODULUS 4, REMAINDER 0) ;  -- 分 4 份，餘數為 0
CREATE TABLE
postgres=> CREATE TABLE hash1b PARTITION OF hash1 FOR VALUES
        WITH (MODULUS 4, REMAINDER 1) ;
CREATE TABLE
postgres=> CREATE TABLE hash1c PARTITION OF hash1 FOR VALUES
        WITH (MODULUS 4, REMAINDER 2) ;
CREATE TABLE
postgres=> CREATE TABLE hash1d PARTITION OF hash1 FOR VALUES
        WITH (MODULUS 4, REMAINDER 3) ;
CREATE TABLE
```



資料表分割強化 (Table Partitioning)

1-1. Hash Partition

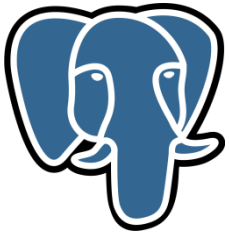
```
postgres=> \d hash1
                    Table "public.hash1"
Column |          Type          | Collation | Nullable | Default
-------+------------------------+-----------+----------+---------
c1     | numeric                |           |          |
c2     | character varying(10)  |           |          |
Partition key: HASH(c1)
Number of partitions: 4 (Use \d+ to list them.)

postgres=> \d hash1a
                    Table "public.hash1a"
Column |          Type          | Collation | Nullable | Default
-------+------------------------+-----------+----------+---------
c1     | numeric                |           |          |
c2     | character varying(10)  |           |          |
Partition of: hash1 FOR VALUES WITH (modulus 4, remainder 0)
```



資料表分割強化 (Table Partitioning)

1-1. Hash Partition

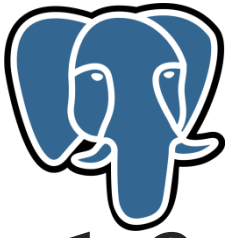
```
--查詢排除
postgres=> EXPLAIN SELECT * FROM hash1 WHERE c1 = 1000 ;
QUERY PLAN
------------------------------------------------------------------
Append  (cost=0.00..20.39 rows=4 width=70)
-> Seq Scan on hash1c (cost=0.00..20.38 rows=4 width=70)
    Filter: (c1 = '1000'::numeric)
(3 rows)
```



資料表分割強化 (Table Partitioning)

1-1. Hash Partion

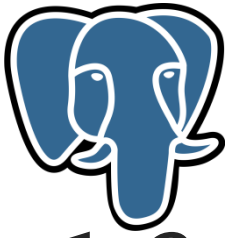
```
-- pg_class 中 pg_get_expr(ession)查詢設定
postgres=> SELECT pg_get_expr(relpartbound, oid) FROM pg_class WHERE
relname='hash1a' ;
          pg_get_expr
----------------------------------------
FOR VALUES WITH (modulus 4, remainder 0)
(1 row)

-- pg_partitioned_table 查詢 partitioning strategy
postgres=> SELECT partstrat FROM pg_partitioned_table WHERE
partrelid='hash1'::regclass ;
partstrat
-----------
h                    -- l = list partitioned table, r = range partitioned table
(1 row)
```



資料表分割強化 (Table Partitioning)

1-2. Default partition (Range & List)

```
-- Default partition 處理未被區別的 Table partition 值
postgres=> CREATE TABLE plist1 (c1 NUMERIC, c2 VARCHAR(10))
           PARTITION BY LIST (c1) ;
CREATE TABLE
postgres=> CREATE TABLE plist11 PARTITION OF plist1 FOR VALUES IN (100) ;
CREATE TABLE
postgres=> CREATE TABLE plist12 PARTITION OF plist1 FOR VALUES IN (200) ;
CREATE TABLE
postgres=> CREATE TABLE plist1d PARTITION OF plist1 DEFAULT ;
CREATE TABLE

-- Attach DEFAULT partition
postgres=> CREATE TABLE plist2d (c1 NUMERIC, c2 VARCHAR(10)) ;
CREATE TABLE
postgres=> ALTER TABLE plist2 ATTACH PARTITION plist2d DEFAULT ;
ALTER TABLE
```



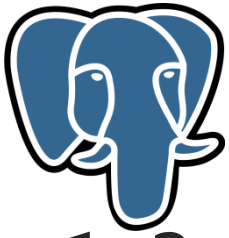
資料表分割強化 (Table Partitioning)

1-2. Default partition (Range & List) (續)

DEFAULT partition 的限制：

1. 無法設定多個 DEFAULT partitions
2. 新增 partitions 若所屬範圍之資料已存在於DEFAULT partition中，則無法新增。
3. 若要增加一個現存的資料表成為 DEFAULT partition，必須確認無重複值。
4. DEFAULT partition 不可使用在 HASH Partitioned Table

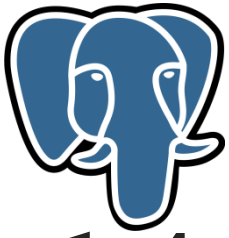
```
-- 上述第二項限制之範例
postgres=> CREATE TABLE plist1(c1 NUMERIC, c2 VARCHAR(10)) PARTITION BY LIST(c1) ;
CREATE TABLE
postgres=> CREATE TABLE plist11 PARTITION OF plist1 FOR VALUES IN (100) ;
CREATE TABLE
postgres=> CREATE TABLE plist1d PARTITION OF plist1 DEFAULT ;
CREATE TABLE
postgres=> INSERT INTO plist1 VALUES (100, 'v1'),(200, 'v2') ;
INSERT 0 2
postgres=> CREATE TABLE plist12 PARTITION OF plist1 FOR VALUES IN (200) ;
ERROR:  updated partition constraint for default partition "plist1d" would be
violated by some row
```



資料表分割強化 (Table Partitioning)

1-3. Update partition key

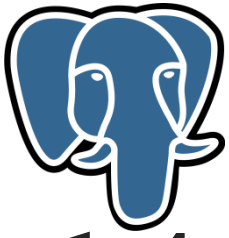
```
postgres=> CREATE TABLE part1(c1 INT, c2 VARCHAR(10)) PARTITION BY LIST(c1) ;
CREATE TABLE
postgres=> CREATE TABLE part1v1 PARTITION OF part1 FOR VALUES IN (100) ;
CREATE TABLE
postgres=> CREATE TABLE part1v2 PARTITION OF part1 FOR VALUES IN (200) ;
CREATE TABLE
postgres=> INSERT INTO part1 VALUES (100, 'data100');
INSERT 0 1
postgres=> INSERT INTO part1 VALUES (200, 'data200');
INSERT 0 1

-- 將c2為’data200’之c1值改為100，資料將從 part1v2 移到 part1v1
postgres=> UPDATE part1 SET c1=100 WHERE c2='data200' ;
UPDATE 1
```



資料表分割強化 (Table Partitioning)

1-4. 自動創建 index

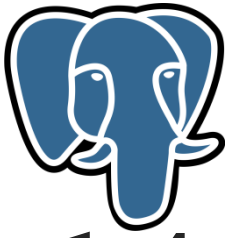
```
postgres=> CREATE TABLE part1(c1 NUMERIC, c2 VARCHAR(10)) PARTITION BY LIST(c1) ;
CREATE TABLE
postgres=> CREATE TABLE part1v1 PARTITION OF part1 FOR VALUES IN (100) ;
CREATE TABLE
postgres=> CREATE TABLE part1v2 PARTITION OF part1 FOR VALUES IN (200) ;
CREATE TABLE
postgres=> CREATE INDEX idx1_part1 ON part1(c2) ;
CREATE INDEX
postgres=> \d part1
            Table "public.part1"
Column | Type                   | Collation | Nullable | Default
----------+------------------------+-----------+----------+---------
c1       | numeric                |           |          |
c2       | character varying(10)  |           |          |
Partition key: LIST (c1)
Indexes:
    "idx1_part1" btree (c2)   -- 手動建在母表上
Number of partitions: 2 (Use \d+ to list them.)
```



資料表分割強化 (Table Partitioning)

1-4. 自動創建 index (續)

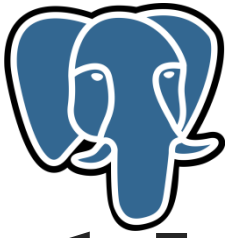
```
postgres=> \d part1v1
            Table  "public. part1v1 "
Column | Type                        | Collation | Nullable | Default
-----------+-----------------------------+-----------+----------+---------
C1         | numeric                     |           |          |
C2         | character  varying(10)  |           |          |
Partition of: part1 FOR VALUES IN ('100')
Indexes:
        "part1v1_c2_idx" btree  (c2)     -- index自動生成在v1子表中

postgres=> \d part1v2
...(略)
Partition of: part1 FOR VALUES IN ('200')
Indexes:
        "part1v2_c2_idx" btree  (c2)   -- index自動生成在v2子表中
```



資料表分割強化 (Table Partitioning)

1-4. 自動創建 index (續2)

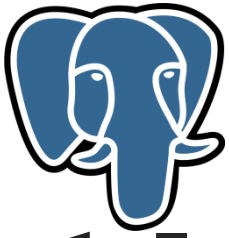
```
-- 個別刪除自動創建的index是不允許的
postgres=> DROP INDEX part1v1_c2_idx ;
ERROR:  cannot drop index part1v1_c2_idx because index idx1_part1 requires it
HINT:  You can drop index idx1_part1 instead.
```



資料表分割強化 (Table Partitioning)

1-5. 建立 unique constraint

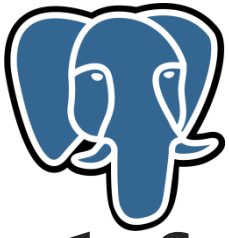
```
-- 建立 PRIMARY KEY 與 UNIQUE KEY，在子表即會自動建
postgres=> CREATE TABLE part1(c1 NUMERIC, c2 VARCHAR(10)) PARTITION BY RANGE(c1) ;
CREATE TABLE
postgres=> ALTER TABLE part1 ADD CONSTRAINT pk_part1 PRIMARY KEY (c1) ;
ALTER TABLE
postgres=> \d part1
              Table  "public. part1 "
Column | Type                        | Collation | Nullable | Default
----------+-----------------------------+----------+----------+---------
C1       | numeric                     |           | not null|
C2       | character  varying(10)  |           |           |
Partition key: RANGE (c1)
Indexes:
"pk_part1"  PRIMARY  KEY,  btree  (c1)  --剛才手動建
Number of partitions: 0
```



資料表分割強化 (Table Partitioning)

1-5. 建立 unique constraint (續)

```
-- 建立 PRIMARY KEY 與 UNIQUE KEY，在子表即會自動建
postgres=> CREATE TABLE part1v1 (LIKE part1) ;
CREATE TABLE
postgres=> ALTER TABLE part1 ATTACH PARTITION part1v1 FOR VALUES FROM
(100) TO (200) ;
ALTER TABLE
postgres=> \d part1v1
              Table "public. part1v1"
Column | Type                        | Collation | Nullable | Default
----------+-----------------------------+----------+----------+---------
C1       | numeric                     |           | not null|
C2       | character varying(10)  |           |          |
Partition of: part1 FOR VALUES FROM ('100') TO ('200')
Indexes:
"part1v1_pkey" PRIMARY KEY, btree (c1)   -- 自動建置
```

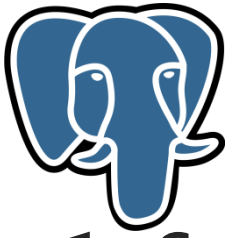


資料表分割強化 (Table Partitioning)

1-6. Partition-Wise Join & Aggregate

Features	Parameter name	Default value
Partition-Wise Join	enable_partitionwise_join	off
Partition-Wise Aggregate	enable_partitionwise_aggregate	off

```
-- 未開啟 partition-wise join
postgres=> EXPLAIN SELECT COUNT(*) FROM pjoin1 p1 INNER JOIN pjoin2 p2 ON p1.c1 = p2.c1 ;
                          QUERY PLAN
------------------------------------------------------------------------------------
Finalize Aggregate  (cost=79745.46..79745.47 rows=1 width=8)
->  Gather  (cost=79745.25..79745.46 rows=2 width=8)
      Workers Planned: 2
    ->  Partial Aggregate  (cost=78745.25..78745.26 rows=1 width=8)
        ->  Parallel Hash Join  (cost=36984.68..76661.91 rows=833333 width=0)
            Hash Cond: (p1.c1 = p2.c1)
            ->  Parallel Append  (cost=0.00..23312.00 rows=833334 width=6)
                  -> Parallel Seq Scan on pjoin1v1 p1 (cost=0.00..9572.67 ...)
                  -> Parallel Seq Scan on pjoin1v2 p1_1 (cost=0.00..9572.67 ...)
        ->  Parallel Hash  (cost=23312.00..23312.00 rows=833334 width=6)
            ->  Parallel Append (cost=0.00..23312.00 rows=833334 width=6)
                  -> Parallel Seq Scan on pjoin2v1 p2  (cost=0.00..9572.67  ...)
                  -> Parallel Seq Scan on pjoin2v2 p2_1  (cost=0.00..9572.67  ...)
(13 rows)
```

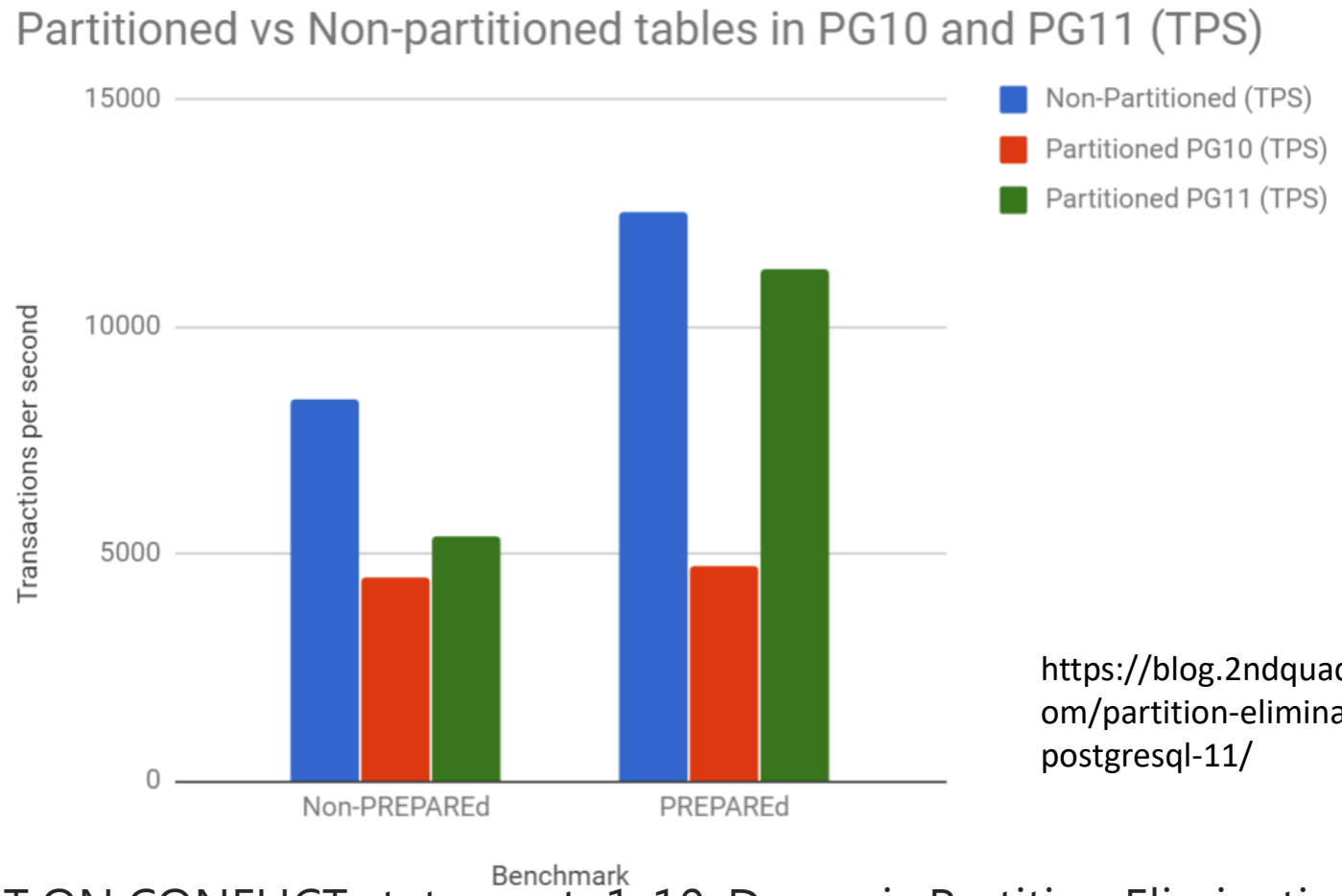


資料表分割強化 (Table Partitioning)

1-6. Partition-Wise Join & Aggregate

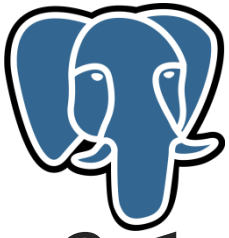
```
postgres=> SET enable_partitionwise_join = on ;
SET
postgres=> EXPLAIN SELECT COUNT(*) FROM pjoin1 p1 INNER JOIN pjoin2 p2 ON p1.c1 = p2.c1 ;
                              QUERY PLAN
-----------------------------------------------------------------------------------
Finalize Aggregate  (cost=75578.78..75578.79 rows=1 width=8)
->  Gather  (cost=75578.57..75578.78 rows=2 width=8) Workers Planned: 2
   ->  Partial Aggregate  (cost=74578.57..74578.58 rows=1 width=8)
   ->  Parallel Append  (cost=16409.00..72495.23 rows=833334 width=0)
         -> Parallel Hash Join  (cost=16409.00..34164.28 rows=416667 width=0)
               Hash Cond: (p1.c1 = p2.c1)
            -> Parallel Seq Scan on pjoin1v1 p1  (cost=0.00..9572.67 rows=416667 width=6)
            -> Parallel Hash  (cost=9572.67..9572.67 rows=416667 …)
                  -> Parallel Seq Scan on pjoin2v1 p2 (cost=0.00..9572.67…)
         -> Parallel Hash Join  (cost=16409.00..34164.28 rows=416667 width=0)
               Hash Cond: (p1_1.c1 = p2_1.c1)
            -> Parallel Seq Scan on pjoin1v2 p1_1  (cost=0.00..9572.67 rows=416667 width=6)
            -> Parallel Hash (cost=9572.67..9572.67 rows=416667 …)
                  -> Parallel Seq Scan on pjoin2v2 p2_1 (cost=0.00..9572.67…)
```

資料表分割強化 (Table Partitioning)



https://blog.2ndquadrant.com/partition-elimination-postgresql-11/

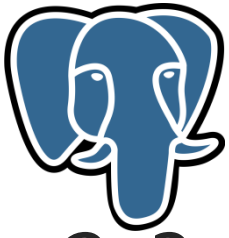
1-7. INSERT ON CONFLICT statement
1-8. FOR EACH ROW trigger
1-9. FOREIGN KEY support
1-10. Dynamic Partition Elimination
1-11. Control Partition Pruning



平行查詢強化 (Parallel Queries)

2-1 . Parallel Hash / 2-2. Parallel Append

```
postgres=> EXPLAIN SELECT COUNT(*) FROM hash1 INNER JOIN hash2 ON hash1.c1 = hash2.c1 ;
                                  QUERY PLAN
------------------------------------------------------------------------------
Finalize Aggregate        (cost=368663.94..368663.95 rows=1 width=8)
-> Gather  (cost=368663.73..368663.94  rows=2  width=8) Workers Planned: 2
     -> Partial  Aggregate (cost=367663.73..367663.74        rows=1 width=8)
          -> Parallel  Hash  Join       (cost=164082.00..357247.06 rows=4166667 width=0)
Hash  Cond:  (hash2.c1  =  hash1.c1)
               -> Parallel  Seq  Scan  on  hash2 (cost=0.00..95722.40 rows=4166740 width=6)
               -> Parallel  Hash       (cost=95721.67..95721.67 rows=4166667 width=6)
                    ->    Parallel  Seq  Scan  on  hash1 (cost=0.00..95721.67 rows=4166667 width=6)
(9 rows)
postgres=> EXPLAIN SELECT COUNT(*) FROM  data1 UNION ALL  SELECT COUNT(*) FROM data2 ;
                                  QUERY  PLAN
------------------------------------------------------------------------------
Gather   (cost=180053.25..180054.25  rows=2  width=8) Workers  Planned:  2
->  Parallel  Append   (cost=179053.25..179054.05  rows=1  width=8)
    ->   Aggregate   (cost=179054.02..179054.04  rows=1  width=8)
         ->   Seq  Scan  on  data1   (cost=0.00..154054.22  rows=9999922  width=0)
    ->   Aggregate   (cost=179053.25..179053.26  rows=1  width=8)
         ->   Seq  Scan  on  data2   (cost=0.00..154053.60  rows=9999860  width=0)
(7  rows)
```



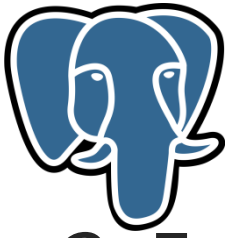
平行查詢強化 (Parallel Queries)

2-3 . CREATE TABLE AS SELECT statement

```
postgres=> EXPLAIN CREATE TABLE para1 AS SELECT COUNT(*) FROM data1 ;
QUERY PLAN
------------------------------------------------------------------------
Finalize Aggregate       (cost=11614.55..11614.56 rows=1 width=8)
->          Gather       (cost=11614.33..11614.54  rows=2  width=8)
        Workers Planned: 2
->          Partial  Aggregate        (cost=10614.33..10614.34  rows=1  width=8)
->          Parallel Seq Scan on data1 (cost=0.00..9572.67 rows=416667 width=0)
(5 rows)
```

2-4 . CREATE MATERIALIZED VIEW statement

```
postgres=> EXPLAIN CREATE MATERIALIZED VIEW mv1 AS SELECT COUNT(*) FROM data1 ;
QUERY PLAN
------------------------------------------------------------------------------
Finalize Aggregate   (cost=11614.55..11614.56 rows=1 width=8)
->  Gather   (cost=11614.33..11614.54 rows=2 width=8)
     Workers Planned: 2
->  Partial Aggregate   (cost=10614.33..10614.34 rows=1 width=8)
->          Parallel Seq Scan on data1      (cost=0.00..9572.67 rows=416667 width=0)
(5 rows)
```

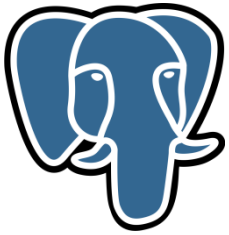


平行查詢強化 (Parallel Queries)

2-5 . SELECT INTO statement

```
postgres=> EXPLAIN SELECT COUNT(*) INTO val FROM data1 ;
QUERY PLAN
------------------------------------------------------------------------
Finalize Aggregate     (cost=11614.55..11614.56 rows=1 width=8)
->  Gather   (cost=11614.33..11614.54 rows=2 width=8)
    Workers Planned: 2
->  Partial Aggregate   (cost=10614.33..10614.34 rows=1 width=8)
->            Parallel Seq Scan on data1 (cost=0.00..9572.67 rows=416667 width=0)
(5 rows)
```

2-6 . CREATE INDEX statement



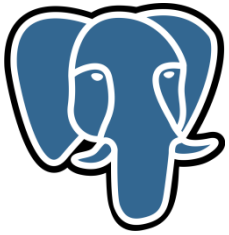
邏輯複製強化 (Logical Replication)

3-1. 支援 TRUNCATE 語法

```
postgres=> CREATE PUBLICATION pub1 FOR TABLE data1
           WITH (publish='INSERT, DELETE, UPDATE, TRUNCATE') ;
CREATE PUBLICATION
```

3-2. pg_replication_slot_advance

```
postgres=# SELECT pg_replication_slot_advance('sub1', pg_current_wal_lsn()) ;
pg_replication_slot_advance
----------------------------
(sub1,0/5B63E18)
(1 row)
```

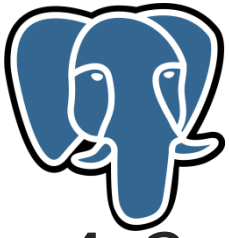


架構面 (Architecture)

4-1. catalogs 調整

Catalog name	Added column	Data Type	Description
pg_aggregate	aggfinalmodify	char	Whether the aggfinalfn function changes the value
	aggmfinalmodify	char	Whether the aggmfinalfn function changes the value
pg_attribute	atthasmissing	bool	Have a default value that has not updated the page
	attmissingval	anyarray	Default value not updating page
pg_class	relrewrite	oid	OID when the new relation is created during DDL execution
pg_constraint	conparentid	oid	Parent partition constraint OID
	conincluding	smallint[]	Non-constrained column number list
pg_index	indnkeyatts	smallint	Number of key columns
pg_partitioned_table	partdefid	oid	OID of the default partition
pg_proc	prokind	char	Object kind f: function p: procedure a: aggregate function w: window function
pg_publication	pubtruncate	boolean	TRUNCATE propagation
pg_stat_wal_receiver	sender_host	text	Connection destination hostname
	sender_port	integer	Connection destination port number
information_schema.table_constraints	enforced	yes_or_no	Reserved for future use

Catalog name	Deleted column	Description
pg_class	relhaspkey	Have primary key
pg_proc	proisagg	Is aggregate function
	proiswindow	Is Window function



架構面 (Architecture)

4-2. 新增 ROLE

Role name	Description
pg_execute_server_program	Execute programs on the server
pg_read_server_files	Read files on the server
pg_write_server_files	Write files on the server

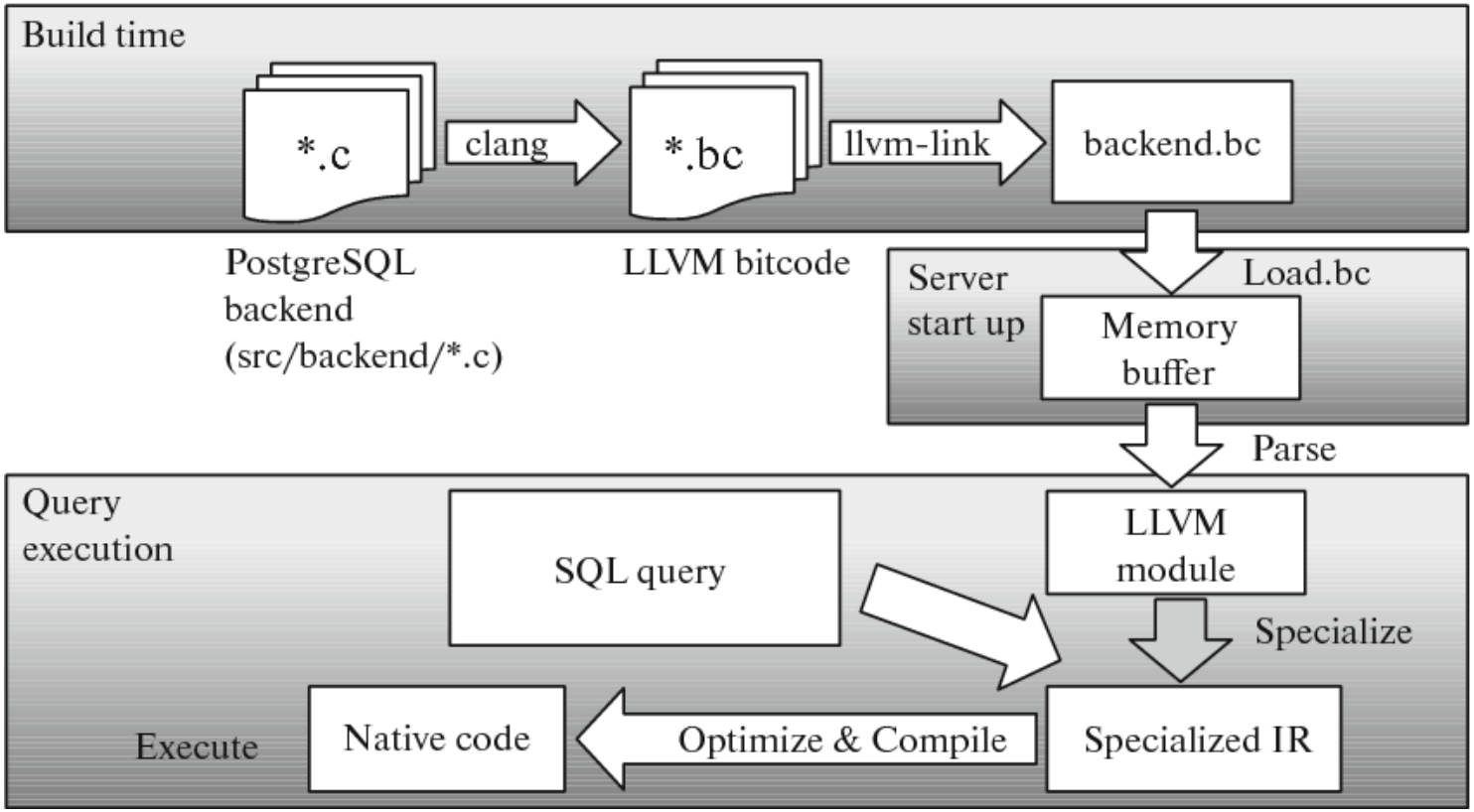
※ 使用於 COPY 指令與file_fdw

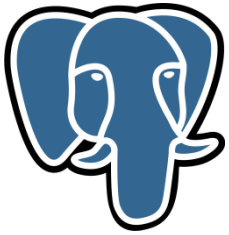
```
postgres=# GRANT pg_read_server_files TO user1 ;
GRANT ROLE

postgres(user1)=> COPY copy1 FROM '/tmp/copy1.csv' CSV ;
COPY 2000
```

架構面 (Architecture)

4-3. 整合LLVM

- PostgreSQL 11支持使用LLVM進行JIT編譯，以加速由處理器瓶頸引起的長時間運行的SQL語句。估計超過一定數量成本的SQL語句會事先編譯然後執行。





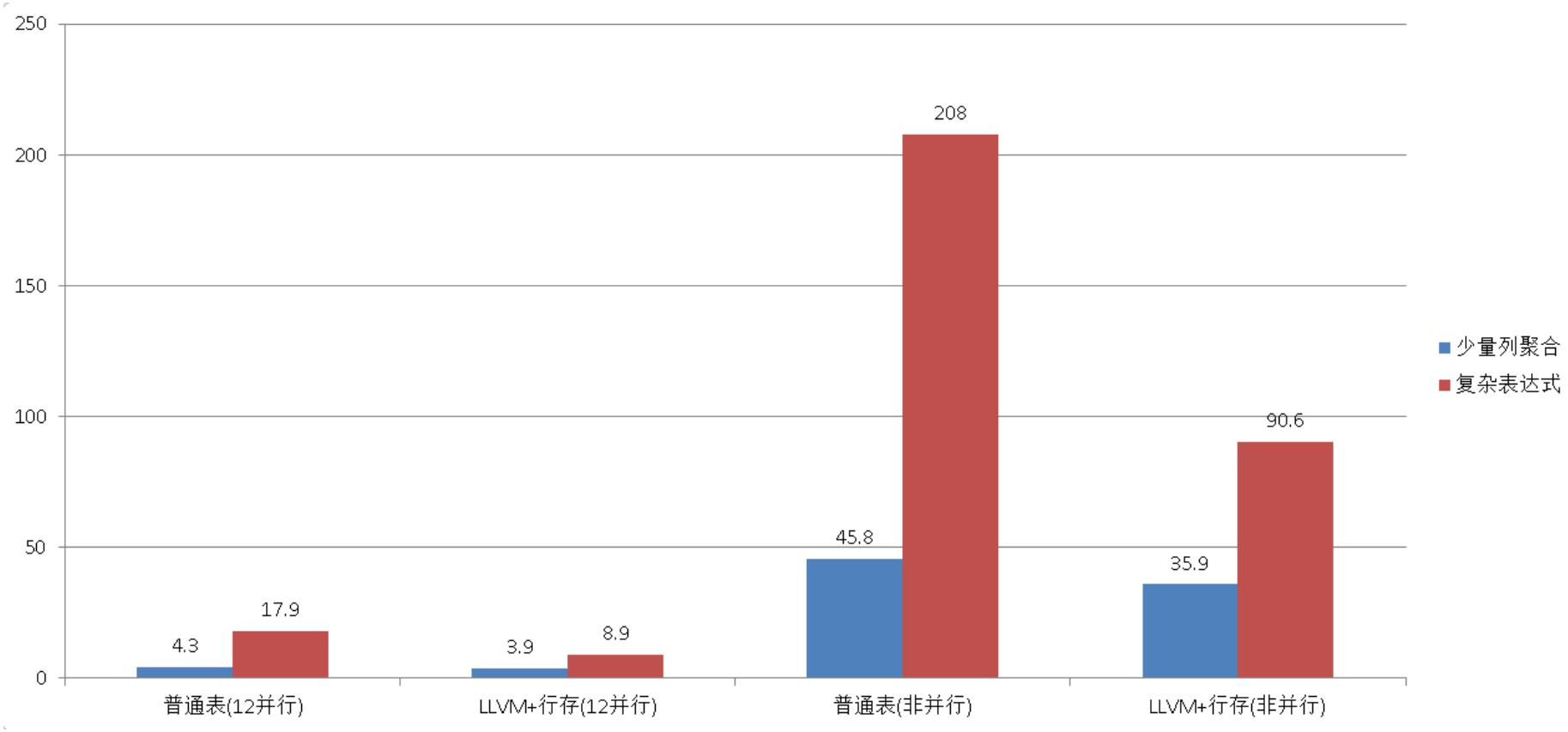
架構面 (Architecture)

4-3. 整合LLVM

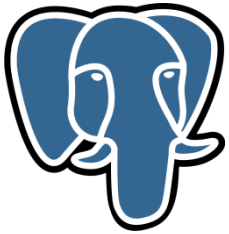
```
postgres=> EXPLAIN ANALYZE SELECT COUNT(*) FROM jit1 ;
                                  QUERY PLAN
-------------------------------------------------------------------------------
Aggregate (cost=179053.25..179053.26 rows=1 width=8)
          (actual time=2680.558..2680.559 rows=1 loops=1)
       -> Seq Scan on jit1        (cost=0.00..154053.60 rows=9999860 width=0)
                        (actual time=0.022..1424.095 rows=10000000 loops=1)
Planning Time: 0.024 ms
JIT:
   Functions:        2
   Generation Time: 1.505 ms
   Inlining:         false
   Inlining Time: 0.000 ms
   Optimization: false
   Optimization Time: 0.594 ms
   Emission Time: 8.688 ms
   Execution Time: 2682.166 ms
(12 rows)
```

架構面 (Architecture)

4-3. 整合LLVM



https://github.com/digoal/blog/blob/master/201612/20161216_01.md?spm=a2c4e.11153940.blogcont69418.59.2405611bDQqapl&file=20161216_01.md

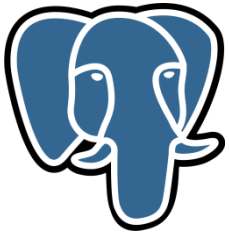


架構面 (Architecture)

4-4. Predicate locking for GIN / GiST / HASH index
4-5. Enhanced LDAP authentication
4-6. Extended Query timeout
4-7. Change of backup label file
4-8. Using Huge Pages under Windows environment
4-9. Remove secondary checkpoint information
4-10. Error code list

Others

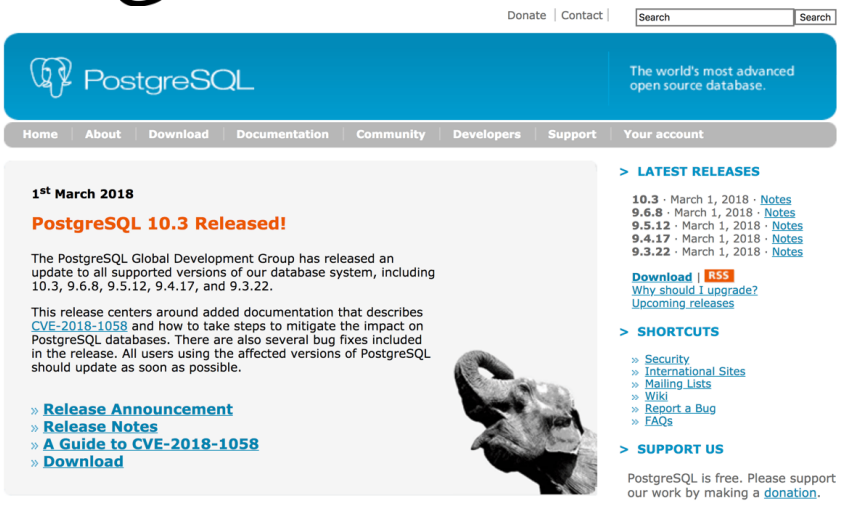
5. SQL語句 (SQL Statement)
   LOCK Table / STATISTICS of function index
6. PL/pgSQL (PL/pgSQL language)
   PROCEDURE object…
7. 參數設定 (Configuration parameters)
8. 常用指令 (Utilities / Commands)
9. 預設模組 (Contrib Modules)



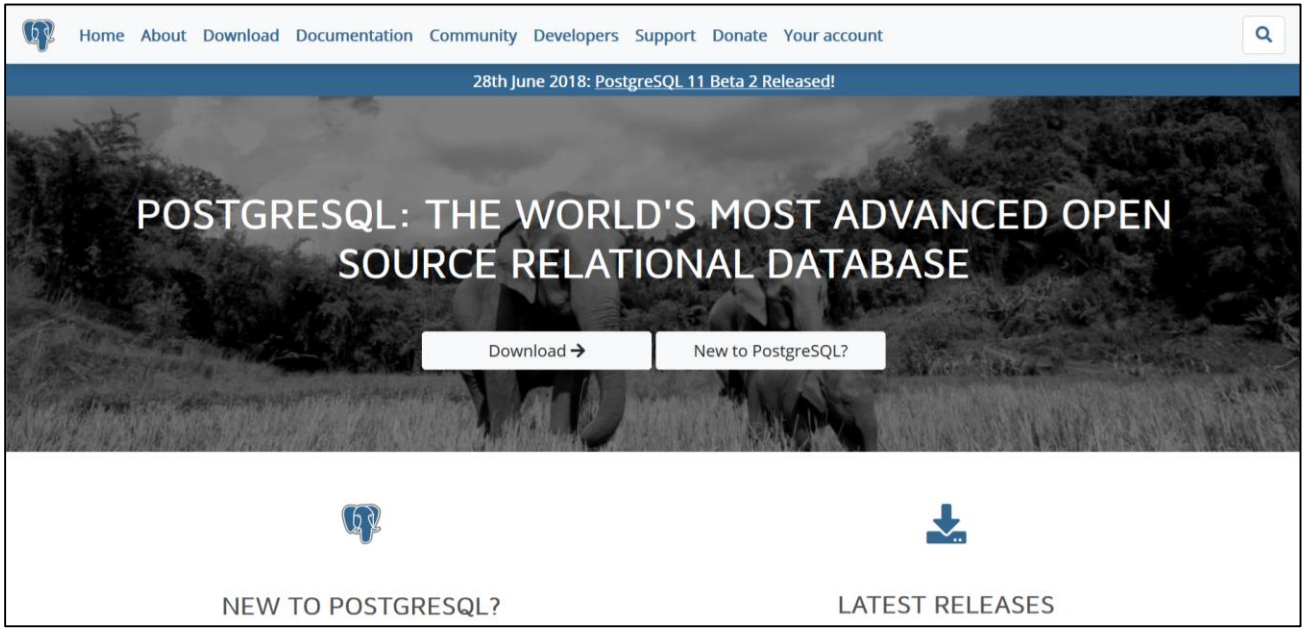
PostgreSQL 11 新特性

- 參考資料/來源：
  - PostgreSQL 11 New Features With Examples (Beta 1) Noriyoshi Shinoda, HP-JP：
    - https://h50146.www5.hpe.com/products/software/oe/linux/mainstream/support/lcc/pdf/PostgreSQL_11_New_Features_beta1_en_20180525-1.pdf

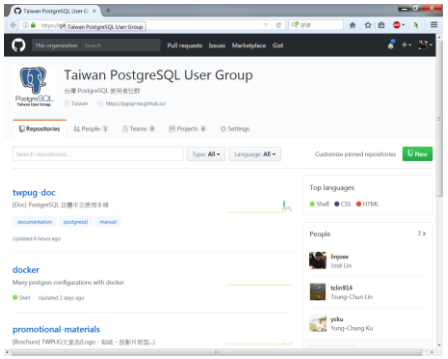
PostgreSQL.org 網站更新



Community
https://www.postgresql.org/community/
Commit Fest
https://commitfest.postgresql.org/
Bug Summit
https://www.postgresql.org/account/submitbug/
Git
https://git.postgresql.org/gitweb/



台灣PostgreSQL使用者社群



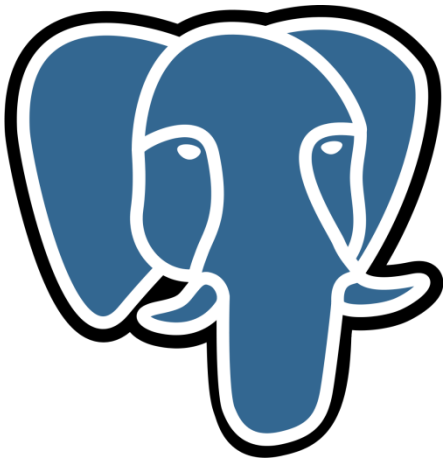
- PostgreSQL.tw
- Github: pgsql-tw
- Postgresql正體中文使用手冊

FB社群: PostgreSQL.TW
(提問交流)

- FB粉絲專頁: @pgsqlTaiwan
  (初創者: 小郭-郭朝益先生)

PostgreSQL
Taiwan User Group

一起來為PG增添台灣味!



Thank you.

歡迎加入台灣PostgreSQL使用者社群

Github : pgsql-tw

Website : pgsql-tw.github.io

Facebook : @pgsqlTaiwan