

PostgreSQL 9.5 最新事情

Noriyoshi Shinoda

February 20, 2016



自己紹介

篠田典良（しのだのりよし）



– 所属

- 日本ヒューレット・パカード株式会社 テクノロジーコンサルティング事業統括

– 現在の業務

- PostgreSQLをはじめOracle Database, Microsoft SQL Server, Vertica, Sybase ASE等 RDBMS全般に関するシステムの設計、チューニング、コンサルティング
- オープンソース製品に関する調査、検証
- Oracle Database関連書籍の執筆
- 弊社講習「Oracle DatabaseエンジニアのためのPostgreSQL入門」講師

– 関連する URL

- 「PostgreSQL 虎の巻」シリーズ
 - <http://h30507.www3.hp.com/t5/user/viewprofilepage/user-id/838802>
- Oracle ACEってどんな人？
 - <http://www.oracle.com/technetwork/jp/database/articles/vivadeveloper/index-1838335-ja.html>



Agenda

PostgreSQL 9.5最新事情

1. PostgreSQL 9.5概要
2. アプリケーション開発を容易にする機能
3. 大規模環境に対応する機能
4. 運用を容易にする機能
5. 罨
6. PostgreSQL 10?
7. まとめ





1. PostgreSQL 9.5概要



1. PostgreSQL 9.5概要

PostgreSQLの歴史

- 1974年 Ingres プロトタイプ
 - HP NonStop SQL, SAP Sybase ASE, Microsoft SQL Serverの元になる
- 1989年 POSTGRES 1.0～
- 1997年 PostgreSQL 6.0～
 - GEQO, MVCC, マルチバイト
- 2000年 PostgreSQL 7.0～
 - WAL, TOAST
- 2005年 PostgreSQL 8.0～
 - 自動VACUUM, HOT, PITR
- 2010年 PostgreSQL 9.0～
 - レプリケーション, 外部表, JSON, マテリアライズド・ビュー
- 2016年1月 PostgreSQL 9.5
- 2016年2月 Security Update Release 9.5.1 ← [いまここ](#)



1. PostgreSQL 9.5概要

本日説明する新機能

- アプリケーション開発関連
 - INSERT ON CONFLICT文
 - Row Level Security機能
 - UPDATE SET文
 - SELECT SKIP LOCK文
 - JSONB型に関する新機能
- 大規模環境に対応する機能
 - BRIN Index
 - SELECT TABLESAMPLE文
 - CREATE FOREIGN TABLE INHERITS文
- 運用を容易にする機能
 - pg_rewindコマンド
 - ALTER TABLE SET UNLOGGED文



1. PostgreSQL 9.5概要

本日説明しない主な新機能

– パフォーマンスの向上

- ロック制御の改善によるマルチ・プロセッサ環境におけるスループット向上
- ソート処理の高速化

– ユーティリティの改善

- vacuumdbコマンドの--jobsパラメータ
- pgbenchコマンドの--latency-limitパラメータ

– SQL文の新機能

- GROUPING SETS / ROLLUP / CUBE句
- IMPORT FOREIGN SCHEMA文
- CREATE EVENT TRIGGERの拡張
- REINDEX SCHEMA文
- PL/pgSQL ASSERT文
- etc





2. アプリケーション開発を容易にする機能

INSERT ON CONFLICT

INSERT文で制約違反が発生したらUPDATE文を実行

```
INSERT INTO employees VALUES (1000, 'Shinoda', 'shinoda@hpe.com')  
ON CONFLICT (empid)  
DO UPDATE SET ename = EXCLUDED.ename, email = EXCLUDED.email
```

- 主キー列empid = 1000のタプルが存在しなければ
 - INSERT INTO employees VALUES (1000, 'Shinoda', 'shinoda@hpe.com') が実行される
- 主キー列empid = 1000のタプルが存在すれば
 - UPDATE employees SET ename='Shinoda', email='shinoda@hpe.com' WHERE empid=1000 が実行される
- EXCLUDED句は、INSERT INTOで指定した値を指す
- 使用できる制約
 - PRIMARY KEY, UNIQUE, EXCLUSION（排他制約）



INSERT ON CONFLICT

INSERT文で制約違反が発生したらUPDATE文を実行

- その他の構文
 - 「DO NOTHING」を記述すると、制約違反が発生しても何もしない
 - 「DO NOTHING」指定時は制約列名を省略できる
 - 「ON CONFLICT ON CONSTRAINT 制約名」を記述すると、制約名を指定できる
- VALUES句だけではなく、INSERT SELECT文でも実行可能

```
INSERT INTO instab1 SELECT * FROM instab2  
ON CONFLICT (pk_col1)  
DO UPDATE SET valcol1 = EXCLUDED.valcol1
```

INSERT ON CONFLICT

トリガーの動作

- INSERT ON CONFLICT文はINSERT文なのか？ UPDATE文なのか？
- トリガーが特殊な動作になる
- EACH ROWトリガーの動作（数字は実行順）

| Trigger | INSERT 成功 | DO NOTHING | DO UPDATE (更新あり) | DO UPDATE (更新なし) |
|---------------|--------------|---------------|---------------------|---------------------|
| BEFORE INSERT | ①実行 | ①実行 | ①実行 | ①実行 |
| AFTER INSERT | ②実行 | - | - | - |
| BEFORE UPDATE | - | - | ②実行 | - |
| AFTER UPDATE | - | - | ③実行 | - |

- DO UPDATE (更新なし)
 - DO UPDATE句にWHERE句を指定し、更新されなかった場合。

INSERT ON CONFLICT

トリガーの動作

– EACH STATEMENT トリガーの動作（数字は実行順）

| Trigger | INSERT 成功 | DO NOTHING | DO UPDATE (更新あり) | DO UPDATE (更新なし) |
|---------------|--------------|---------------|---------------------|---------------------|
| BEFORE INSERT | ①実行 | ①実行 | ①実行 | ①実行 |
| AFTER INSERT | ④実行 | ②実行 | ④実行 | ④実行 |
| BEFORE UPDATE | ②実行 | - | ②実行 | ②実行 |
| AFTER UPDATE | ③実行 | - | ③実行 | ③実行 |

INSERT ON CONFLICT

MySQLとの違い

- MySQLのINSERT ON DUPLICATE KEY UPDATE文に相当
 - PRIMARY KEY制約、UNIQUE制約に対応
 - INSERT SELECT文でも使用可能
- MySQLとの違い
 - そもそも構文が違う

```
INSERT INTO employees VALUES (1000, 'Shinoda', 'shinoda@hpe.com')  
ON DUPLICATE KEY UPDATE  
ename = VALUES(ename), email = VALUES(email)
```

- PostgreSQLにはREPLACE文/INSERT IGNORE文に相当する機能は無い
- 原則として制約名または制約列の指定が必要（DO NOTHING指定時以外）
- 制約違反が発生したら「何もしない=DO NOTHING」ことができる



Row Level Security

レコード単位のデータ参照設定

– GRANT文によるアクセス制御

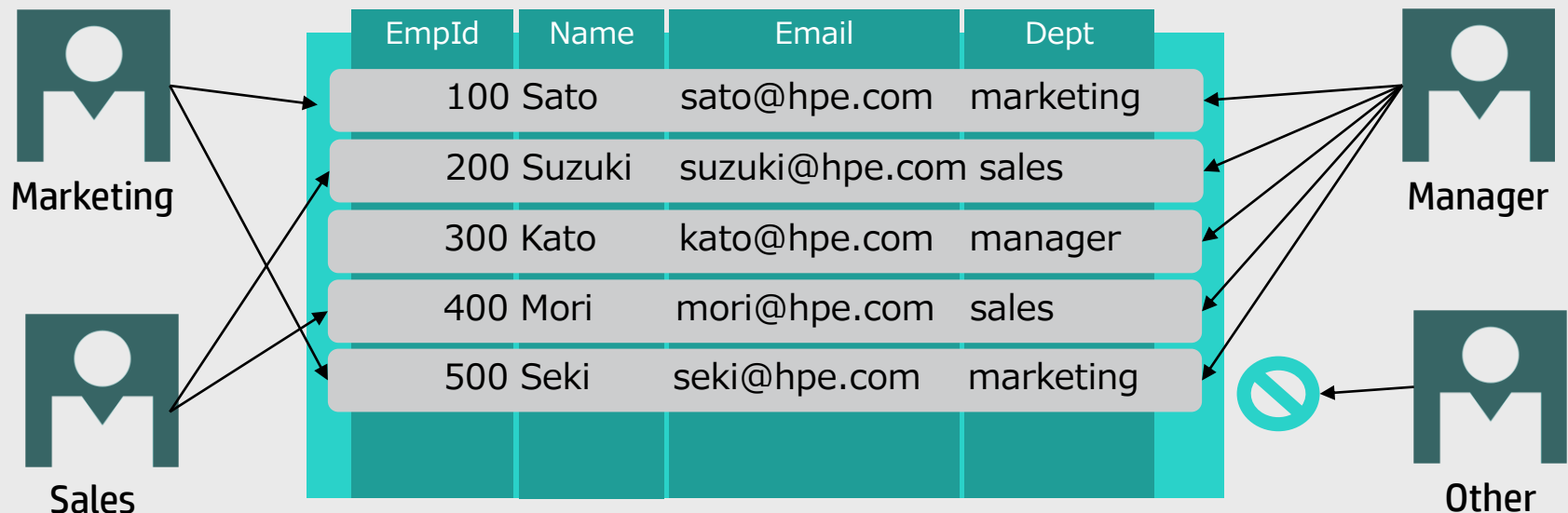
– テーブル単位

– 列単位

– Row Level Security (RLS)

– レコード単位のアクセス制御

– GRANTによる許可を置き換えるものではなく、GRANTされた範囲を絞るもの



Row Level Security

利用方法

– テーブルに対してRLSの有効化

```
ALTER TABLE table_name ENABLE ROW LEVEL SECURITY
```

– ポリシーの作成

- 対象となるテーブル (ON)
- 対象となる操作 (FOR)
- 対象となるロール (TO)
- 許可する検索条件 (USING) → WHERE句条件
- 許可する更新条件 (WITH CHECK) → WHERE句条件

```
CREATE POLICY policy_name ON table_name  
[ FOR { ALL | SELECT | INSERT | UPDATE | DELETE } ]  
[ TO role_name | PUBLIC [, ...] ]  
[ USING (expression) ]  
[ WITH CHECK (expression) ]
```



Row Level Security

実行例

– ポリシーの作成とRLS有効化

```
postgres(owner)=> ALTER TABLE employees ENABLE ROW LEVEL SECURITY ;  
ALTER TABLE  
postgres(owner)=> GRANT SELECT ON employees TO PUBLIC ;  
postgres(owner)=> CREATE POLICY pol1 ON employees  
                   FOR SELECT TO PUBLIC  
                   USING (dept = current_user) ;  
CREATE POLICY
```

```
postgres(marketing)=> SELECT name, dept FROM employees ;  
name   | dept  
-----+-----  
Sato    | marketing  
Hayashi | marketing  
(2 rows)
```



Row Level Security

実行例

– 実行計画

```
postgres=> \d+ rls1
```

Table "public.rls1"

| Column | Type | Modifiers | Storage | Stats target | Description |
|--------|-----------------------|-----------|----------|--------------|-------------|
| c1 | character varying(10) | | extended | | |
| c2 | character varying(10) | | extended | | |

Policies:

```
POLICY "rls1_p1" FOR ALL
```

```
USING (((c1)::name = "current_user"()))
```

```
POLICY "rls1_p2" FOR ALL
```

```
USING (((c2)::text = 'data'::text))
```

```
postgres=> EXPLAIN SELECT * FROM rls1 ;  
QUERY PLAN
```

```
-----  
Seq Scan on rls1 (cost=0.00..22.00 rows=6 width=108)
```

```
Filter: (((c2)::text = 'data'::text) OR ((c1)::name = "current_user"()))
```

```
(2 rows)
```

Row Level Security

注意点

- ポリシー名
 - ポリシー名の名前空間はテーブル単位
- ポリシー作成テーブル
 - ENABLE ROW LEVEL SECURITYの設定が無くてもPOLICYの作成は可能
 - でもポリシーは動作せず、GRANTによる許可のみ
- 複数ポリシー
 - 異なる条件（USING）を持つ複数のポリシーが適用されている場合にはOR条件になる
- 制約との整合性
 - ポリシーが適用されているテーブルではレコードが隠蔽されるため、SELECT文で検索して存在しないことを確認しても制約違反が発生する可能性がある。
- パラメーターrow_security = on (default) | off | force
 - forceに設定すると、ポリシーをテーブル所有者やSuperuserにも適用する
- BYPASSRLSロール
 - ポリシー設定をバイパスできる



SELECT SKIP LOCKED

ロックされていないタプルのみ検索

- ロックが競合する場合の動作
 - 待機するかエラーにする
 - SELECT FOR UPDATE同士
 - SELECT FOR UPDATEとSELECT FOR SHARE
 - NOWAITを指定するとエラー
 - ロックしていないタプルのみ検索する
 - PostgreSQL 9.5新機能

```
SELECT ... FROM table_name FOR UPDATE SKIP LOCKED
```



UPDATE SET

結合結果による複数列の同時更新構文

- 複数列の一括更新の記述が可能に
- PostgreSQL 9.4 まで

```
UPDATE upd2
SET c2 = upd1.c2, c3 = upd1.c3
FROM
    (SELECT * FROM upd1) AS upd1 WHERE upd1.c1 = upd2.c1
```

- PostgreSQL 9.5

```
UPDATE upd2
SET (c2, c3) =
    (SELECT c2, c3 FROM upd1 WHERE upd1.c1 = upd2.c1)
```

JSONB

演算子と関数の追加

- 「||」 演算子

- 要素の追加／更新を行う

```
postgres=> SELECT '{"key":"key1", "val1":"1000"}'::jsonb ||  
           '{"val2":"2000"}'  
           ?column?
```

```
-----  
{"key": "key1", "val1": "1000", "val2": "2000"}  
(1 row)
```

- 「-」 演算子

- 要素の削除を行う

- 入れ子構造の要素を削除する「 #- 」演算子も追加

```
postgres=> SELECT '{"key":"key1", "val1":"1000"}'::jsonb - 'key' ;  
           ?column?
```

```
-----  
{"val1": "1000"}  
(1 row)
```

JSONB

演算子と関数の追加

– jsonb_set関数

– 要素の置換／更新を行う

```
postgres=> SELECT
  jsonb_set('{"key":"key1", "val1":"1000"}'::jsonb, '{"val1"}', '2000') ;
  jsonb_set
-----
{"key": "key1", "val1": 2000}
(1 row)
```



JSONB

演算子と関数の追加

– その他

| 関数名 | 機能 | 備考 |
|-------------------|-----------|------|
| jsonb_pretty | 整形 | |
| jsonb_strip_nulls | NULL要素の削除 | |
| jsonb_concat | 結合 | 演算子 |
| jsonb_delete | 削除 | -演算子 |



3. 大規模環境に対応する機能

BRIN Index (Block Range Index)

大規模環境におけるストレージ量とパフォーマンスのバランス

– B-treeインデックス

- B-treeインデックスは、列値とタプルID (TID) のセットをソートして保存
- 範囲検索は非常に高速
- 大規模環境では、ストレージ使用量が多い

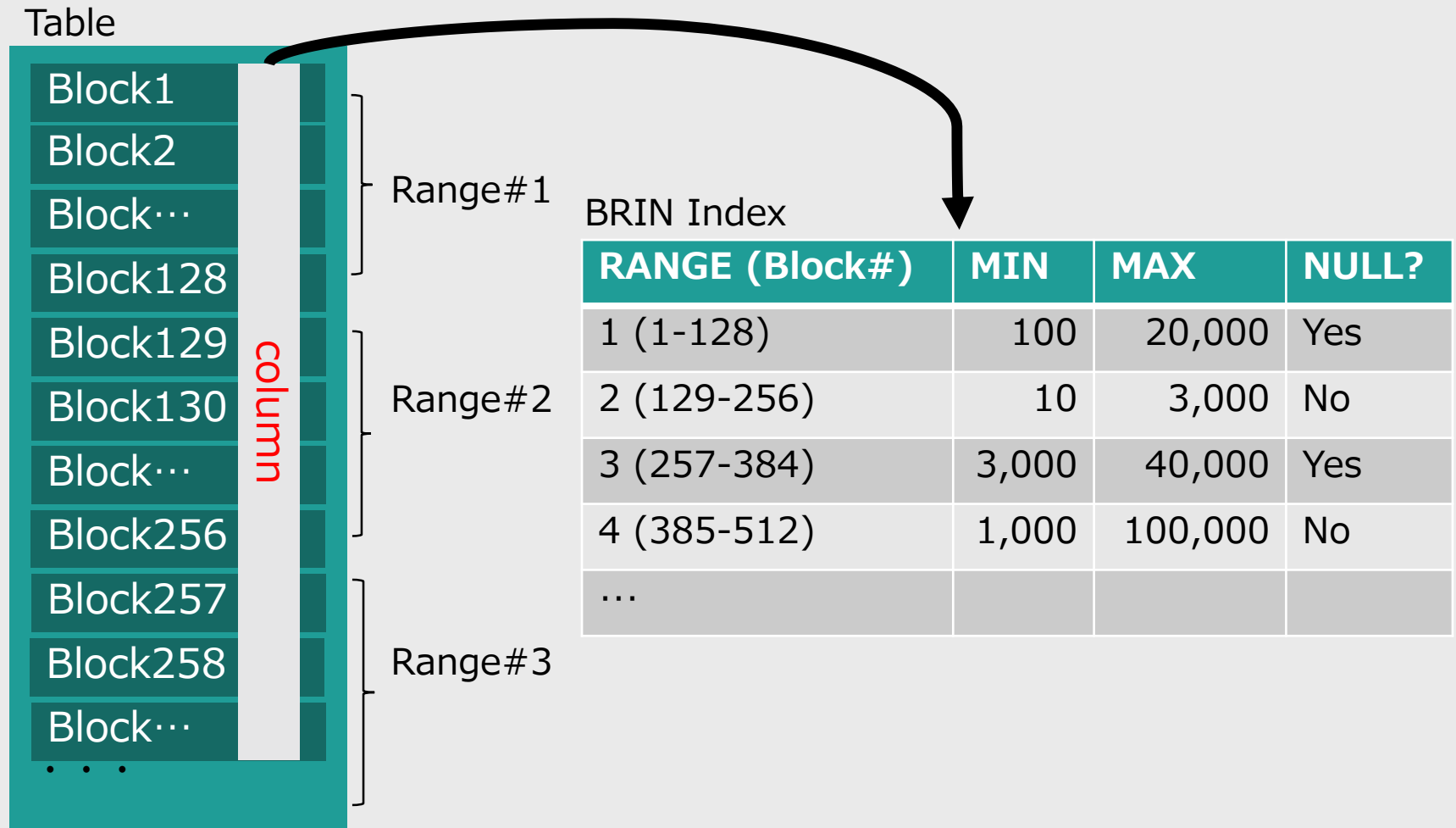
– BRINインデックス

- 隣接する複数ブロックを 1 レンジとして、レンジ単位に最大値／最小値／NULL値の有無を保持
- レンジ内のブロック数はインデックスのオプションpages_per_rangeで決定（デフォルト値 128）
- ストレージ使用量が非常に少ない
- B-treeインデックスよりも低速だが、全件検索よりも高速
- 時系列データ等、連続して増減する列に対して特に有効

```
CREATE INDEX index_name ON table_name USING BRIN (column_name)
[ WITH (pages_per_range = #of pages) ]
```

BRIN Index (Block Range Index)

BRIN Indexの構造



BRIN Index (Block Range Index)

実行例

```
postgres=> CREATE TABLE brin1 (c1 NUMERIC, c2 NUMERIC) ;
CREATE TABLE
postgres=> INSERT INTO brin1 VALUES (generate_series(1, 100000000),
    generate_series(1, 100000000)) ;
INSERT 0 100000000
postgres=> CREATE INDEX idx_btree ON brin1 (c1) ;
CREATE INDEX
postgres=> CREATE INDEX idx_brin ON brin1 USING BRIN (c2) ;
CREATE INDEX
postgres=> SELECT relname, pg_size_pretty(pg_relation_size(oid)) FROM
    pg_class ;
relname  | pg_size_pretty
-----+-----
brin1    | 4223 MB
idx_btree | 2142 MB
idx_brin  | 160 kB
```

BRIN Index (Block Range Index)

実行計画

```
postgres=> EXPLAIN SELECT * FROM brin1 WHERE c1 > 5000000000 ;
```

QUERY PLAN

Index Scan using idx_btree on brin1 (cost=0.57..4.58 rows=1 width=12)

Index Cond: (c1 > '5000000000'::numeric)

(2 rows)

```
postgres=> EXPLAIN SELECT * FROM brin1 WHERE c1 > 50000000 ;
```

QUERY PLAN

Bitmap Heap Scan on brin1 (cost=384339.57..1544654.07 rows=49581880 width=12)

Recheck Cond: (c1 > '50000000'::numeric)

-> **Bitmap Index Scan on idx_brin** (cost=0.00..371944.10 rows=49581880 width=0)

Index Cond: (c1 > '50000000'::numeric)

(4 rows)

```
postgres=> EXPLAIN SELECT * FROM brin1 WHERE c1 > 5000000 ;
```

QUERY PLAN

Seq Scan on brin1 (cost=0.00..1790541.10 rows=94859126 width=12)

Filter: (c1 > '5000000'::numeric)

(2 rows)

SELECT TABLESAMPLE

サンプリング検索

```
SELECT ... FROM table_name
```

```
TABLESAMPLE {SYSTEM | BERNOULLI} (percent) [ REPEATABLE (seed) ]
```

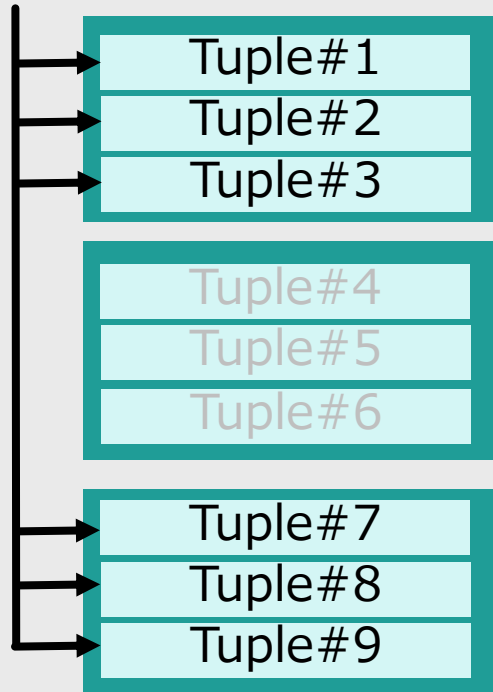
- テーブル内の一部をサンプリング
 - percent でサンプリング割合をパーセンテージで指定する (0～100)
 - WHERE句を指定した場合は、サンプリング後に評価される
 - SYSTEM
 - ブロック単位でサンプリング (ランダム・スキャン)
 - ブロック内の全タプルを使用
 - BERNOULLI
 - タプルの単位でサンプリング (シーケンシャル・スキャン)
 - SYSTEMよりも正確だがI/O負荷が高い
 - REPEATABLE
 - サンプリング・アルゴリズムに使用する数値を指定
 - 省略時はSQL実行時に乱数が適用される



SELECT TABLESAMPLE

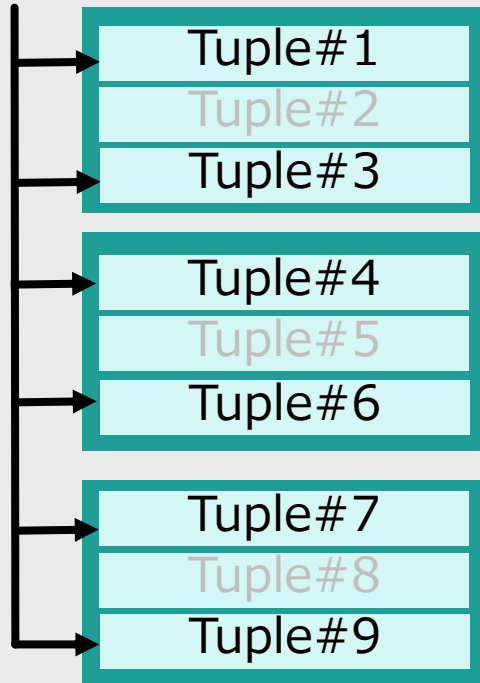
サンプリング検索

SYSTEM



サンプル率が1% 以上の
場合、Bulk Read

BERNOULLI



常にBulk Read

SELECT TABLESAMPLE

サンプリングのI/O負荷

– 同一内容（約5,000ブロック）の2テーブルに対して10%のサンプル検索

```
postgres=> SELECT COUNT(*) FROM samplesys TABLESAMPLE
           SYSTEM (10) ;
postgres=> SELECT COUNT(*) FROM sampleber TABLESAMPLE
           BERNOULLI (10) ;
postgres=> SELECT relname, heap_blks_read FROM pg_statio_user_tables ;
 relname | heap_blks_read
-----+-----
samplesys |           533
sampleber |          4759
(2 rows)
```



CREATE FOREIGN TABLE INHERITS

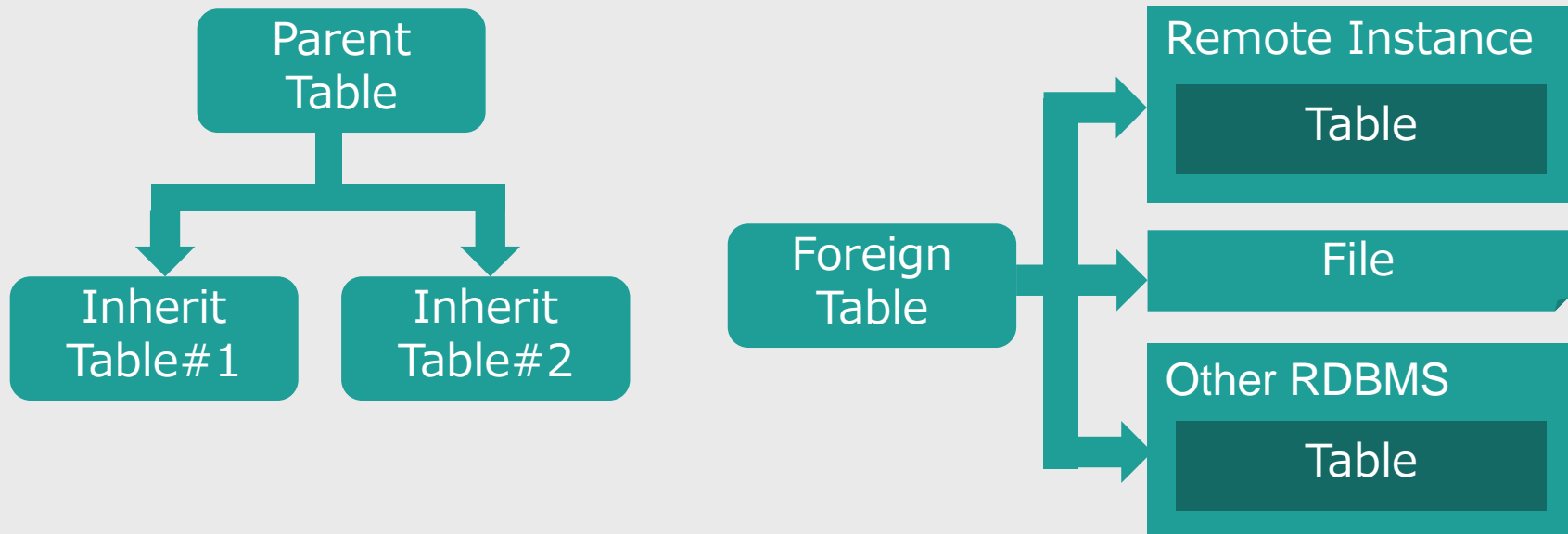
複数リモート・インスタンスに処理をオフロード

– 継承テーブル (INHERITS)

- 親子関係を持つテーブル (パーティション・テーブルと呼ぶことも)
- WHERE句により自動的に子テーブルを参照することで負荷分散が可能

– 外部テーブル (FOREIGN TABLE / FOREIGN DATA WRAPPER)

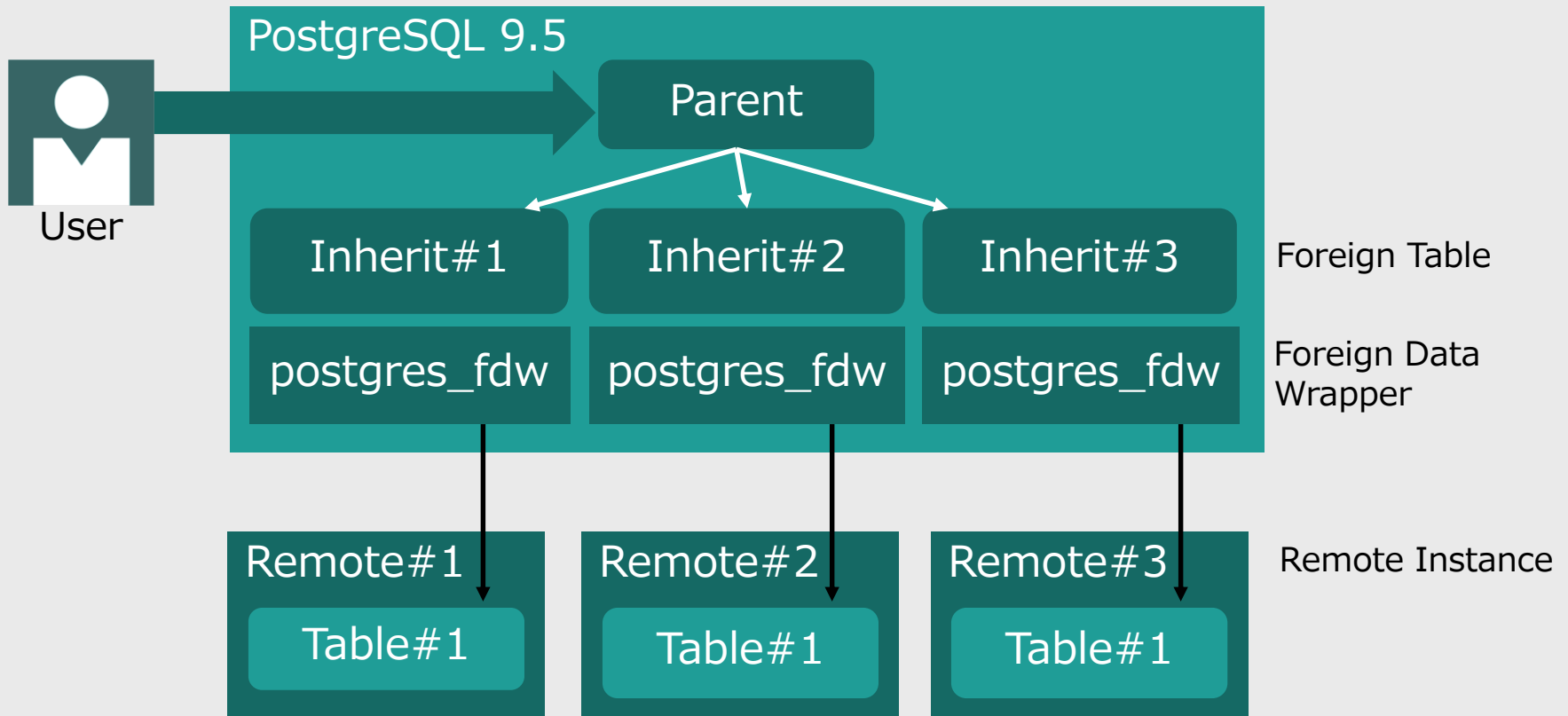
- リモート・インスタンスのテーブルやファイルをテーブルとして参照できる機能 (ファイルや他システム等)



CREATE FOREIGN TABLE INHERITS

複数リモート・インスタンスに処理をオフロード

– PostgreSQL 9.5では、外部テーブルと継承テーブルを混在可能に



CREATE FOREIGN TABLE INHERITS

実行例

```
postgres=> CREATE TABLE parent_table (col1 NUMERIC, ...) ;


postgres=# CREATE SERVER remote1 FOREIGN DATA WRAPPER
      postgres_fdw OPTIONS (host 'remsvr1', dbname 'userdb1', port '5432') ;

postgres=# CREATE USER MAPPING FOR public SERVER remote1 OPTION
      (user 'demo', password 'secret') ;

postgres=> CREATE TABLE inherit_table1 INHERITS (parent_table) ;
      SERVER remote1 ;
```

- 集計処理 (COUNT / AVG / SUM / GROUP BY etc) 、結合処理、LIMIT句はローカル・インスタンスで実施される





4. 運用を容易にする機能

pg_rewind

レプリケーション環境の再同期

– PostgreSQL 9.4まで

- ① マスター・インスタンスの異常終了
- ② スレーブ・インスタンスを昇格
- ③ 旧マスター・インスタンスのデータを削除し、**全データをコピーし再設定**

– PostgreSQL 9.5

- ① マスター・インスタンスの異常終了
- ② スレーブ・インスタンスを昇格
- ③ pg_rewindコマンドで旧マスターを**差分更新**



pg_rewind

レプリケーション環境の再同期

– 実行条件

- 旧マスター・インスタンス側で起動
- wal_log_hints = on (デフォルトoff)
- full_page_writes = on (デフォルトon) またはチェックサムの有効化
- 旧マスター (ターゲット) インスタンスが**正常**終了していること
 - pg_controlファイルの「Database cluster state」が「shut down」状態

```
$ pg_rewind
--target-pgdata={旧マスターのクラスター}
--source-server={新マスター接続情報}
--dry-run シミュレーション実行
--progress 進捗状況の出力
--debug 追加情報の出力
--help 使用方法の表示
```



5. 罨

ALTER TABLE SET UNLOGGED / LOGGED

更新時のWAL出力量を制御

- 更新処理（INSERT, UPDATE, DELETE）実行時にはWALが出力
 - データベース障害時の復旧に使用
 - pg_xlogディレクトリ内の16MBのファイル群
 - OLTP環境ではパフォーマンス・ボトルネック
- PostgreSQL 9.5ではテーブル単位にWALの生成を切り替え可能に

```
ALTER TABLE テーブル名 SET UNLOGGED
```

```
ALTER TABLE テーブル名 SET LOGGED
```

- UNLOGGEDテーブルはpg_classカタログのrelpersistence = 'u'
 - TEMPORARY TABLEからの変更は不可



ALTER TABLE SET UNLOGGED / LOGGED

更新時のWAL出力量を制御

– 内部実装は？

- 新規テーブルの作成とデータのコピーを実行
- ALTER TABLE WITH[OUT] OID文, ALTER TABLE ALTER TYPE文と同じ動作

```
postgres=> SELECT pg_relation_filepath('logged1') ;  
pg_relation_filepath
```

```
-----  
base/16385/41347  
(1 row)
```

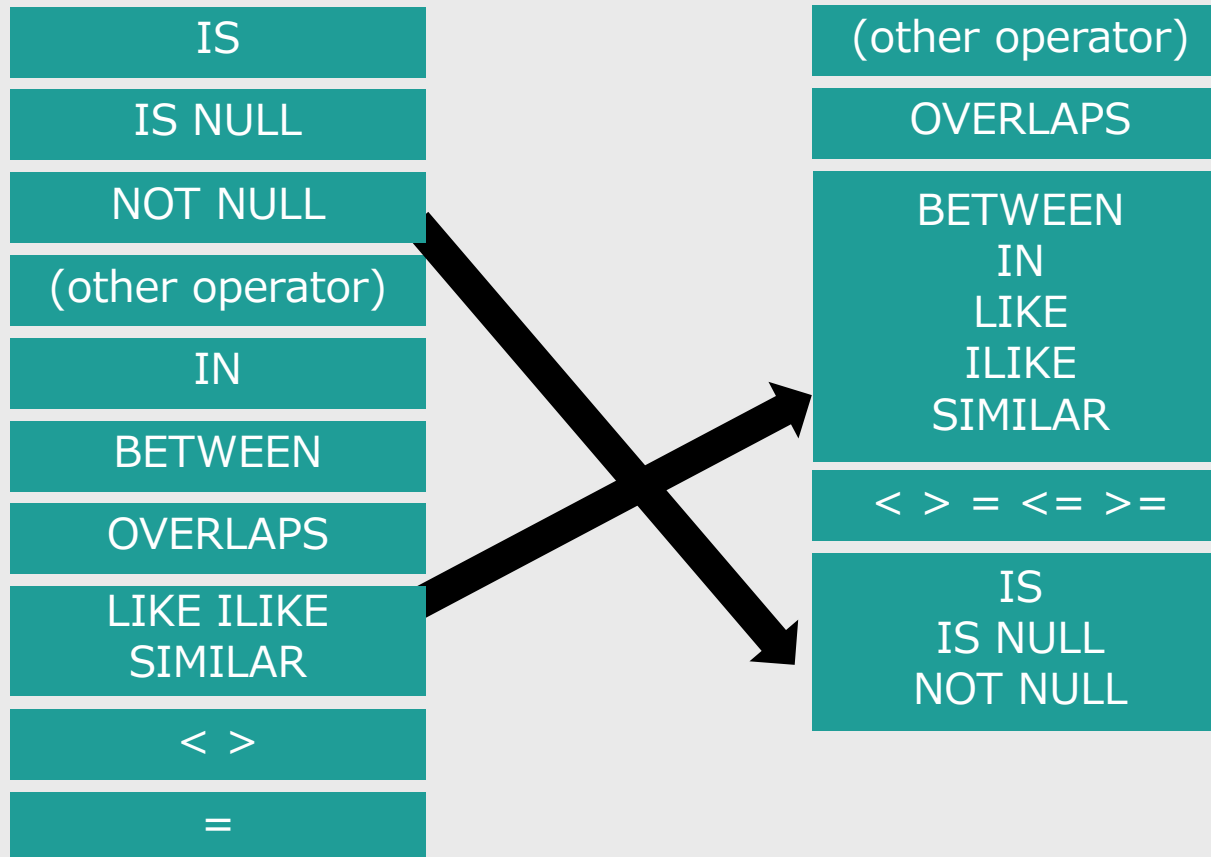
```
postgres=> ALTER TABLE logged1 SET UNLOGGED ;  
ALTER TABLE
```

```
postgres=> SELECT pg_relation_filepath('logged1') ;  
pg_relation_filepath
```

```
-----  
base/16385/41353  
(1 row)
```


優先順位の変更

演算子の優先順位が変更された



優先順位の変更

演算子の優先順位が変更された

- 優先順位の変更により影響を受けるSQLに警告を出力するパラメーター

operator_precedence_warning (デフォルトoff)

- 実行例

```
postgres=> SET operator_precedence_warning = on ;
SET
postgres=> SELECT COUNT(*) FROM sample1 WHERE c1 > 10 IS true ;
WARNING: operator precedence change: IS is now lower precedence than >
LINE 1: SELECT COUNT(*) FROM sample1 WHERE c1 > 10 IS true ;

count
-----
999990
(1 row)
```



インスタンス停止モード デフォルト値の変更

– pg_ctl stopコマンドの停止モードのデフォルトが変更された

PostgreSQL 9.4までのデフォルト

- pg_ctl stop -D data -m **smart**

PostgreSQL 9.5のデフォルト

- pg_ctl stop -D data -m **fast**

パラメーターの変更

追加されたパラメーター

| パラメータ名 | 説明 |
|-----------------------------|--|
| max_wal_size | チェックポイントの開始サイズ (checkpoint_segments 廃止) |
| min_wal_size | WALリサイクルを行うサイズ |
| cluster_name | プロセス名の指定 |
| gin_pending_list_limit | GINインデックスの待機リスト最大値 |
| row_security | Row Level Security機能の有効化 |
| track_commit_timestamp | トランザクションのコミット時間の出力 |
| wal_compression | WAL圧縮機能の有効化 |
| log_replication_commands | レプリケーション関連ログの出力 |
| operator_precedence_warning | 優先順位の変更影響に関する警告 |
| wal_retrieve_retry_interval | WALデータの再取得間隔の指定 |

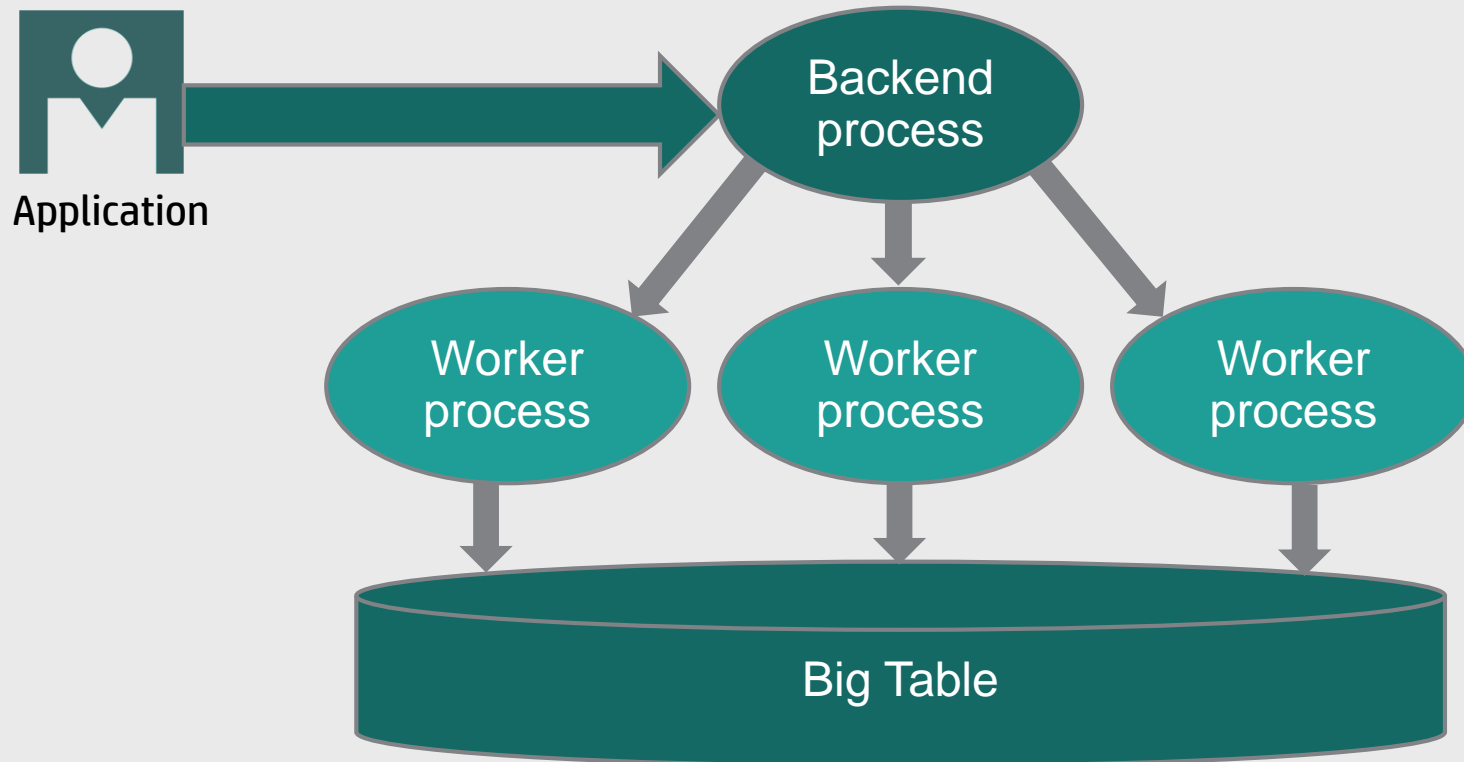




6. PostgreSQL 10?

Parallel Seq Scan / Parallel Join

並列検索



Parallel Seq Scan / Parallel Join

並列検索

– 実行計画

```
postgres=# SET max_parallel_degree = 10 ;
SET
postgres=> EXPLAIN SELECT * FROM data1 WHERE col2='1000' ;
               QUERY PLAN
-----
Gather (cost=1000.00..40789.71 rows=1 width=11)
  Number of Workers: 4
    -> Parallel Seq Scan on data1 (cost=0.00..39789.61 rows=1 width=11)
        Filter: ((c2)::text = '1000'::text)
```

– 永安さんのブログで紹介

– <http://pgsqldeepdive.blogspot.jp/2015/12/parallel-seq-scan.html>

Parallel Seq Scan / Parallel Join

並列検索

- 現状の実装では並列度の最大値はパラメーターmax_parallel_degree (Superuserのみ変更可) で決定される。ただしパラメーターmax_worker_processesを超えられない。
- 3,000ブロック、9,000ブロック、27,000ブロックごとにワーカー・プロセスを増加させる実行計画を作成する。
- 最新のSnapshotにコミット済み



その他の新機能

Commitfestsから

| Patch | Status | 説明 |
|-------------------------------------|---------------------|--|
| Declarative partitioning | Needs Review | ネイティブ・パーティション・テーブルの作成 |
| postgres_fdw | Committed | postgres_fdwにfetch_sizeオプション追加、extensionsオプション追加 |
| Idle In Transaction Session Timeout | Ready for Committer | アイドル状態のトランザクション・タイムアウト指定パラメーター |
| generate_series | Ready for Committer | date型の生成に対応 |
| Audit Extension | Needs Review | 監査機能の拡張 |

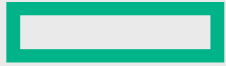


7. まとめ



まとめ

- PostgreSQL 9.5には、魅力的な新機能が数多く採用されました。
 - パフォーマンスの向上
 - 大規模環境に対応した新機能
 - アプリケーション開発を容易にする新機能
- 参考URL
 - Commitfests
<http://commitfest.postgresql.org/>
 - PostgreSQL 9.5新機能紹介（澤田さん）
<http://www.slideshare.net/hadoopxnttdata/postgresql-95-new-features-nttdata>
 - Michael Paquierさんのブログ
<http://michael.otacoo.com/>
 - めこ@横浜さんのブログ
http://d.hatena.ne.jp/nuko_yokohama/
 - Performance improvements in PostgreSQL 9.5 (and beyond)
<http://www.slideshare.net/fuzzycz/performance-improvements-in-postgresql-95-and-beyond>



Hewlett Packard
Enterprise

Thank you

noriyoshi.shinoda@hpe.com