AWS

SUMMIT

Oracle Database から Aurora & Redshift に移行するための実践ガイド

アマゾン ウェブ サービス ジャパン株式会社
柴田竜典
2017/6/1

amazon web services

自己紹介

柴田竜典［シバタツ］

- データベース関連の
  相談ごと何でも担当
    - AWSへの移行を機に
      RDBMSをAuroraに
      乗り換えたい
    - オンプレミスOracleを
      AWSにフォーク
      リフティングしたい
- 好きなAWSのサービス: S3

@rewse

Oracleからの移行を検討しているお客様の声

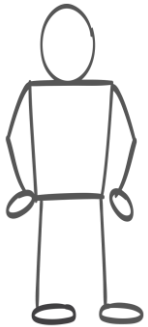
- クラウドにシステム全体を移行するのであれば、RDBMSもクラウドネイティブなものにしたい
- RDBMSもオートスケールさせたいがCPUライセンスだとそれができない
- IT予算の多くをOracleライセンスが占めている

などなど

移行にあたっての不安や悩み

業務部門 / アプリ開発部門

- アプリケーションへの影響は
  どれくらいあるのだろうか
- 移行のための業務停止は
  できるだけ短くしたい



IT部門 / インフラ管理部門

- 業務部門に何をガイドして
  よいのか分からない
- 従来の管理手法が
  どう変わるのだろうか
- 移行のための費用は
  あまりかけたくない

移行にあたって決めなくてはいけないこと

1. What?
   対象システムは？
2. Why?
   移行理由は？
3. How?
   移行戦略は？
4. Where?
   移行先は？
5. When?
   期限は？
6. Who?
   担当者は？

AWSの考えるクラウドへの6つの移行戦略 (1/3)

Re-Host | ホスト変更

サーバーやアプリケーションを
オンプレミス環境からクラウドに
そのまま持ってくる

オンプレミスのOracleを
そのままEC2に持ってくる

Re-Platform | プラットフォーム変更

クラウド移行の一環として
プラットフォームの
アップグレードを行なう

オンプレミスの Oracle 10g を
EC2に構築した12cに移行する

AWSの考えるクラウドへの6つの移行戦略 (2/3)

Re-Purchase | 買い換え

クラウド対応したライセンス
またはアプリケーションに
買い換える

オンプレミスのOracleを
ライセンス込みの
RDS for Oracle に買い換える

Refactor | 書き換え

クラウド環境で最適に動作する
ようにアプリケーションを
書き換える

オンプレミスのOracleを
AuroraやRedshiftに変更する

AWSの考えるクラウドへの6つの移行戦略 (3/3)

Retire | 廃止

サーバーやアプリケーションを
廃止する

平行運用していた
古いシステムをDBごと
このタイミングで廃止する

Retain | 保持

オンプレミス環境を
引きつづき使用する

Oracle9iに依存している
アップグレードできない
パッケージアプリケーションが
ある

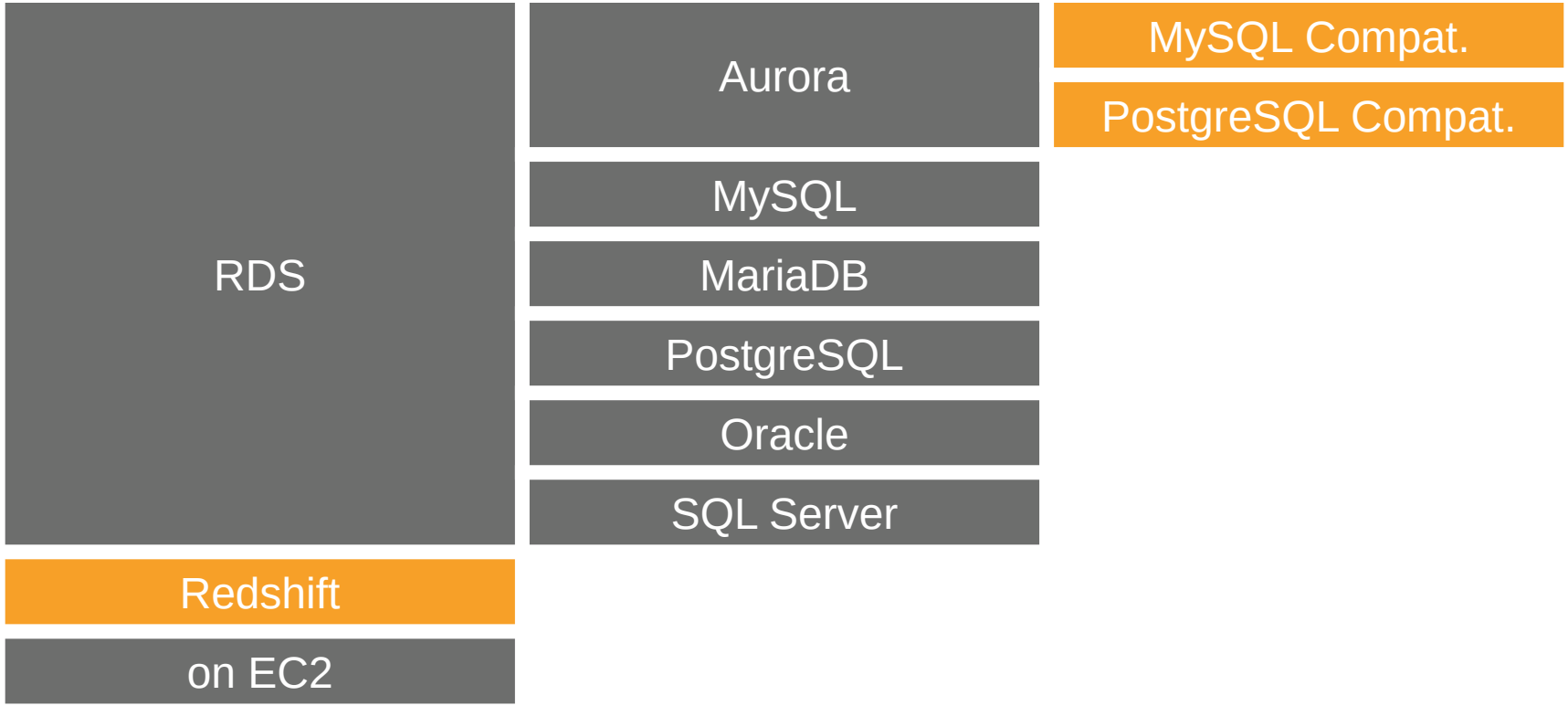
クラウド移行戦略を移行の複雑さで比較

- Retain | 保持
- Retire | 廃止
- Re-Host | ホスト変更
- Re-Purchase | 買い換え
- Re-Platform | プラットフォーム変更
- Refactor | 書き換え

移行にあたって決めなくてはいけないこと

1. What?
   対象システムは？
2. Why?
   移行理由は？
3. How?
   移行戦略は？
4. Where?
   移行先は？
5. When?
   期限は？
6. Who?
   担当者は？

本セッションでご紹介する移行先



3つのRDBMSの特性概要

	Aurora MySQL	Aurora PostgreSQL	Redshift
OLTP性能	★★★	★★★	-
OLAP性能	-	★★	★★★
目標レスポンス時間	数ミリ秒から数十秒	数ミリ秒から数十分	数秒から数時間
結合方法 *1	ネステッドループ *2	ネステッドループ、 ハッシュ、ソートマージ	ハッシュ、 ソートマージ
インデックス *1	Bツリー、空間、 全文	Bツリー、関数、空間、 全文、ゾーンマップ	ゾーンマップ
制約 *1	NOT NULL、PK、 UNIQUE、FK	NOT NULL、PK、 UNIQUE、FK、CHECK	NOT NULL *3
Oracleとの文法互換性	★★	★★★	★★★

*1 移行を前提としているため、Oracle Databaseにない機能は記載省略
*2 Block Nested Loop 結合と Batched Key Access 結合を含む
*3 PK、UNIQUE、FKは定義できるが強要されない

Statspack/AWRから現行ワークロードを評価

- 短時間のSQLが大量に流れているのか、
  長時間のSQLが少量流れているのかを見極める
- SQL ordered by Elapsed Time を調査
  - Elapsed Time は期間中の合計時間
  - Elapsed Time per Exec が短い = 短時間のSQLが大量
  - Elapsed Time per Exec が長い = 長時間のSQLが少量

移行にあたって決めなくてはいけないこと

1. What?
   対象システムは？
2. Why?
   移行理由は？
3. How?
   移行戦略は？
4. Where?
   移行先は？
5. When?
   期限は？
6. Who?
   担当者は？

期限と担当者を決めるには工数見積もりが必要
工数 = 移行先との違いの量

AWSが提供する移行支援サービス

- AWS TCO Calculator
- AWS Application Discovery Service

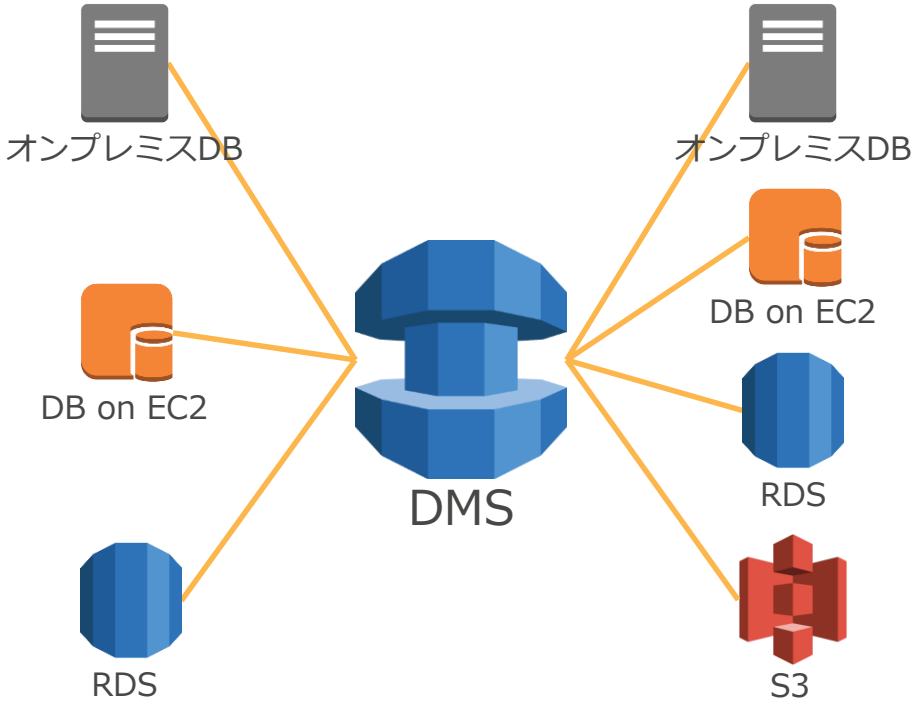
- VM Import/Export
- AWS Server Migration Service
- **AWS Database Migration Service**
- AWS Storage Gateway
- AWS Snowball
- VMware on AWS

- AWS CloudWatch
- AWS Config
- AWS CloudFormation
- AWS CloudTrail
- AWS Service Catalog
- AWS Trusted Advisor
- AWS サポート

AWS Database Migration Service (DMS)

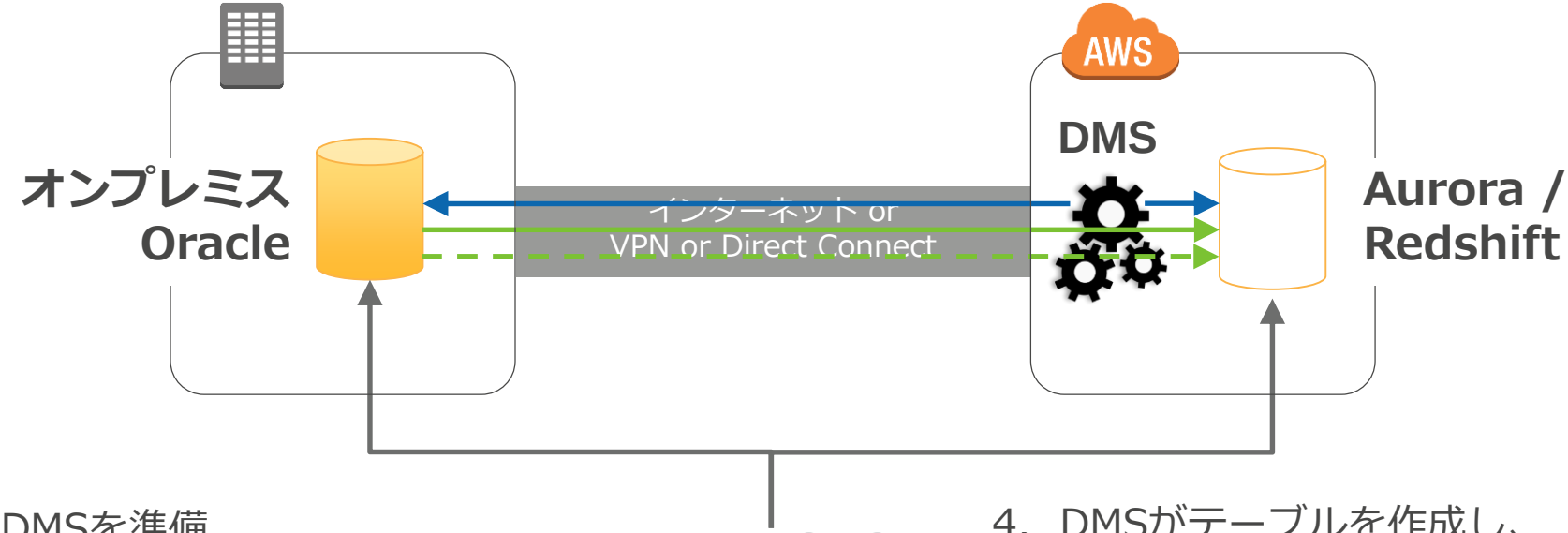
既存のデータベースを
最小限のダウンタイムで
マイグレーションする
サービス

同種はもちろん
異種プラットフォームの
移行にも対応



※オンプレミス to オンプレミスは非対応

移行中もアプリケーションは稼働したまま



1. DMSを準備
2. DMSがソースDBと
   ターゲットDBに接続
3. 対象のテーブル、スキーマ
   などを選択
4. DMSがテーブルを作成し、
   データをロードし、
   レプリケーション開始
5. 任意のタイミングで
   アプリケーションを
   ターゲットDB側に切り替え

AWS Database Migration Service の特徴

使用が簡単
MCで数回クリックするだけ

最小限のダウンタイム
オンラインでの
継続的レプリケーション対応

豊富な
対応プラットフォーム
Oracle, Microsoft SQL Server,
SAP ASE, MySQL, MariaDB,
PostgreSQL, Aurora, Redshift,
S3, MongoDB, DynamoDB

簡単なセットアップ
ソースDBへの変更はほぼ不要

高い信頼性
マルチAZ可能なインスタンス

低コスト
c4.largeインスタンスで
0.196USD/時間

参考: https://www.slideshare.net/AmazonWebServicesJapan/auroraaws-database-migration-service-schema-conversion-tool

AWS Schema Conversion Tool

ソースDBのスキーマ、ビュー、
ファンクション、
ストアドプロシージャの大部分を
自動的にターゲットDB互換
フォーマットに変換できる
デスクトップアプリケーション

AWS Schema Conversion Tool (SCT) の特徴

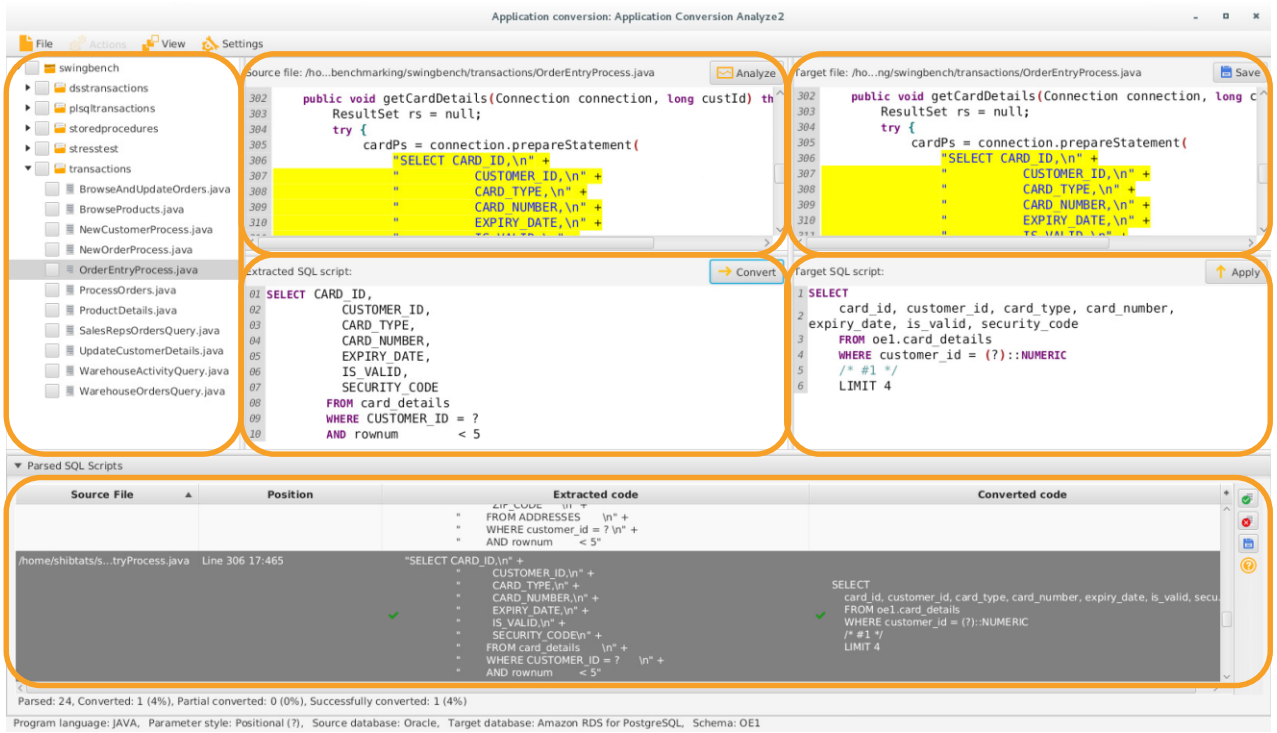
手動変換の補助
自動変換できない個所とその理由を明示

評価レポートの作成
何割のオブジェクトが自動変換可能か
などのPDFレポートを数クリックで
作成でき、変換工数の事前見積もりを
補助

アプリケーションSQLに対応
アプリケーションソースコードを
スキャンして変換

豊富な対応プラットフォーム
Oracle, Microsoft SQL Server, Teradata,
Netezza, Greenplum, Vertica,
MySQL, MariaDB, PostgreSQL,
Aurora, Redshift

機械的に変換できるSQLはSCTで対応

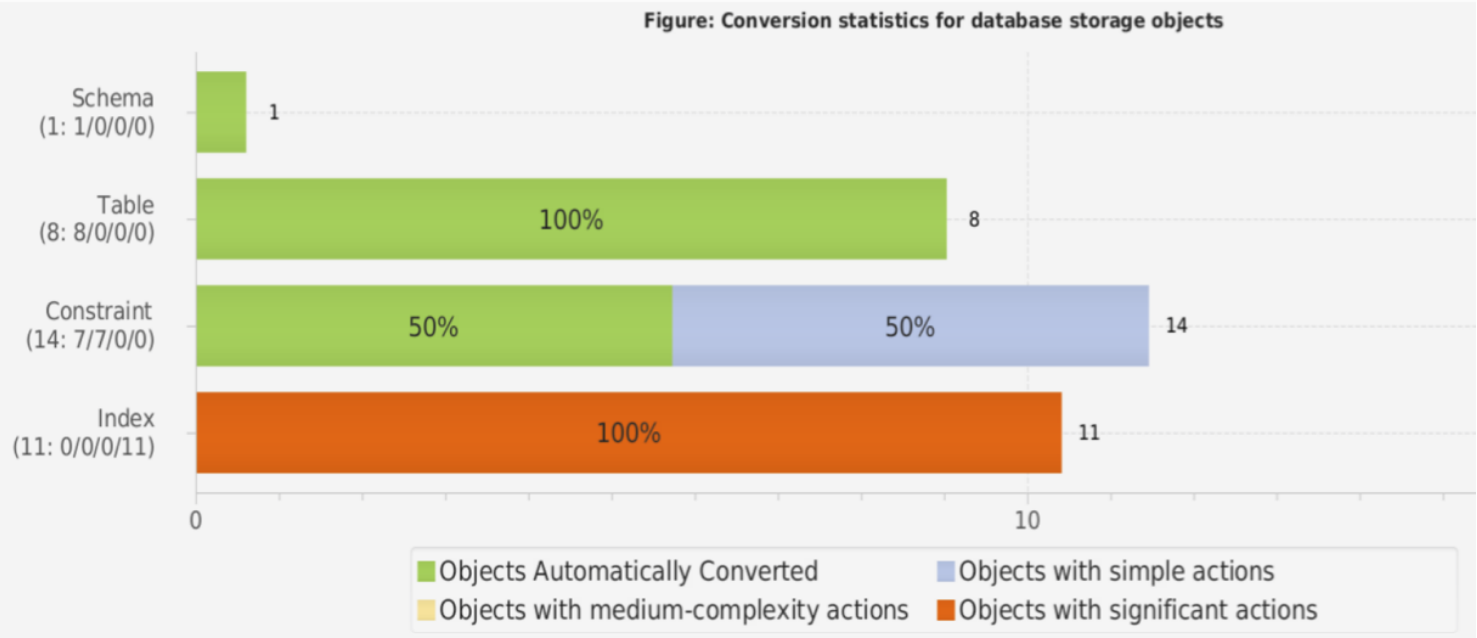


- (+) 結合
- ROWNUM
- 関数の多く

など

評価レポート

完全自動変換できるまたは手動変更工数ごとに
色分けされたグラフなどを含んだレポート



参考: https://www.slideshare.net/AmazonWebServicesJapan/auroraaws-database-migration-service-schema-conversion-tool

Oracle Database との違いのアジェンダ

1. オブジェクトの主な違い
2. SELECTでの主な違い
3. DMLでの主な違い
4. データの移行
5. 構築と運用

ここからの前提

選択バイアスが掛かっているだけなので
ご安心ください

- SCTやDMS自動化できない部分または
  注意を要する部分だけを抽出しています
- Oracle Database にないが Aurora & Redshift には
  ある機能については、このセッションでは扱いません

Oracle RAC から Aurora MySQL への移行事例

レコチョク様

- 全サービスで横断して使用している1000万会員のDB
- DMS / SCT リリース前に手動で移行完了
- 移行した結果
    - Auroraによる障害はこれまでゼロ
    - DBサーバー運用に工数をほとんど割かなくてよい
    - 性能は問題なし（むしろ速くなった）

SlideShare: Oracle RAC から Aurora MySQL への移行

https://www.slideshare.net/recotech/oracle-racaurora-my-sql

オブジェクトの主な違い

オブジェクトの主な違い

	Aurora MySQL	Aurora PostgreSQL	Redshift
インデックス *1	Bツリー、空間、全文	Bツリー、関数、空間、全文、ゾーンマップ	ゾーンマップ
制約 *1	NOT NULL、PK、UNIQUE、FK	NOT NULL、PK、UNIQUE、FK、CHECK	NOT NULL *2
シーケンス	AUTO_INCREMENT	✓	
マテリアライズドビュー		フルリフレッシュ	
データベースリンク	*3	✓	
パーティショニング	✓	RANGE、LIST	
ストアドプロシージャ	ストアドルーチン	PL/pgSQLなど	Python

*1 移行を前提としているため、Oracle Databaseにない機能は記載省略
*2 PK、UNIQUE、FKは定義できるが強要されない
*3 AuroraではないMySQLの場合はFEDERATEDエンジンで代替可能

✓: ほぼ同様の機能があるもの

MySQLのシーケンス

- 列にAUTO_INCREMENT属性をつけ、
  その列にNULLまたは0を入れる
  または何も入れないと、
  その列内の最大値 + 1 が入る
- CURRVALに相当する値は
  LAST_INSERT_ID()関数を使用
    - 直近のINSERTで採番された値を返す

参考: https://dev.mysql.com/doc/refman/5.6/ja/example-auto-increment.html

PostgreSQLのマテリアライズドビュー

- リフレッシュは手動のフルリフレッシュのみ
- 読み取り専用のみ対応

参考: https://www.postgresql.jp/document/9.6/html/rules-materializedviews.html

PostgreSQLのデータベースリンク

- 関数として実装されている
  - ```
    SELECT * FROM dblink(
      'dbname=mydb user=u1',
      'select c1 from t1'
    ) AS t(c text);
    ```
  - ```
    CREATE VIEW rt1 AS
    SELECT * FROM dblink('dbname=mydb',
      'select c1 from t1') AS t(c text);
    ```

参考: https://www.postgresql.jp/document/9.6/html/contrib-dblink-function.html

Redshift → PostgreSQL のdblink

- アドホックな分析をする少数ユーザー用の
  データウェアハウスとしてRedshiftを使用
- ダッシュボードを閲覧する多数ユーザー用の
  データマートとして Aurora PostgreSQL を使用

MySQLとPostgreSQLのパーティショニング

- MySQL
    - RANGE、HASH、LIST、KEY
- PostgreSQL
    - RANGE、LIST
    - サブパーティションとEXCHANGEは非対応

参考: https://dev.mysql.com/doc/refman/5.6/ja/partitioning.html
https://www.postgresql.jp/document/9.6/html/ddl-partitioning.html

ストアドプロシージャ / ストアドファンクション

- PostgreSQLのPL/pgSQLとMySQLの
  ストアドルーチンは
  PL/SQLに似ているが完全に同じではない
  - 参考: PostgreSQL 9.6.2文書: 41.12. Oracle PL/SQL からの移植
    https://www.postgresql.jp/document/9.6/html/plpgsql-porting.html
- RedshiftはPythonによるストアドファンクションのみ
- Aurora MySQL、Aurora PostgreSQL、Redshiftは
  Javaプロシージャ非対応

参考: https://dev.mysql.com/doc/refman/5.6/ja/stored-programs-views.html
https://www.postgresql.jp/document/9.6/html/xplang.html

SELECTでの主な違い

互換性がないとどうなるのか

A) 文法エラーになる

→ エラーに地道に対応していけば良い

B) 文法エラーが出ずに、結果が異なる

→ 想定している結果なのかも評価しないと
非互換に気づかない

やっかいなのはパターンB

一連のバッチの最終結果しかなかったりすると、
どのSQLで結果が変わってしまったのか調査が非常に大変

結果相違を起こしやすい2大ポイント

- NULLと空文字の扱いの違い
- CHARの埋められた空白の扱いの違い

NULLと空文字の扱いの違い (1/2)

```
INSERT INTO t1 (id, val) VALUES (1, '');

SELECT id FROM t1 WHERE val IS NULL;
```

Oracle Database

空文字はNULLと同じ
→ id=1が返る

MySQL、PostgreSQL、Redshift

NULLと空文字を区別する
→ id=1は返らない

'' でgrep

NULLと空文字の扱いの違い (2/2)

```
1. SELECT NULL || 'aws' FROM t1;
2. SECLECT CONCAT(NULL, 'aws') FROM t1;
```

Oracle Database

NULLは空文字と同じ
→ awsが返る

MySQL

1. || は論理OR演算子
2. CONCATはNULLが返る
   → COALESCEで対策

PostgreSQL、Redshift

1. || はNULLが返る
   → COALESCEで対策
2. CONCATはawsが返る

外部結合したあとの || に特に注意

CHARの末尾に埋められた空白の扱いの違い

```
CREATE TABLE t1 (v char(5));
INSERT INTO t1 values ('aws');
```

```
1. SELECT v || 'cloud' from t1;
2. SELECT CONCAT(v, 'cloud') from t1;
3. SELECT LENGTH(v) from t1;
```

Oracle Database

1. aws cloud
2. aws cloud
3. 5

PostgreSQL、Redshift

1. awscloud
2. aws cloud
3. 3

MySQL

2. awscloud
3. 3

CHARをCONCATしている部分に
RPADを追加
→ CONCAT(RPAD(v, 5, ' '), 'cloud')

その他の機械的に変更しにくい代表的な非互換

	Aurora MySQL	Aurora PostgreSQL	Redshift
結合方法 *1	ネステッドループ *2	ネステッドループ、 ハッシュ、ソートマージ	ハッシュ、 ソートマージ
MERGE	ON DUPLICATE KEY	ON CONFLICT	
ヒント句 *1	索引、結合順序 *3	索引、スキャン方法、 結合方式、結合順序 *4	
ウインドウ関数		✓	✓
完全外部結合		✓	✓
パラレル実行		SELECT	✓

*1 移行を前提としているため、Oracle Databaseにない機能は記載省略
*2 Block Nested Loop 結合と Batched Key Access 結合を含む
*3 MySQL 5.7 からスキャン方法や結合方法にも対応
*4 pg_hint_plan拡張モジュールによってサポート

✓: ほぼ同様の機能があるもの

完全外部結合を使ったMERGE文の書き換え

Oracle Database

```
MERGE INTO t1
  USING t2
  ON (t1.c1 = t2.c1)
WHEN MATCHED THEN
  UPDATE SET t1.c2 = t2.c2
WHEN NOT MATCHED THEN
  INSERT (t1.c1, t1.c2)
  VALUES (t2.c1, t2.c2);
```

PostgreSQL、Redshift

```
CREATE TABLE t1_new
AS SELECT
  CASE
    WHEN t1.c1 IS NOT NULL
    THEN t1.c1 ELSE t2.c1
  END c1,
  CASE
    WHEN t2.c1 IS NOT NULL
    THEN t2.c2 ELSE t1.c2
  END c2
FROM t1
  FULL OUTER JOIN t2
    ON t1.c1 = t2.c1;
```

DMLでの主な違い

トランザクション分離レベル (1/2)

- Oracle Database
  - デフォルトは READ COMMITTED
  - READ UNCOMMITTED と
    REPEATABLE READ には非対応
- MySQL (InnoDB)
  - デフォルトは REPEATABLE READ
    - Phantom Read は発生しない
  - READ COMMITTED でも
    Non Repeatable Read (Fuzzy Read) は発生しない

参考: https://dev.mysql.com/doc/refman/5.6/ja/innodb-transaction-model.html

トランザクション分離レベル (2/2)

- PostgreSQL
  - デフォルトは READ COMMITTED
  - 4つの分離レベルをすべてサポート
- Redshift
  - SERIALIZABLEのみ
    - その他の3つのレベルは設定可能だが SERIALIZABLEのエイリアスになっている

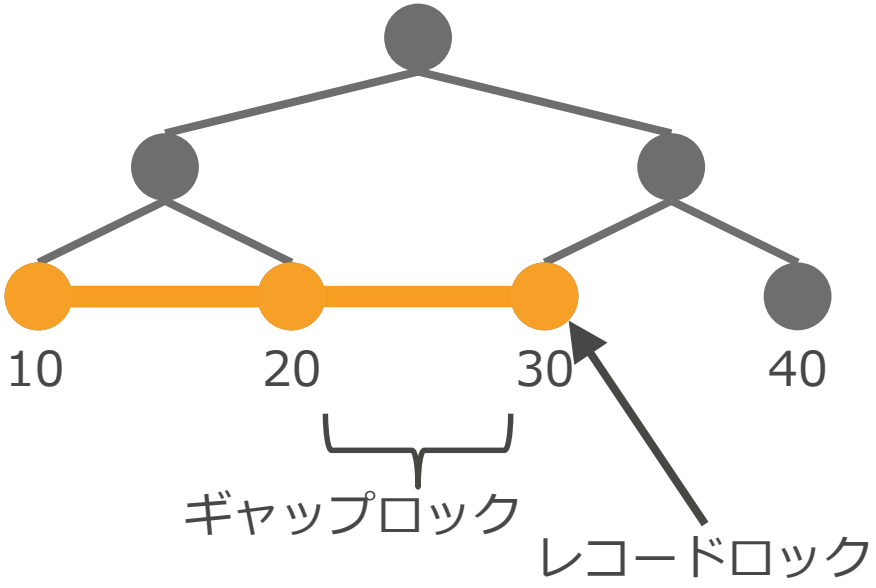
参考: https://www.postgresql.jp/document/9.6/html/transaction-iso.html
http://docs.aws.amazon.com/ja_jp/redshift/latest/dg/c_serial_isolation.html

MySQL (InnoDB) 独自のロック

レコードロック + ギャップロック = ネクストキーロック

- SELECT FOR UPDATE での範囲検索などの場合 範囲外の次のキーまで ロックする
- READ COMMITTED の場合、ギャップロックは 取得しない

```
SELECT id FROM t1
WHERE id BETWEEN 10 AND 20
FOR UPDATE;
```



参考:
https://dev.mysql.com/doc/refman/5.6/ja/innodb-record-level-locks.html
http://dbstudy.info/files/20140907/mysql_lock_r2.pdf

PostgreSQLとRedshiftのVACUUM (1/2)

- PostgreSQLとRedshiftでは、DELETEされた行と関連するインデックスの領域は即座には再利用されない
- PostgreSQLとRedshiftではUPDATEは古い行のDELETE + 新しい行のINSERTとして実行される
- DELETE / UPDATEが多いワークロードではVACUUMコマンドによって不要な領域を解放する必要がある

参考: https://www.postgresql.jp/document/9.6/html/routine-vacuuming.html
http://docs.aws.amazon.com/ja_jp/redshift/latest/dg/t_Reclaiming_storage_space202.html

PostgreSQLとRedshiftのVACUUM (2/2)

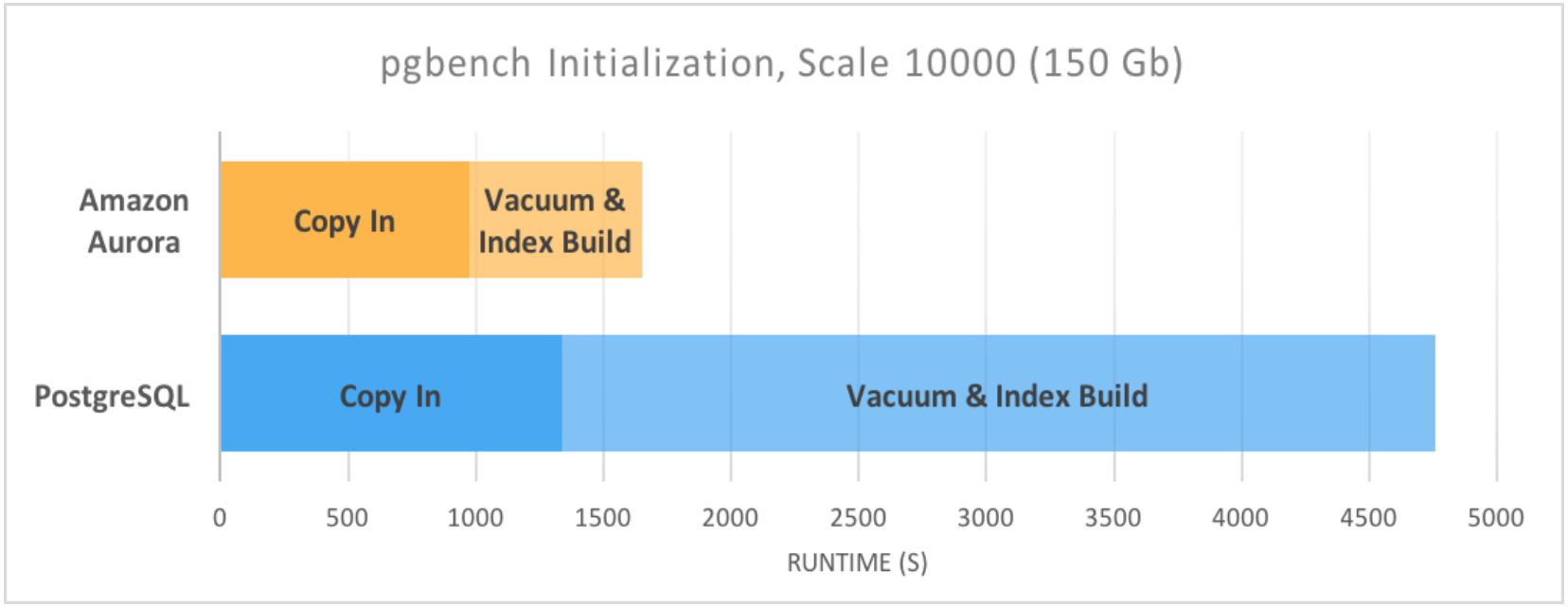
PostgreSQL

- VACUUM
  - 排他ロックを取得しない
  - OSには領域を戻さない
  - デフォルトは自動実行
- VACUUM FULL
  - OSに領域を戻す
  - 排他テーブルロックを取得する
  - IO負荷が非常に高い

Redshift

- VACUUM [FULL]
  - 不要な領域の解放と未ソートデータのソート
- VACUUM DELETE ONLY
  - 不要な領域の解放のみ
- VACUUM REINDEX
  - インターリーブソートキーのメンテナンス
  - IO負荷が非常に高い

Aurora PostgreSQL のVACUUM



Command: `pgbench -i -s 2000 –F 90`

通常のPostgreSQLに比べて Aurora PostgreSQL のVACUUMは高速

構築と運用

マネージド型RDBMSなので 構築と運用は容易に

- インストール
  数クリックで完了
- 災対サイト　Redshift除く
  セレクトボックスから
  選択するだけで
  複数DC冗長構成
- スケールアップ /
  スケールアウト
  数クリックで完了
- バックアップ / リストア
  1クリックで
  スナップショット
  取得 / 復旧
- ハードウェア障害
  起動しなおすだけで復旧
- バージョンアップ
  セレクトボックスから
  選択するだけ

参考: https://www.slideshare.net/AmazonWebServicesJapan/20170510awsblackbeltrds

まとめ

移行にあたって決めなくてはいけないこと

1. What?
   対象システムは？
2. Why?
   移行理由は？
   スケーラビリティや
   コスト削減
3. How?
   移行戦略は？
   リファクタリング
4. Where?
   移行先は？
   SQL ordered
   by Elapsed Time
5. When?
   期限は？
6. Who?
   担当者は？
   主な差分のご紹介

オンプレミスOracleからの移行事例

- AWSパートナー
  事例大全集 Vol.2 にて
  15社の移行事例が掲載

- パミール3Fの
  EXPO展示会場にて配布中

Amazon Web Services ブログで今すぐ公開

- Oracle Database を Amazon Aurora に移行する方法

  https://aws.amazon.com/jp/blogs/news/how-to-migrate-your-oracle-database-to-amazon-aurora/

- オンプレミスや Amazon EC2 上の
  Oracle Database を Amazon Redshift に移行

  https://aws.amazon.com/jp/blogs/news/migrating-oracle-database-from-on-premises-or-amazon-ec2-instances-to-amazon-redshift/

最後に……

- 移行するシステムの順序を決定する要素
  - 複雑かどうか
  - ミッションクリティカルかどうか
  - **得られる利益が大きいかどうか**
- データベースの移行で得られる利益は大きい

お客様の実際のシステムを対象としたアセスメントや
AWSパートナーのご紹介も可能ですので
いつでもご相談ください

AWS

SUMMIT

Thank You!

アマゾン ウェブ サービス ジャパン株式会社
柴田竜典 | シバタツ

amazon web services

本セッションのFeedbackをお願いします

受付でお配りしたアンケートに本セッションの満足度やご感想などをご記入ください
アンケートをご提出いただきました方には、もれなく素敵なAWSオリジナルグッズを
プレゼントさせていただきます

アンケートは受付、パミール3FのEXPO展示会場内にて回収させて頂きます

AWSソリューションDay 2017: Database Day

-すでに始まっている！「クラウドへのデータベース移行」と「データレイクを軸としたビッグデータ活用」-

- Database Day とは？

  ユーザー企業 / パートナー / AWSによる導入事例や活用動向また技術情報を
  ご紹介するIT部門（エンジニア / 管理者など）向けのカンファレンス
- 開催日時 / 会場
  - 2017年7月5日(水) 10:00-17:30 （9:30開場予定）
  - 大崎ブライトコアホール（JR大崎駅より徒歩5分）
- セッション

  基調講演 + 2トラック構成ブレイクアウトセッション
    - トラック1: データベース移行
    - トラック2: データレイク
- お申込み

  https://aws.amazon.com/jp/about-aws/events/2017/solutiondays20170705/