BigData-JAWS 勉強会#2

EmbulkとDigdagで作るRedshiftデータマート

2016/09/26

クラスメソッド株式会社
データインテグレーション部
川崎照夫

自己紹介

- 所属/氏名
  - クラスメソッド株式会社 データインテグレーション部
  - 川崎照夫
- 仕事
  - 前職：BIツール（Pentaho）の代理店でコンサルタント職
  - 現職：データ分析基盤構築のPM
- 好きなAWSサービス
  - Redshift

前職で
分析チームに
お世話になる

JDMC SINCE 2011 Japan Data Management Consortium

前職で「日本データマネジメント・コンソーシアム（JDMC）」の研究会に参加

クラスメソッド株式会社

設立： 2004年、13期目

場所：秋葉原、札幌、上越、バンクーバ、ベルリン

事業：

AWSコンサル	：	約45名の専門チーム
モバイルアプリサービス開発	：	約30名の専門チーム
データ分析環境開発	：	約15名の専門チーム

体制：約100名のエンジニア集団

技術：クラウドでのデータ分析基盤に強み

全世界で14,000社以上あるAWSコンサルティングパートナーの内、

・プレミアコンサルティングパートナー（全世界で31社）

・**ビッグデータコンピテンシー　（全世界で20社）**

・モバイルコンピテンシー　（全世界で 6社）

に認定され、実績を積み重ねております。

Amazon Redshift の代表的な導入事例

はたらくを楽しもう。 intelligence マーケティング分析環境の構築支援	NTT docomo しゃべってコンシェル	dentsu Dentsu.io	SHISEIDO データ分析基盤	Human ヒューマンホールディングス データ分析基盤の構築
スシロー 「うまいすしを、腹一杯。うまいすしで、心も一杯。」	すかいらーくグループ 全国約3,000店舗のPOSデータ数十億件をリアルタイム分析する基盤を開発	無印良品 無印良品	PARCO POCKET PARCO	イオンイーハート データ分析基盤の構築

http://classmethod.jp/cases/

本日のアジェンダ

- データ分析基盤構築の方法論について
  - （議論の土台）
- データマート作成について
  - Embulk/Digdag

#1の振り返り

- EMRの紹介と他のHadoop製品との使い分け
- ドコモのRedshiftの使わせ方

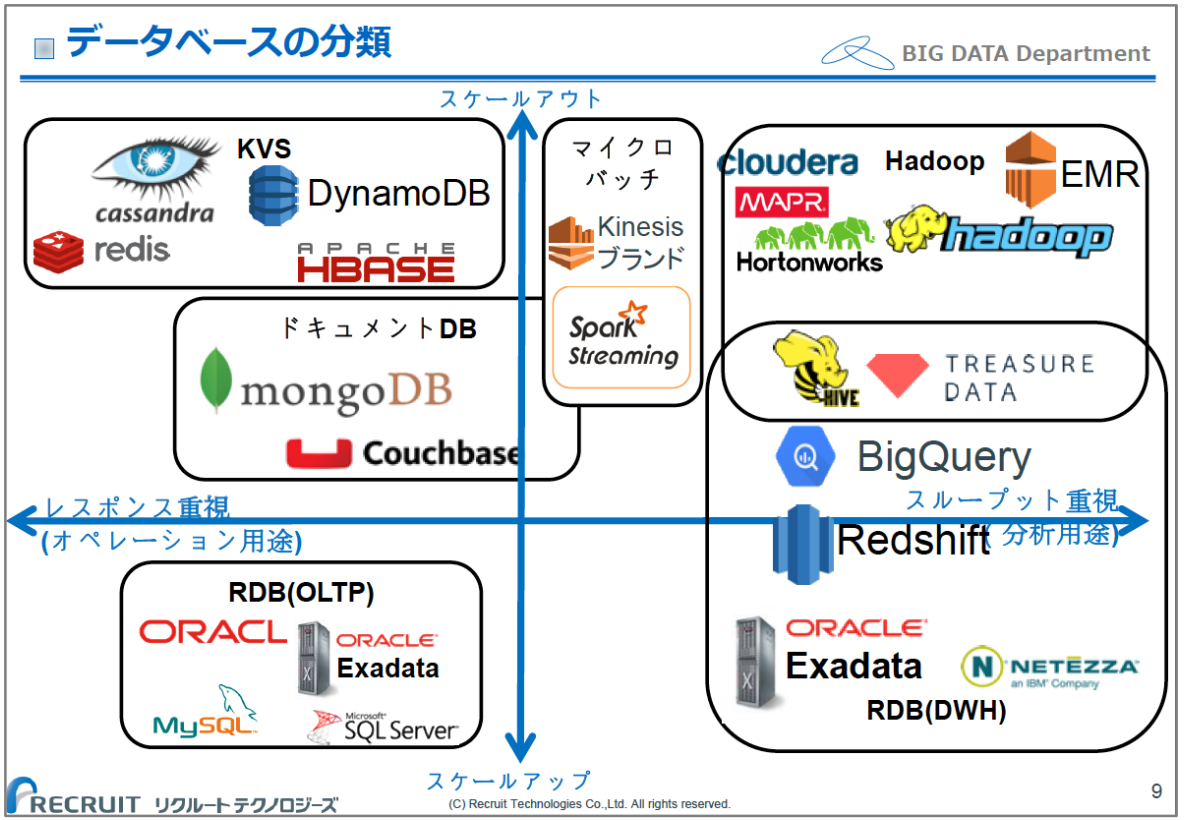
【参考】
BigData-JAWS 勉強会#1 参加レポート #bdjaws | Developers.IO
http://dev.classmethod.jp/cloud/aws/event-report-bigdata-jaws-1/

ビッグデータの分類

- OLTP系⇔情報系、DWH系
- スケールアウト⇔スケールアップ

会場で挙手してもらう
OLTP系⇔情報系
の二択では「情報系」
の担当の方が多い

前回の渡部さんのスライドより



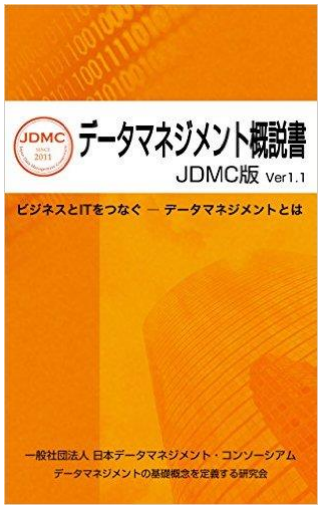
私がビッグデータ支部でやりたいこと

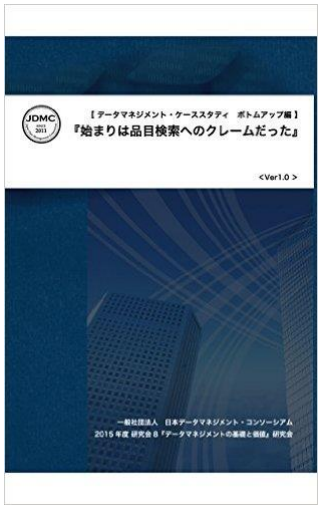
- データ分析基盤構築の方法論を議論
  - ツールの話
  - DWH構築手法
  - データマネジメントの方法論
  - オープンデータの活用

データマネジメントの方法論

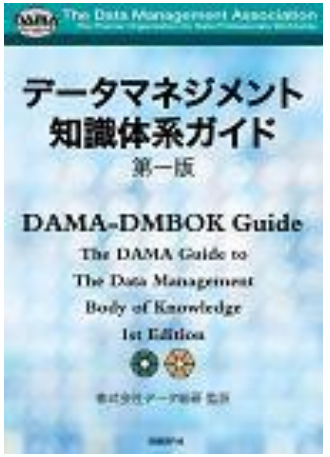
データマネジメントの普及団体JDMC(日本データマネジメント・コンソーシアム)
http://japan-dmc.org/
（データマネジメント製品の）ベンダー企業会員は年会費が必要だが、ユーザー会員は無償で参加可能

出版物：
「データマネジメント概説書」
ケーススタディ2冊





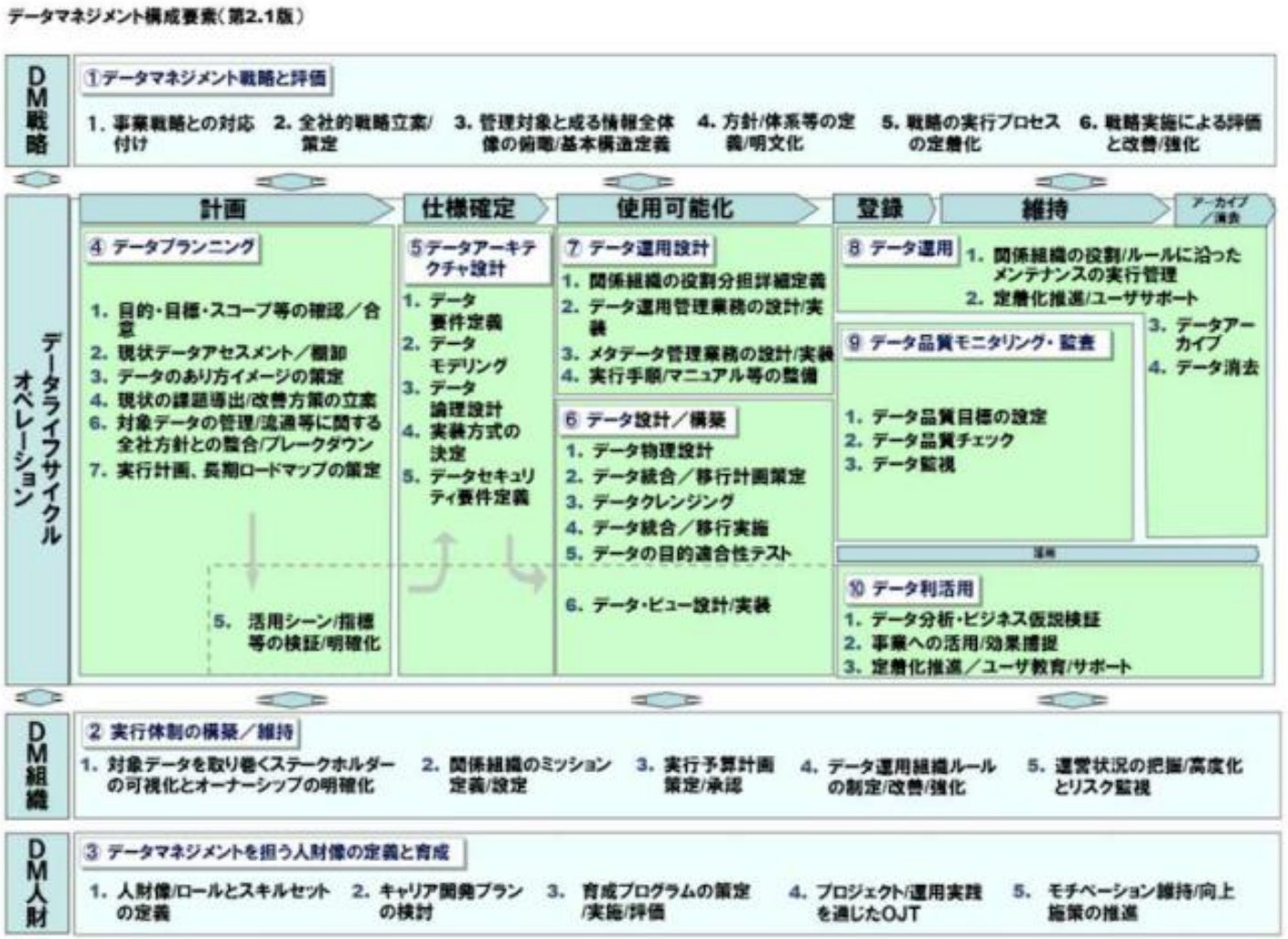
どれもKindle版で400円ほど。（kindle unlimitedにも対応）



DMBOK Guide（原典）

データマネジメントの方法論

データマネジメント構成要素（データマネジメント概説書より）



データマネジメントの方法論

データマネジメントあるある（データマネジメント概説書より）

データ構造は建物のようには見えない

データ構造は、建物のように誰の目にもみえるようには示されていない。 建物の場合であれば、間取りが実情と合わない、外壁にひびがあるなど、 何をすべきか合意も対処も容易である。しかし、データ構造は目に見えにくく、 問題を合意すること自体が困難である。これは、データの責任管理の問題につながる。 建物全体の管理は、設備管理の範疇だが、日々の活動の中で常に快適な 作業環境を維持する責任は利用者にある。 フロア内の什器や席の配置を決めているのも利用部門のことが多い。 データについても同じように情報システム部門と事業部門の責任分担がある。

全部やろうとするからいけない

情報は活動する上での血液として企業／組織をめぐるので、 データマネジメントに関する業務範囲や関連部門が広くなる。 また、対象となるデータは過去の蓄積から膨大にある。このため、後からデータを整理するのは、非常に困難である。

そのため、もう二度とやらなくてすむように、全部きれいにしてしまおうと考えるのだが、これがかえってうまくいかない理由である。 費用と時間と合意の全てについてほぼ確実にその量に負け、 結果的に断念せざるをえなくなる。 期待効果を明確にした部分的な導入を、真剣に考えなくてはいけない。

「構成要素」の全体を把握した上で
「スモールスタート」が必要。

データ分析基盤構築の方法論

著者：青木峰郎
出版社：SBクリエイティブ
発売日：2015/06/30

- クックパッド青木峰郎さんの「10年戦えるデータ分析入門」の11章を参照
  - 我が意を得たり、の心境
  - 知らないノウハウが多数あり、非常に勉強になった

データ分析基盤構築の方法論

著者：青木峰郎
出版社：SBクリエイティブ
発売日：2015/06/30

【参考】14章 「参考書籍」の一冊
書名：コーポレート・インフォメーション・ファクトリー
著者：W.H.インモン(他)
出版社：海文堂出版
発売日：1999年5月

【参考】ネットで参照できる青木さんの講演資料
Cookpad TechConf 2016 - DWHに必要なこと
http://www.slideshare.net/mineroaoki/cookpad-techconf-2016-dwh

講演ビデオ：CookpadTechConf2016 vol.11 - YouTube
https://www.youtube.com/watch?v=UYHGLTlIggE

AWS Summit Tokyo 2014での講演
Amazon Redshiftによるリアルタイム分析サービスの構築
http://www.slideshare.net/mineroaoki/at-aws-summit-tokyo-2014

講演ビデオ：Amazon Redshiftによるリアルタイム分析サービスの構築（AWS Summit Tokyo 2014 | TC-02）
https://www.youtube.com/watch?v=gPa85i55u74

「SQL中心アーキテクチャ」（出典：10年戦えるデータ分析入門）

- 次の3つの条件を満たした分析システム
  1. 必要なデータを1つのデータベースに集める
  2. データの加工には原則としてSQLだけを用いる
  3. データベース内を論理的にソースデータ層・DWH層・アプリケーション層の3層に区分する
  - 詳しくは、書籍をご覧ください

クラウドでデータ分析基盤構築

- ユースケースで構成が変わるが、、
- Redshiftは企業のDWHの中心になるはず

- 今回は「データマート」を作成していく
- ソフトウェア構成
  - ETLツール
  - スケジュール実行の仕組み
  - 可視化、分析
  - ドキュメント管理、メタデータ管理

ソフトウェア構成

- RDB - Redshift
- ETLツール - Embulk
- スケジュール実行の仕組み - Digdag
- 可視化、分析 - Tableau
- ドキュメント管理、メタデータ管理 - dmemo

今回用意できた範囲

【参考】クックパッド小室さんが開発されているOSS
データベースドキュメント管理システム dmemo のご案内
http://techlife.cookpad.com/entry/2016/08/08/103906

データベースドキュメント管理システム dmemoを立ち上げてみた。| Developers.IO
http://dev.classmethod.jp/tool/launch_dmemo/

データベースドキュメント管理システム dmemoを実際に触ってみた。| Develoers.IO
http://dev.classmethod.jp/tool/tried_dmemo/

【余談】メタデータ管理

メタデータ管理（リクルートさんの事例）
リクルートのビッグデータ活用基盤とデータ活用に向けた取組み
http://www.slideshare.net/recruitcojp/ss-58636898
http://www.slideshare.net/recruitcojp/ss-58636898/36

問い合わせ対応の工数削減に
大きな効果あり、とのこと（p.41）

リクルートのビッグデータ活用基盤とビッグデータ活用のためのメタデータ管理Webのご紹介
http://www.slideshare.net/recruitcojp/web-52520360

データ定義情報の管理とWebによる公開
http://www.slideshare.net/recruitcojp/20150216-datamanagement

Embulk/Digdagのご紹介

- どちらもTreasureData発のOSS
- Embulk
  - バルクデータローダー
- Digdag
  - ワークフローエンジン
- プラグイン対応で多数のプラグインが開発されている

※Digdagのアイコンはまだ無いようです

公式サイト
Embulk(最新版 0.8.13)
http://www.embulk.org/
Digdag(最新版 0.8.15)
http://www.digdag.io/

Embulk/Digdagに注目する理由

- TreasureData発のOSS
  - fluentdの成功
  - バッチ版のfluentdを期待
- ETLツールの代替
- テキストベースの設定ファイル
  - バージョン管理しやすい
- Embulkのguess（推測）機能
  - テーブル定義が無いケースで重宝
  - 次メジャーバージョンで機能アップ？

【参考】過去のブログもご覧ください

【Embulk】

[Embulk] Embulkについての個人的なまとめ
http://dev.classmethod.jp/tool/embulk-matome-01/

[Embulk] guess機能を試してみた【追記】あり
http://dev.classmethod.jp/tool/embulk-guess-01/

[Embulk] guess機能を試してみた Redshift編
http://dev.classmethod.jp/tool/embulk-guess-02/

[Embulk] タイムスタンプのカラムを追加する
http://dev.classmethod.jp/tool/embulk-guess-03/

[Embulk] タイムスタンプのカラムを追加する add_timeプラグイン編
http://dev.classmethod.jp/tool/embulk-guess-04/

[Embulk] guess機能を試してみた (テーブルが存在する場合)
http://dev.classmethod.jp/tool/embulk-guess-05/

【Digdag】

Embulk界隈で話題になっている分散ワークフローエンジン「DigDag」について調べてみた #digdag
http://dev.classmethod.jp/tool/embulk-digdag-01/

Treasure Data社のOSSワークフローエンジン『Digdag』を試してみた #digdag
http://dev.classmethod.jp/server-side/getting-started-digdag/

Digdag + EmbulkによるTSVファイルのS3→Redshiftロード #digdag
http://dev.classmethod.jp/cloud/aws/load-tsv-from-s3-to-redshift-using-embulk-digdag/

今回のチュートリアルに近い内容です

今回のチュートリアルの構成

- 意図
  - なるべくそのまま実案件で使えそうなチュートリアル
- 手順
  - データファイルをS3にアップ
  - ロード
  - 加工
  - テーブルチューニング
  - 定型バッチ化
  - 初回ロード、その後の差分ロード

今回用意できた範囲

ETLからELTへ

- ETLの派生形、ELT
  - E：Extract（抽出）
  - T：Transform（変形・加工）
  - L：Load（ロード）
- ELTでは、先にデータベースにロード、データベースのリソースを活用して変形・加工を行う
  - まずはRedshiftにデータを入れてしまい、Redshift上で変形・加工する
  - EmbulkのFilterプラグインの機能（レコードの加工）は使わない

サンプルデータ

- TPC-H（データベースのベンチマーク標準）
- SSB(スタースキーマベンチマーク)

【参考】

TPC-H

http://www.tpc.org/tpch/

→「スケールファクター」パラメータで任意のサイズのサンプルデータ作成

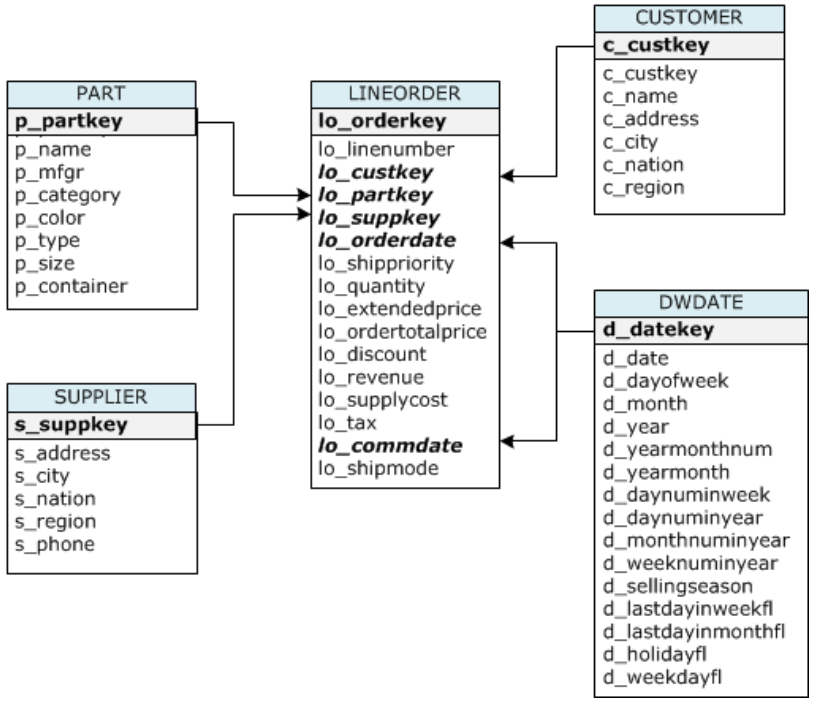
Star Schema Benchmark （原典の論文）

http://www.cs.umb.edu/~poneil/StarSchemaB.PDF

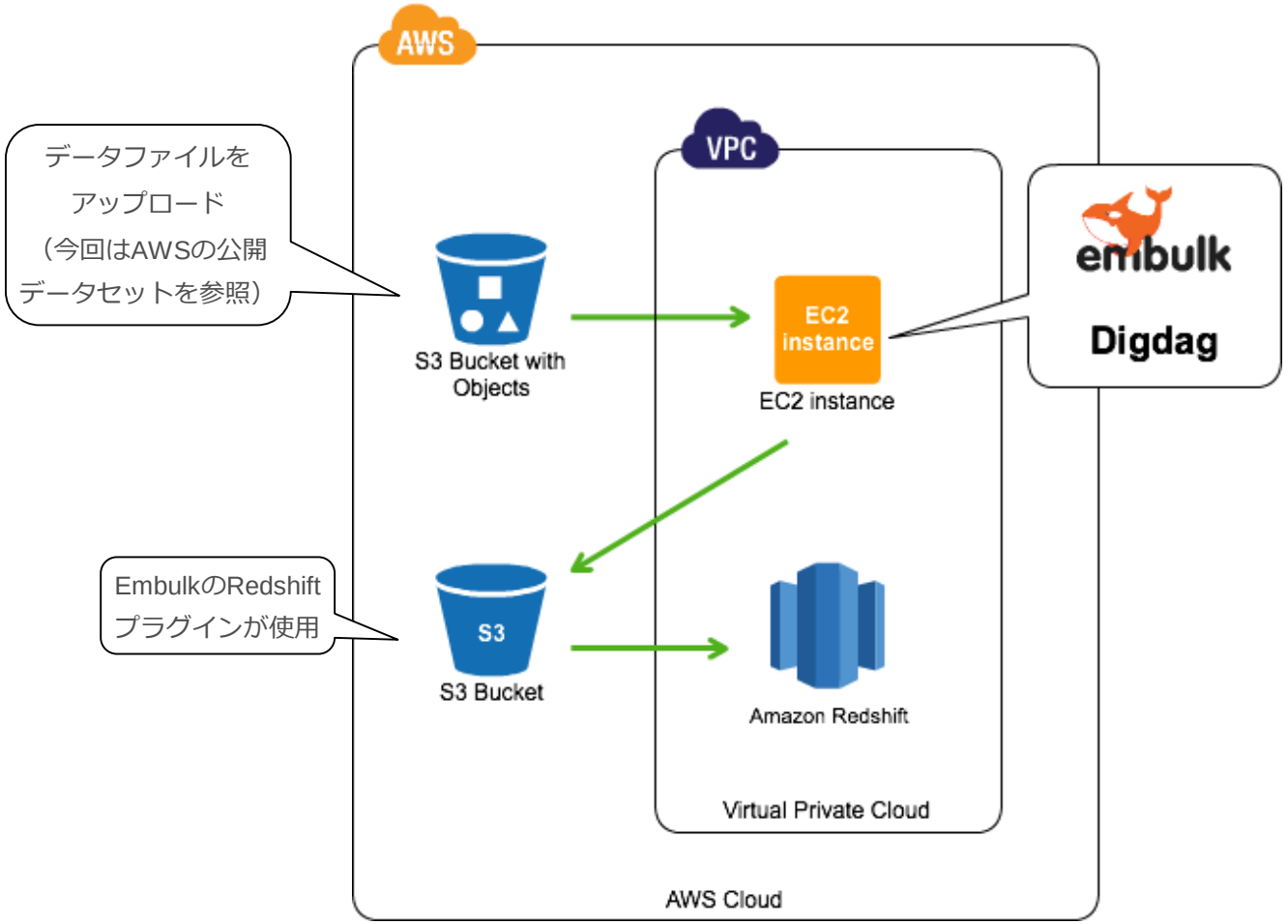
→TPC-Hのデータをスタースキーマに変形

→AWSの公開データセットが利用可能

SSBのテーブル構成



システム構成


AWS
VPC
データファイルを
アップロード
（今回はAWSの公開
データセットを参照）
S3 Bucket with
Objects
EC2
instance
EC2 instance
embulk
Digdag
EmbulkのRedshift
プラグインが使用
S3
S3 Bucket
Amazon Redshift
Virtual Private Cloud
AWS Cloud

環境構築

- CloudFormationを使い、Redshiftクラスタを一発起動する
  - 今回使ったテンプレートは、後日ブログで紹介予定
  - チュートリアルの詳しい手順もブログに掲載します

【参考】

最短時間でRedshiftクラスターを立てる方法 | Developers.IO

http://dev.classmethod.jp/server-side/db/cloudformation_for_redshift_01/

Embulkインストール

◇CloudFormationで作成されたEC2インスタンスにsshでログインし、下記コマンドを実行

```
$ curl --create-dirs -o ~/.embulk/bin/embulk -L "http://dl.embulk.org/embulk-latest.jar"
$ chmod +x ~/.embulk/bin/embulk
$ echo 'export PATH="$HOME/.embulk/bin:$PATH"' >> ~/.bashrc
$ source ~/.bashrc
```

◇続けて、今回の手順で使うプラグインをインストール

```
$ embulk gem install embulk-input-s3
$ embulk gem install embulk-output-redshift
```

Embulkのサンプル

```
$ embulk example ./try1
2016-09-25 14:25:50.503 +0000: Embulk v0.8.13
Creating ./try1 directory...
  Creating ./try1/
  Creating ./try1/csv/
  Creating ./try1/csv/sample_01.csv.gz
  Creating ./try1/seed.yml

Run following subcommands to try embulk:

  1. embulk guess ./try1/seed.yml -o config.yml
  2. embulk preview config.yml
  3. embulk run config.yml
```

◇guess機能を使うための設定

```
$ cat seed.yml
in:
  type: file
  path_prefix: "/home/ec2-user/example/try1/csv/sample_"
out:
  type: stdout
```

左記の1～3の手順でサンプルを実行可能

Embulkのサンプル

◇guessコマンドで生成された設定ファイル

```
$ cat config.yml
in:
  type: file
  path_prefix: /home/ec2-user/example/try1/csv/sample_
  decoders:
  - {type: gzip}
  parser:
    charset: UTF-8
    newline: CRLF
    type: csv
    delimiter: ','
    quote: '"'
    escape: '"'
    null_string: 'NULL'
    trim_if_not_quoted: false
    skip_header_lines: 1
    allow_extra_columns: false
    allow_optional_columns: false
    columns:
    - {name: id, type: long}
    - {name: account, type: long}
    - {name: time, type: timestamp, format: '%Y-%m-%d %H:%M:%S'}
    - {name: purchase, type: timestamp, format: '%Y%m%d'}
    - {name: comment, type: string}
out: {type: stdout}
```

Digdagインストール

【参考】Oracle JDKを一発でダウンロードするwgetのオプション
http://qiita.com/ajis_ka/items/b3d9030e2aaf199a20b7

◇Digdagの実行には、JDK8u72以上が必要となります
（最近のAmazon Linuxだと、インストールが必要）

```
$ curl -o ~/bin/digdag --create-dirs -L "https://dl.digdag.io/digdag-latest"
$ chmod +x ~/bin/digdag
$ echo 'export PATH="$HOME/bin:$PATH"' >> ~/.bashrc
```

Digdagのサンプル

◇サンプルの実行手順

```
$ digdag init mydag
$ cd mydag
$ digdag run mydag.dig
```

◇実行状況

```
$ digdag init mydag
2016-09-25 14:32:48 +0000: Digdag v0.8.13
  Creating mydag/.gitignore
  Creating mydag/query.sql
  Creating mydag/mydag.dig
Done. Type `cd mydag` and then `digdag run mydag.dig` to
run the workflow. Enjoy!
```

◇initコマンドで生成された設定ファイル

```
$ cat mydag/mydag.dig
timezone: UTC

+query:
  td>: query.sql
  database: sample_datasets
```

tdオペレータは、TreasureDataへのクエリなので、環境が無い場合はrunコマンドでエラーになります。

データロード（Embulk）

◇ guess-redshift-output.ymlの内容（lineorderテーブルをロードする例）

```
in:
  type: s3
  bucket: awssampledbapnortheast1
  path_prefix: ssbgz/lineorder0001
  endpoint: s3-ap-northeast-1.amazonaws.com
  access_key_id: xxxxxx
  secret_access_key: xxxxxx
out:
  type: redshift
  host: xxxx-redshiftcluster.ap-northeast-1.redshift.amazonaws.com
  user: xxxx
  password: xxxx
  database: defaultdb
  table: lineorder
  access_key_id: xxxxxx
  secret_access_key: xxxxxx
  iam_user_name: xxxx
  s3_bucket: cm-csk-dev-20160914-1100
  s3_key_prefix: temp/redshift
  mode: insert
  batch_size: 1024000000
```

データロード（Digdag）

◇今回のフォルダ構成

```
ssbgz
 ├── ssbgz.dig
 └── embulk
     └── guess-redshift-output.yml
```

◇ssbgz.digの内容

```
timezone: UTC
+guess:
  sh>: embulk guess embulk/guess-redshift-output.yml -o embulk/config-redshift-output.yml
+load:
  embulk>: embulk/config-redshift-output.yml
```

◇Digdagを実行

```
$ digdag run ssbgz.dig
```

成功すれば、最後に下記のようなメッセージが表示される。

```
Success. Task state is saved at .digdag/status/20160914T000000+0000 directory.
  * Use --session <daily | hourly | "yyyy-MM-dd[ HH:mm:ss]"> to not reuse the last session time.
  * Use --rerun, --start +NAME, or --goal +NAME argument to rerun skipped tasks.
```

列長調査

Embulk	説明	Redshift
boolean	真偽値	BOOLEAN
long	整数型	BIGINT
timestamp	タイムスタンプ	TIMESTAMP
double	浮動小数点	DOUBLE PRECISION
string	文字列	VARCHAR

EmbulkでTimestamp型のロードに時間がかかる問題があるので注意！

【参考】EmbulkのTimestampが遅すぎて死にそうなのでFilterを作ったよ

http://qiita.com/arenahito/items/5ea63bb0327a2f9b9e95

入力データのstring型は、Redshift側でVARCHAR(65535)に設定される
Redshiftで大きすぎるVARCHAR型を修正

【参考】

Amazon Redshift: テーブル作成時の列データ型や桁数を検討する為に入力データの最大値・最大桁数を求める | Developers.IO

http://dev.classmethod.jp/cloud/aws/amazon-redshift-sqlquery-get-maxvalue-or-maxlength/

列名、列長を変更してCREATE TABLE

```
CREATE TABLE lineorder (
lo_orderkey BIGINT
, lo_linenumber BIGINT
, lo_custkey BIGINT
, lo_partkey BIGINT
, lo_suppkey BIGINT
, lo_orderdate BIGINT
, lo_orderpriority VARCHAR(44)
, lo_shippriority BIGINT
, lo_quantity BIGINT
, lo_extendedprice BIGINT
, lo_ordertotalprice BIGINT
, lo_discount BIGINT
, lo_revenue BIGINT
, lo_supplycost BIGINT
, lo_tax BIGINT
, lo_commitdate BIGINT
, lo_shipmode VARCHAR(28)
);
```

COPY、自動圧縮

◇Redshiftのcopyコマンドを使い、データを再ロード。自動圧縮オプションを設定することで、列の圧縮が有効に。（psql上で実行）

```
copy lineorder
from's3://awssampledbapnortheast1/ssbgz/lineorder0001_part_00.gz'
credentials 'aws_access_key_id=xxxxxx;aws_secret_access_key=xxxxxx'
gzip
COMPUPDATE ON
;
```

ソートキー追加（要：テーブル再作成）

- ソートキー選択のベストプラクティス
  - 最新のデータが最も頻繁にクエリ処理される場合は、タイムスタンプ列をソートキーの主要な列として指定します。
  - 1 つの列に対して範囲フィルタリングまたは等価性フィルタリングを頻繁に実行する場合は、その列をソートキーとして指定します。
  - （ディメンション）テーブルを頻繁に結合する場合は、結合列をソートキーとして指定します。

【参考】

ステップ 3: ソートキーを選択する

http://docs.aws.amazon.com/ja_jp/redshift/latest/dg/tutorial-tuning-tables-sort-keys.html

分散キーについて（シェアードナッシング）

AWS Black Belt Online Seminar Amazon Redshift

http://www.slideshare.net/AmazonWebServicesJapan/aws-black-belt-online-seminar-amazon-redshift

MPPとシェアードナッシングがスケールアウトの鍵

- **MPP：Massive Parallel Processing**
  - １つのタスクを複数のノードで分散して実行する仕組み
  - Redshiftではリーダーノードがタスクをコンピュートノードに分散して実行する
  - ノードを追加する（スケールアウト）でパフォーマンス向上可能
- **シェアードナッシング**
  - ディスクをノードで共有**しない**構成
  - ディスクを共有するとノード数が増えた時にボトルネックになるため、それを回避
  - ノードとディスクがセットで増えていく

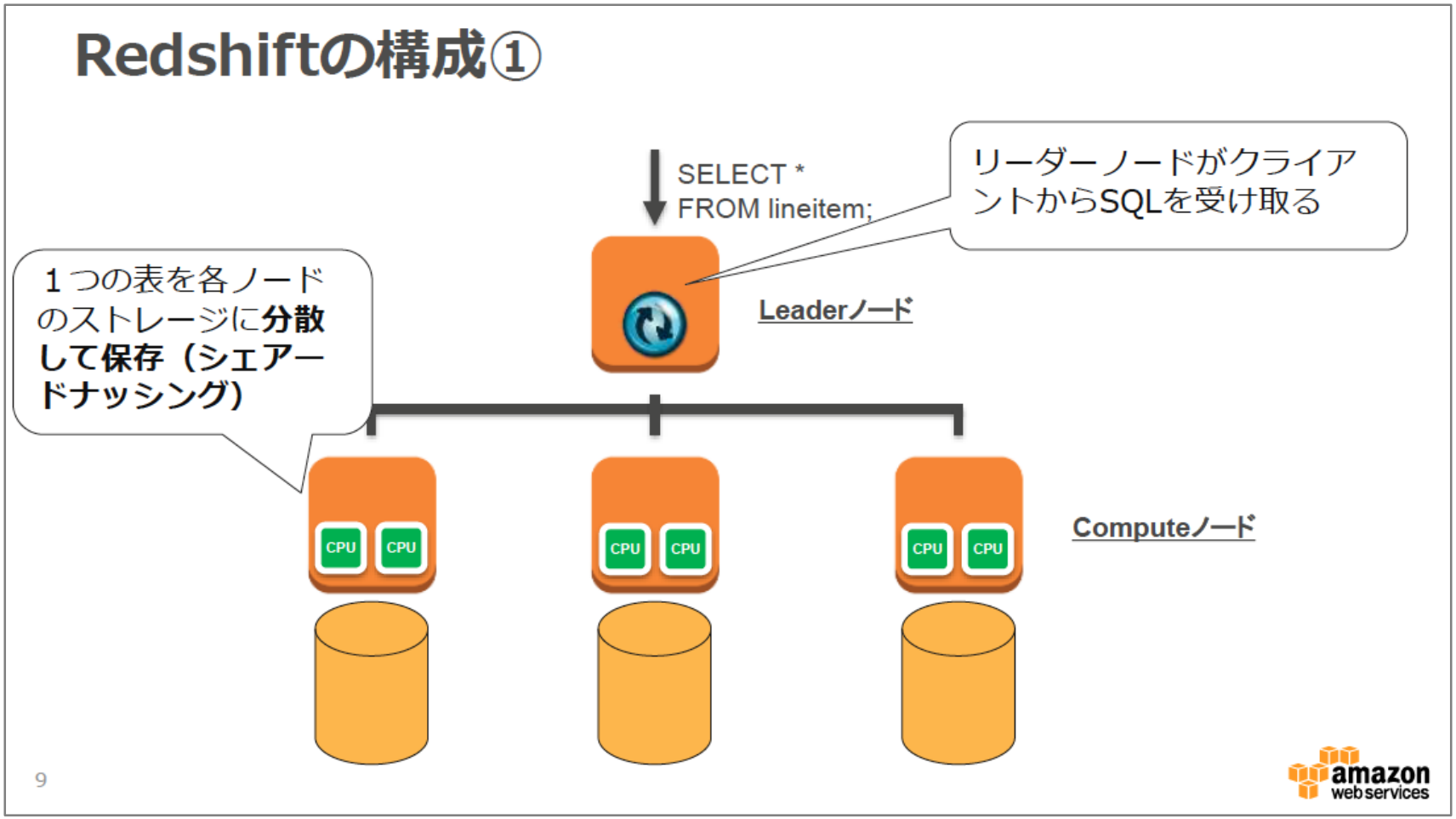
8

amazon web services

分散キーについて（シェアードナッシング）

AWS Black Belt Online Seminar Amazon Redshift

http://www.slideshare.net/AmazonWebServicesJapan/aws-black-belt-online-seminar-amazon-redshift

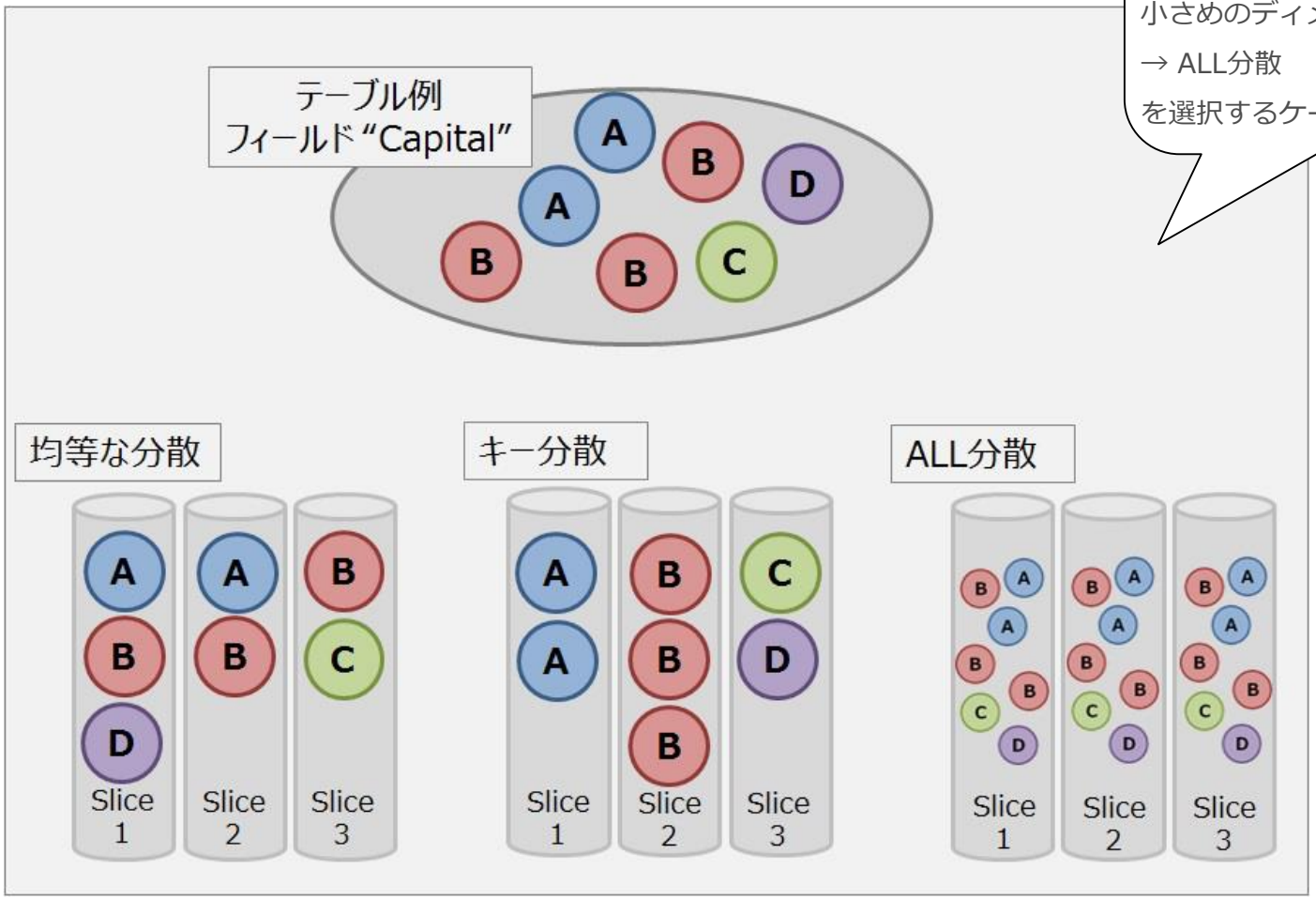


分散キーの種類

DACエンジニアブログ：アドテクゑびす界
Amazon Redshiftのパフォーマンスチューニング #1 アーキテクチャ概要
http://yebisupress.dac.co.jp/2014/12/09/amazon-redshift-のパフォーマンスチューニング1/
「分散スタイル図解」

ファクトテーブル
→ キー分散

小さめのディメンションテーブル
→ ALL分散
を選択するケースが多い



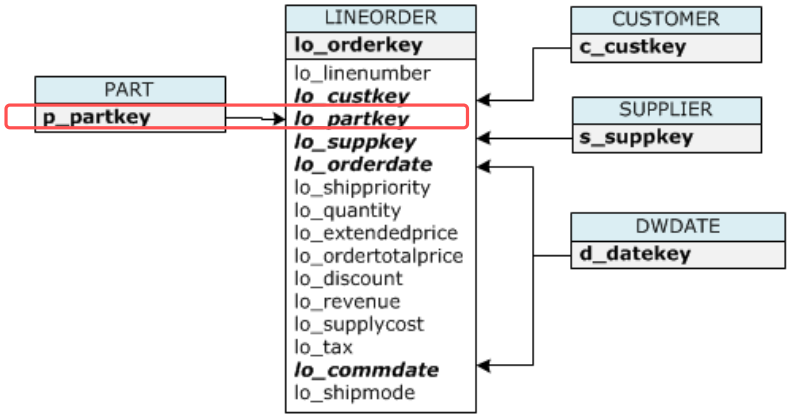
分散キー追加（要：テーブル再作成）

分散スタイルを選択するには

http://docs.aws.amazon.com/ja_jp/redshift/latest/dg/tutorial-tuning-tables-distribution.html

2. ファクトテーブルと 1 つのディメンションテーブルをそれらの共通の列に基づいて分散させます。

以下の図に示しているのは、SSB スキーマ内のファクトテーブル、LINEORDER、ディメンションテーブル間の関係です。



各テーブルに使用できる分散キーは 1 つのみです。つまり、スキーマ内の 1 組のテーブルのみをそれらの共通の列に基づいてコロケーションできます。中央のファクトテーブルをまず選択するのは明らかです。その組となる 2 つ目のテーブルには、ファクトテーブルを一般的に結合する最大サイズのディメンションを選択します。この設計では、LINEORDER がファクトテーブルであり、PART が最大サイズのディメンションです。PART はそのプライマリキー `p_partkey` に基づいて LINEORDER に結合されます。

LINEORDER の分散キーとして `lo_partkey`、PART の分散キーとして `p_partkey` を指定します。これにより、結合キーの一致する値がデータのロード時に同じスライスにコロケーションされるようになります。

最終的なCREATE TABLE文

```
◇ファクトテーブルの例
CREATE TABLE lineorder (
lo_orderkey BIGINT
, lo_linenumber BIGINT
, lo_custkey BIGINT
, lo_partkey BIGINT
, lo_suppkey BIGINT
, lo_orderdate BIGINT
, lo_orderpriority VARCHAR(44)
, lo_shippriority BIGINT
, lo_quantity BIGINT
, lo_extendedprice BIGINT
, lo_ordertotalprice BIGINT
, lo_discount BIGINT
, lo_revenue BIGINT
, lo_supplycost BIGINT
, lo_tax BIGINT
, lo_commitdate BIGINT
, lo_shipmode VARCHAR(28)
)
DISTSTYLE KEY
DISTKEY (lo_partkey)
SORTKEY(lo_orderdate)
;
```

Redshift Try！（キャンペーンのお知らせ）

最後にお知らせです

Amazon Redshiftの利用料が最大$2,000無料に！導入検証支援プログラム「Redshift Try!」を開始します
http://dev.classmethod.jp/cloud/aws/redshift-try/

Amazon Redshiftの事例紹介セミナー
東京：2016年9月27日(火)15:00～17:30
http://dev.classmethod.jp/news/bigdata-20160915-27/