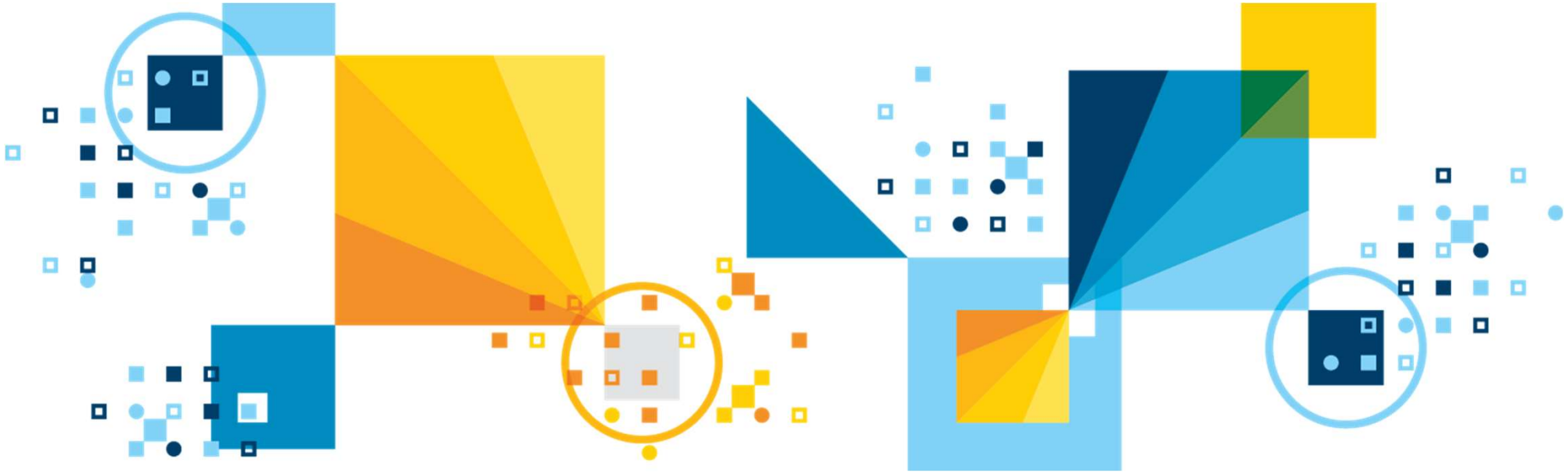
Db2 Warehouse

パフォーマンス・チューニングガイド



目次

- Db2 Warehouseの概要
- データの分散について
  - 複数DBパーティション上での表データの分散
  - 表の分散キーについて
  - 表の圧縮について
  - 索引の利用
  - データ・スキッピング
  - ヒープ利用について
- 稼働状況の確認
  - 稼働状況の確認方法（リアルタイム／履歴）
  - ワークロード管理の設定、確認
  - （参考）カラム・オーガナイズ表専用のモニター項目

目次

- アクセスパスの確認
  - Visual Explainによる確認
  - db2exfmtによる確認
  - 選択されるJoin方法と特徴について
- チューニングのための表変更方法
  - External Tableを利用したデータの格納／抽出
  - 方法①：CTAS利用
  - 方法②：カーソルロード
  - 方法③：並列INSERT + NOT LOGGED

注意事項

本資料掲載事項は、ある特定の環境・使用状況においての正確性がIBMによって確認されていますが、すべての環境において同様の結果が得られる保証はありません。これらの技術を自身の環境に適用する際には、自己の責任において十分な検証と確認を実施いただくことをお奨めいたします。

本ガイド資料は Db2 Warehouse V1.11.1をベースに検証を実施した内容を元に作成しています。

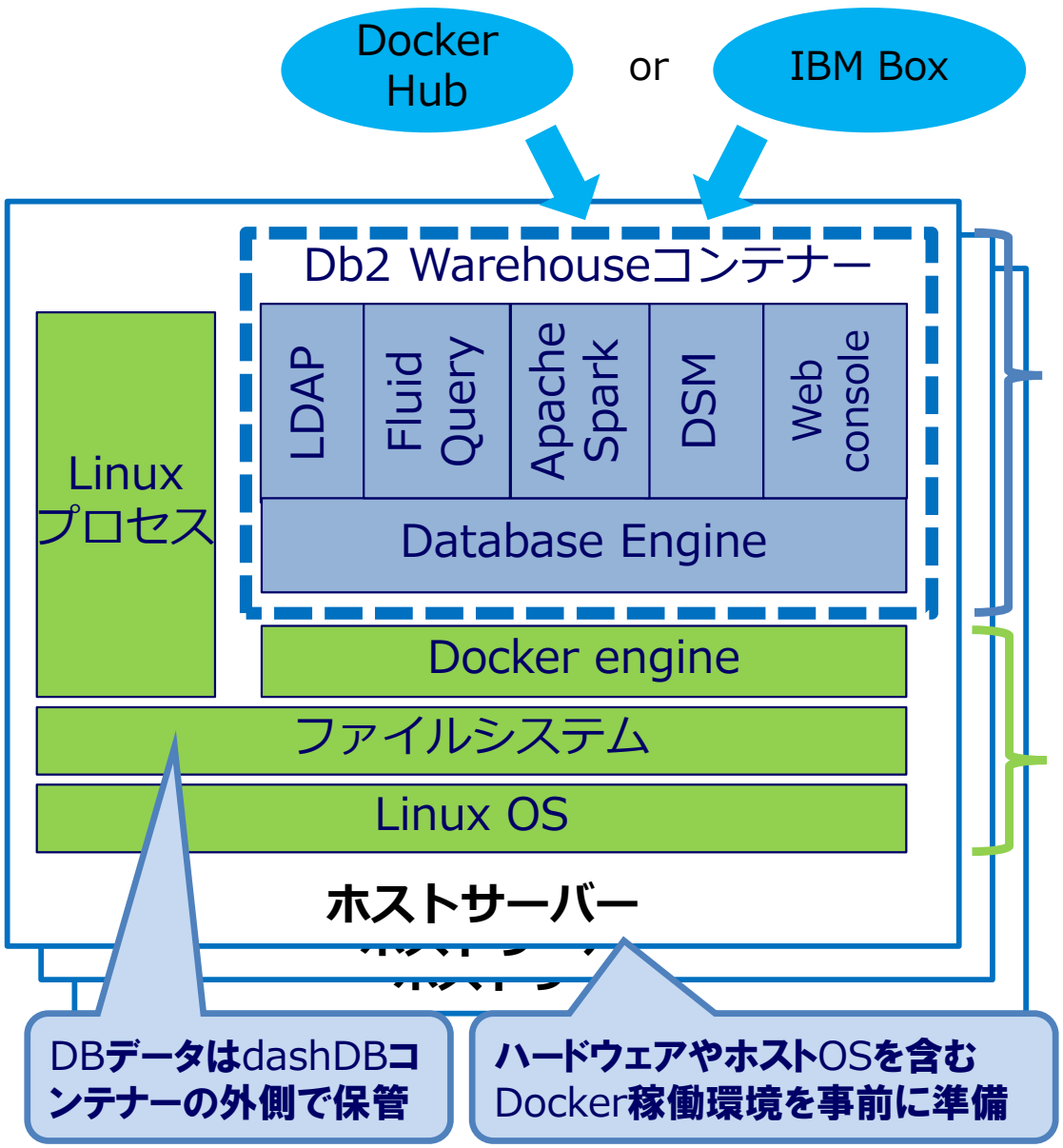
dashDB Local V1.0との変更点は補足で記載していますが、必要に応じて下記マニュアル参照の上導入ください。

Db2 Warehouse Knowledge Center:
https://www.ibm.com/support/knowledgecenter/SS6NHC/com.ibm.swg.im.dashdb.kc.doc/welcome.html

Db2 Warehouse概要

Db2 Warehouseアーキテクチャ概要

Docker Hub上のリポジトリーから取得
もしくはIBMのBoxからダウンロード



Db2 Warehouseの稼働環境

- パブリッククラウド
- オンプレミス (Docker Hubへの接続は必要)

Db2 Warehouseが提供する部分

- Databaseエンジン
- ユーザー管理
- データ投入
- Sparkの稼働環境
- オブジェクト管理
- 負荷モニター

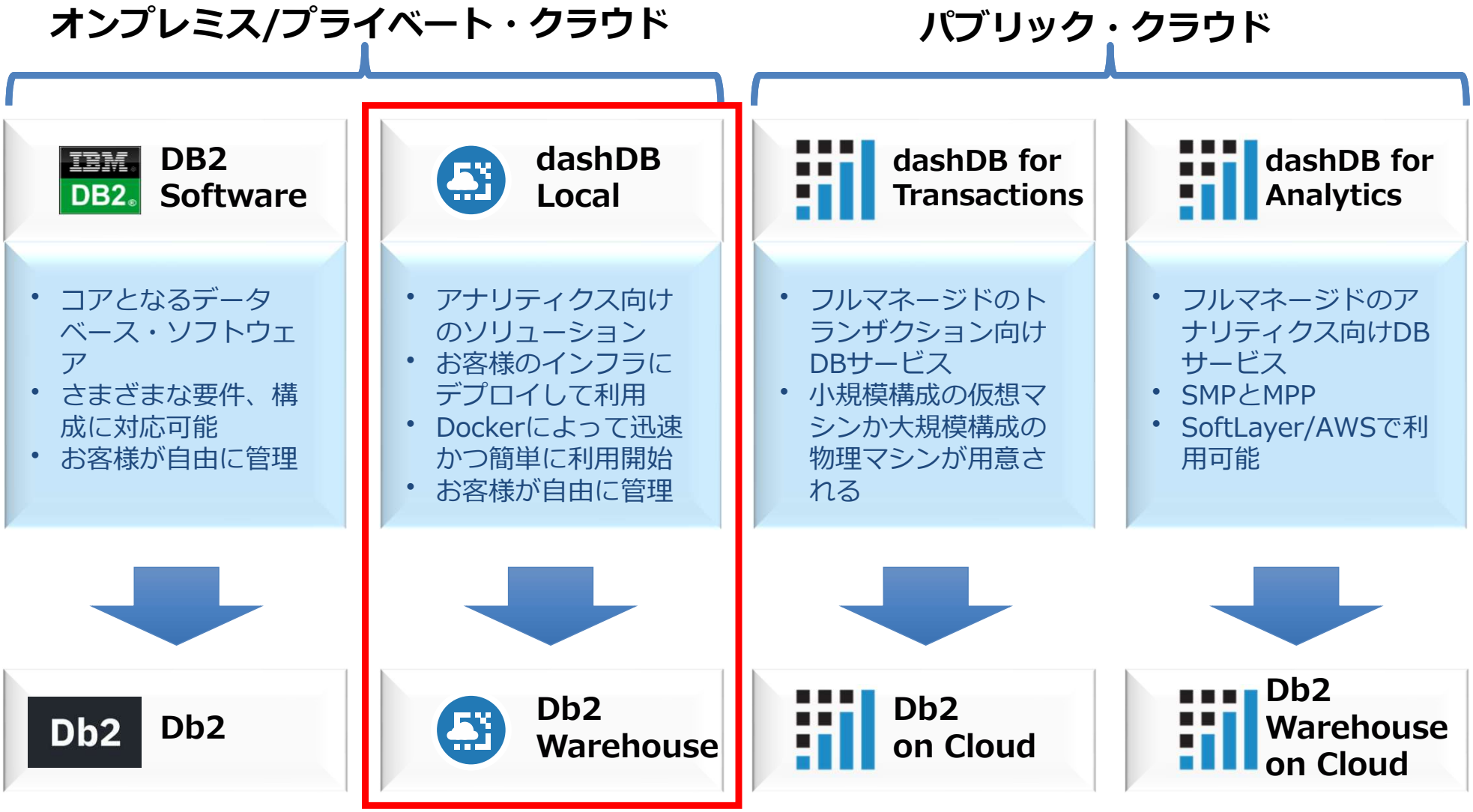
事前に用意する部分

- Dockerエンジン
- ホストサーバー（H/W、OS）
- ファイルシステム
  (MPPの場合は共用ファイルシステム)

ホストOS(ノード)の台数

シングルノード(SMP)またはマルチノード(MPP)
- SMPの場合：1ノード
- MPPの場合：3ノードから24ノード or 60ノード

dashDB LocalからDb2 Warehouseへ名称変更



IBMのデータベースソリューションをDb2ブランドで統一

Db2 Warehouseのベースはカラム・オーガナイズ表

汎用RDBMSでの行ストア（行オーガナイズ表）

ID	商品名	価格	サイズ	発売日
1001	商品A	1000	L	2017-01-20
1002	商品B	2000	XS	2015-07-07
1003	商品C	1500	M	2016-10-31
1004	商品D	3000	S	2017-04-11

- １ブロックにすべての列のデータを格納
- クエリーの結果導出に不要な列も読み込む必要がある。

以下の表では利用できない

パーティション表、MDC表、テンポラル表、型付き表

カラム・オーガナイズ表

ID
1001
1002
1003
1004

商品名
商品A
商品B
商品C
商品D

価格
1000
2000
1500
3000

サイズ
L
XS
M
S

発売日
2017-01-20
2015-07-07
2016-10-31
2017-04-11

- 列データ毎に別ブロックに格納
- 参照処理における不要列データの読み込みが無くなり、必要なデータのみ効率良くメモリーへロード
- 同一列内には特定のデータが繰り返し現れることが多く、圧縮効率が良い

```
CREATE TABLE  MYTABLE ( ID INT, NAME CHAR(10)･･･ )
ORGANIZE BY COLUMN
```

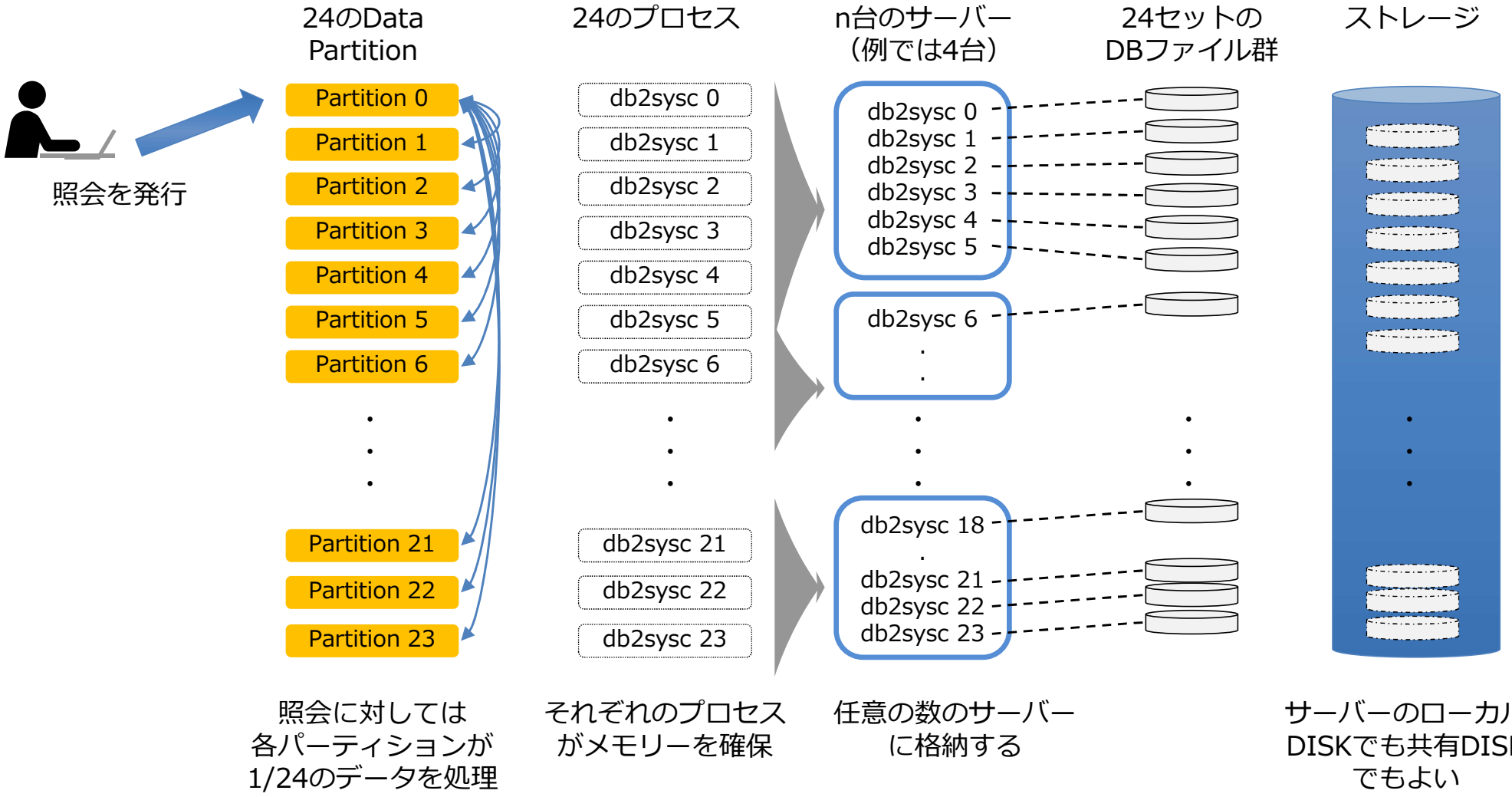
または

```
UPDATE DB CFG FOR MYDB USING DFT_TABLE_ORG COLUMN
```

指定なしのデフォルトでもCOLUMNとなる

さらに、Db2 Warehouseのベースは複数DBパーティション構成

Db2 MPP環境はn個のDB partitionで構成され、それぞれのDB partitionが1/nのデータを分散して保持する



データの分散について

複数DBパーティション上での表データの分散

表作成時に指定する分散キーによって、レコードがどのDBパーティションに格納されるのかが決められる

```
CREATE TABLE CUSTOMER(
 cust_id INT,
 name VARCHAR(80),
 gender  CHAR(5))
DISTRIBUTE BY HASH(cust_id);
```

表作成時の分散キーの指定が重要

DISTRIBUTE BY RANDOM ではDb2が自動的に分散キー設定

CUSTOMER

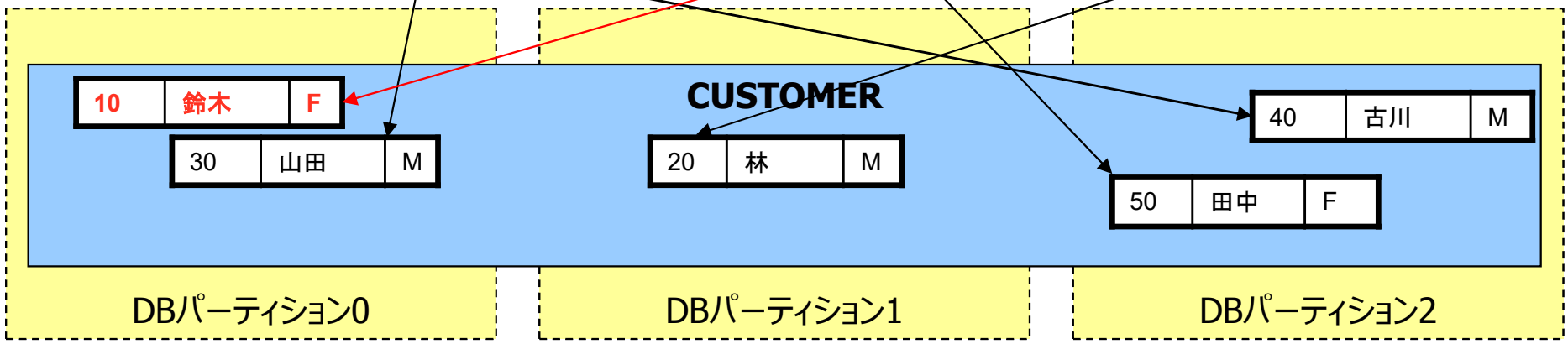
cust_id	name	gender
10	鈴木	F
20	林	M
30	山田	M
40	古川	M
50	田中	F

分散キーの値から得られるハッシュ値と分散マップによって対象のパーティションが決定する

Hash(cust_id)

分散マップ

0	1	2	3	4	5	6	7	…	32768
0	1	2	0	1	2	0	1	2	0

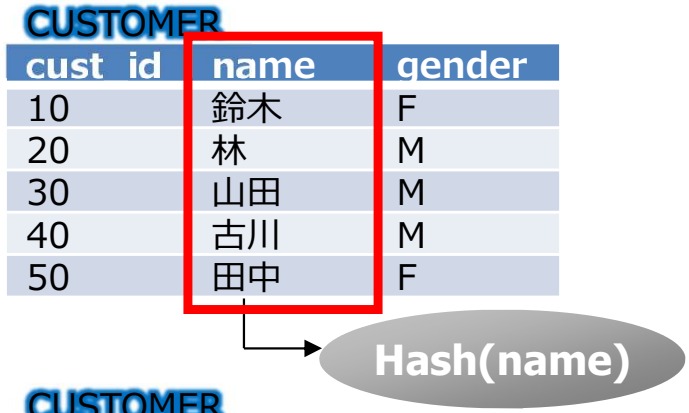


表作成時の分散キー指定方法（その１）

分散キーは性能を考慮し明示的に指定する。（Db2にお任せすることもできる）

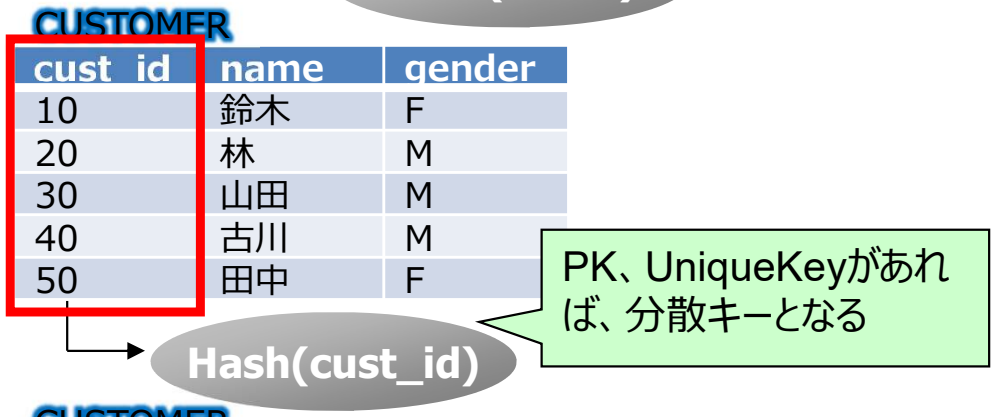
①分散キーをHASH指定する方法

```
CREATE TABLE CUSTOMER(
 cust_id INT,
 name VARCHAR(80),
 gender  CHAR(5))
DISTRIBUTE BY HASH(name);
```



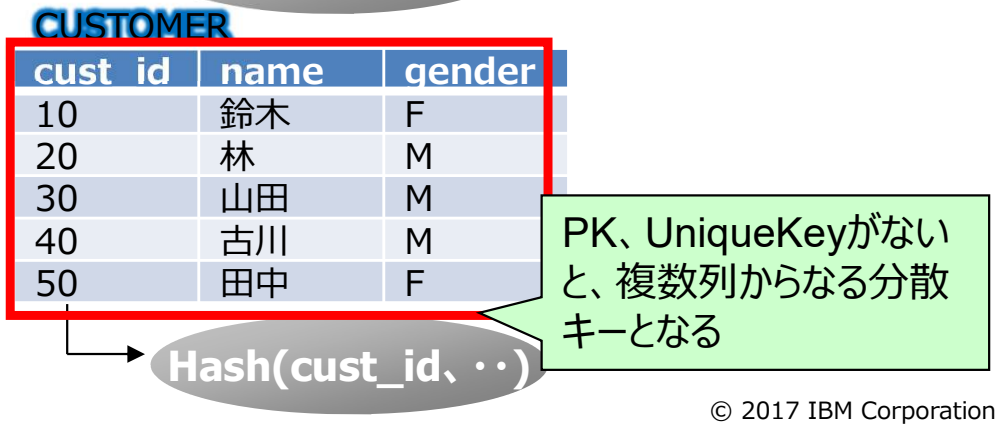
②-1分散キーを指定しなかった場合(PKあり)

```
CREATE TABLE CUSTOMER(
 cust_id INT PRIMARY KEY,
 name VARCHAR(80),
 gender  CHAR(5))
;
```



②-2分散キーを指定しなかった場合(PKなし)

```
CREATE TABLE CUSTOMER(
 cust_id INT,
 name VARCHAR(80),
 gender  CHAR(5))
;
```



表作成時の分散キー指定方法（その２）

分散キーは性能を考慮し明示的に指定する。（Db2にお任せすることもできる）

③-1分散キーをRAMDOM指定する方法(PKあり)

```
CREATE TABLE CUSTOMER(
 cust_id INT PRIMARY KEY,
 name VARCHAR(80),
 gender  CHAR(5))
DISTRIBUTE BY RANDOM;
```

CUSTOMER

cust_id	name	gender
10	鈴木	F
20	林	M
30	山田	M
40	古川	M
50	田中	F

DISTRIBUTE BY RANDOM ではDb2が自動的に分散キー設定

Hash(name)

③-2分散キーをRAMDOM指定する方法(PKなし)

```
CREATE TABLE CUSTOMER(
 cust_id INT,
 name VARCHAR(80),
 gender  CHAR(5))
DISTRIBUTE BY RANDOM;
```

CUSTOMER

RANDOM_DISTRIBUTION_KEY	cust_id	name	gender
2043504668	10	鈴木	F
314294847	20	林	M
610592156	30	山田	M
335981432	40	古川	M
96519872	50	田中	F

Hash(RANDOM_・・・)

隠し列(IMPLICITLY HIDDEN)として
INTEGERの分散キー用の列が追加される

どの列を分散キーとすべきか

分散キーを適切に選択することで、パーティション間に均一に負荷が分散され、最適なパフォーマンスを得ることができる

★分散キーの候補となる列

- **カーディナリティ（値のばらつき）の高い列** → 各パーティションへ格納される行の均一化
  - ○： 取引番号、顧客番号など
  - ×： 有効フラグ、性別
- **特定の値にアクセスが集中しないような列** → 各区分への均等なデータアクセスの発生
  - ○： 取引番号、顧客番号など
  - ×： 取引日付など
- **結合列として使われる列** → 結合処理を最も効率よく行うことができる

★分散キーの選択時の考慮点

- **多くの列を指定しない** → ハッシングのコスト軽減のため
- **プライマリー・キー、ユニーク索引には、必ず分散キーを含める** → 分散キー選択の際の制約
- **LOB、LONG、XML列は使用不可** → 分散キー選択の際の制約
- **分散キーの定義変更は不可** → 分散キー選択の際の制約

分散状況の確認（その１）

指定した分散キーで、データが均一に分散しているか確認することができる

①表単位で各DBパーティションのレコード件数を確認する方法
DBパーティション番号を戻す DBPARTITIONNUM(列名) 関数を利用し、DBパーティション毎の表件数を求める。

```
select DBPARTITIONNUM(SALES_YEAR) as DBP_NUM, count(*) as CNT
from GOSALES.SALES_TARGET
group by DBPARTITIONNUM(SALES_YEAR)
order by DBPARTITIONNUM(SALES_YEAR);
```

```
DBP_NUM     CNT
----------- --------------------
          1                11246.
          2                 8439.
          3                 8673.
          4                14070.
          5                11141.
          6                 9708.
...
         17                12355.
         18                14449.
         19                 7641.
         20                10675.
         21                 9340.
         22                 5172.
         23                10795.

  23 record(s) selected.
```

DBPARTITIONNUM(列名)の列に関しては表のどの列でも良く、同じ結果となる

表の各DBパーティションの件数が出力される。

データが均一に分散されていれば、最適な性能が望める

分散状況の確認（その２）

指定した分散キーで、データが均一に分散しているか確認することができる

②表単位で分散状況（MAX、MIN、AVG）を確認する方法
DBパーティション番号を戻す DBPARTITIONNUM(列名）関数を利用し、MAX、MIN、AVGの件数を求める。

```
with ROW_CNT AS (
    select DBPARTITIONNUM(SALES_YEAR) as DBP_NUM, count(*) as CNT
    from GOSALES.SALES_TARGET
    group by DBPARTITIONNUM(SALES_YEAR)
)
select MAX(CNT) as MAX_CNT,MIN(CNT) as MIN_CNT,AVG(CNT) as AVG_CNT
from ROW_CNT ;
```

WITH句にて、各DBパーティションの件数を求めている

```
MAX_CNT                     MIN_CNT                     AVG_CNT
--------------------------- --------------------------- ---------------------------
                      15088.                      5172.                      10157.

  1 record(s) selected.
```

表のデータの偏りの傾向がわかる。

MAXとMINがAVGに近ければ、最適な性能が望める

分散状況の確認（その３）

指定した分散キーで、データが均一に分散しているか確認することができる

③表スペース単位で分散状況を確認する方法
表スペース利用状況を戻す MON_TBSP_UTILIZATION管理ビューを利用し、DBパーティション毎の表スペース状況を調べる。

```
select substr(TBSP_NAME,1,16) as TBSP_NAME,MEMBER,
   TBSP_TOTAL_SIZE_KB,TBSP_USABLE_SIZE_KB,TBSP_UTILIZATION_PERCENT
from sysibmadm.MON_TBSP_UTILIZATION
order by TBSP_NAME,MEMBER ;
```

```
TBSP_NAME        MEMBER TBSP_TOTAL_SIZE_KB   TBSP_USABLE_SIZE_KB  TBSP_UTILIZATION_PERCENT
---------------- ------ -------------------- -------------------- ------------------------
DSMSPACE              0               393216               393088                    95.37
IDAX_USERTEMPSPA      1                   32                   32                   100.00
IDAX_USERTEMPSPA      2                   32                   32                   100.00
IDAX_USERTEMPSPA      3                   32                   32                   100.00
IDAX_USERTEMPSPA      4                   32                   32                   100.00
IDAX_USERTEMPSPA      5                   32                   32                   100.00
IDAX_USERTEMPSPA      6                   32                   32                   100.00
IDAX_USERTEMPSPA      7                   32                   32                   100.00
IDAX_USERTEMPSPA      8                   32                   32                   100.00
...
```

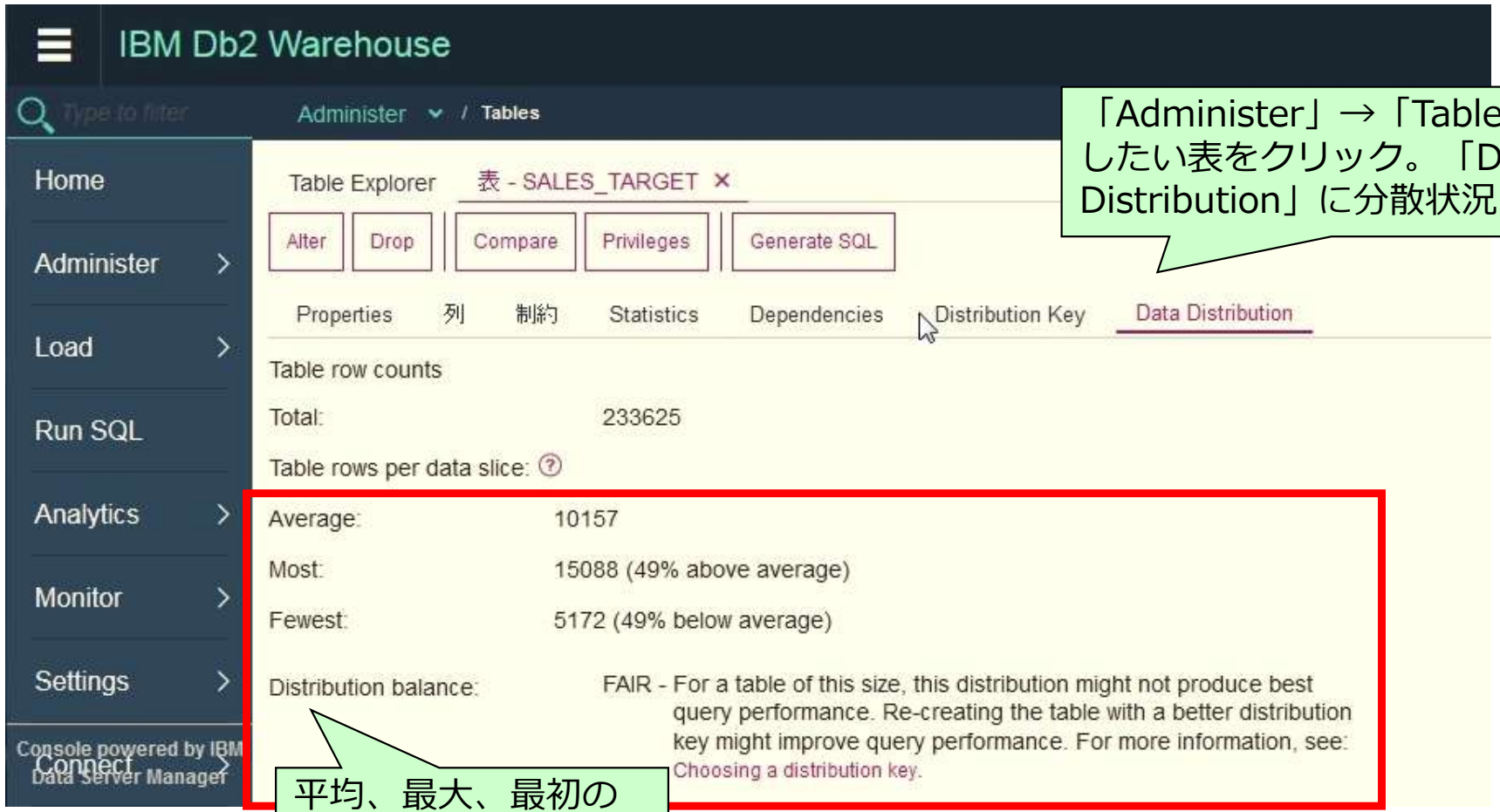
DBパーティション番号

表スペース毎にDBパーティションのサイズを出力

分散状況の確認（その４）

指定した分散キーで、データが均一に分散しているか確認することができる

④表単位でWebコンソールから分散状況を確認する方法
Webコンソールで表単位の分散状況を確認できる。平均、最大、最小のサイズと、その評価が確認できる。



分散状況の確認（その５）

指定した分散キーで、処理が均一に分散しているか確認することができる

⑤Webコンソールからトランザクションの分散状況を確認する方法
Webコンソールでトランザクションが各DBパーティションでどのように処理されているかを確認できる。CPU、処理量、アクセス行数などが確認できる。

データベース・パーティション	CPU 時間 (%)	コミット数/分		取り行数/戻された行数	読み取り行数/分	変更行数/分
0	1.64%	346.55		2.32	125,446.25	37,648.27
1	0.58%	0.00		--	44,561.12	2,654.14
2	0.55%	0.00	0.00	--	39,705.52	2,648.56
3	0.55%	0.00	0.00	--	39,979.00	2,648.90
4	0.54%	0.00	0.00	--	40,721.22	2,662.92
5	0.54%	0.00	0.00	--	39,528.09	2,657.47
6	0.56%	0.00	0.00	--	40,139.43	2,659.63
7	0.56%	0.00	0.00	--	40,668.62	2,645.43
8	1.04%	0.84	9.69	1.06	48,851.56	8,005.98
9	0.60%	0.00	0.00	--	39,479.63	2,654.15
10	0.60%	0.00	0.00	--	40,562.77	2,656.44
11	0.60%	0.00	0.00	--	39,951.61	2,642.05

表の圧縮処理

カラム・オーガナイズ表の圧縮タイミング

以下のタイミングでカラム・オーガナイズ・ディクショナリーが作成され圧縮される。

・表にある程度の件数をINSERT

```
2017-10-03-16.26.41.715552+540 I15230974E574          LEVEL: Info
PID     : 7077                 TID : 47992073611008   PROC : db2sysc 4
INSTANCE: db2inst1             NODE : 004             DB   : BLUDB
APPHDL  : 4-638                APPID: *N4.DB2.171003072641
AUTHID  : DB2INST1             HOSTNAME: node1i.jp.ibm.com
EDUID   : 415                  EDUNAME: db2taskp (BLUDB) 4
FUNCTION: DB2 UDB, CDE Data, EvolvedDictionaryBuilder::reportSuccess, probe:100
DATA #1 : String, 88 bytes
ADC: ADC SUCCESS "DB2INST1.SALES_TARGET" Elapsed Time: 00:00:00.544 Rows sampled: 138255
```

db2diag.logにADC(自動辞書作成)のログが出力される

・LOADのANALYZEフェーズ
ANALYZEフェーズは以下のLOAD時に発生する。

LOAD処理	ANALYZE有無
LOAD with REPLACE	YES
LOAD with INSERT(new、empty table、Load /dev/null)	YES
LOAD with INSERT(row delete、truncate table、Import /dev/null)	NO

```
SQL3500W  The utility is beginning the "ANALYZE" phase at time "10/03/2017
15:33:36.148905".
```

Loadのメッセージファイルで
ANALYZEフェーズの発生が確認
できる

表の圧縮状況の確認（その１）

各表の圧縮状況を確認することができる

①SYSCAT.TABLESの情報から確認する方法
表に対してRunstatsされた状態で、SYSCAT.TABLESを確認する。
カラム・オーガナイズ表の場合、ADMIN_GET_COMPRESS_INFO表関数では情報を取得できない。

```
select substr(TABSCHEMA,1,16) as TABSCHEMA,substr(TABNAME,1,16) as TABNAME,
    CARD,NPAGES,PCTPAGESSAVED
from syscat.tables where tabname = 'SALES_TARGET' ;
```

Runstats時の状況が出力される

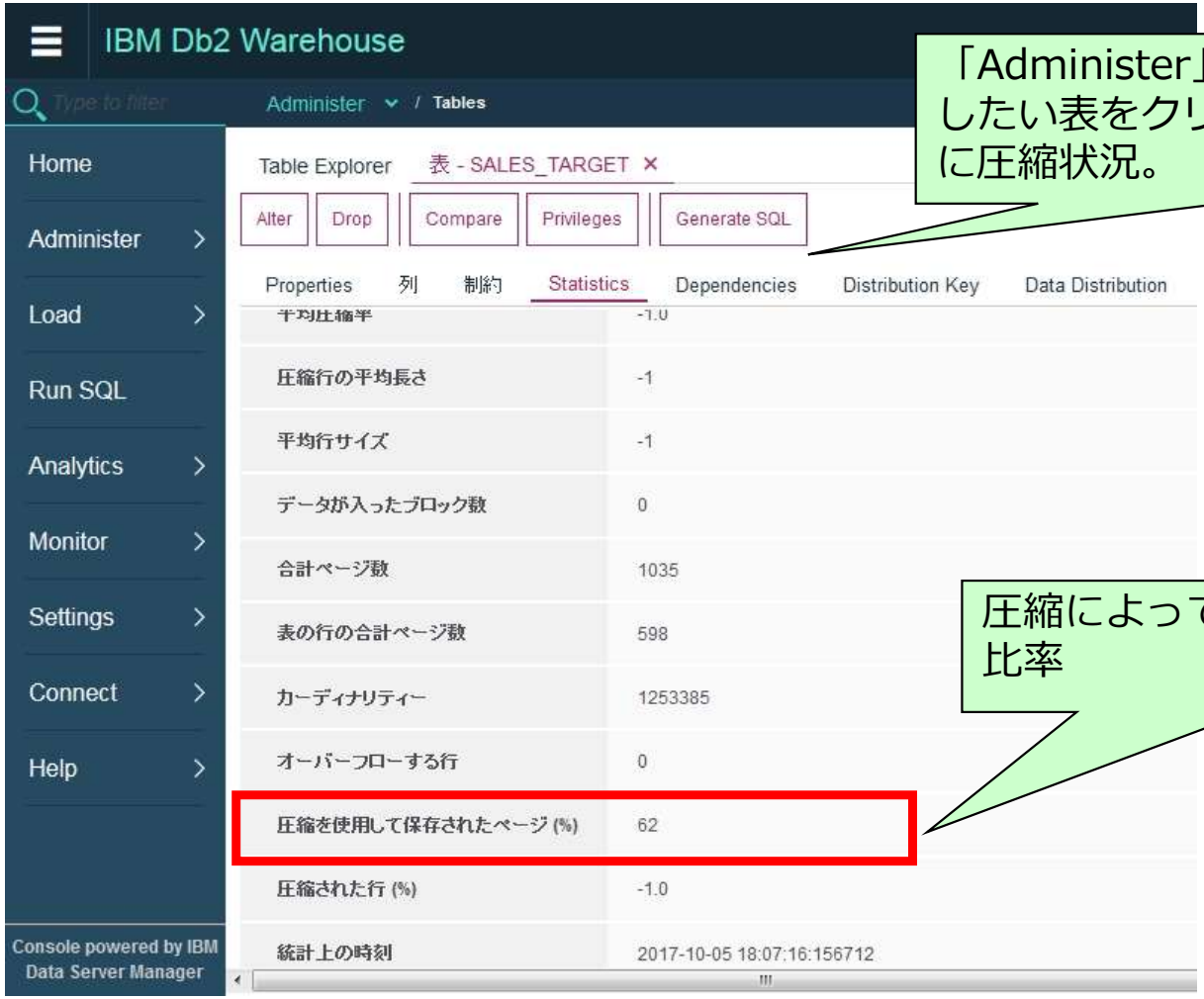
```
TABSCHEMA        TABNAME          CARD                NPAGES              PCTPAGESSAVED
---------------- ---------------- ------------------- ------------------- -------------
GOSALES          SALES_TARGET                258658                 253            50
DB2INST1         SALES_TARGET               1253385                 598            62

  2 record(s) selected.
```

「PCTPAGESSAVED」が圧縮状況。
行オーガナイズ表のサイズを元に、どのくらい圧縮されるか表示される。
62ならば、行オーガナイズに比べて、38%のサイズの表となる。

表の圧縮状況の確認（その２）

各表の圧縮状況を確認することができる

②表単位でWebコンソールから圧縮状況を確認する方法
Webコンソールで表単位の圧縮状況を確認できる。行圧縮ではないため、「平均圧縮率」「圧縮行の平均の長さ」「平均行サイズ」「圧縮された行」などは、すべて「-1」と出力される。



「Administer」→「Tables」で確認したい表をクリック。「Statistics」に圧縮状況。

圧縮によって節約されるサイズの比率

索引の利用

CREATE、ALTER TABLEでPrimaryKey、UniqueKeyを作成できる

CREATE/ALTER INDEXはエラー（SQL1667N）となる。
全列を条件に指定した際、アクセスパスでその索引を利用することができる。
（例え先頭列であっても、一部の列のみの条件だと索引は利用されない。）

```
create table distcust11(
cust_id int not null,
name char(20) not null,
gender char(5) not null,
primary key(cust_id,name) ENFORCED
)
DISTRIBUTE BY HASH (cust_id,name)
;
```

キー重複を許可させない設定。デフォルトはNOT ENFORCEDのため、データのチェックはしない。

PKには、分散キーをすべて含める必要がある

CREATE/ALTER INDEXはエラー（SQL1667N）となる。

```
SQL1667N  The operation failed because the operation is not supported with the
type of the specified table.  Specified table: "DB2INST1.DISTTEST1".  Table
type: "ORGANIZE BY COLUMN".  Operation: "CREATE INDEX".  SQLSTATE=42858
```

（参考）照制約（Informational制約）の利用

Informational制約としての参照制約を作成できる

表間のリレーションがわかるため、Joinなどで正確な見積りができるようになり、より適切なアクセスパスが選択される。

```
alter table DISTCUST11
add constraint FK_DISTCUST11
foreign key(CUST_ID,NAME) references DISTCUST12 (CUST_ID,NAME)
NOT ENFORCED NOT TRUSTED
;
```

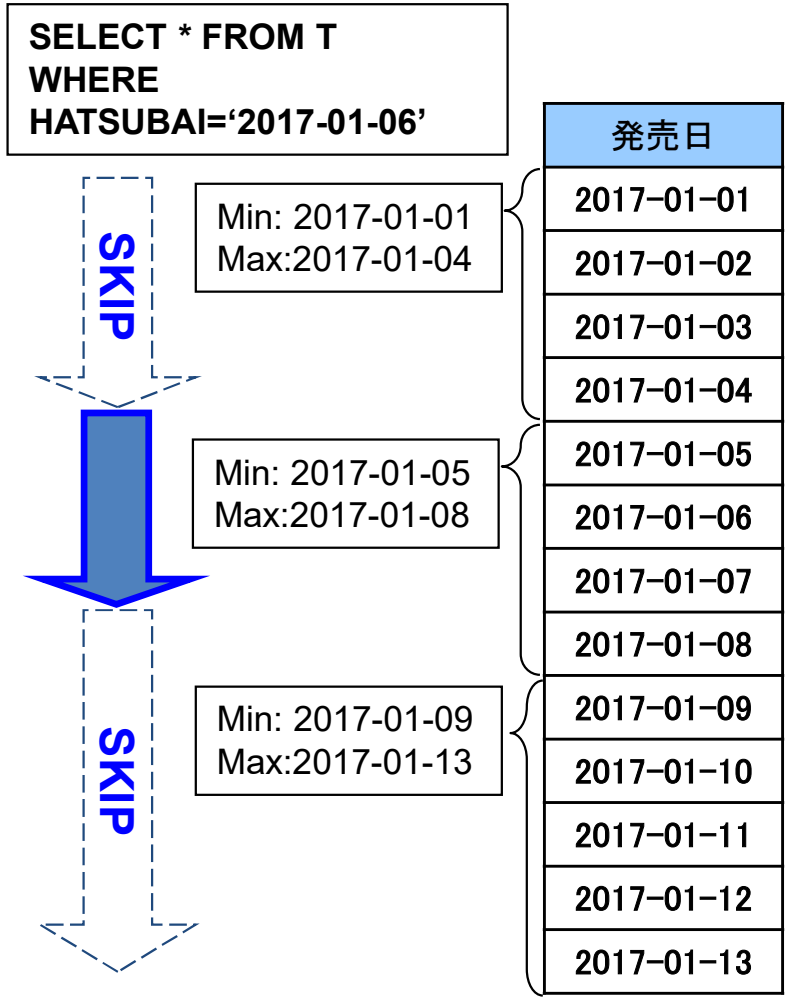
NOT ENFORCED：データの追加検査はされず、SQLコンパイラーに利用されるのみ
NOT TRUSTED：データが、制約に適合していない可能性があることを指定

参照制約によって、Joinでの絞込みが適切に評価される。

```
select a.* from DISTCUST11 a,DISTCUST12 b
where a.CUST_ID=b.CUST_ID and a.NAME=b.NAME
;
```

データ・スキッピングを利用した効率的なアクセス

- 一定のデータ件数毎に、各列に出現したデータの最大値と最小値を保持
- 条件に適合しないデータブロックを自動的にスキップ
- I/O量、メモリー容量、CPU時間を大幅削減
- データ・スキッピングのための管理情報はすべて自動メンテナンスされ、利用者による管理は一切不要



条件に合致するデータを含んだ領域のみをスキャン

※ データ・スキッピングの対象データタイプは、数値型のデータ・タイプ（DATE型, TIMESTAMP型も含む）のみに有効だったが、V10.5 FP4の機能拡張にて文字型（CHAR、VARCHAR、GRAPHIC、VARGRAPHIC）も追加された。

シノプシス表（概要表）によるデータ・スキッピングの管理

■SYNOPSIS表(シノプシス表または概要表)
- –カラム・オーガナイズ表を作成すると自動的に作成される
- –カラムごとの、一定データ件数ごとの最大値/最小値を持つ（日時型、数値型、文字型）

```
select substr(TABSCHEMA,1,16) as TABSCHEMA, substr(TABNAME,1,40) as TABNAME, TABLEORG
from SYSCAT.TABLES where TABLEORG='C' ;
```

```
TABSCHEMA       TABNAME                                  TABLEORG
--------------- ---------------------------------------- --------
SAMPLES         ILNEITEM                                 C
SYSIBM          SYN170914172633799432_LINEITEM           C
GOSALES         PRODUCT_BRAND                            C
SYSIBM          SYN170914173030923553_PRODUCT_BRAND      C
...
```

SYSIBMスキーマのシノプシス表が自動的に作成される

シノプシス表

TSN0～TSN1023

TSNMIN	TSNMAX	L_ORDERKEY MIN	L_ORDRKEY MAX	…
0	1023	123456	345678	
1024	2047	400000	673876	

L_ORDERKEY	L_QUANTITY
123456	123
187609	4567
192765	890
:	:
:	:
456789	1357

アクセスする際のヒープ利用の特徴

- カラム・オーガナイズ表では多くのソートヒープが利用される
  - カラム・オーガナイズ表では、すべての結合がハッシュ・ジョインとなる
  - GROUP BYをハッシュによって行う
  - カラム処理から列処理にデータを受け渡すキュー(CTQ)の処理にソートヒープを使用する
- 同時実行ユーザー数に応じ、バッファープールと同等、またはバッファープール以上の領域をソートヒープに割り当てること
  - SHEAPTHRES_SHR = DBパーティションでのSORTHEAP合計サイズ上限
  - SORTHEAP = １つ１つのソート処理で利用可能なソートヒープ
    - SORTHEAPが小さいとソートオーバーフローが発生しパフォーマンスが低下する
    - SORTHEAPとSHEAPTHRES_SHRの割合で、SHEAPTHRES_SHRが小さいとソートヒープが確保できず、クエリーが異常終了する場合がある

同時利用ユーザーが少ない場合（20ユーザー未満）	同時利用ユーザーが多い場合（20ユーザー以上）
バッファープール（全メモリーの40%）	バッファープール（全メモリーの25%）
SHEAPTHRES_SHR（全メモリーの40%）	SHEAPTHRES_SHR（全メモリーの50%）
SORTHEAP=SHEAPTHRES_SHR/5	SORTHEAP=SHEAPTHRES_SHR/20

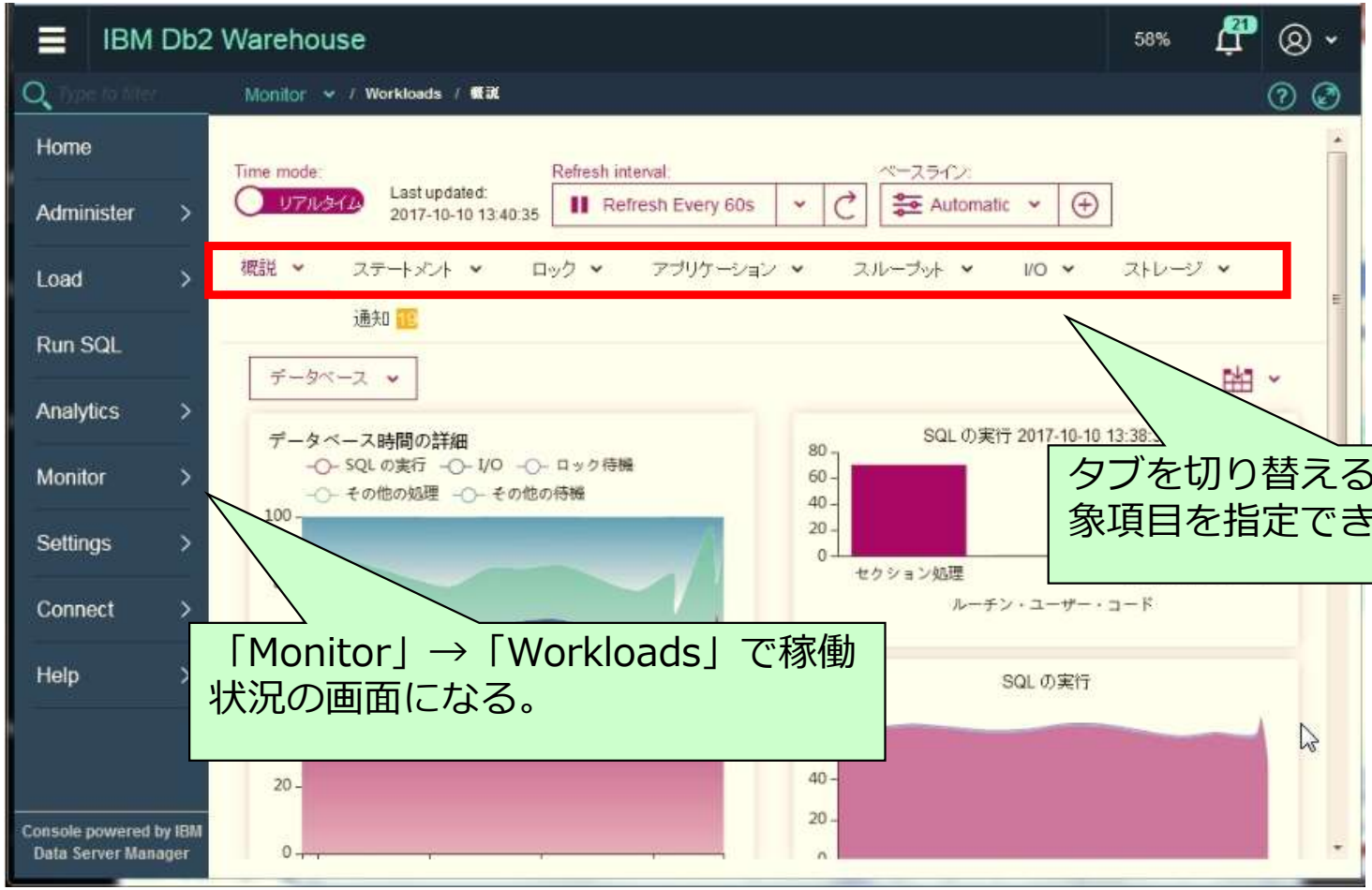
※各ヒープはDB全体の合計サイズではなく、DBパーティション当りのサイズのため注意。

稼働状況の確認

稼動状況の確認（リアルタイム）

リアルタイムで、DBの稼動状況が確認できる

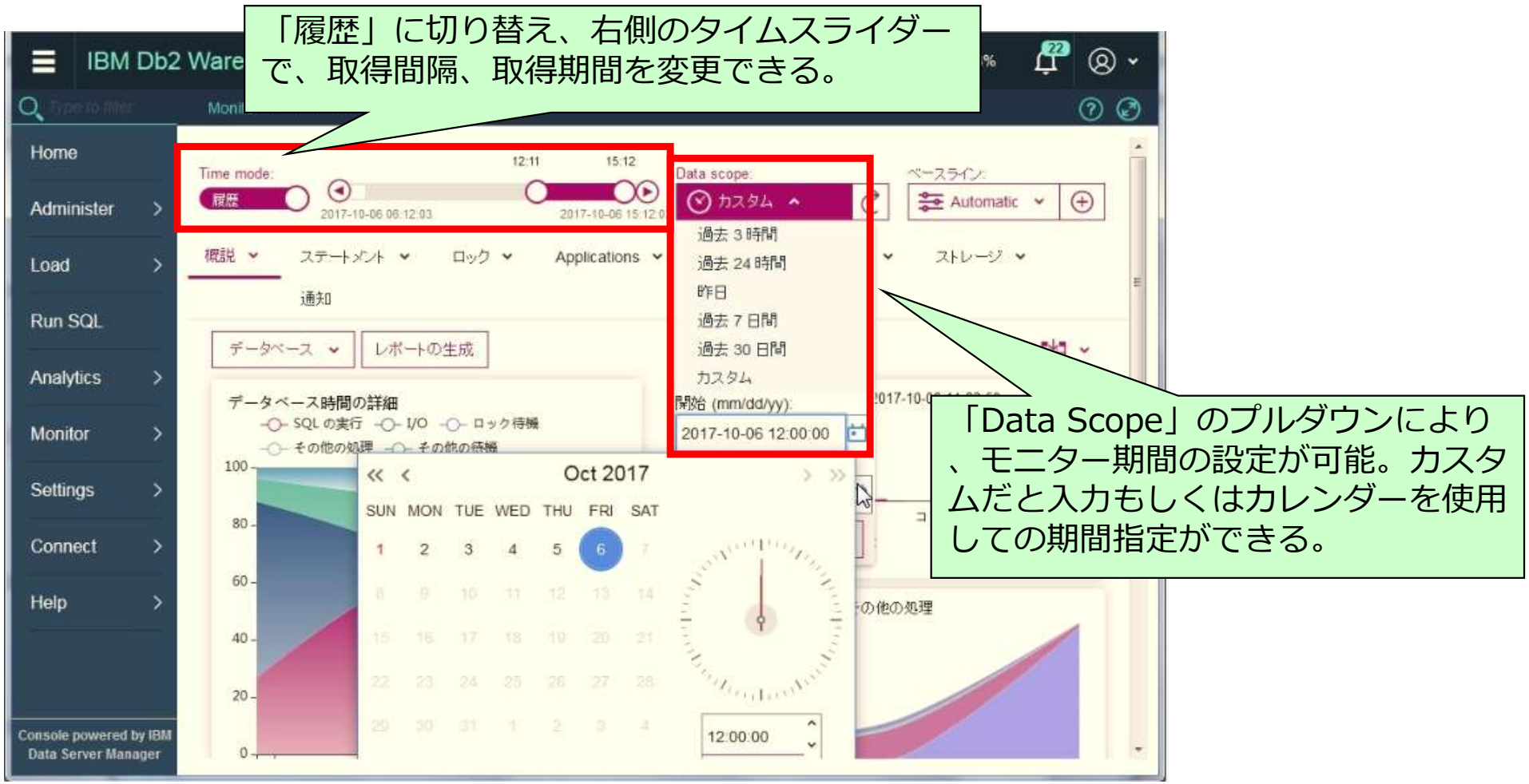
Webコンソールで現在の稼働状況をグラフィカルに確認できる。
まずは「概説」を確認することで、ワークロードのボトルネック、処理量、CPU使用が確認できる。タブを変更することで、さらに詳細なモニター項目の指定ができる。



稼動状況の確認（履歴）

DBの稼動状況の履歴が確認できる

Webコンソールのモニター画面で、「Time mode」を「履歴」に切り替え、右にあるタイムスライダーで確認期間を変更することで、時間を遡って状況を確認することができる。



ワークロードの識別

SQL処理時間やアプリ名でアクティビティが識別される構成になっている

■処理時間によって、サービス・サブクラスが分かれている
- –SYSDEFAULTSUBCLASS
  - •処理時間：0秒 ～ 30秒
  - •CPUSHARE：2,500
- –SYSMEDIUMSUBCLASS
  - •処理時間：30秒 ～ 10分
  - •CPUSHARE：2,500
- –SYSCOMPLEXSUBCLASS
  - •処理時間：10分 ～
  - •CPUSHARE：3,500

ワークロードを識別する方法として処理時間による「**THRESHOLD**」を利用している。

THRESHOLDの閾値を処理時間とし、その際のアクションとして、ワークロードのサブクラスを変更（REMAP）している。サブクラス毎にCPUSHAREの重みを設定することで、例えば暴走クエリーがいても軽い処理のCPUは確保される設定となっている。

※CPUSHAREについては後述

■処理タイプによって、サービス・サブクラスが分かれている
- –SYSLOADSUBCLASS
  - •処理タイプ：LOAD
  - •CPUSHARE：1,500

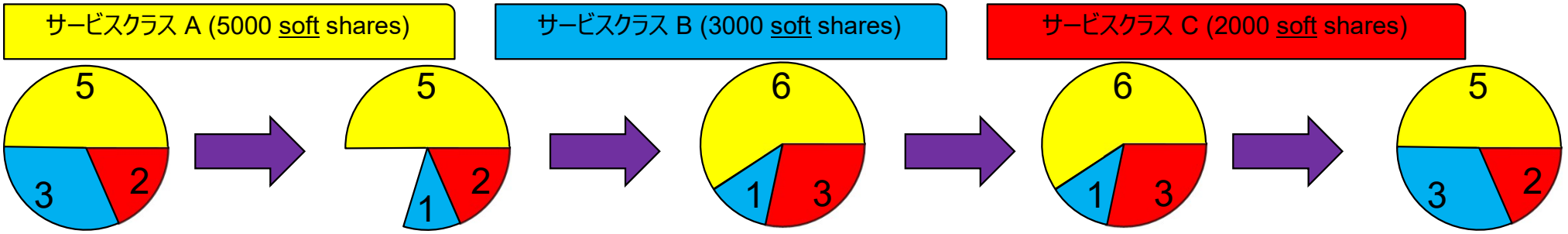
Db2 WarehouseのモニターツールであるDSM(Data Server Manager)の処理を、専用のワークロードとして識別している。その結果、通常の業務処理とDSMの処理を分類してモニターすることができる。

■アプリケーション名によって、ワークロード識別されている
- –DSM_WORKLOAD
  - •DSM(Data Server Manager)の処理がアプリ名で識別されている
  - •アプリケーション名が「‘DSMAu*’,‘DSMOQT’,‘DSMRt*’,‘DSSNAP*’,‘DS_ConnMgt*’」

（参考）ワークロードのCPUSHAREによる優先制御

CPU使用率を、シェアまたはリミットで制御できる

■CPUシェア
- –サービスクラスへの**CPUリソースの割り当てを比率割合**で決める
- –ソフトCPUシェア
  - •CPU競合が発生する場合に有効な制限。CPUに空きがあれば制限を越えて利用可能
- –ハードCPUシェア
  - •CPU使用量に対する制限。CPUに空きがあっても制限されるが、同じ環境にてソフトCPUシェアの処理が実行されていない場合にはCPUを無制限に利用することができる



すべてのサービスクラスが各シェアの最大値まで使用している

サービスクラスBが一部のリソースを解放する

サービスクラスAとCが余ったリソースを追加で使用する

サービスクラスBが再度リソースを必要とする状況が発生する

サービスクラスAとCの処理が完了すると、Bにリソースが割り当てられる

■CPUリミット
- –サービスクラスが利用可能なシステム**CPUリソースの最大使用率（%）**を指定する

ワークロードのモニター

サービスクラス、ワークロード単位で処理状況がモニターできる

■サービスクラス毎の処理状況確認
- 「Monitor」→「Workloads」の「スループット」タブから選択
- 識別されたサービスクラス毎に、CPU時間、Trans数、アクセス行数など確認できる

ワークロードのモニター

サービスクラス、ワークロード単位で処理状況がモニターできる

■ワークロード毎の処理状況確認
- 「Monitor」→「Workloads」の「スループット」タブから選択
- 識別されたワークロード毎に、CPU時間、Trans数、アクセス行数など確認できる

カラム・オーガナイズ表専用のモニター項目が追加されています。

（参考）モニタリング情報の確認：カラムデータの処理時間

■カラムデータの処理時間が占める割合
- 経過時間のうち、カラムデータの処理に費やされた時間の割合を確認できる
- 一般に、この割合が高い方が、カラム・オーガナイズ処理エンジンが有効活用されている

```
SELECT
 TOTAL_SECTION_TIME, TOTAL_COL_TIME,
 DEC((FLOAT(TOTAL_COL_TIME)/ FLOAT(NULLIF(TOTAL_SECTION_TIME,0)))*100,5,2) AS PCT_COL_TIME,
 STMT_TEXT
FROM TABLE(MON_GET_PKG_CACHE_STMT(NULL,NULL,NULL,-1)) AS T
WHERE STMT_TEXT LIKE '%TEST01.TABLE01%'
```

調査対象ステートメントの一部あるいは全部を条件に指定

TOTAL_SECTION_TIME	TOTAL_COL_TIME	PCT_COL_TIME	STMT_TEXT
3509	606	17.26	select dbpartitionnum(seq) as partnum,* from TEST01.TABLE01 WHERE D..
611340	270349	44.22	SELECT C04, C05, C06, COUNT_BIG(*) FROM DECGOK A,TEST01.TABLE01 B W..
26741390	15255571	57.04	select POW(10,0) * t0.c1 + POW(10,1) * t1.c1 + POW(10,2) * t2.c1 + ..
2792	2655	95.09	INSERT INTO ibmpdq.datastats (tableid, tbspaceid, hashid, rowcnt, r..
5591	5468	97.80	UPDATE ibmpdq.datastats SET rowavg = (SELECT (avg(case when "BASE_..

各ステートメントのセクション実行時間のうち、カラム処理に費やした時間の割合

「total_col_time」
カラム・オーガナイズ表のデータアクセスに費やされた合計経過時間(ミリ秒)

「total_section_time」
エージェントがセクション実行にかかった時間の合計(ミリ秒)

カラム・オーガナイズ表専用のモニター項目が追加されています。

（参考）モニタリング情報の確認：バッファー・プールヒット率 (1/2)

■カラムデータのバッファープールヒット率①
- MON_BP_UTILIZATIONを利用して確認できる。
- 各DBパーティションにおける、データベース活動化時点以降のヒット率が表示される

```
SELECT SUBSTR(BP_NAME,1,15) BP_NAME
  ,MEMBER
  ,COL_HIT_RATIO_PERCENT
  ,DATA_HIT_RATIO_PERCENT
  ,INDEX_HIT_RATIO_PERCENT
  ,COL_PHYSICAL_READS
  ,DATA_PHYSICAL_READS
  ,INDEX_PHYSICAL_READS
FROM SYSIBMADM.MON_BP_UTILIZATION WHERE BP_NAME='IBMDEFAULTBP'
ORDER BY MEMBER ASC
```

BP_NAME	MEMBER	COL_HIT_RATIO_PERCENT	DATA_HIT_RATIO_PERCENT	INDEX_HIT_RATIO_PERCENT	COL_PHYSICAL_READS	DATA_PHYSICAL_READS	INDEX_PHYSICAL_READS
IBMDEFAULTBP	0	94.27	99.95	99.47	154667	3279856	716699
IBMDEFAULTBP	1	88.61	89.64	93.02	994181	252237	343061
IBMDEFAULTBP	2	87.22	89.92	92.70	703237	247051	302290
IBMDEFAULTBP	3	87.20	89.96	92.56	708394	247133	306739
..	..	..	..	..	..	..	..

カラムデータに関するBPヒット率(メンバー毎)

カラムデータの物理読取ページ数
(*pool_col_p_reads + pool_temp_col_p_reads*)

参考：Monitoring buffer pool activity
https://www.ibm.com/support/knowledgecenter/SS6NHC/com.ibm.swg.im.dashdb.admin.mon.doc/doc/r0007606.html

カラム・オーガナイズ表専用のモニター項目が追加されています。

（参考）モニタリング情報の確認：バッファー・プールヒット率 (2/2)

■カラムデータのバッファープールヒット率②

– DB全体でのヒット率を確認する場合は、関連エレメント値から計算可能
(全メンバーの読取ページ数を合算するため、MON_BP_UTILIZATION は使用しない)

WITH .. AS(共通表式)にて行/列の読取ページ数を取得

```
WITH POOL_DATA( ROW_DATA_PAGES_FOUND, COL_DATA_PAGES_FOUND,
                ROW_DATA_L_READS,     COL_DATA_L_READS )
AS (SELECT SUM(POOL_DATA_LBP_PAGES_FOUND - POOL_ASYNC_DATA_LBP_PAGES_FOUND)   AS ROW_DATA_PAGES_FOUND,
           SUM(POOL_COL_LBP_PAGES_FOUND  - POOL_ASYNC_COL_LBP_PAGES_FOUND)   AS COL_DATA_PAGES_FOUND,
           SUM(POOL_DATA_L_READS + POOL_TEMP_DATA_L_READS)                    AS ROW_DATA_L_READS,
           SUM(POOL_COL_L_READS  + POOL_TEMP_COL_L_READS)                     AS COL_DATA_L_READS
     FROM TABLE(MON_GET_DATABASE(-2)) AS T)
SELECT
  CASE WHEN ROW_DATA_L_READS > 0
       THEN DEC((FLOAT(ROW_DATA_PAGES_FOUND) / FLOAT(ROW_DATA_L_READS)) * 100, 5, 2) ELSE NULL
  END AS ROW_DATA_PAGE_HIT_RATIO,
  CASE WHEN COL_DATA_L_READS > 0
       THEN DEC((FLOAT(COL_DATA_PAGES_FOUND) / FLOAT(COL_DATA_L_READS)) * 100, 5, 2) ELSE NULL
  END AS COL_DATA_PAGE_HIT_RATIO
FROM POOL_DATA
```

後半のSELECT .. 部分でヒット率を計算

```
ROW_DATA_PAGE_HIT_RATIO COL_DATA_PAGE_HIT_RATIO
----------------------- -----------------------
                  99.80                   87.14
```

カラムデータに関するBPヒット率(DB全体)

$$\text{BPヒット率} = \frac{(\text{LBP\_PAGES\_FOUND} - \text{ASYNC\_LBP\_PAGES\_FOUND})}{(\text{L\_READS} + \text{TEMP\_L\_READS})}$$

カラム・オーガナイズ表専用のモニター項目が追加されています。

（参考）モニタリング情報の確認：プリフェッチ

■プリフェッチ状況確認

–カラムデータに対するプリフェッチの発生状況は以下で確認可能

–大量データを連続読込するようなクエリーでは、プリフェッチャーの方がI/O効率が良い

```
SELECT
   MEMBER,
   POOL_ASYNC_COL_READS,
   POOL_COL_P_READS,
   DEC((FLOAT(POOL_ASYNC_COL_READS)/FLOAT(NULLIF(POOL_COL_P_READS,0)))*100,5,2) AS COL_PREFETCH_RATIO,
   SKIPPED_PREFETCH_UOW_COL_P_READS,
   POOL_FAILED_ASYNC_COL_REQS
FROM TABLE(MON_GET_DATABASE(-2))
ORDER BY MEMBER ASC
```

既にバッファー・プールにページが存在したため、プリフェッチをスキップした回数

MEMBER	POOL_ASYNC_COL_READS	POOL_COL_P_READS	COL_PREFETCH_RATIO	SKIPPED_PREFETCH_UOW_COL_P_READS	POOL_FAILED_ASYNC_COL_REQS
0	134457	154667	86.93	186	0
1	642894	703237	91.41	1709	0
2	656632	708394	92.69	1691	0
3	620021	713287	86.92	1748	0
..					

カラムデータ読取のうちプリフェッチされた割合

プリフェッチが失敗した回数(Prefetchキューがフルだった場合などに発生する)

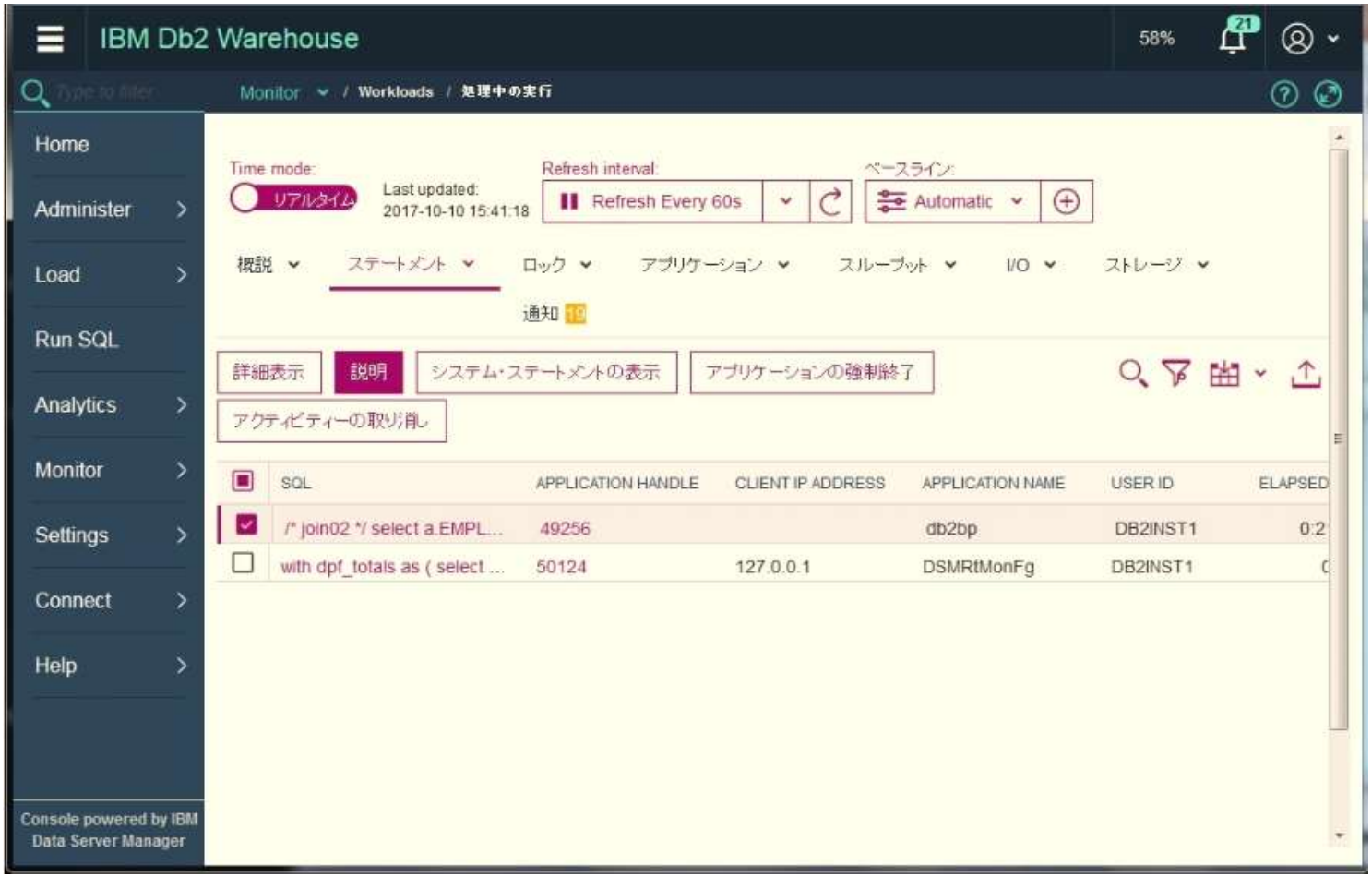
$$\text{プリフェッチの割合} = \frac{\text{POOL\_ASYNC\_COL\_P\_READS}}{\text{POOL\_COL\_P\_READS}}$$

アクセスパスの確認

アクセスパスの確認：VisualExplain

実行したSQLのアクセスパスが適切か確認することができる

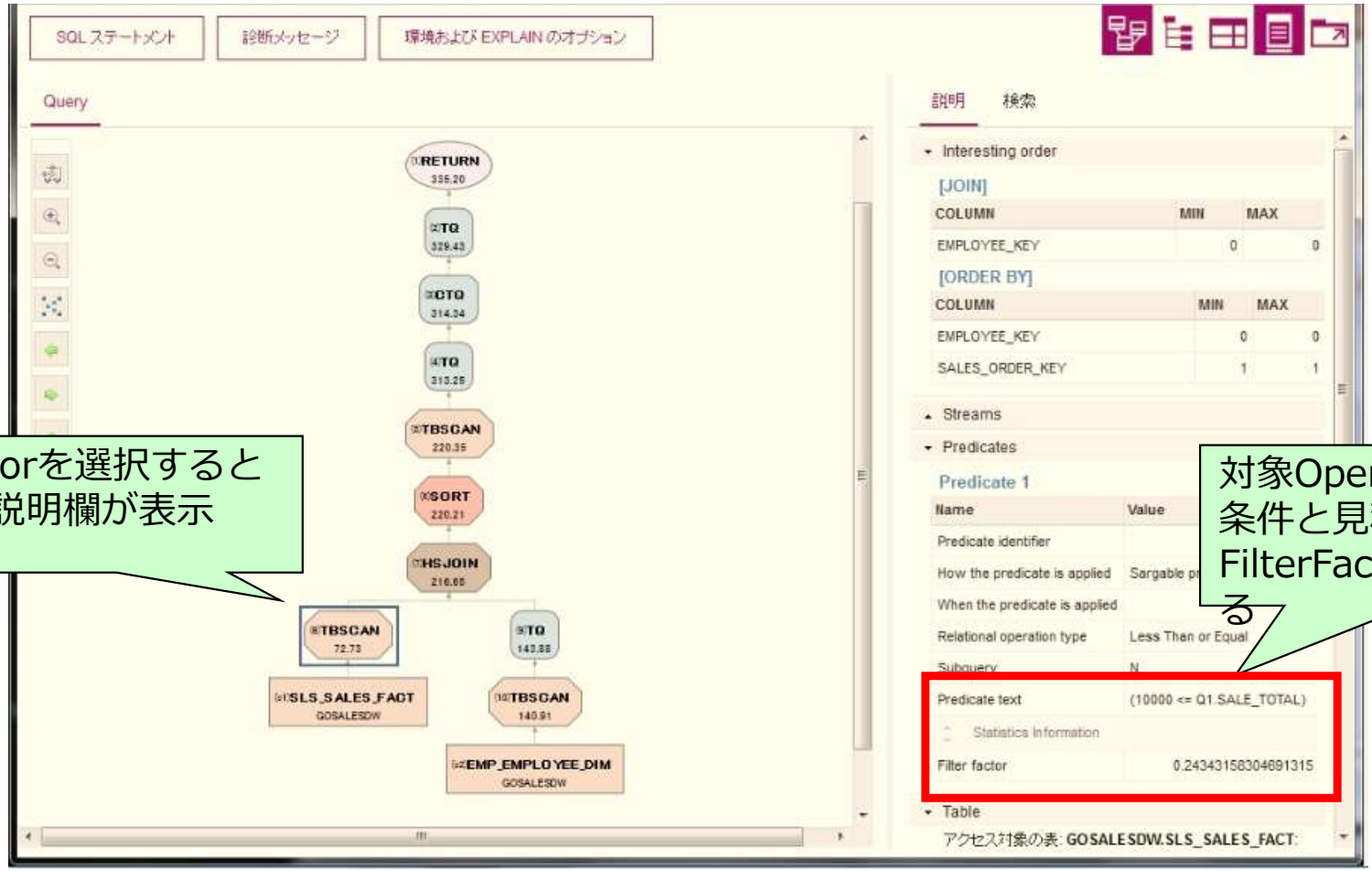
Explainの取得方法は、従来同様でEXPALIN表を作成し、db2exfmtでフォーマットする方法や、WebコンソールからVisualExplainを確認することができる。
①SQL単位でWebコンソールからアクセスパスを確認する方法
確認すべきSQLを期間指定などで探し、「説明」を選択することでVisualExplainを表示。



アクセスパスの確認：VisualExplain

実行したSQLのアクセスパスが適切か確認することができる

VisualExplainとして、グラフ表示され、確認したいOperatorを選択することで、右側に詳細な情報が表示される。
Joinの順序、WHERE条件の適用箇所、見積もり行数、見積もりコストが想定どおりか確認できる。



Operatorを選択すると、右に説明欄が表示

対象Operatorの、絞り込み条件と見積もり絞込み率（FilterFactor）が確認できる

アクセスパスの確認：VisualExplain

実行したSQLのアクセスパスが適切か確認することができる

表示モードを、「グラフ・ビュー」「ツリー・ビュー」「表形式ビュー」から選択できる。グラフ・ビューでは全体の流れが確認でき、ツリーや表形式では、各オペレータでの見積もり行数やCPUコストの確認がしやすい。

Operation	Estimated cardinality	Actual cardinality	Cumulative total cost (timeron)	Cumulative CPU cost (timeron)	Cumulative I/O cost (timeron)	Total cost (timeron)	CPU cost (timeron)	I/O cost (timeron)	Node
Return	109,727.758	-	335.20	460,588,608.00	55.00	5.77	48,829,280.00	0.00	
Table Queue	109,727.758	-	329.43	411,759,328.00	55.00	15.10	127,847,616.00	0.00	
Column Table Queue	109,727.758	-	314.34	283,911,712.00	55.00	1.09	9,222,144.00	0.00	3
Table Queue	109,727.758	-	313.25	274,689,568.00	55.00	92.90	233,148,576.00	0.00	4
Table scan	4,770.772	-	220.35	41,540,992.00	55.00	0.14	1,194,228.00	0.00	5
Sort rows	4,770.772	-	220.21	40,346,764.00	55.00	3.56	30,129,639.00	0.00	6
Hash join	4,770.772	-	216.65	10,217,125.00	55.00	0.04	314,346.00	0.00	7
Table scan - GOSALESDW.SLS_SALES_FACT	4,770.772	-	72.73	8,552,926.00	35.00	72.73	8,552,926.00	35.00	8
Table Queue	943.00	-	143.88	1,349,853.00	20.00				
Table scan - GOSALESDW.EMP_EMPLOYEE_DIM	41.00	-	140.91	73,983.40	20.00				

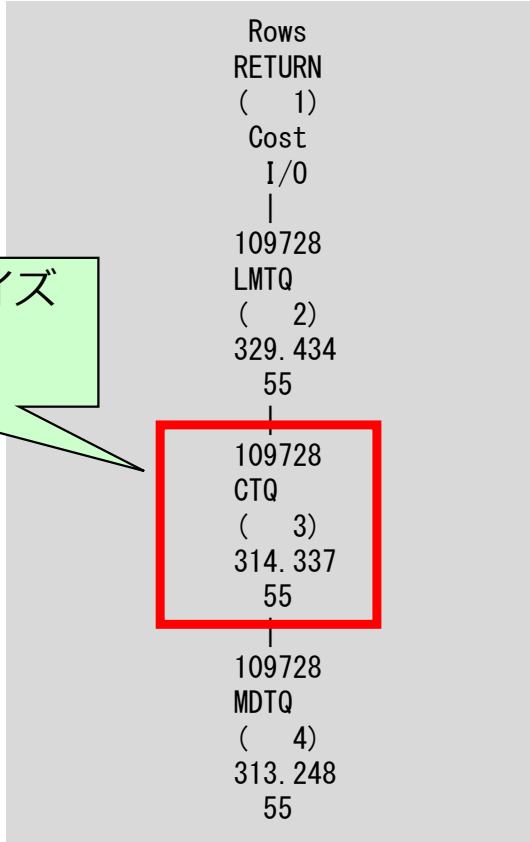
アクセスパスの確認：db2exfmt

実行したSQLのアクセスパスが適切か確認することができる

①db2topを利用、もしくはSET CURRENT EXPLAIN MODE EXPLAINを利用して、アクセスパスをEXPLAIN表に格納し、db2exfmtで抽出して確認する方法

```
db2 SET CURRENT EXPLAIN MODE EXPLAIN
db2 -vtf join01.sql
db2 SET CURRENT EXPLAIN MODE NO
db2exfmt -d BLUDB -1 -o join01.exfmt
```

■カラム・オーガナイズ表のExplain情報
・CTQ
カラム・オーガナイズデータ処理と、行オーガナイズデータ処理の間の移行処理。
■パーティションDBのExplain情報
・DTQ（指示表キュー）
レコードを再ハッシュしてキューに入れる
・BTQ（ブロードキャスト表キュー）
レコードを再ハッシュせずに全てキューに入れる
・LTQ（イントラパーティション表キュー）
表キューをパーティション内で利用する
・MTQ（マージ表キュー）
ソートのために表キューを利用する

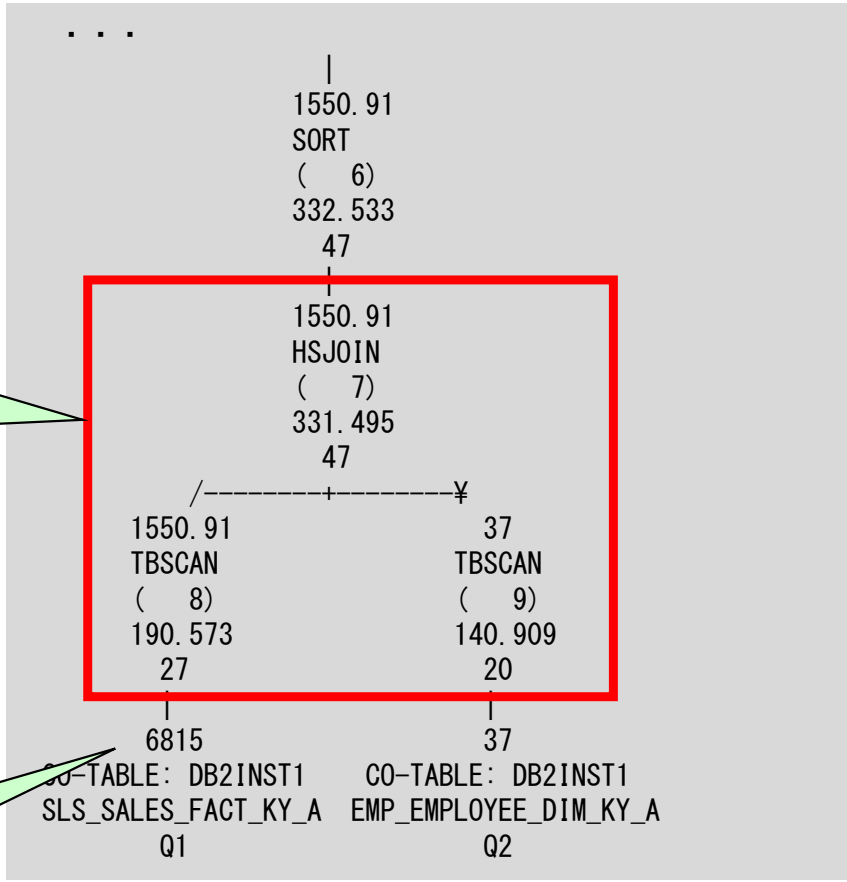


アクセスパスの確認：最適なJoin方法

分散キーがJoinキーと一致している場合が最も効率的なJoin

DBパーティション内でJoin処理が完結するため、Joinのためのレコードの移動が発生しない。

■コロケーテッド結合
結合処理がDBパーティション内で完結できるため、結合のためにDBパーティション間をレコードが送受信されることがない



表キューを使用せず、表スキャン後にそのままJoinしている

見積もり件数は、1つのDBパーティション内のレコード数を表す

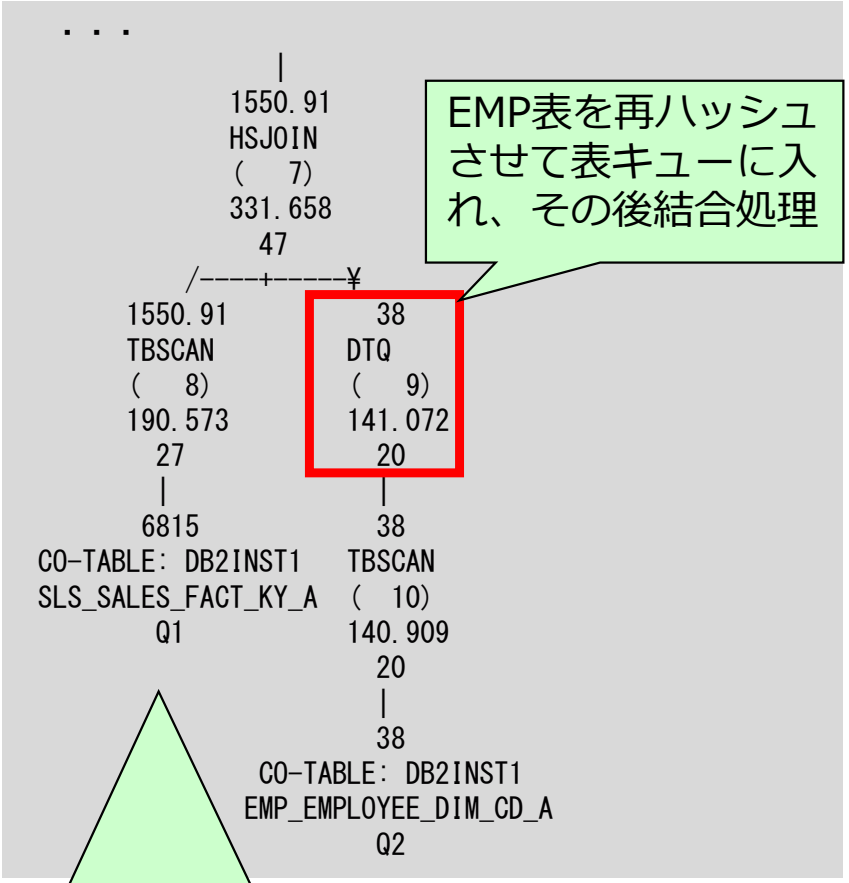
アクセスパスの確認：指示結合

分散キーとJoinキーが異なる場合、結合のためにレコードが送受信される

DBパーティション内でJoin処理が完結できないため、Joinのためのレコードの移動が発生する。

■指示結合
分散キーとJoinキーが一致しない表のレコードを再ハッシュさせて、対象パーティションにDTQを利用して送信。

※対象レコードが少なければ、コストの高い処理ではない。使用されている条件によるレコードの絞込みの見積もりが適切に評価されていれば問題ない。



アクセスパスの確認：指示結合

DTQを使用して、どのように送受信されるか確認できる

db2exfmtの対象のDTQの情報を確認することで、どのDBパーティショングループに向けてどのようにデータが送受信されるか確認できる。

```
  . . .
Input Streams:
-------------
        4) From Operator #10

                Estimated number of rows:       38
                Partition Map ID:                1
                Partitioning:                   (MULT )
                                                Multiple Partitions
                Number of columns:              5
                Subquery predicate ID:          Not Applicable
```

「Input Stream」は受信

「Output Stream」は送信

「MULT」というキーワードを含んでいるかどうか、というアクセスパス確認方法もある。

「Partition Map ID:　1」
IDが1は、IBMDEFAULTGROUP。
db2 LIST DB PARTITION GROUPS SHOW DETAIL　コマンドにてPMAP_IDと、対象のDBパーティションNoを確認できる。

「Partitioning: MULT」
データが複数のDBパーティションに向けて送受されることが確認できる。
各DBパーティションから複数のDBパーティションに向けてデータが送受信されるため、対象データが多い場合には高コストの処理となる傾向がある。

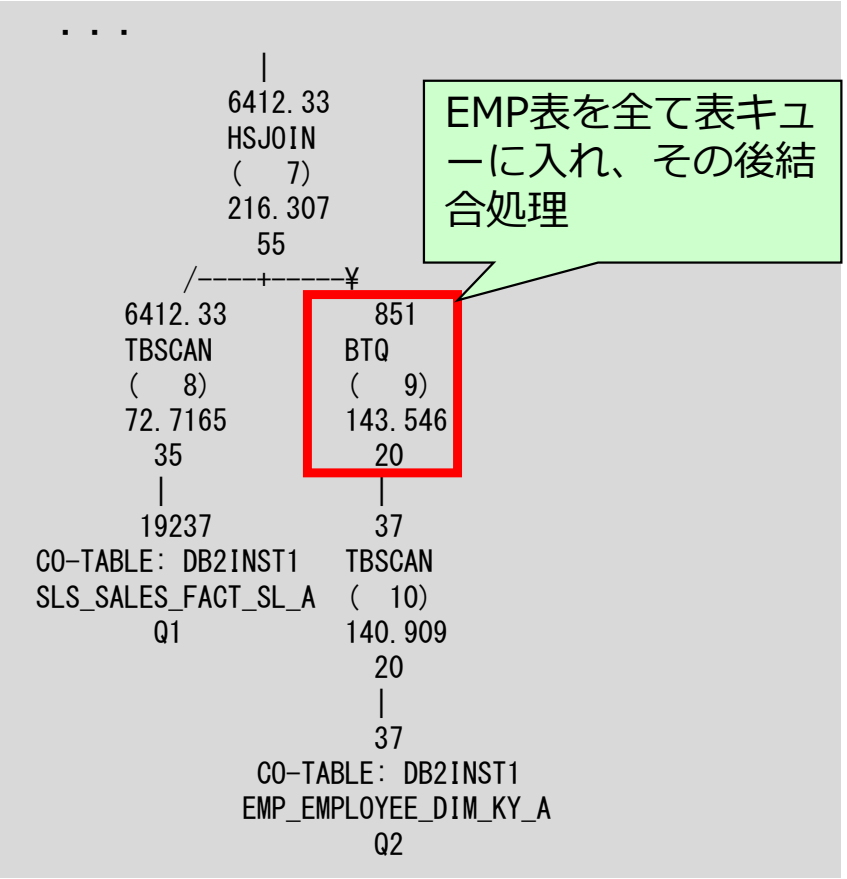
アクセスパスの確認：ブロードキャスト結合

分散キーとJoinキーが異なる場合、結合のためにレコードが送受信される

DBパーティション内でJoin処理が完結できないため、Joinのためのレコードの移動が発生する。

■ブロードキャスト結合
分散キーとJoinキーが一致しない表の全てのレコードを対象パーティションにBTQを利用して送信。

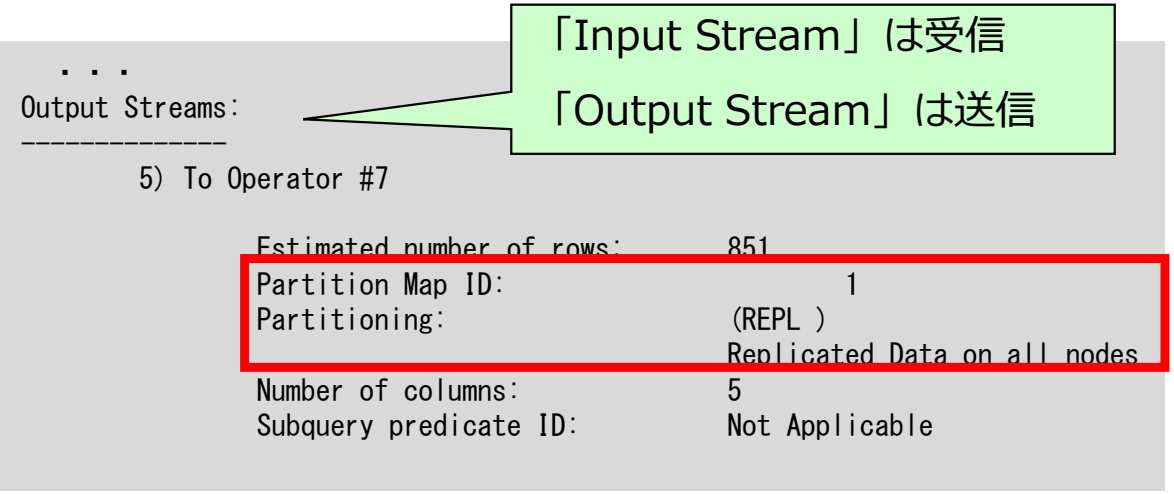
※小さな表がBTQ により送信されるならば、コストの高い処理ではない。BTQで送信されるレコード数が適切に評価されていれば問題ない。



アクセスパスの確認：ブロードキャスト結合

BTQを使用して、どのように送受信されるか確認できる

db2exfmtの対象のBTQの情報を確認することで、どのDBパーティショングループに向けてどのようにデータが送受信されるか確認できる。



「Partition Map ID:　1」
IDが1は、IBMDEFAULTGROUP。
db2 LIST DB PARTITION GROUPS SHOW DETAIL　コマンドにてPMAP_IDと、対象のDBパーティションNoを確認できる。

「Partitioning: REPL」
データが複数のDBパーティションに向けてコピーされることが確認できる。
同じデータが各DBパーティションに向けてが送信される。対象データが多い場合には高コストの処理となる傾向がある。

チューニングのための分散キー変更方法

External Tableを利用したデータの格納／抽出

ETLプロセスを単純なSQL処理で実現できる

■CSVファイルを仮想表のように定義することができる
- –CREATE EXTERNAL TABLEステートメントで、ファイルを表のように定義
  - •その表をソースとしてInsert で利用すれば、Importのように利用可能
  - •その表をターゲットとしてInsertすれば、Exportのように利用可能
- –LIKE節を使えば、既存表のレイアウトでExternal Tableを定義

LIKEではなく、列定義することも可能

```
create external table EXT_SALES_TARGET
like SALES_TARGET
using (dataobject('/mnt/blumeta0/home/db2inst1/db2look/SALES.del'))
;
```

■External Tableでのデータ格納

通常のLOAD処理よりも高速

```
insert into SALES_TARGET select * from EXT_SALES_TARGET
;
```

■External Tableでのデータ抽出

```
insert into EXT_SALES_TARGET select * from SALES_TARGET
;
```

カラム・オーガナイズ表の分散キーの変更方法（その１）

■方法①：CTAS（Create Table As Select）を利用
- –CREATE TABLEのAS SELECT節を利用して作成＆コピー
- –分散キーを変更

```
create table SALES_TARGET_TMP
as ( select * from SALES_TARGET ) WITH DATA
DISTRIBUTE BY HASH (SALES_STAFF_CODE)
;
```

Load処理を利用せず、並列にSelectされたレコードが並列でInsertされる

WITH NO DATA指定の場合、データアクセスせずに表定義のみコピーされる

カラム・オーガナイズ表の分散キーの変更方法（その２）

■事前処理：分散キーのみ変更した表を作成
- –CREATE TABLEのLIKE節を利用して作成
- –分散キーを変更

CTASの、WITH NO DATA を利用して表作成することも可能

```
create table SALES_TARGET_TMP like SALES_TARGET
DISTRIBUTE BY HASH (SALES_STAFF_CODE)
;
```

■方法②：カーソルロード
- – 元表のカーソルを定義し、そのカーソルを読み込んでLOADする
- – 各DBパーティションで並列に読み込み、LOAD処理が実行される

```
declare c1 cursor for select * from SALES_TARGET;
load from c1 of cursor messages ./load/load_sales.msg
replace into db2inst1.SALES_TARGET_TMP;
```

カラム・オーガナイズ表の分散キーの変更方法（その３）

■方法③：並列INSERT + NOT LOGGED
- –INSERT with SELECTを利用する
- –NOT LOGGED INITIALLYを利用する
  - •同じ作業単位内であれば、ログなしでの高速処理
  - •INSERTがエラーとなると、表はドロップ・ペンディングとなる（DROPのみ可能）
- –各DBパーティションで並列に読み込み、INSERT処理が実行される

既存表への追加INSERT処理は避けるべき。

```
alter table db2inst1.SALES_TARGET_TMP activate not logged initially;
insert into db2inst1.SALES_TARGET_TMP
select * from SALES_TARGET;
```

■事後処理：表の置き換え
- –RENAME TABLEを利用して、既存表から新規表へ置き換え
- –Viewや権限設定などを合わせる

```
rename table db2inst1.SALES_TARGET to SALES_TARGET_OLD;
rename table db2inst1.SALES_TARGET_TMP to SALES_TARGET;
```

置き換え先の表名にはスキーマは指定できない

参考資料

Db2 Warehouse Webサイト
https://www.ibm.com/us-en/marketplace/db2-warehouse

Db2 Warehouse Knowledge Center:
https://www.ibm.com/support/knowledgecenter/SS6NHC/com.ibm.swg.im.dashdb.kc.doc/welcome.html