

Advanced Replication

アドバンスト -DPROPR V8 Product Details-

<第2.00版>2006年01月

お断り: 当資料は、DB2 SQL Replication または DpropR V8 をベースに作成されています。

(C)日本IBMシステムズ・エンジニアリング(株) インフォメーション・マネージメント

1



Agenda

- ▶ レプリカ (*Update Anywhere*)
- ▶ レプリケーション対象データの操作
- ▶ レプリケーション処理の操作
- ▶ データ共用・区分化対応
- ▶ 特殊な表、データタイプのレプリケーション
- ▶ マルチベンダー・レプリケーション
- ▶ その他





► レプリカ (*Update Anywhere*)

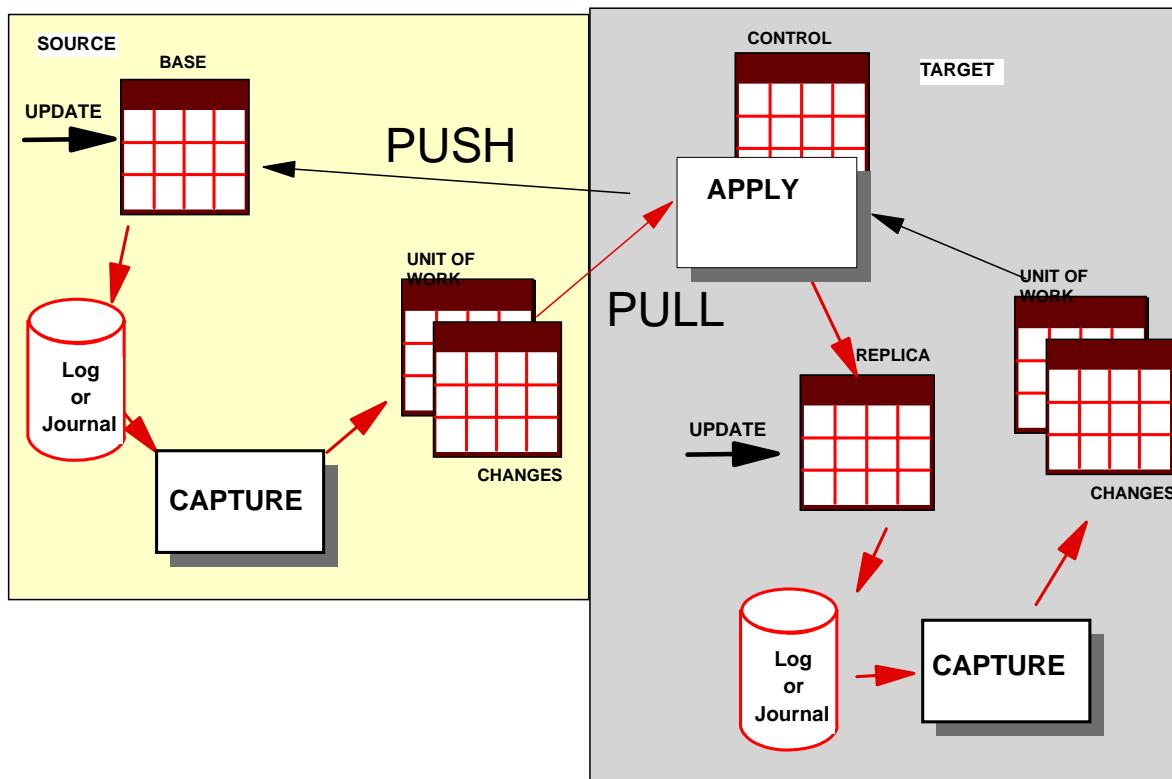
- レプリケーション対象データの操作
- レプリケーション処理の操作
- データ共用・区分化対応
- 特殊な表、データタイプのレプリケーション
- マルチベンダー・レプリケーション
- その他



blank page



レプリカ(Update Anywhere : 双方向レプリケーション)



解説:

■ UPDATE ANYWHEREとは

UPDATE ANYWHEREとは、複製ソース表からか、または複製サブスクリプションに定義されているレプリカ・ターゲット表からの変更データの複製のことです。

1方向の複製と違って、ターゲット表（レプリカ）は、更新することができ、ターゲット・サーバーの Capture プログラムが変更を取り込み、Apply プログラムがその変更を元のソース・サーバーの複製ソースにプッシュします。

複製サブスクリプション内のすべてのレプリカの更新は、まとめて複製されます。

ソース・サーバーとターゲット・サーバーの両方で Capture プログラムが実行されている場合、オリジナルのユーザー表（複製ソース・オブジェクト）とその更新可能コピー（レプリカ）の両方にに対する変更を取り込むことができ、Apply プログラムは、これらの変更を一方のサーバーから他方のサーバーに複製して、データの現行性を維持します。

1 対多の構成（1 つのソース表から多数のレプリカへの）についても同じことが言えます。つまり、複製ソース表での変更を、そのレプリカのすべてにコピーすることができ、そのレプリカのいずれかでの変更を、複製ソース表にコピーすることができます。

次に、その変更を他のすべてのレプリカにコピーできます。

Update Anywhereを実現する機能

■衝突の自動検知

-Conflict、Cascade Conflict

■衝突したDataの補償

-Before-Image Column in Consistent Change Data表

■拒否されたDataの理由コード提供

-IBMSNAP_REJ_CODE in UOW表

■Exitルーチンの提供

-ASNDONE Exit by Apply

■双方向Replication

-Apply PULL & PUSH Replication

■変更Dataの無限Loop回避

-APPLY_QUAL in UOW表

■拒否されたDataのプルーニング

-Retention_Limitのみ、通常プルーニングから除外

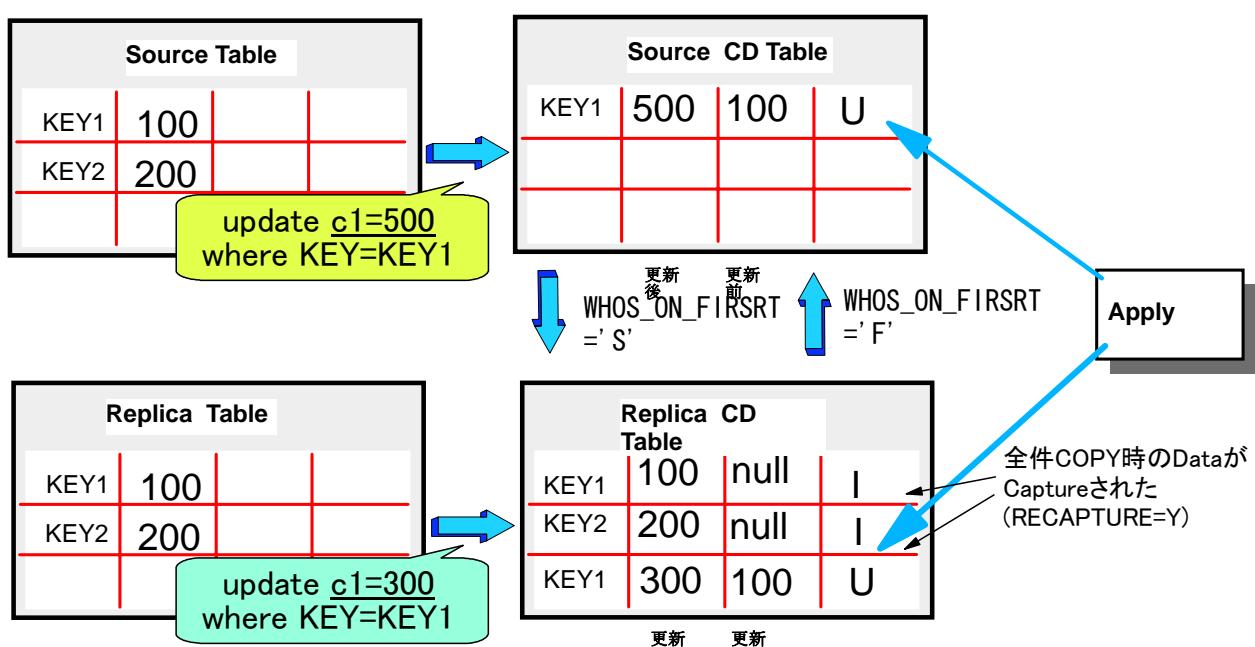
■衝突検知レベル設定可能

-CONFLICT_LEVEL(0, 1, 2) in REGISTER表(Registration時)

解説:

■衝突(Conflict)の自動検知

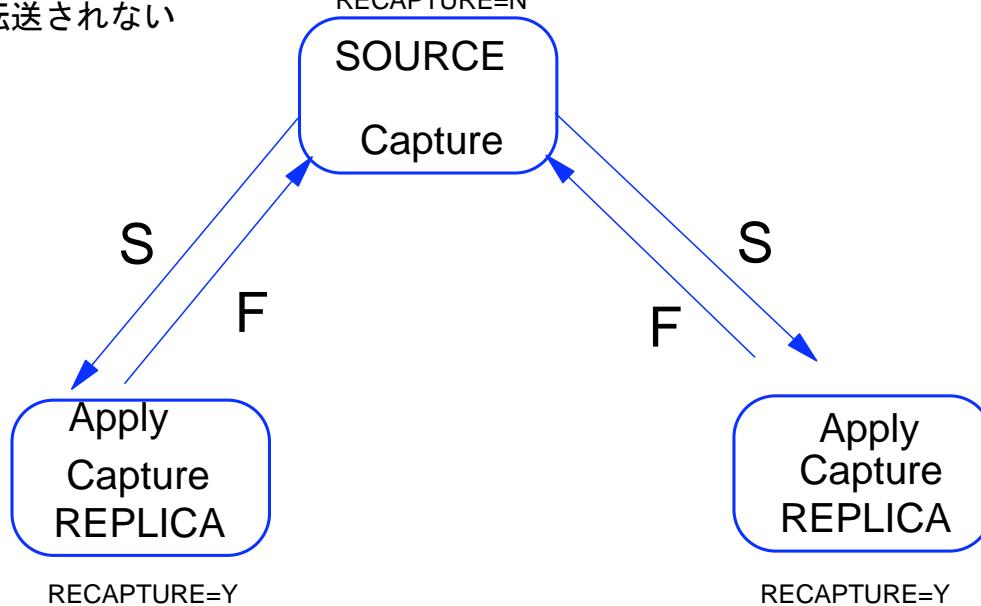
以下の例ではKEY1の行に対してConflictが発生している。ApplyはCOPY実行時CD表の内容とSpill Fileへ書き出された内容で同じKEYが見つかるとConflictが発生したと判断する
 Conflictが発生した場合、Replica側が必ず敗者となり、CD表のBEFORE-IMAGEを利用しUNDO処理を行ない、最終的にはSOURCE表の変更内容がReplica側へ反映されることになる。
 UNDOされる内容は同じLUW内の変更処理すべてとなり、UNDO処理自身も1LUWで行われる



変更Recaptureの防止 (Update Anywhere)

■ RECAPTURE (Y/N)

- CaptureがApplyによって行われた変更を変更データとしてCaptureするかどうか
 - (使用例) 他のレプリカで発生した変更をマスター側でCaptureされず他のレプリカに変更が転送されない



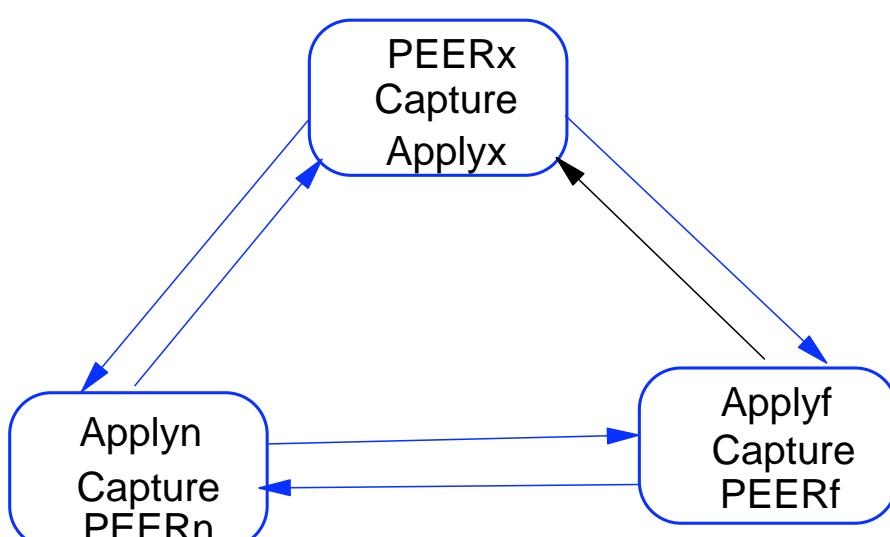
解説:

■ RECAPTURE=Nの使用例 (Peer to Peer Replication)

現行のUpdateAnywhereは競合が発生した場合、必ず“マスター”で発生した更新がレプリカへ反映される。一方ApplyはPUSH, PULLの両方向のReplicationを実施するが、PUSH方向のデータ量が多いとPerformanceに問題が出る。またお客様の要求でも競合の検知の必要性がなく(ex. 異なるKEYでのInsert only)、データの鮮度を高める要求が多くあった。そこで対等レプリケーション(Peer to Peer Replication)のサポートがされることになった。

対等Replicationでは

- 競合検知はしない (RECAPTURE=N)
- Capture/Applyはおのとのサイトで実行し、PULLかつWHENS_ON_FIRST='S' 方向のみのレプリケーションが実施
- 全件COPYはApplyでは実行せず、手動で実施する必要がある
- GUI (RC) ではサポートされず、手動で設定する必要がある





- ▶ レプリカ (*Update Anywhere*)
- ▶ レプリケーション対象データの操作
- ▶ レプリケーション処理の操作
- ▶ データ共用・区分化対応
- ▶ 特殊な表、データタイプのレプリケーション
- ▶ マルチベンダー・レプリケーション
- ▶ その他

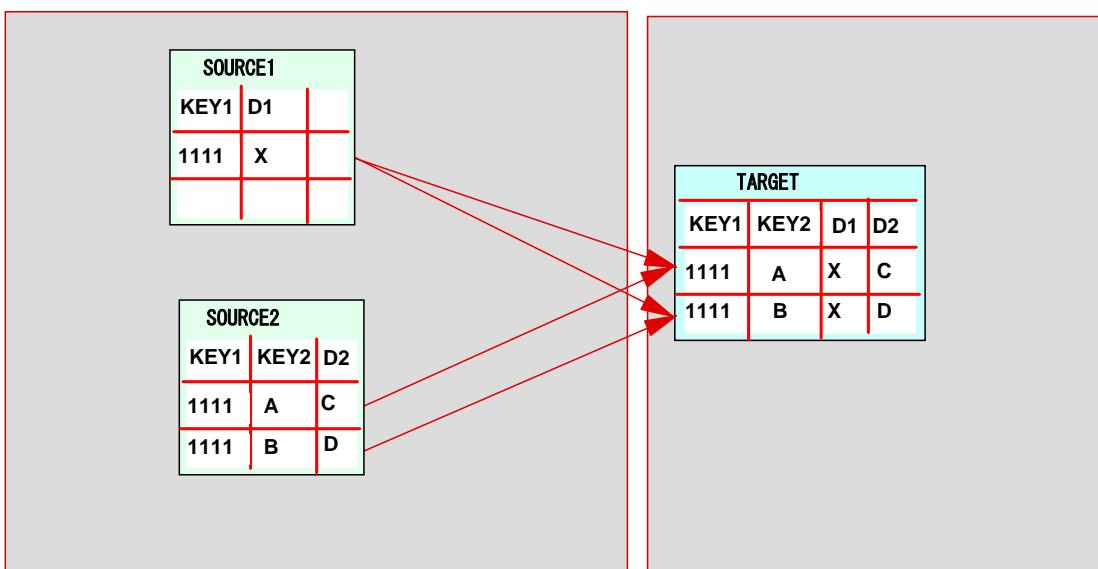


内容

- JOIN VIEW レプリケーション
- UNION レプリケーション
- レプリケーション・データのフィルタリング
 - PREDICATES
 - UOW_CD_PREDICATES
- レプリケーション・データの加工
- レプリケーション・ターゲット・キーの更新への対応
 - CHG_UPD_TO_DEL_INS
 - TARGET_KEY_CHG



JOIN VIEWレプリケーション



• SOURCE1, SOURCE2をJoinした形でTarget表をReplicationし反映させたい
SOURCE側にViewを作成する

1. VIEW1 = SOURCE1 + SOURCE2_CD ... SOURCE2の変更をTargetへ反映させる
2. VIEW2 = SOURCE2 + SOURCE1_CD ... SOURCE1の変更をTargetへ反映させる

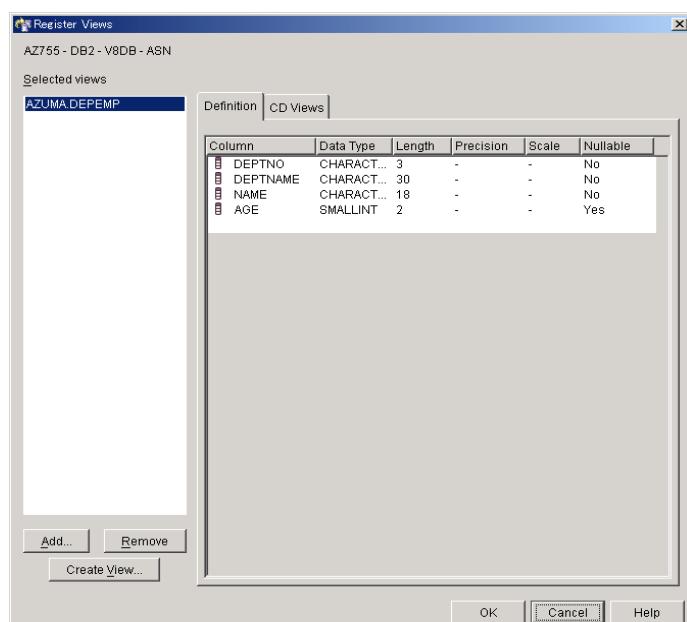
解説:

他の表の視点となる複製ソースを定義できます。

視点に含まれる各複製ソース表を定義し終わったら、視点を定義し、視点複製ソースを作成できます。
すると、その視点複製ソースは、ターゲット表へのコピーに使用可能になります。

V7では、既存の視点を複製ソースとして定義することはできませんでしたがV8から可能です。

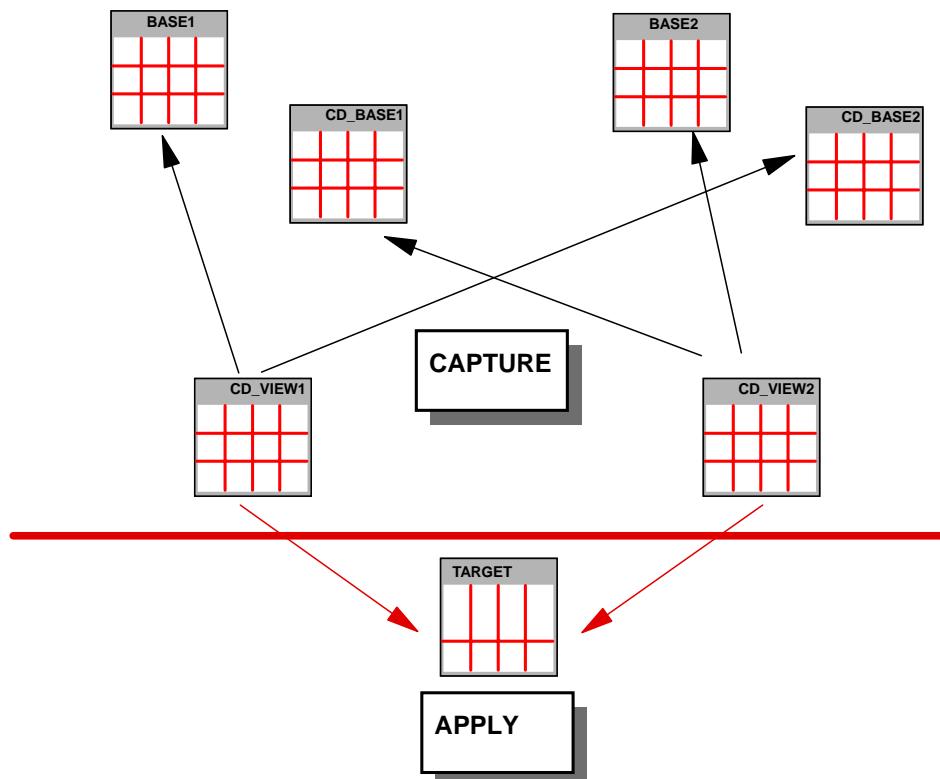
ただしその視点内で参照されている表名にスキーマをあらかじめ付けて作成する必要があります。(ASN1704E RC=11)



- 上記のようにDEPT表とEMP表をJOINする場合はあらかじめDEPT表、EMP表をRegistrationしておき、“ビューの登録”で定義する

解説:

- VIEWはREGISTER表に登録され、CAPTUREによってCD_OLD_SYNCHPOINT, CD_NEW_SYNCHPOINTもPHYSICAL_TABLE同様更新される



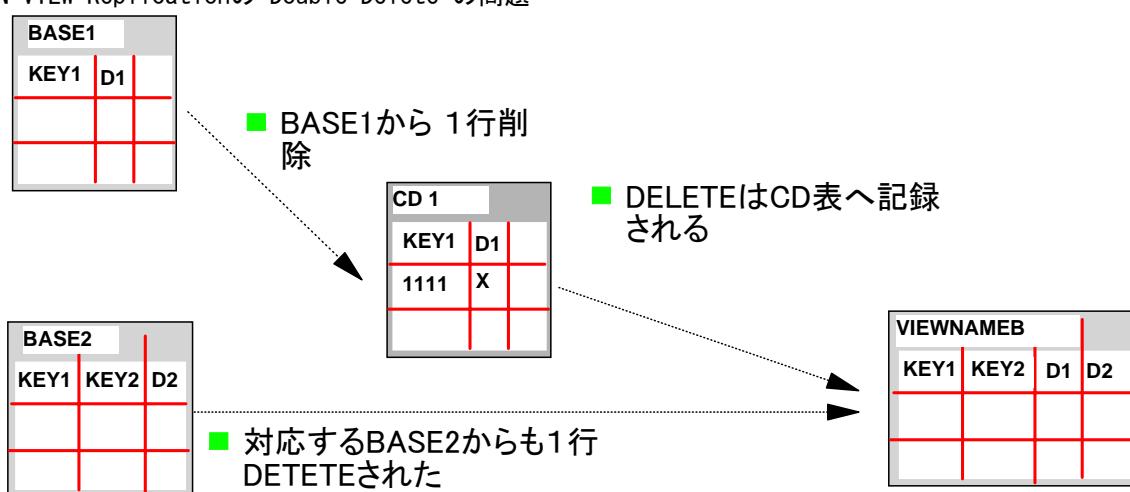
(C)日本IBMシステムズ・エンジニアリング(株) インフォメーション・マネージメント

15

DB2 Universal Database

解説:

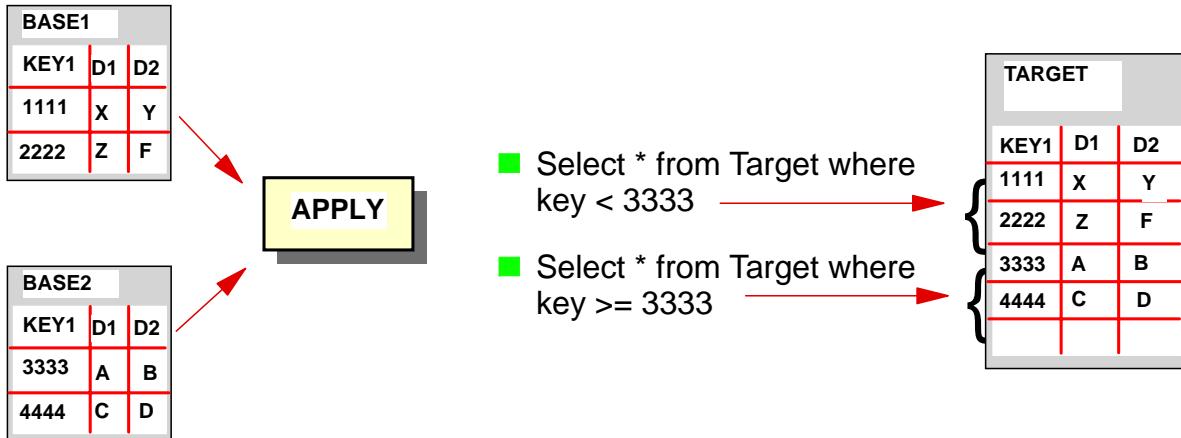
- JOIN VIEW Replicationの"Double Delete"の問題



- 上記の例でBASE2とCD1をJOINしてもJOIN VIEWは対応する行が見つからない為、DELETEはTARGET表には反映されない
この解決策としてはCD1, CD2にもVIEWを作成してそのVIEW経由のReplicationを定義することで回避可能
例)--View Registration (DEPTCD & EMPCD)

```
CREATE VIEW DEPCDEMPCD
  (IBMSNAP_UOWID, IBMSNAP_INTENTSEQ, IBMSNAP_OPERATION,
  DEPTNO, DEPTNAME, NAME, AGE) AS SELECT
  B. IBMSNAP_UOWID, B. IBMSNAP_INTENTSEQ, B. IBMSNAP_OPERATION,
  A. DEPTNO, A. DEPTNAME, B. NAME, B. AGE
  FROM DEPCD A, EMPCD B
  WHERE A. DEPTNO=B. DEPTNO
  AND A. IBMSNAP_OPERATION='D' AND B. IBMSNAP_OPERATION='D' ;
```

UNION レプリケーション



解説:

- VIEWをTARGET表側に作成、SOURCE側には作成しない
- INSERT, UPDATE, DELETEはそのまま伝播される
- FULLREFRESHもKEY RANGEでTARGET表がDELETEされるので問題なし
- コントロールセンターからはTARGET表側のVIEW定義できないので事前作成必要

レプリケーション・データのフィルタリング

■ レプリケーション・データのフィルタリング方法

- ソース表の列の値によるフィルタリング
 - ▶ 例：LOCATIONが‘東京’のものだけ、レプリケーションしたい
 - ▶ 方法：APPLYのレプリケーションの条件で指定
レプリケーション・センターで指定可能
- CD表またはUOW表固有の列 (IBMSNAP_OPERATION等) の値によるフィルタリング
 - ▶ 例：削除のデータはレプリケーションしたくない
 - ▶ 方法：SUBS_MEMBER表のUOW_CD_PREDICATES列を利用
レプリケーション・センターでは指定不可
- CD表に書き出す時点でのフィルタリング
 - ▶ 例：削除のデータはCD表にも書き出さない
 - ▶ 方法：CD表にトリガーを作成

解説：

- レプリケーション・データをフィルタリングする方法としては、以下の3つの方法が適用できます。
 - ▶ ソース表の列の値によるフィルタリング
 - ▶ CD表またはUOW表固有の列 (IBMSNAP_OPERATION等) の値によるフィルタリング
 - ▶ CD表に書き出す時点でのフィルタリング

ソース表の列の値によるフィルタリング

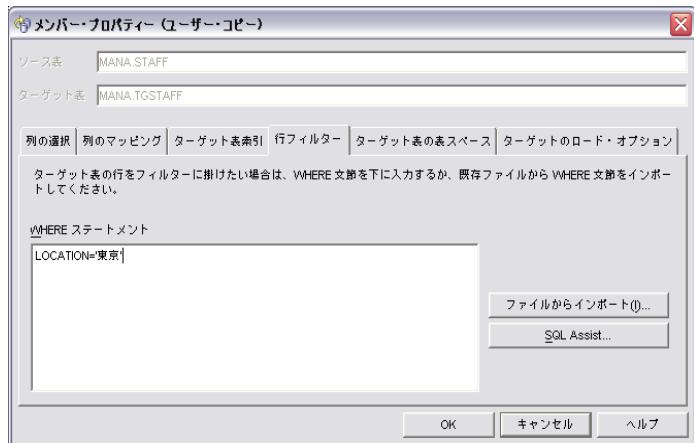
■APPLYのレプリケーションの条件指定

- レプリケーション・センターでは、メンバー・プロパティの行フィルターで指定

例：LOCATIONが‘東京’のものだけ、レプリケーションしたい

- 指定した条件はSUBS_MEMBER表のPREDICATES列の値として入る

- ▶ここに指定された条件は差分コピー時だけでなく、FULLREFRESH時にも適用される
- ▶CD表やUOW表にしか存在しない列に対する条件を指定すると、FULLREFRESH時にエラーになってしまう
⇒UOW_CD_PREDICATES列を利用



解説：

- 指定した条件は、SUB_MEMBER表のPREDICATES列の値に入る

```
INSERT INTO ASN.IBMSNAP_SUBS_MEMBER (
  APPLY_QUAL, SET_NAME, WHOS_ON_FIRST,
  SOURCE_OWNER, SOURCE_TABLE, SOURCE_VIEW_QUAL,
  TARGET_OWNER, TARGET_TABLE, TARGET_STRUCTURE,
  TARGET_CONDENSED, TARGET_COMPLETE,
  PREDICATES, UOW_CD_PREDICATES, JOIN_UOW_CD,
  MEMBER_STATE, TARGET_KEY_CHG, LOADX_TYPE,
  LOADX_SRC_N_OWNER, LOADX_SRC_N_TABLE
) VALUES (
  'TEST',
  'TEST',
  'S',
  'MANA',
  'STAFF',
  0,
  'MANA',
  'TGSTAFF',
  8,
  'Y',
  'Y',
  'LOCATION='東京'
  null,
  null,
  'N',
  'N',
  null,
  null,
  null
);
```

CD表またはUOW表固有の列の値によるフィルタリング

■ PREDICATES列の問題点

- 削除のデータはレプリケーションしたくない場合などは、CD表のIBMSNAP_OPERATION列に条件を指定すればよい。
- しかしCDまたはUOW表固有のカラムをAPPLYのレプリケーション条件(PREDICATES)に指定した場合、FULLREFRESHを手動で実行しないと、そのようなカラムがソース表には存在しない為、SQLエラー(SQLCODE -206)が発生

■ 解決策 : UOW_CD_PREDICATES列を利用

- SUBS_MENBR表のUOW_CD_PREDICATES列に条件を指定
 - Fullrefresh時、APPLYはUOW_CD_PREDICATES列で指定された条件は無視する
 - 差分反映時、PREDICATES列およびUOW_CD_PREDICATES列を結合した条件でSQL文を実行する
 - ターゲット表がUSERCOPYでUOW_CD_PREDICATES列にUOW表の列(IBMSNAP_AUTHIDなど)を条件に指定する場合は、SUBS_MENBR表のJOIN_UOW_CD列を'Y'に設定する必要がある
 - レプリケーションセンターではPREDICATEは指定可能であるがUOW_CD_PREDICATESは手動で設定する必要がある

解説:

■ UOW_CD_PREDICATESの使用例

サブスクリプション定義後に手動で設定する

```
UPDATE ASN. IBMSNAP_SUBS_MEMBR SET UOW_CD_PREDICATES =
  'IBMSNAP_OPERATION <> ''D''
 WHERE APPLY_QUAL='QUAL1';
```

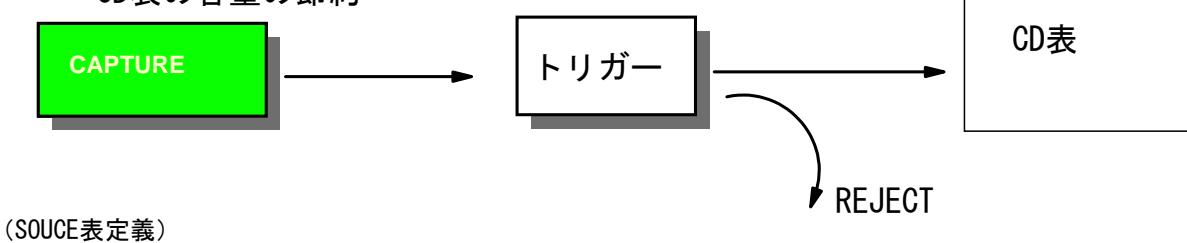
APPLYの実行時

```
-----
 mem_info i = 0
-----
 SOURCE_OWNER      = AZUMA
 SOURCE_TABLE      = TEST_SOURCE
 SOURCE_VIEW_QUAL  = 0
 TARGET_OWNER      = AZUMA
 TARGET_TABLE      = TGTEST_SOURCE
 TARGET_CONDENSED  = Y
 TARGET_COMPLETE   = Y
 TARGET_STRUCTURE  = 8
 PREDICATES       = null
 MEMBER_STATE     = S
 TARGET_KEY_CHG   = N
 JOIN_UOW_CD      is null
UOW_CD_PREDICATES = IBMSNAP_OPERATION <> 'D'
LOADX_TYPE        is NULL
-----
```

CD表に書き出す時点でのフィルタリング

■特定のデータをCD表に書かせないTriggerを作成

- レプリケーション対象でないデータがターゲット側に取込まれることを防止
 - 例：削除のデータはCD表にも書き出したくない
 - CD表の容量の節約



```

CREATE TABLE TEST_SOURCE
(ROW_NUMBER SMALLINT NOT NULL,
TEST_DATA SMALLINT,
DESCRIPTION CHAR(20))
DATA CAPTURE CHANGES ;
CREATE UNIQUE INDEX TEST_SOURCE_X ON TEST_SOURCE
(ROW_NUMBER);

INSERT INTO TEST_SOURCE VALUES (1,1,'this is row');
INSERT INTO TEST_SOURCE VALUES (5,5,'this is row5');
INSERT INTO TEST_SOURCE VALUES (6,6,'this is row6');
DELETE FROM TEST_SOURCE;
  
```

(トリガー例)

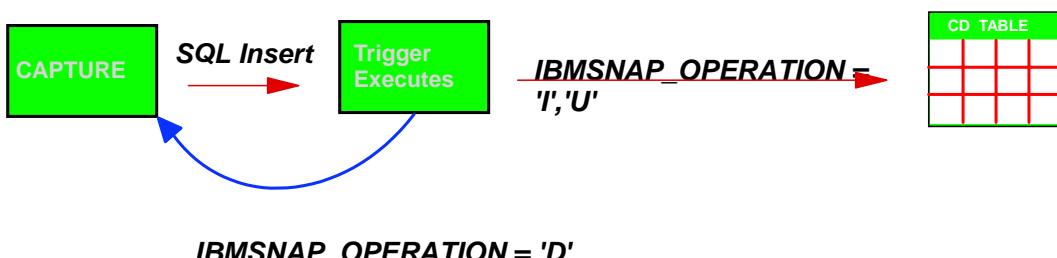
```

CREATE TRIGGER CD_FILTER_TRIGGER
NO CASCADE BEFORE INSERT ON CD01
REFERENCING NEW AS N
FOR EACH ROW MODE DB2SQL
WHEN (N. IBMSNAP_OPERATION = 'D')
SIGNAL SQLSTATE '99999'
  
```

```
SET MESSAGE_TEXT = 'CD INSERT %'
```

解説：

- 特定の変更がターゲット側に取込まれることを防止
 - DELETEの反映を行ないたくない場合などに利用可能



レプリケーション・データの加工

■ レプリケーション・データの加工方法

● APPLYのレプリケーション実行時に加工

- ▶ 例：NAMEの先頭3バイトだけ、レプリケーションしたい
- ▶ 方法：ソース表（CD表）の列に対する関数や演算の利用
レプリケーション・センターで指定可能

● CD表に書き出す時点での加工

- ▶ 例：CD表にデータが入る時点で、連番をふりたい
- ▶ 方法：CD表にトリガーを作成

● ターゲット表を更新する時点での加工

- ▶ 例：ソース側はITEM_NOしかないが、ターゲット表にはターゲット側にあるマスター表と
付き合わせたITEM_NAMEの値も入れたい
- ▶ 方法：ターゲット表にトリガー

✓ トリガーはレプリケーションの負荷への影響を考慮すること



解説：

- レプリケーション・データの加工方法は、以下の3つの時点で可能です。

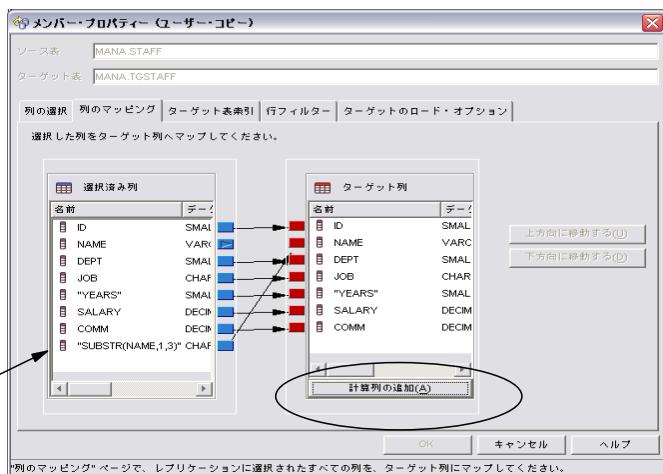
- ▶ APPLYのレプリケーション実行時に加工
- ▶ CD表に書き出す時点での加工
- ▶ ターゲット表を更新する時点での加工



APPLYのレプリケーション実行時に加工

■ ソース表（CD表）に含まれる列に対する関数や演算を指定

- レプリケーション・センターでは、メンバー・プロパティの列のマッピングで計算列を追加
 - ▶ 例：NAMEの先頭3バイトだけ、レプリケーションしたい
計算列として、SUBSTR (NAME, 1, 3) を定義
- 追加した列は、ターゲット列に追加列として定義しても、既存の列とマッピングしてもよい



解説：

- 加工が、ソース表またはCD表に対するSQL操作の範囲であれば、レプリケーション実行時に加工ができます。
- 定義は列のマッピングで行います。

CD表に書き出す時点での加工

■ 変更データを加工しCD表に書かせるTriggerを作成

- 例：CD表に連番をデータとして割り振る



(SOURCE表定義)

```
CREATE TABLE TEST_SOURCE
(ROW_NUMBER SMALLINT NOT NULL,
TEST_DATA SMALLINT,
DESCRIPTION CHAR(20))
DATA CAPTURE CHANGES ;

CREATE UNIQUE INDEX TEST_SOURCE_X ON TEST_SOURCE
(ROW_NUMBER) ;

INSERT INTO TEST_SOURCE VALUES (1,1,'this is row');
INSERT INTO TEST_SOURCE VALUES (5,5,'this is row5');
INSERT INTO TEST_SOURCE VALUES (6,6,'this is row6');
```

(トリガー例)

```
create sequence azseq start with 0%  
  
create trigger cdtrg1  
no cascade  
before insert on cd01  
referencing new as new  
for each row mode db2sql  
begin atomic  
set new.description = 'changed  
datarow' || char(nextval for azseq);  
end%
```



解説：

■CD表にデータとして連番をふる

```
C:\v7test>DB2 INSERT INTO TEST_SOURCE VALUES (12, 12, 'DATA1')  
DB20000I The SQL command completed successfully.
```

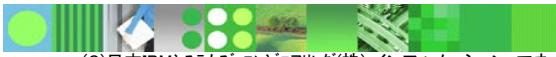
C:\v7test>DB2 INSERT INTO TEST_SOURCE VALUES (12, 12, 'DATA1')
SQL0803N One or more values in the INSERT statement, UPDATE statement, or
foreign key update caused by a DELETE statement are not valid because the
primary key, unique constraint or unique index identified by "1" constrains
table "AZUMA.TEST_SOURCE" from having duplicate rows for those columns.
SQLSTATE=23505

C:\>DB2 INSERT INTO TEST_SOURCE VALUES (13, 13, 'DATA1')
DB20000I The SQL command completed successfully.

C:\\$y7test>DB2 SELECT * FROM CD01 WITH UR

IBMSNAP_UOWID ACTION ROW_NUMBER	IBMSNAP_INTENTSEQ XROW_NUMBER TEST_DATA	IBMSNAP_OPER XDESCRIPTION
x'000000000000000000002175'	x'00000000000004299508'	
12	- 12 - changed data row0	-
x'000000000000000000002195'	x'00000000000004299AF1'	
13	- 13 - changed data row2	-

2 record(s) selected.



ターゲット表を更新する時点での加工

■ ターゲット表にTriggerを作成

- 例：ソース側はITEM_NOしかないが、ターゲット表にはターゲット側にあるマスター表と付き合わせたITEM_NAMEの値も入れたい



(トリガー例)

```
create trigger ins_trg
no cascade before insert on tgitem_no
referencing new as n_row
for each row
when (n_row.item_no is not null)
begin atomic
  set n_row.item_name=(select item_name from
item_no_name where item_no=n_row.item_no);
end
%
```

```
create trigger upd_trg
no cascade before update on tgitem_no
referencing new as n_row
for each row
when (n_row.item_no is not null)
begin atomic
  set n_row.item_name=(select item_name from
item_no_name where item_no=n_row.item_no);
end
%
```

解説：

- ターゲット表にトリガーを作成して、データ加工を行うことも可能です。
- ターゲット表には、必要に応じてINSERTトリガー、UPDATEトリガー、DELETEトリガーを作成する必要があります。
- ただしレプリケーション実行時にトリガーが発行されるため、レプリケーション時間に対する影響や、トリガー実行が失敗すると、それまでのレプリケーションは全てロールバックされるため、それらの対応を考慮する必要があります。

レプリケーション・ターゲット・キーの更新への対応

■ レプリケーション・ターゲット・キーが更新された場合

レプリケーション・キーとは、IS_KEY=Y
が設定されている列の組合せ

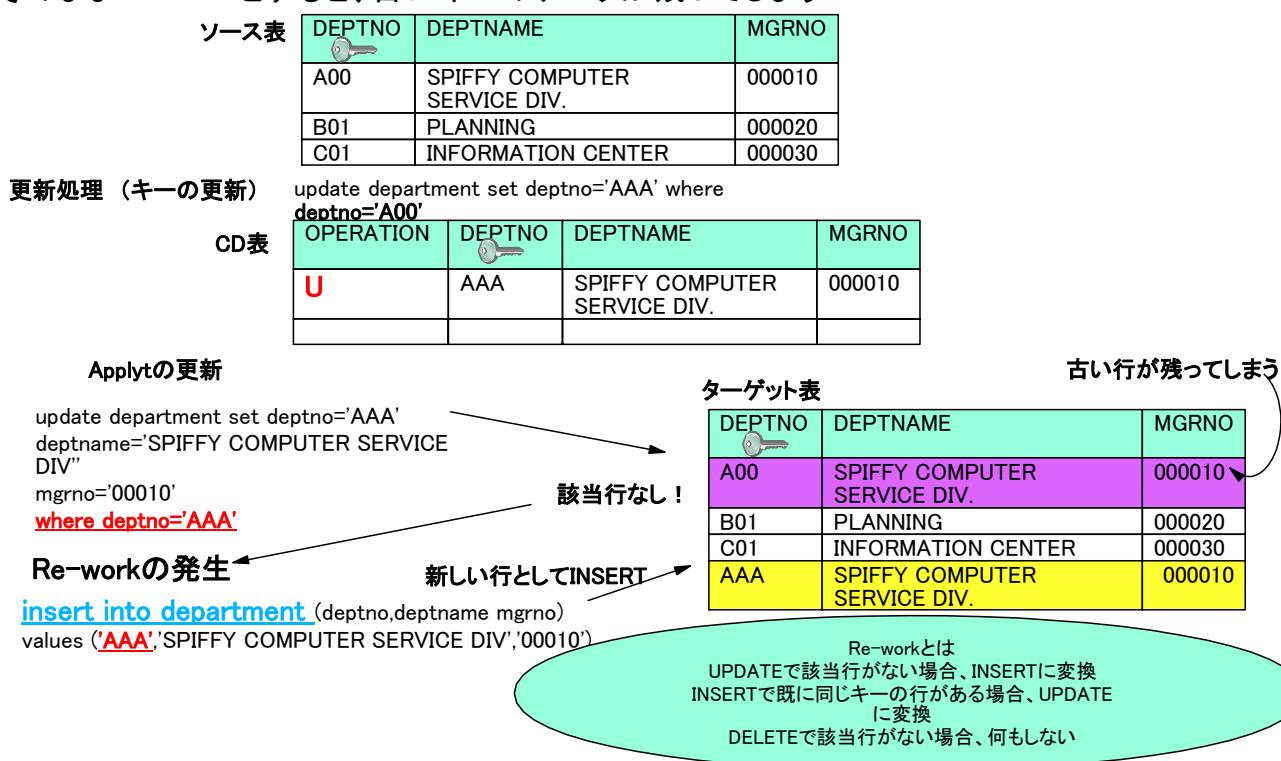
- アプライはレプリケーション・ターゲット・キーとして指定された列(通常はプライマリー・キー)を選択条件に指定して、ターゲット表のUPDATEを行う
 - ▶ この時条件に指定する値は、通常CD表に入っている更新後の値
 - ▶ ただしこれはレプリケーション・ターゲット・キーは更新されないということ前提としている
 - ▶ **レプリケーション・ターゲット・キーが更新された場合、該当行が存在しないことになる**
 - ▶ この場合、アプライはCD表にある更新後の列の値でINSERTを行う
 - ▶ レプリケーション・ターゲット・キーが更新されINSERTに置き換わると、古いキーのデータがそのまま残ってしまう
- これを回避するための機能
 - ▶ 機能①: CHG_UPD_TO_DEL_INS=Y (REGISTER表の列)
 - UPDATEの情報をCD表に書き出す時にDELETEとINSERTに変更
 - ▶ 問題点
 - CD表のボリュームが増加
 - ソース表(CD表)単位でしか、指定できない
 - DELETE RESTRICTの参照制約があり、従属性が存在すると、親の行は削除できない
 - ▶ 機能②: TARGET_KEY_CHG=Y (SUBS_MEMBR表の列)
 - ターゲット表を更新する場合に更新前のプライマリー・キー値を検索条件にする
 - ▶ 利点
 - メンバーごとに指定が可能

解説:

- レプリケーション・ターゲット・キー(プライマリー・キー)の変更
- サブスクリプション・セット・メンバーを作成する場合、ターゲット表で使用する主キーまたはユニーク索引として使用するため、ターゲット表の列を選択します。これらの列はターゲット・キーを構成するものとして考えられます。これらの列を、ソース表の主キーまたはユニーク索引とマッチさせることができます。これらの列は、ソース・キーとして考えられます。ただし、ターゲット・キーの列をソース・キーの列とマッチさせる必要はありません。
- アプライ・プログラムは、非コンデンス CCD 表、および関連付けされたソース表がフル・リフレッシュ・レプリケーション用に登録されているターゲット表以外では、すべてのターゲット表でターゲット・キーを必要とします。アプライ・プログラムがターゲット表に変更を適用する場合、更新するべき正しい行を検出する必要があります。ターゲット・キーの列にある値を、ソース表の関連付けされた列の値とマッチさせて、行を検出します。
- ただし、ターゲット・キーにある列として使用するソース表の列に変更がある場合は、アプライ・プログラムは更新すべきターゲット表の行をターゲット・キーで探し出すことができません。
- この問題を回避するため、レプリケーション・ソースとして表を登録するときに、どちらの列をターゲット・キーとして使用するのかを決めてください。
- ユーザーのアプリケーションが、ソース表の関連付けられた列を変更することが予測される場合は、以下の2つの方法で対応してください。
 - ▶ 機能①: CHG_UPD_TO_DEL_INS=Y (REGISTER表の列)
 - UPDATEの情報をCD表に書き出す時にDELETEとINSERTに変更
 - ▶ 機能②: TARGET_KEY_CHG=Y (SUBS_MEMBR表の列)
 - ターゲット表を更新する場合に更新前のプライマリー・キー値を検索条件にする
 - この場合、更新が予想される列について変更後イメージと同様に変更前イメージも登録してください。
 - アプライ・プログラムがターゲット表の正しい行を探し出そうとしており、そのターゲット・キーに使用しているソース列の値が変更されている場合、アプライ・プログラムは変更前イメージをターゲット・キーとマッチさせることによって行を探し出すことができます。

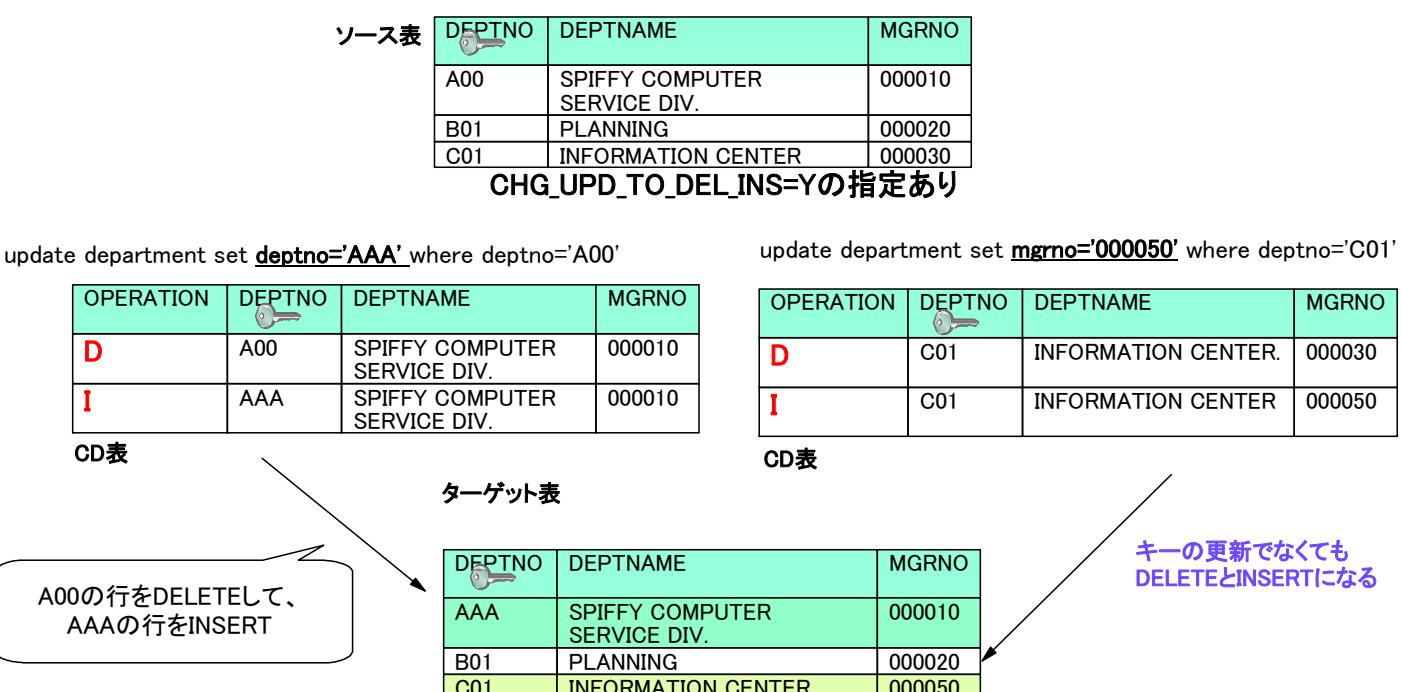
解説:

- ターゲット表のレプリケーション・キーになっている列が更新された場合の問題点
 - そのままUPDATEとすると、古いキーのデータが残ってしまう



解説:

- 機能① : UPDATEをDELETEとINSERTに変更する方法での回避
 - CHG_UPD_TO_DEL_INS=Yの指定による



解説:

- 機能②: ターゲット表を更新する場合に、更新前のレプリケーション・キー値を検索条件にする方法での回避
 - TARGET_KEY_CHG=Yの指定による

ソース表	DEPTNO	DEPTNAME	MGRNO
	A00	SPIFFY COMPUTER SERVICE DIV.	000010
	B01	PLANNING	000020
	C01	INFORMATION CENTER	000030

TARGET_KEY_CHG=Yの指定あり

update department set deptno='AAA' where deptno='A00'

update department set mgrno='000050' where deptno='C01'

OPERATION	XDEPTNO (変更前)	DEPTNO (変更後)	DEPTNAME	MGRNO	OPERATION	XDEPTNO (変更前)	DEPTNO (変更後)	DEPTNAME	MGRNO
U CD表	A00	AAA	SPIFFY COMPUTER SERVICE DIV.	000010	U CD表	C01	C01	INFORMATION CENTER	000050

CD表にはキーの変更前イメージを
保管する列を作成しておく必要がある
ターゲット表

update department set
deptno='AAA'
deptname='SPIFFY COMPUTER
SERVICE DIV.'
mgrno='000010'
where deptno='A00'

変更前のキー値で更新

DEPTNO	DEPTNAME	MGRNO
AAA	SPIFFY COMPUTER SERVICE DIV.	000010
B01	PLANNING	000020
C01	INFORMATION CENTER	000050

update department set
deptno='C01'
deptname='INFORMATION
CENTER',
mgrno='000050'
where deptno='C01'

ターゲット表にはキーの変更前イメージを保管する列は
含まなくてよい

解説:

- このページはブランクです

TARGET_KEY_CHG

■ BEFORE-IMAGEの値を検索条件にして更新(UPDATE)を実行する

- 通常(TARGET_KEY_CHG=N) APPLYは
 - ▶ UPDATE TARGET_TABLE SET ALL_COLS=NEW_COLS WHERE KEY VALUE=NEW KEYで更新

- TARGET_KEY_CHG=Yでは
 - ▶ UPDATE TARGET_TABLE SET ALL_COLS=NEW_COLS WHERE KEY VALUE=OLD KEYで更新
 - ▶ REWORKEDは発生しない
 - ▶ SOURCE表のKEYの更新があるサブスクリプションが対象となる
 - ▶ CHG_UPD_TO_DEL_INSと同時に使用することはできない
 - ▶ IS_KEY=Yの列が計算列の場合には使用できない
 - ▶ CD表BEFORE-IMAGEを含めるようにRegistration定義が必要
 - ▶ SUBS_COLS表にBEFORE-IMAGEカラムをCOL_TYPE='P'でSubscription定義が必要
 - ▶ TARGET表にはBEFORE-IMAGEカラムは必要なし

解説:

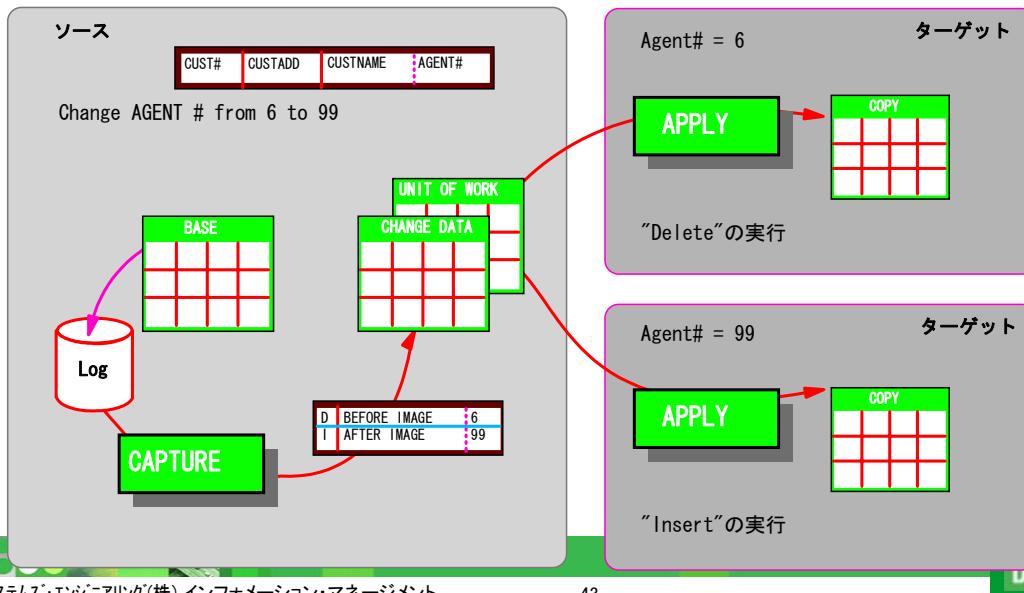
■ SUBS_COLS表の定義

```
INSERT INTO ASN.IBMSNAP_SUBS_COLS (
  APPLY_QUAL, SET_NAME, WHOS_ON_FIRST,
  TARGET_OWNER, TARGET_TABLE, TARGET_NAME,
  COL_TYPE, IS_KEY, COLNO, EXPRESSION
) VALUES (
  'Q1',
  'S1',
  'S',
  'AZUMA',
  'TGTEST_SOURCEXX',
  'XROW_NUMBER',
  'P',
  'N',
  0,
  'XROW_NUMBER'
);
```

TARGET_KEY_CHGでなく、CHG_UPD_TO_DEL_INSが必要な例

■ レプリケーション・キーでなく、レプリケーション条件として指定している列 (PREDICATES) の値が変更になる場合

- 例 : AGENTの番号によってターゲット表を分けています
 - ▶ AGENT番号がUPDATEされると、更新後のAGENT番号に該当する方の表へのレプリケーションされる。
=>古いAGENT番号の表に、データが残ってしまう。
- この場合、TARGET_KEY_CHGでは対応できないので、UPDATEをDELETEとINSERTに変更する
CHG_UPD_TO_DEL_INS=Yの設定が必要



解説:

- レプリケーション・キーでなく、レプリケーション条件として指定している列 (PREDICATES) の値が変更になり、変更後の値がレプリケーションの条件と合致しなければ、その更新データはレプリケーションされません。
- 更新後のデータに合致する条件が指定されたサブスクリプションがあった場合は、そのUPDATEの情報がレプリケーションされ、そこではINSERTにREWORKされ、新規のレコードが発生します。
- この場合、TARGET_KEY_CHGでは古い情報が残ってしまうことを避けることはできないため、UPDATEをDELETEとINSERTに変更するCHG_UPD_TO_DEL_INS=Yの設定が必要です。

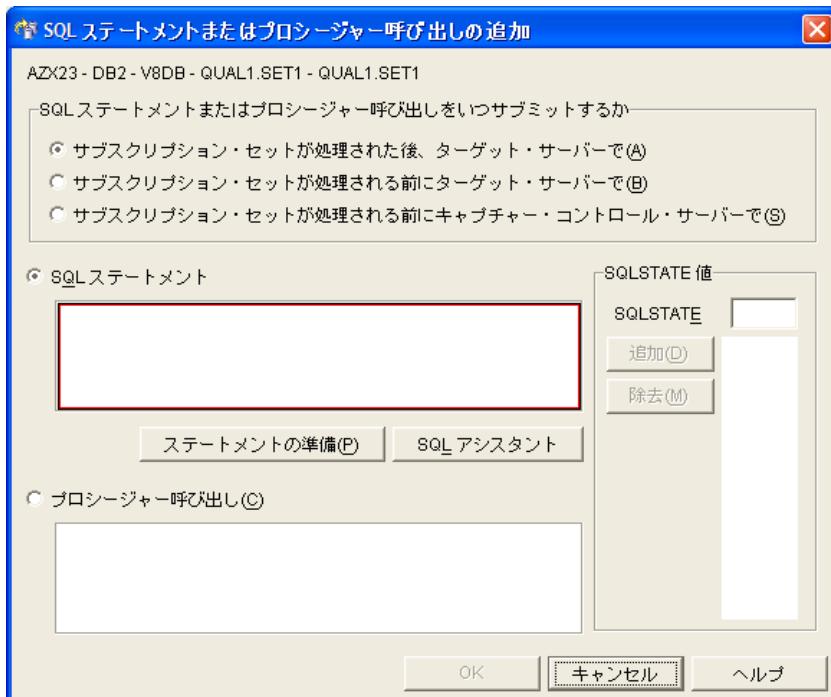
- 
- ▶ *Replica (Update Anywhere)*
 - ▶ レプリケーション対象データの操作
 - ▶ **レプリケーション処理の操作**
 - ▶ データ共用・区分化対応
 - ▶ 特殊な表、データタイプのレプリケーション
 - ▶ マルチベンダー・レプリケーション
 - ▶ その他

内容

- レプリケーション処理前後の操作
-SQL-Before/After
- 変更データのブロック化
-MAX_SYNCH_MINUTES
- メンバーのステータス情報
-MEMBER_STATE
- APPLYのエラー操作
-SQLERRORCONTINUE
- サブスクリプション情報のキャッシュ化
-OPT40NE

レプリケーション処理前後の操作

■SQL-Before/After機能



● Replicationの実行前・後に実行するSQL文またはStored Procedureの複数指定が可能

● Stored Procedureの指定の場合はIN/OUTパラメータ指定不可

● SQL文の実行結果でERRORにしたくないSQLSTATEを複数指定

解説:

■レプリケーション前後の実行時処理

Applyプログラムが複製サブスクリプションを処理する前または後に、SQL Statementおよびストアード・プロシージャを使用して実行時処理Statementを定義することができます。

この機能は、CCD表の整理や複製サブスクリプションが処理される順序の制御に役立ちます。

実行時処理Statementは、サブスクリプションの処理前にソース・サーバーにおいて、およびサブスクリプションの処理後にソース・サーバーとターゲット・サーバーにおいて実行できます。たとえば、データの複製前にソース・データを変換したり、コピー後にターゲット・データを変換したりすることができます。

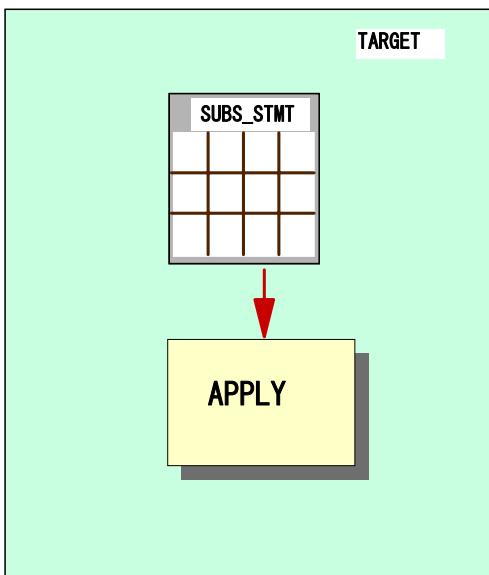
ストアード・プロシージャは、パラメーターなしのSQL CALL Statementを使用します。

実行時プロシージャは、単一の作業単位内でまとめて実行されます。

さらに、それぞれの処理Statementごとに受け入れ可能 SQLSTATE を定義することもできます。

ストアード・プロシージャの呼び出しが、処理前または後のStatementを介して行うことができますが、もっと複雑な変換を行うには、トリガーを使うこともできます。

SQL-Before/AfterとSUBS_STMTS表



SUBS_STMTS

SET_NAME	from SUBS_SET
BEFORE_OR_AFTER	A. AFTEE SQL STATEMENT at TARGET B. BEFORE SQL STATEMENT at TARGET S. BEFORE SQL STATEMENT at
SOURCE	G. For Non-IBM Replication X. For Non-IBM Replication
STMT_NUMBER	Relative order number within scope of BEFORE_OR_AFTER
EI_OR_CALL	E. SQL_STMT by EXEC SQL IMMEDIATE C. SQL_STMT by EXEC SQL CALL
ACCEPT_SQLSTATES	5 bytes SQLSTATE
SUBS_SET	
AUX_STMTS	0. Absence OF SUBS_STMTS n. The number of SUBS_STMT rows

- SQL-BEFOREはReplicationとは別LUW, SQL-AFTERは同じLUWでの実行になる

解説:

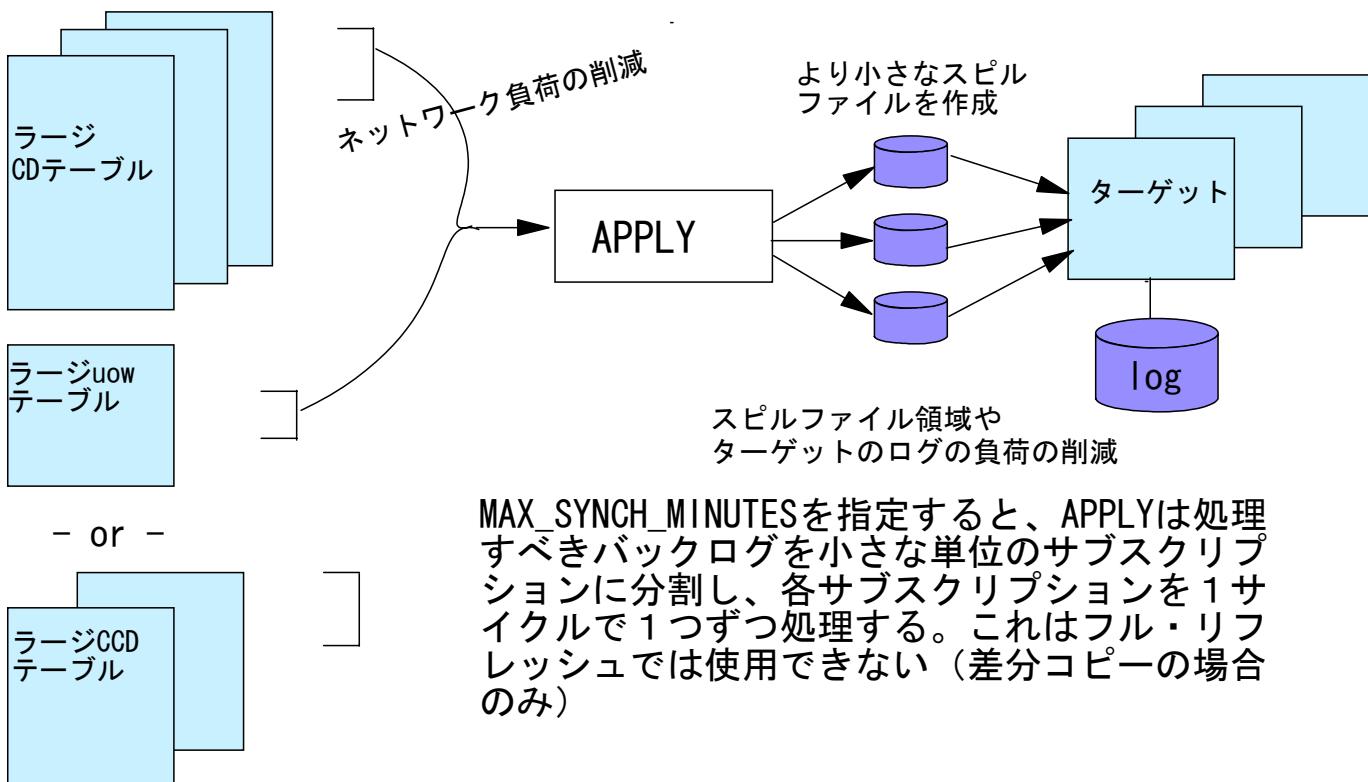
ASN. IBMSNAP_SUBS_STMTS

この表には、複製サブスクリプション・セットで実行されるステートメントに関する情報が入ります。ステートメントまたはストアード・プロシージャーを指定していない場合、ステートメント表には行が入りません。

APPLY_QUAL	このサブスクリプションを実行するプラットフォーム・インスタンス用の Applyプログラムを識別
SET_NAME	サブスクリプション・セットを指定します。
BEFORE_OR_AFTER	以下のことを示すフラグ。 A ステートメントは、すべての応答セット行が適用された後、ターゲット・サーバーで実行されます。 B ステートメントは、応答セット行が適用される前に、ターゲット・サーバーで実行されます。 S ステートメントは、応答セット・カーソルのオープン前に、ソース・サーバーで実行されます。
STMT_NUMBER	BEFORE_OR_AFTER の範囲内での実行の相対順序を定義します。
EI_OR_CALL	以下のことを示すフラグ。 E SQL ステートメントは、ターゲット・サーバーで EXEC SQL EXECUTE IMMEDIATE として実行されます。 C SQL ステートメントには、ターゲット・サーバーで EXEC SQL CALL として実行されるストアード・プロシージャー名が入っています。
SQL_STMT	以下のいずれかの値が入ります。 ステートメント EI_OR_CALL = 'E' の場合に EXEC SQL EXECUTE IMMEDIATE ステートメントとして実行される SQL ステートメント。 プロシージャー EI_OR_CALL = 'C' の場合に EXEC SQL CALL ステートメントとして実行される、パラメーターまたは CALL キーワードなしの SQLストアード・プロシージャーの 8 バイト名。
ACCEPT_SQLSTATES	サブスクリプションの定義時に指定した 1~10 個の 5 バイト SQLSTATE 値。

変更データのブロック化

■ MAX_SYNCH_MINUTES



解説：

■ 変更データのブロック化

大量の変更データを含んだ CD または CCD 表からのレプリケーションでは変更データの反映がしばしば問題になることがあります。

- ターゲット側のDB2アクティブログをFULLにする
- 予備ファイル(SPILL FILE)のオーバーフローさせる

長期にわたる Apply プログラムの停止（たとえば、故障）のために変更データの大きなブロックが CD 表に累積している場合、そのデータのブロックを分割すると、LOGやSPILLファイルのオーバーフローを避けられます。

サブスクリプション・セットの作成で「データブロック化係数」を指定すると、何分間分の変更データをサブスクリプション・サイクルに移すことができるかを指定できます。

指定した分数は、データ・ブロックのサイズを判別するのに使用されます。

変更データの累積がデータ・ブロックのサイズよりも大きい場合、Apply プログラムは1つのサブスクリプション・サイクルを複数のミニサイクルに変換して、バックログを管理可能な量に削減します。

Apply プログラムは、予備ファイルまたはログのオーバーフローを起こさないブロックにコピーするために、作業負荷レベルへの適合を自動的に3回再試行します。

作業負荷に合わせるために、Apply プログラムは、値を半分にカットし、複製が正常に完了するよう応答セットを十分小さいサイズにします。

まだ問題が続く場合、値を4分の1にし、変更データ・バックログを、使用可能なシステム資源に合わせます。それによって、ネットワークおよび DBMS 上の負担が減り、故障のリスクが少なくなります。

ミニサイクルのいずれかにおいて障害が発生した場合、障害が起こったミニサイクルだけを再実行すれば済みます。Apply プログラムが後で複製サブスクリプションを再試行するときには、最後に成功したミニサイクル点から続けて再開します。

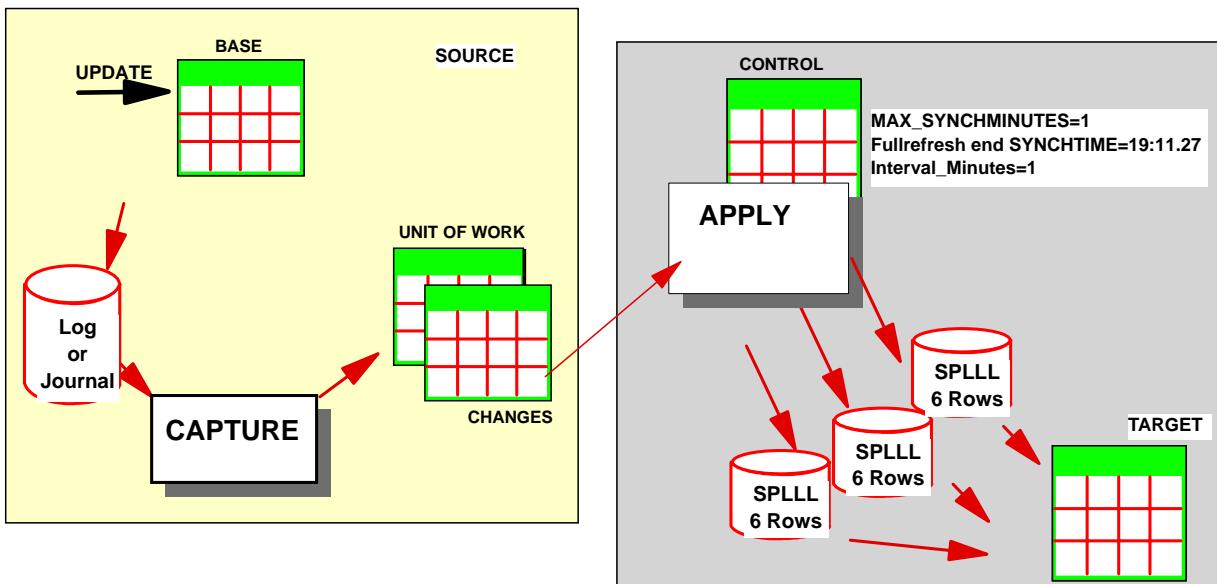
データ・ブロック化のデフォルトは NULL です。この NULL のデフォルト値では、サイズに関係なく、使用可能なすべてのコミット済みデータがコピーされます。設定する分数

は、サブスクリプションのすべてのトランザクションが、SPILLファイルのオーバーフローやログの満杯条件を起こすことなくコピーされるよう、十分に小さくしなければなりません。

SPILLファイルに必要なスペースは、複製サブスクリプションで設定された各応答セットに必要なスペースを合計した大きさが必要となります。

サブスクリプション・セット制御表の列MAX_SYNCH_MINUTES には、指定した値が入ります。

MAX_SYNCH_MINUTESのTEST結果



```
select SET_INSERTED, SET_UPDATED, SET_DELETED, STATUS, LASTRUN, LASTSUCCESS, SYNCPOINT, SYNCTIME from asn.ibmsnap_applytrail order by lastrun
```

SET_INSERTED	SET_UPDATED	SET_DELETED	STATUS	LASTRUN	LASTSUCCESS	SYNCPOINT	SYNCTIME
6	0	0	0	2002-09-01-19.11.27.864002	2002-09-01-19.11.27.864002	-	2002-09-01-19.11.27.894000
0	6	0	2	2002-09-01-19.16.28.857001	2002-09-01-19.16.27.864002	x'3D71E8130000000000000'	2002-09-01-19.12.35.000000
0	6	0	2	2002-09-01-19.16.29.128000	2002-09-01-19.16.29.128000	x'3D71E8520000000000000'	2002-09-01-19.13.38.000000
0	6	0	0	2002-09-01-19.16.29.168002	2002-09-01-19.16.29.168002	x'3D71E8C80000000000000'	2002-09-01-19.15.36.000000

解説:

- 上記の例ではCD表に書かれた18行の変更データが3つのミニサイクルに分割され6行ずつ差分反映されたことを示す。
- APPLYTRAIL上MAX_SYNCH_MINUTESが使用された場合はSTATUS=2で報告される
- 最後のミニサイクルはSTATUS=0
- FP9 Applyより、RCの定義時の省略値はNULL
 - >不要なuow表へのアクセス、パフォーマンストラブルの改善の為
- FP6 ApplyよりMAX_SYNCH_MINUTES=NULLであっても"Self Healing機能追加
 - >Log_FullにHitすると自動的にMAX_SYNCH_MINUTES=20を設定し再実行
 - >問題が解決しない場合MAX_SYNCH_MINUTES=10を設定し再実行

メンバーのステータス情報

■ MEMBER_STATE (Fixpak2からの新機能)

SUBS_MEMBER表のMEMBER_STATE列は、各メンバーの4つの状況を示す

- N (NEW)

▶ そのSubscription Set内で初期状態を示す (Subscription Set定義直後)

- L (Loaded)

▶ Fullrefresh直後の状態、また変更データはターゲットへ反映していない

- S (Synchronized)

▶ N→L状態からそのSubscription Set内すべてのMemberが同期が取られた状態。変更データの反映が実行可能または実行中のステータス

- D (Disabled)

▶ そのメンバーはSET内の処理からスキップされる

意図的に特定のメンバー処理のみ停止したい場合に利用可能

* Apply skips member with status *disabled*, until it's set back to *new*

To disable a member, user issues this sql statement (there is no GUI support currently):

```
UPDATE ASN.1BMSNAP_SUBS_MEMBER
```

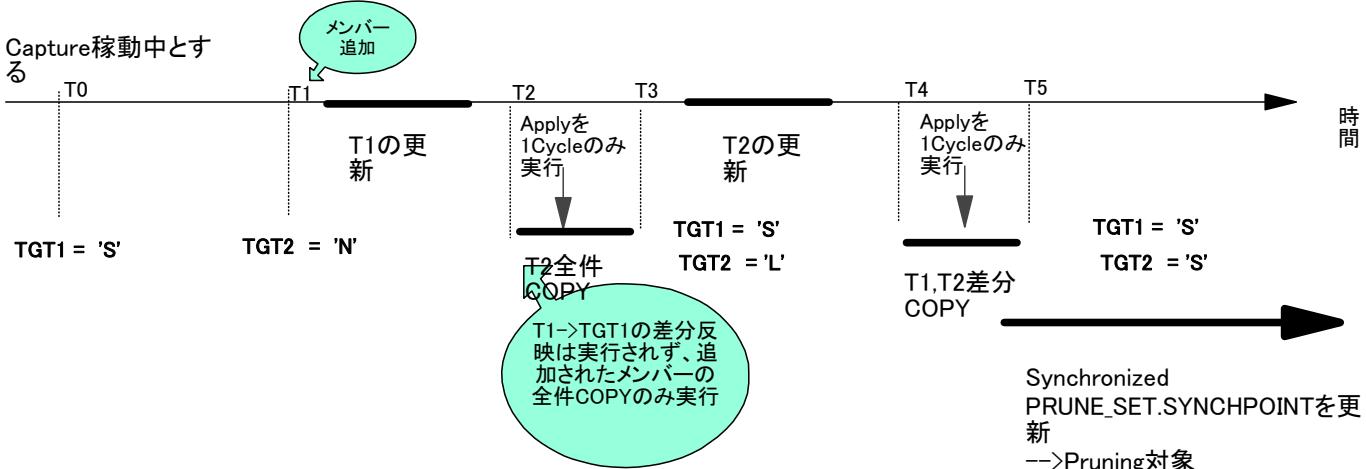
```
SET MEMBER_STATE = 'D'
```

```
WHERE APPLY_QUAL= 'QUAL1'
```

```
AND TARGET_TABLE='TGTEST_SOURCE2' ;
```

解説:

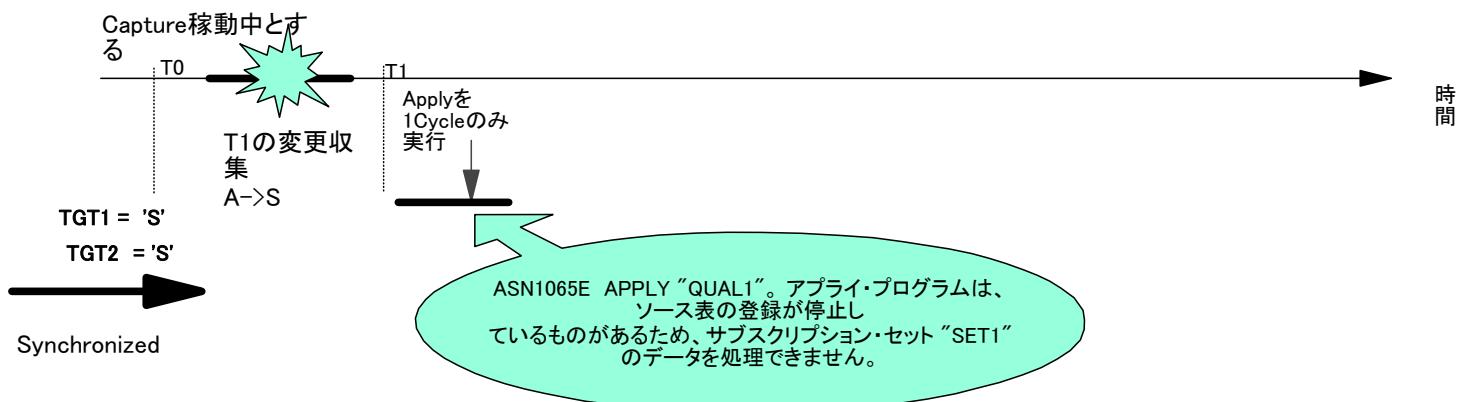
- すべてのメンバーが'S'になるまで変更反映はされない
同じセット内に2つのメンバー、T1→TGT1, T2→TGT2のReplicationを想定



- (1) もしsubs_set synchpointとsubs_set synctimeがNOT NULL --> member_stateはSとなる
- (2) もしsubs_set synchpointがNULLかつsubs_set synctimeがNOT NULL --> member_stateはLとなる
- (3) もしsubs_set synchpointとsubs_set synctimeがNULL --> member_stateはNとなる

解説:

- 'S' 状態でソース表のREGISTER.STATUSが"STOPPED"になると差分反映を中止(新規Message ASN1065E)
同じセット内に2つのメンバー、T1→TGT1, T2→TGT2のReplicationを想定



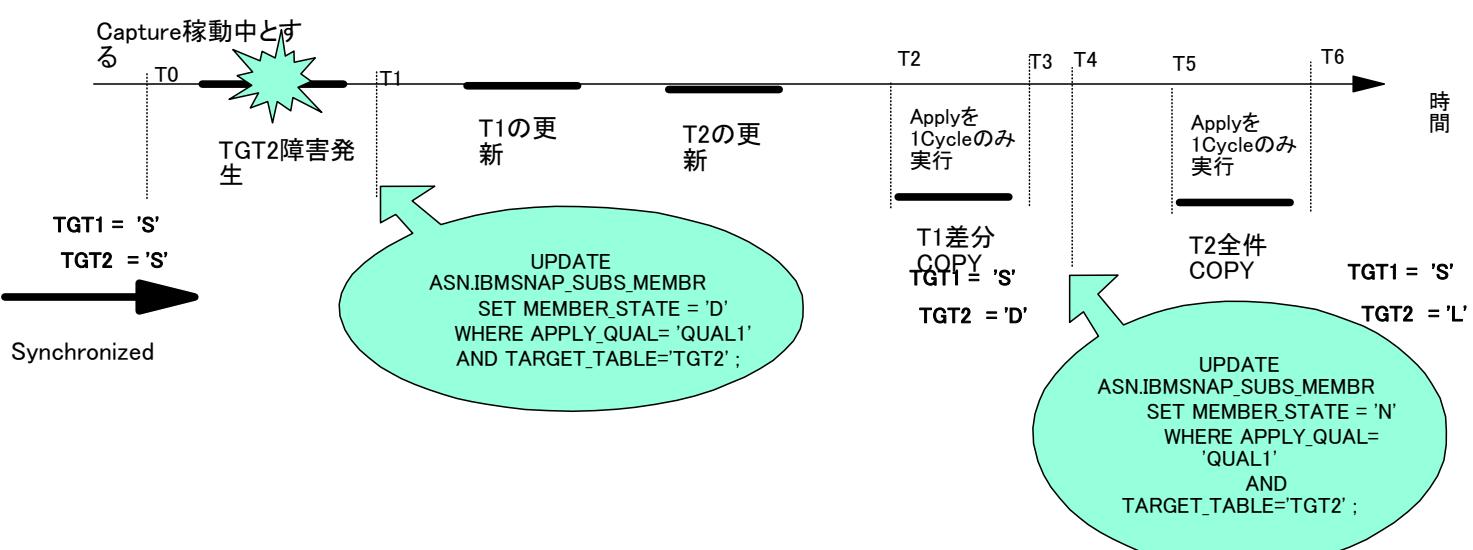
- 原因を取り除き(例: Data Capture Changes属性がAlterで除去された等)、手動でRegister表を更新する必要がある

エラーの登録情報をノーマルへ戻す為には下記のSQLを発行し、CAPTUREのREINITあるいは停止、再始動を行う必要がある

```
update asn.ibmsnap_register
set state='I'
where state='S'
and source_table='TEST_SOURCE';
```

解説:

- 特定のメンバーをDisableにする
同じセット内に2つのメンバー、T1→TGT1, T2→TGT2のReplicationを想定



APPLYのエラー操作

■SQLERRCONTINUE

- ApplyのStartup Option
- Apply実行時のエラー処理が変更される
 - ▶ SQLERRCONTINUE指定なしでは通常STATUS=-1が記録され5分毎にRetry
 - ▶ 指定ありでは<APPLY_QUAL>.SQSファイル内に指定されたSQLSTATEであればERRORを<APPLY_QUAL>.ERRに記録し、STATUS=0が記録され処理はロールバックされない。
- <APPLY_QUAL>.SQSファイル内には20個までの5桁のSQLSTATEを指定しAPPLYに事前定義しておく
- “無視可能な”errorが発生した場合、STATUSは+16が記録され、処理はロールバックされない。“無視不可能な”errorが発生した場合STATUSは-1になり処理はロールバックされる
- デバッグ目的での使用が推奨されている、本番は不向き
- UNIX & Window環境のみサポートされる

解説:

- Target表にConstraintが作成されて反映したい変更にその違反があった場合(SQLSTATE 23513)が発生した場合 STATUS=16にて記録され、SYNCHPOINTは更新される

```

2003-05-04-08.43.05.685000 ASN10451 APPLY "QUAL1". アプライ・プログラムは、データベース "SAMPLE" を使って開始されました。
The NLS msg is ASN10451 APPLY "QUAL1". アプライ・プログラムは、データベース "SAMPLE" を使って開始されました。
INIT: sqlstate file C:\x23backup\%v8test\%QUAL1.SQS is opened
CPINIT: sqlstate string is 23513
INIT: sqlerror file C:\x23backup\%v8test\%QUAL1.ERR is opened
ApplyWorker: mbrStates = 0
--- Process next subscription (1) ---

.....
CLOS: activate = 1
CLOS: status = 0
CLOS: lastrun = 2003-05-04-08.43.06.246001
CLOS: lastsuccess = 2003-05-04-08.42.51.831002
CLOS: Synchpoint is 3eb45334000000010000
CLOS: synctime = 2003-05-04-08.42.52.316000
CLOS: apply_qual = QUAL1
CLOS: set_name = SET1
CLOS: sWhoIsOnFirst = S
PSET: need_ascp is 1; needSrcPcUpd is 0
PSET: currtsrvrType is SQL, currtsrvrVersion is 08
PSET: connect to SAMPLE; rc = 1
APS: Synchtime(T) is 2003-05-04-08.42.52.316000
PSET: connect to SAMPLE; rc = 1
PSET: currtsrvrType is SQL, currtsrvrVersion is 08
SAT: ASNLOAD = N, EFFECT_MEMBERS = 0
SAT: FULL_REFRESH = N
SAT: SET_INSERTED = 0
SAT: SET_DELETED = 0
SAT: SET_UPDATED = 0
SAT: SET_REWORKED = 0
SAT: SET_REJECTED_TRXS = 0
SAT: STATUS = 16
SAT: LASTRUN = 2003-05-04-08.43.06.246001
SAT: LASTSUCCESS = 2003-05-04-08.42.51.831002
SAT: SYNCHPOINT is 3eb45334000000010000
SAT: SYNTIME is 2003-05-04-08.42.52.316000

```

サブスクリプション情報のキャッシュ化

■ OPT4ONE

- APPLYのパフォーマンス向上の為、SUBS_MEMBER表とSUBS_COLS表の情報をメモリ内に持ちAPPLYの実行サイクル毎に両テーブル情報を読み込まない。
 - ▶ SUBS_MEMBERおよびSUBS_COLS表の変更はAPPLYが停止するまで認識されない
 - ▶ QUALIFIERあたり 1 SETが定義されている場合のみ有効
 - ▶ OPT4ONEを起動パラメーターで呼び出した場合、追加されたメンバーはApplyに認識されない

解説:

■ OPT4ONEの指定なしAPPLY TRACE

```
--- Process next subscription (1) ---
CEXPc: connect to V7DB
ABIND: sqlca.sqlcode is 0
CPGCST: Control server timestamp is
2002-04-21-09.52.09.251001
  RINES: No eligible named event subscription at this
moment
Compiled(P) at 11:57:27 on Dec 11 2001 (Level 70121a)
  CPGCI: numKeys is 1
  GMI: set type is READ_ONLY
  set_info
    ACTIVATE      = 1
    APPLY_QUAL   = QUAL1
    SET_NAME     = SET1
    WHOS_ON_FIRST = S
    SOURCE_SERVER = V7DB
    SOURCE_ALIAS  = V7DB
    TARGET_SERVER = V7DB
    TARGET_ALIAS  = V7DB
    STATUS        = 0
    LASTRUN       = 2002-04-21-09.51.59.467000
    REFRESH_TIMING = R
    SLEEP_MINUTES = 1
    EVENT_NAME    = null
    LASTSUCCESS   = null
    SYNCPOINT     = null
    SYNCHTIME     = null
    MAX_SYNCH_MINUTES is null
    AUX_STMTS     = 0
    ARCH_LEVEL    = 0201
  -----
  mem_info i = 0
    SOURCE_OWNER   = AZUMA
    SOURCE_TABLE   = TEST_SOURCE
    SOURCE_VIEW_QUAL = 0
    TARGET_OWNER   = AZUMA
    TARGET_TABLE   = TEST_SOURCE_PIT
    TARGET_CONDENSED = Y
    TARGET_COMPLETE = Y
    TARGET_STRUCTURE = 8
    PREDICATES    = TEST_DATA BETWEEN 1 and 200
  -----
  col_info i = 0
    COL_TYPE      = A
    TARGET_NAME   = ROW_NUMBER
    IS_KEY        = Y
    COLNO         = 0
    EXPRESSION    = ROW_NUMBER
  -----
  col_info i = 1
    COL_TYPE      = A
    TARGET_NAME   = TEST_DATA
    IS_KEY        = N
    COLNO         = 1
    EXPRESSION    = TEST_DATA
  -----
  col_info i = 2
    COL_TYPE      = A
    TARGET_NAME   = DESCRIPTION
    IS_KEY        = N
    COLNO         = 2
    EXPRESSION    = DESCRIPTION
  -----
PSET: Commit1 ok
```

解説:

■OPT4ONEの指定ありAPPLY TRACE (初回実行時ではない)

```
--- Process next subscription (2) ---
CEXP: connect to V7DB
CPGCST: Control server timestamp is
2002-04-21-09.52.09.291000
RINES: No eligible named event subscription at this
moment
RIRTS: No eligible relative timer driven subscription at
this moment
TTOS: slpDay is 0; slpHour is 0, slpMin is 0, slpSec is
59, slpMSec is 960001
TTOS1: sleep_time = 60
TTOS: delay_seconds = 60
Delay_seconds = 60 seconds
REST: Sleep time = 60 seconds.
MSGF: MsgNumber is 44.
The NLS msg is ASN10441: The Apply program will become
inactive for 1 minutes and 0 seconds.

SLP: Sleep for 60 seconds.
--- Process next subscription (3) ---
CEXP: connect to V7DB
CPGCST: Control server timestamp is
2002-04-21-09.53.09.297000
RINES: No eligible named event subscription at this
moment
Compiled(P) at 11:57:27 on Dec 11 2001 (Level 70121a)
-----  

set_info
-----  

APPLY_QUAL    = QUAL1
SET_NAME      = SET1
WHOS_ON_FIRST = S
```

```
PSET: Commit1 ok
CEXP: connect to V7DB
ABIND: sqlca.sqlcode is 0
CPGST: Source server timestamp is
2002-04-21-09.53.09.307000
LCKTB:1 continue=1; pLockNotFound=0
LCKTB:2 continue=1
RGRS: GLB_SP = 000000000000043f3245
RGRS: GLB_ST = 2002-04-19-19.26.23.000000|  

-----  

src reg info for mem (0)  

-----  

GLOBAL_RECORD   = N
SOURCE_OWNER    = AZUMA
SOURCE_TABLE    = TEST_SOURCE
SOURCE_VIEW_QUAL= 0
SOURCE_STRUCTURE= 1
SOURCE_CONDENSED= Y
SOURCE_COMPLETE = Y
CD_OWNER        = AZUMA
CD_TABLE        = CDO1
PHYS_CHANGE_OWNER = AZUMA
PHYS_CHANGE_TABLE = CDO1
CD_OLD_SYNCHPOINT IS is 000000000000043c540b
CD_NEW_SYNCHPOINT IS is 000000000000043c83d5
DISABLE_REFRESH = 0
CCD_OWNER       =
CCD_TABLE       =
CCD_OLD_SYNCHPOINT is null
SYNCHPOINT is null
SYNCHTIME is null
CCD_CONDENSED  is null
CCD_COMPLETE   is null
ARCH_LEVEL     = 0201
DESCRIPTION    is null
BEFORE_IMG_PREFIX is X
CONFLICT_LEVEL is 1
PARTITION_KEYS_CHG is N
```

解説:

■このページはブランクです

- ▶ *Replica (Update Anywhere)*
- ▶ レプリケーション対象データの操作
- ▶ レプリケーション処理の操作
- ▶ **データ共用・区分化対応**
- ▶ 特殊な表、データタイプのレプリケーション
- ▶ マルチベンダー・レプリケーション
- ▶ その他



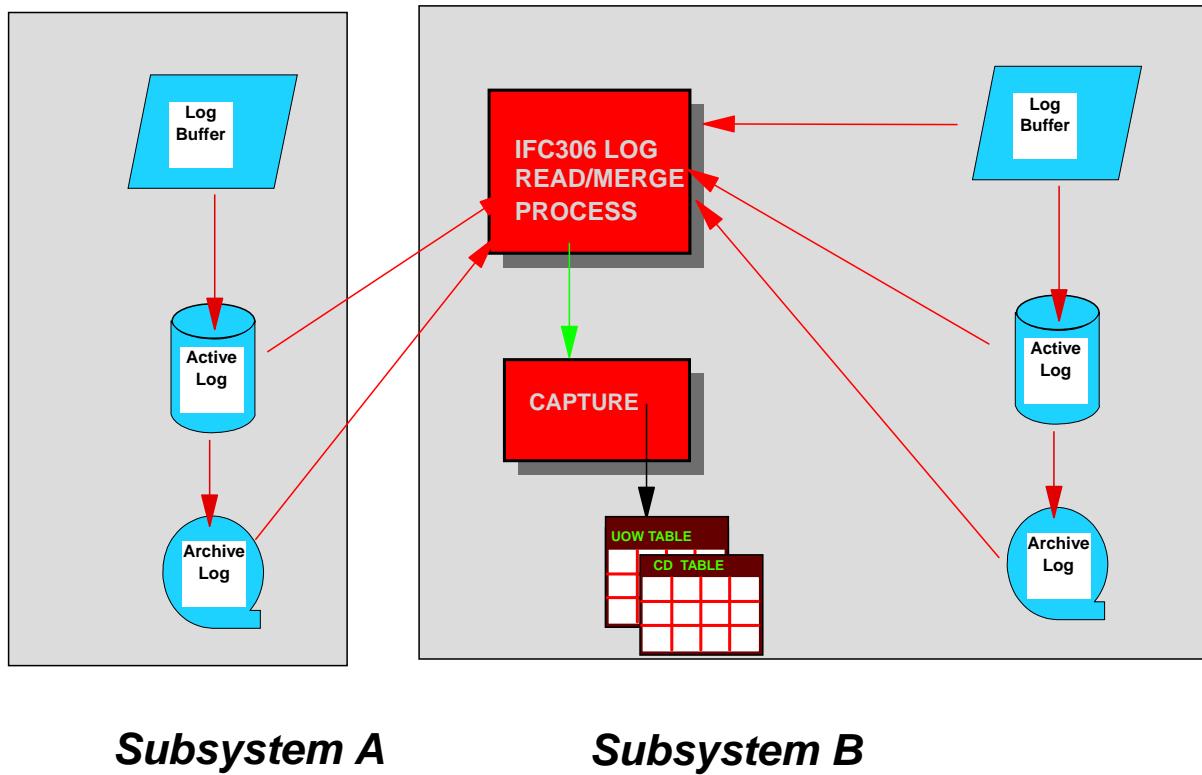
内容

- データ共用環境
- DB2 UDB ESE 区分化環境



データ共用環境

■ Data Sharing Log mergeプロセス

**Subsystem A****Subsystem B**

(C)日本IBMシステムズ・エンジニアリング(株) インフォメーション・マネージメント

67

DB2 Universal Database

解説:

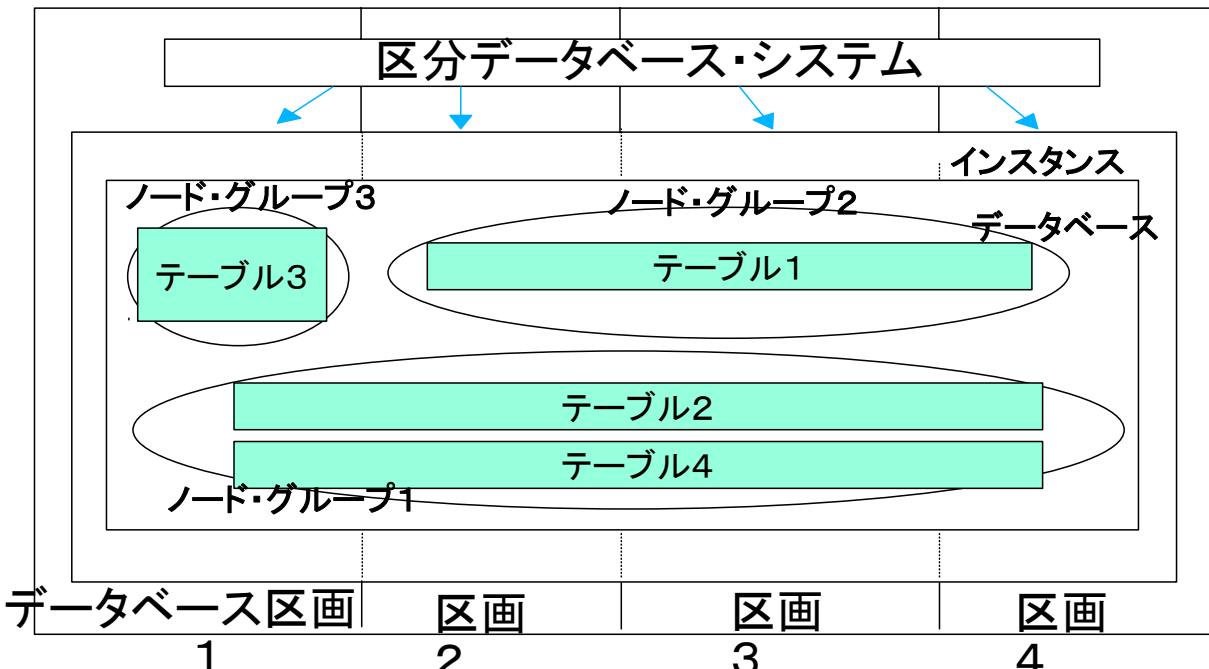
- DB2/MVS Data sharing環境（複数のDB2サブシステムが1つの論理的なDB2シングルイメージを提供する）では各DB2サブメンバーでCaptureを稼動させるのではなく、一つのサブシステム上でCapture/MVSを稼動させる。
- Capture/MVSは現在稼動しているDB2/MVSを介して他のメンバーのLOGを受け取り、CD, UOW表へINSERTを実行する
- Capture/MVSはDB2 APIの制約の為、稼動しているローカルにあるDB2 Buffer上にある変更Dataを読み取ることは可能であるが他のメンバー上にあるLog Buffer内の変更Dataを読み取ることはできない、一旦DISK上(Active Log)へForce-Writeされたデータを読み取る
- DB2 PE, UDB EEE(Extended Enterprise Edition)などの環境では複数Node上にまたがったソース表の変更Dataを読み取ることはできない(UDB V8.1 FP1まで)
 - > Sysplex DB2/MVSが提供するDB2 API(Log Mergeを提供)がないため
 - > DB2 PE, UDB EEEではソース表はカタログNode上のみになければならない、data Change Capture属性がCatalog Nodeのみにしか許されているため。
 - > UDB V8 Fixpak 2にてCaptureがEEE(DPF)で稼働予定

DB2 UDB ESE 区分化環境

■ Capture ESE 区分化環境(DPF) サポート

- UDB V8.1 Fixpak2より

■ UDB ESE DPFの概念（参考）



解説:

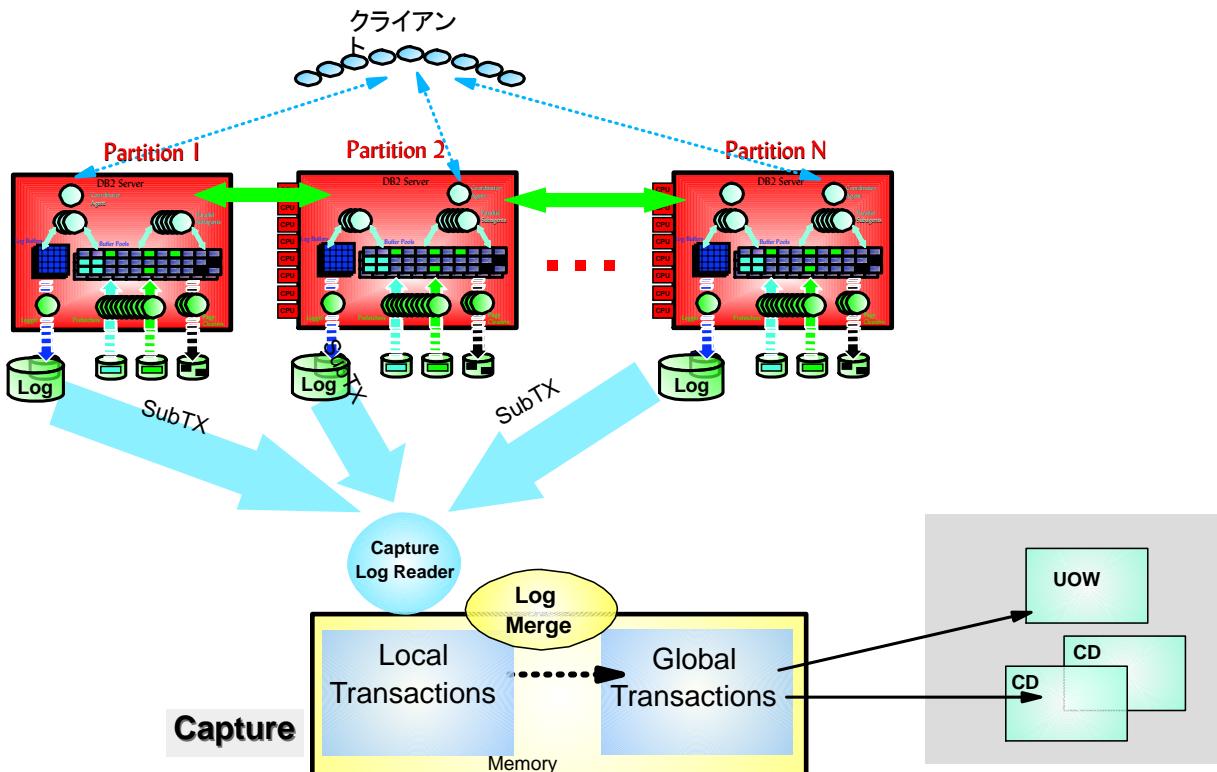
- Capture ESE (DPF) サポートは UDB V8.1 Fixpak2 から提供される新機能
Fixpak2 以前または以前のバージョンでは Data Captuer Changes 属性がカタログノードのみに存在する表しか付与ができず実質的に使用することができなかった。

ESEの概念(参考)

- ノード
クラスター内の各マシンの事をノードと呼びます。
- インスタンス
インスタンスとは一つのデータベース・マネージャー構成ファイルを使用して稼動するデータベース・マネージャー環境の事をさします。データベースを特定のRDB製品独自のフォーマットでデータが格納されたファイルであると考えると、ユーザーからのSQLを解釈し、独自のフォーマットで格納されたファイルからデータを効率的に取り出すためのプログラムをデータベースエンジンと言うことができます。DB2 UDBではdb2syscというプロセスがこのエンジンとしての役割を担っています。このプロセスはデータベース・マネージャー構成ファイルに設定された各種パラメーターを使用して起動します。つまり、このプロセスと、このプロセスの下で管理されているデータベースを含めた環境をインスタンスと呼びます。通常、DB2 UDB Enterprise EditionやWorkgroup Editionでは一つの筐体の中だけでデータベース・システムを構築するため、一つのdb2syscが一つのインスタンスを形成します。一方、DB2 UDB EEEではクラスターを構成する各ノード上で稼動するdb2syscが協調して一つのインスタンスを構成します。一つのインスタンスの中には複数のデータベースを作成することができます。
- データベース
データベースとはユーザーの表や索引などのデータや、これらを管理するためのカタログ・テーブルを格納したファイルの集まりです。DB2 UDB EEEを使用した区分データベース・システムにおいては、クラスター/S P 内のノードをまたがってデータベースが作成されます。
- ノードグループ
クラスター/S P 内のどのノードを使用してテーブルを作成するか指定するために、一つ以上のノードの組み合わせを使用してノード・グループというグループを定義することができます。任意のノードの組み合わせで複数のノード・グループをデータベース内に定義することができます。
- テーブル・スペース
各ノードにおいて物理的にどこをデータの格納に使用するかは、テーブルスペースの定義により指定します。一つのノード・グループ内に複数のテーブル・スペースを定義することができます。

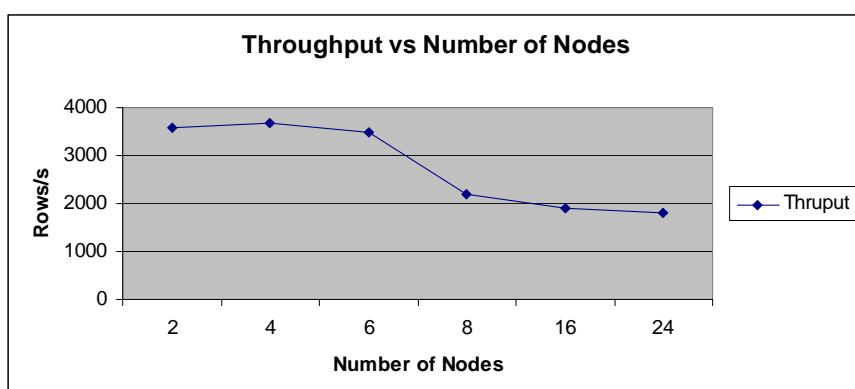
DB2 Partitioned database systems (DB2 MPP) Capture

■ESE上でのCaptureの稼働



解説:

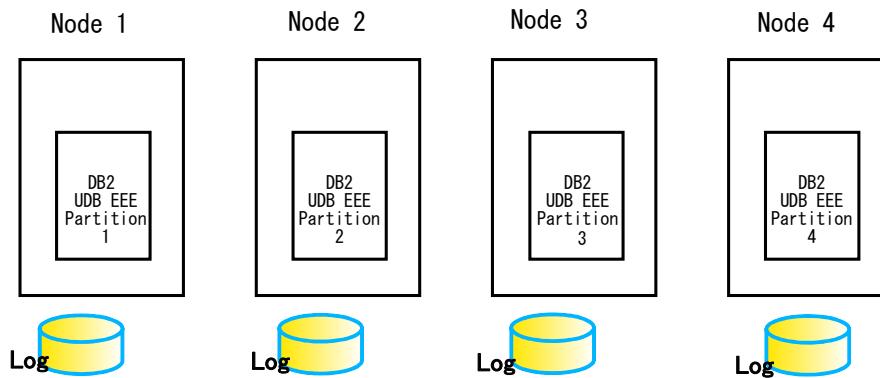
- Captureコントロールテーブル
Catalog Node上の表スペースのみに配置が望ましい
CD, UOW表はCaptureが稼働するノード上に配置が望ましい
CD, UOW表の区分表スペースに配置するとパフォーマンスが劣化する
- パフォーマンス
7ノード以上の構成ではPerformance劣化が発生する
- 制約
MQTはCaptureの対象にはできない
RI制約がある親子表の親表と子表の行が同じ区画、ノード上にない場合RI違反となる場合がある



UDB ESE 環境でのCapture

■各ノード内のLogをCapture自身でマージする

- Captureを各ノードで実行する必要はない
- データベース毎にLOGRETAIN=ONが必要
 - ▶ export DB2NODE=1
 - ▶ db2 update db cfg for v8db using logretain on
- 新規コントロール表(自動作成)
 - ▶ "schema"."IBMSNAP_PARTITIONINFO"
 - Capture起動時自動生成、RCから作成不可



解説:

- デフォルトのデータベース区画は、db2nodes.cfg内のPORT番号の0を持つ区画です。デフォルトではオペレーションはデフォルトのデータベース区画に接続します。接続する区画を変更する場合にはDB2NODE環境変数を使用します。
- DB2NODE環境変数をexportした後には、terminateコマンドを実行してください。
- 現在、どのパーティションに接続しているかは db2 " values current node"ステートメントまたはdb2 list applications show detailの結果の Coordinating Node Number で確認できます。
- DB2 UDB EEE環境においてはデータベース構成ファイルはデータベース毎/データベース・パーティション毎に存在するため、データベース構成パラメーターの変更はそれぞれのデータベース・パーティション毎に行う必要があります。
- db2_allを使用すれば、すべてのデータベース・パーティションのデータベース構成パラメーターを一度に照会/変更することができます。
- Captureは各ノードの読み取り状況を保持(RESTART表と同等) するため存在しなければ下記の表を他のコントロール表が存在する表スペースに自動生成する。

```
-- DDL Statements for table "ASN"."IBMSNAP_PARTITIONINFO"
-----  

CREATE TABLE "ASN"."IBMSNAP_PARTITIONINFO" (
    "PARTITIONID" INTEGER NOT NULL ,
    "USAGE" CHAR(1) NOT NULL ,
    "SEQUENCE" CHAR(10) FOR BIT DATA NOT NULL ,
    "STATUS" CHAR(1) ,
    "LAST_UPDATE" TIMESTAMP )
IN "DPROPRTS" ;  

-- DDL Statements for indexes on Table "ASN"."IBMSNAP_PARTITIONINFO"
CREATE UNIQUE INDEX "ASN"."IBMSNAP_PARTUIX" ON "ASN"."IBMSNAP_PARTITIONINFO"
("PARTITIONID" ASC,
 "USAGE" ASC);
```

UDB ESE (DPF) のトランザクションログレコード

■SQLコミット処理ログとTwo Phase Commit(2相コミット)

- ▶ DPFではアプリケーションが接続している区画を“コーディネーター”とよぶ
- ▶ コーディネーターはそのトランザクション内に参加している他の区画（サボーディネータ）をコーディネートする (Two Phase Commit)
- ▶ IUD更新Logにtimestampは含まれないがCommit Logにtimestampは含まれる

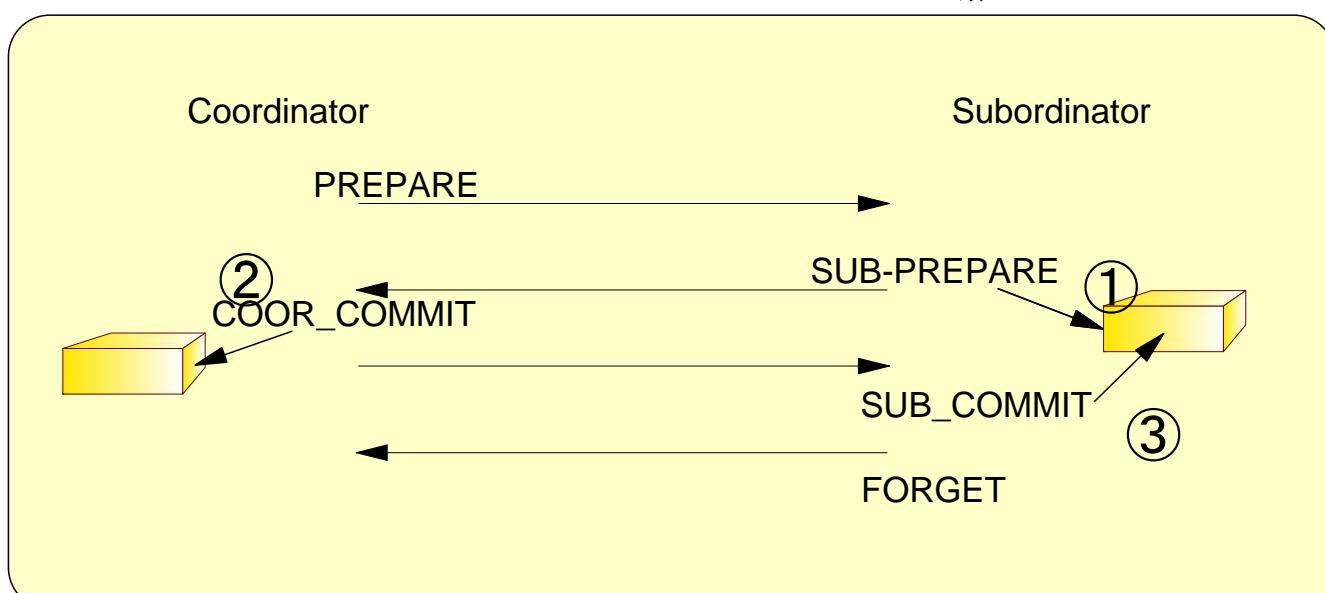
● 3つのトランザクションコミット Log

- ▶ そのトランザクション内の更新がコーディネーター区画のみであればコミットログは EE(Serial)と同じログを書く
- ▶ そのトランザクション内の更新がひとつのサボーディネータ区画であればログは SUB-PREPARE + SUB-COMMITのみ書かれ最適化が実施される
- ▶ そのトランザクション内の更新がコーディネーター区画およびサボーディネータ区画を含む場合は2相コミットを実施する
 1. コーディネータからすべての区画へGID(Global Transaction ID)と共にPrepareリクエストを送付
 2. サボーディネータ区画側でCommit準備をし、PREPARE LOGをTimestamp, GIDとLogに書き出す。返信する
 3. コーディネータからすべての区画から返信を受け取り、返信されたMAX TimestampをGlobal Transactionとして使用する。またCOORDINATOR COMMITログを書き出し、すべての区画にCOMMIT命令を送信する
 4. サボーディネータ区画側はCOMMIT命令を受け取り、SUBORDINATOR COMMITログを書き出す

👉 SUBORDINATOR PREPARE < COORDINATOR COMMIT < SUBORDINATOR COMMITの順序

解説:

- コーディネーターからの2PCフロー
 - 👉 SUBORDINATOR PREPARE < COORDINATOR COMMIT < SUBORDINATOR COMMITの順



解説:

■SQL処理とログ

```

INSERT INTO TEST_SOURCE VALUES(1,1,'this is row1')
DB20000I The SQL command completed successfully.

COMMIT
DB20000I The SQL command completed successfully.

INSERT INTO TEST_SOURCE VALUES (2,2,'this is row2')
DB20000I The SQL command completed successfully.

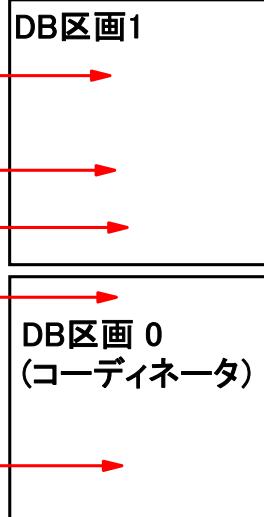
UPDATE TEST_SOURCE SET TEST_DATA=TEST_DATA+1
WHERE ROW_NUMBER=2
DB20000I The SQL command completed successfully.

INSERT INTO TEST_SOURCE VALUES (5,5,'this is row5')
DB20000I The SQL command completed successfully.

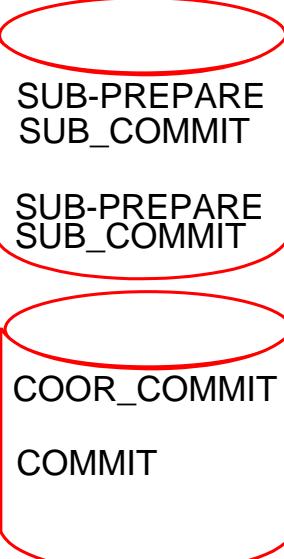
COMMIT
DB20000I The SQL command completed successfully.

INSERT INTO TEST_SOURCE VALUES (7,7,'this is row7')
DB20000I The SQL command completed successfully.

commit
DB20000I The SQL command completed successfully.
  
```



ログ



各行がどちらの区画にハッシュングされたか?

```
'select row_number, nodenumber (row_number) as nodenumber from test_source'
ROW_NUMBER nodenumber
```

```

5      0
7      0
1      1
2      1
  
```

4 record(s) selected.

解説:

```

INSERT INTO TEST_SOURCE VALUES(1,1,'this is row1')
DB20000I The SQL command completed successfully.

COMMIT
DB20000I The SQL command completed successfully.

INSERT INTO TEST_SOURCE VALUES (2,2,'this is row2')
DB20000I The SQL command completed successfully.

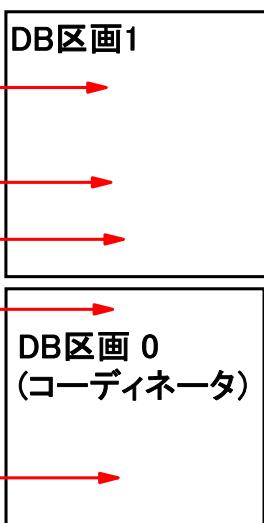
UPDATE TEST_SOURCE SET TEST_DATA=TEST_DATA+1
WHERE ROW_NUMBER=2
DB20000I The SQL command completed successfully.

INSERT INTO TEST_SOURCE VALUES (5,5,'this is row5')
DB20000I The SQL command completed successfully.

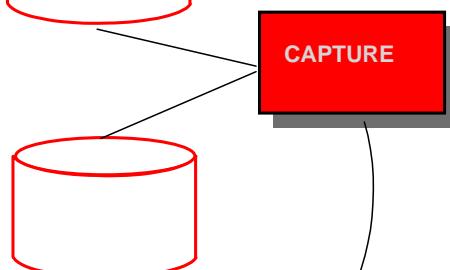
COMMIT
DB20000I The SQL command completed successfully.

INSERT INTO TEST_SOURCE VALUES (7,7,'this is row7')
DB20000I The SQL command completed successfully.

commit
DB20000I The SQL command completed successfully.
  
```



ログ



CD
表

```
$ db2 'select * from cd01 order by ibmsnap_commitseq, ibmsnap_intentseq'
IBMSNAP_COMMITSEQ    IBMSNAP_INTENTSEQ
x'3EA8C8A000000050001'  x'0001000000009B1AC8C'
x'3EA8C8A000000080000'  x'0000000000006D6B44C'
x'3EA8C8A000000080000'  x'0001000000009B1AE3C'
x'3EA8C8A000000080000'  x'0001000000009B1ACE0'
x'3EA8C8A0000000C0000'  x'0000000000006D6B529'

 5 record(s) selected.
```

IBMSNAP_OPERATION ROW_NUMBER TEST_DATA DESCRIPTION

IBMSNAP_OPERATION	ROW_NUMBER	TEST_DATA	DESCRIPTION
I	1	1	this is row1
I	5	5	this is row5
I	2	2	this is row2
U	2	3	this is row2
I	7	7	this is row7

トランザクション順序は保証されるが区画をまたぐ更新のトランザクション内の順序は保証されない

解説:

■実際のログレコード(DB区画 0)

```

Record 1      Log Page Offset = 000C = 12
              Log File Offset = 0000200C = 8204
              Record LSN = 0000 0697 800C
              Record Size = 0048 = 72
              Record Type = 4E = Normal
              lrecflags = 0002
              Record TID = 0000000000BF0
              Back Pointer LSN = 0000 0000 0000
              Originator Code = 01 = Data File Manager
              Function ID = 76 = 118 : INSREC_DP
              Pool ID = 2 Object ID = 4
              RID = 00000004 Pool Page = 0 Slot = 4
              reclen = 34 free_sp = 2840
              recoff = 2850
              oRecType = NORMAL
              oTotRecLen = 34
              TRHEAD Rec Type = FIXEDVAR
# Bytes In Fix Length Data = 26

Log Record Data Follows:
00 05 00 05 00 74 68 69 73 20 69 73 20 72 6F 77 *....this is row*
35 20 20 20 20 20 20 20 20 00 *5 . *

```

解説:

■実際のログレコード(DB区画 0)

```

Record 2      Log Page Offset = 0054 = 84
              Log File Offset = 00002054 = 8276
              Record LSN = 0000 0697 8054
              Record Size = 003C = 60
              Record Type = 4E = Normal
              lrecflags = 0000
              Record TID = 0000000000BF0
              Back Pointer LSN = 0000 0697 800C
              Originator Code = 02 = Index Manager
              Function ID = 65 = 101 : ADDKEY_DP
              Pool ID = 2 Object ID = 4
              Parent Pool ID = 2 Parent Object ID = 4
              Index Root Page = 00000002 = 2

Record 3      Log Page Offset = 0090 = 144
              Log File Offset = 00002090 = 8336
              Record LSN = 0000 0697 8090
              Record Size = 003D = 61
              Record Type = 86 = Commit PE Coordinator
              lrecflags = 0002
              Record TID = 0000000000BF0
              Back Pointer LSN = 0000 0697 8054
              Time Stamp = 3EAD1017 = 1051529239 2003-04-28-11.27.19 GMT
              Nanoseconds = 7
              gxid data = 00000000000000BF0 (Node 0, TID 0000000000BF0)
              nodeInfo = 1
              Authid Length = 16384

```

解説:

■実際のログレコード(DB区画 0)

```

Record 4      Log Page Offset = 00CD = 205
  Log File Offset = 000020CD = 8397
    Record LSN = 0000 0697 80CD
    Record Size = 001C = 28
    Record Type = 88 = Forget
      lrecflags = 0000
      Record TID = 0000000000BF0
  Back Pointer LSN = 0000 0697 8090
    Time Stamp = 3EAD1017 = 1051529239 2003-04-28-11.27.19 GMT
    Nanoseconds = 10

```



解説:

■実際のログレコード(DB区画 0)

```

Record 5      Log Page Offset = 00E9 = 233
  Log File Offset = 000020E9 = 8425
    Record LSN = 0000 0697 80E9
    Record Size = 0048 = 72
    Record Type = 4E = Normal
      lrecflags = 0002
      Record TID = 0000000000BF1
  Back Pointer LSN = 0000 0000 0000
  Originator Code = 01 = Data File Manager
    Function ID = 76 = 118 : INSREC_DP
      Pool ID = 2 Object ID = 4
      RID = 00000005 Pool Page = 0 Slot = 5
      reclen = 34 free_sp = 2804
      recoff = 2816
      oRecType = NORMAL
      oTotRecLen = 34
    TRHEAD Rec Type = FIXEDVAR
  # Bytes In Fix Length Data = 26

  Log Record Data Follows:
  00 07 00 07 00 74 68 69 73 20 69 73 20 72 6F 77 *....this is row*
  37 20 20 20 20 20 20 20 20 00 *7 . *

```



解説:

■実際のログレコード(DB区画 0)

```

Record 6      Log Page Offset = 0131 = 305
  Log File Offset = 00002131 = 8497
    Record LSN = 0000 0697 8131
    Record Size = 003C = 60
    Record Type = 4E = Normal
      lrecflags = 0000
      Record TID = 0000000000BF1
    Back Pointer LSN = 0000 0697 80E9
    Originator Code = 02 = Index Manager
      Function ID = 65 = 101 : ADDKEY_DP
      Pool ID = 2 Object ID = 4
    Parent Pool ID = 2 Parent Object ID = 4
    Index Root Page = 00000002 = 2

Record 7      Log Page Offset = 016D = 365
  Log File Offset = 0000216D = 8557
    Record LSN = 0000 0697 816D
    Record Size = 0026 = 38
    Record Type = 84 = Commit SE
      lrecflags = 0002
      Record TID = 0000000000BF1
    Back Pointer LSN = 0000 0697 8131
      Time Stamp = 3EAD1017 = 1051529239 2003-04-28-11.27.19 GMT
      Nanoseconds = 11
      Authid Length = 8
      Authid = V8EEE

```

解説:

■実際のログレコード(DB区画 1)

```

Record 1      Log Page Offset = 000C = 12
  Log File Offset = 0000200C = 8204
    Record LSN = 0000 0947 000C
    Record Size = 0048 = 72
    Record Type = 4E = Normal
      lrecflags = 0002
      Record TID = 00000000B2F6
    Back Pointer LSN = 0000 0000 0000
    Originator Code = 01 = Data File Manager
      Function ID = 76 = 118 : INSREC_DP
      Pool ID = 2 Object ID = 4
      RID = 00000004 Pool Page = 0 Slot = 4
      reclen = 34 free_sp = 2840
      reoff = 2850
      oRecType = NORMAL
      oTotRecLen = 34
    TRHEAD Rec Type = FIXEDVAR
# Bytes In Fix Length Data = 26

Log Record Data Follows:
00 01 00 01 00 74 68 69 73 20 69 73 20 72 6F 77 *....this is row*
31 20 20 20 20 20 20 20 20 20 20 20 20 20 20 00 *1 . *

```

解説:

■実際のログレコード(DB区画 1)

```

Record 2      Log Page Offset = 0054 = 84
  Log File Offset = 00002054 = 8276
    Record LSN = 0000 0947 0054
    Record Size = 003C = 60
    Record Type = 4E = Normal
      lrecflags = 0000
      Record TID = 00000000B2F6
    Back Pointer LSN = 0000 0947 000C
      Originator Code = 02 = Index Manager
        Function ID = 65 = 101 : ADDKEY_DP
          Pool ID = 2 Object ID = 4
        Parent Pool ID = 2 Parent Object ID = 4
      Index Root Page = 00000002 = 2

Record 3      Log Page Offset = 0090 = 144
  Log File Offset = 00002090 = 8336
    Record LSN = 0000 0947 0090
    Record Size = 0038 = 56
    Record Type = 4C = Lock
      lrecflags = 0020
      Record TID = 00000000B2F6
    Back Pointer LSN = 0000 0947 0054
      Size = 36 = 2 locks
      First Lock = 000200040000000400000000 Type R Mode 5 Flags 0

```

解説:

■実際のログレコード(DB区画 1)

```

Record 4      Log Page Offset = 00C8 = 200
  Log File Offset = 000020C8 = 8392
    Record LSN = 0000 0947 00C8
    Record Size = 00B8 = 184
    Record Type = 81 = PE Prepare
      lrecflags = 0002
      Record TID = 00000000B2F6
    Back Pointer LSN = 0000 0947 0090
      Time Stamp = 3EAD1017 = 1051529239 2003-04-28-11.27.19 GMT
      Nanoseconds = 2
        logspace = 0
        logspace2 = 204
      coorNextLsn = 0000 0697 800C
      gxit data = 00000000000000BEF (Node 0, TID 00000000BEF)

Record 5      Log Page Offset = 0180 = 384
  Log File Offset = 00002180 = 8576
    Record LSN = 0000 0947 0180
    Record Size = 003C = 60
    Record Type = 85 = Commit PE Subordinator
      lrecflags = 0002
      Record TID = 00000000B2F6
    Back Pointer LSN = 0000 0947 00C8
      Time Stamp = 3EAD1017 = 1051529239 2003-04-28-11.27.19 GMT
      Nanoseconds = 4
      gxit data = 00000000000000BEF (Node 0, TID 00000000BEF)
      nodeInfo = 1
      Authid Length = 8
      Authid = 4

```

解説:

■実際のログレコード(DB区画 1)

```

Record 6      Log Page Offset = 01BC = 444
  Log File Offset = 000021BC = 8636
    Record LSN = 0000 0947 01BC
    Record Size = 0048 = 72
    Record Type = 4E = Normal
      lrecflags = 0002
      Record TID = 00000000B2F7
    Back Pointer LSN = 0000 0000 0000
    Originator Code = 01 = Data File Manager
      Function ID = 76 = 118 : INSREC_DP
      Pool ID = 2 Object ID = 4
        RID = 00000005 Pool Page = 0 Slot = 5
        reclen = 34 free_sp = 2804
        recoff = 2816
        oRecType = NORMAL
        oTotRecLen = 34
      TRHEAD Rec Type = FIXEDVAR
# Bytes In Fix Length Data = 26

Log Record Data Follows:
00 02 00 02 00 74 68 69 73 20 69 73 20 72 6F 77 *....this is row*
32 20 20 20 20 20 20 20 20 00 *2 . *

```



解説:

■実際のログレコード(DB区画 1)

```

Record 7      Log Page Offset = 0204 = 516
  Log File Offset = 00002204 = 8708
    Record LSN = 0000 0947 0204
    Record Size = 003C = 60
    Record Type = 4E = Normal
      lrecflags = 0000
      Record TID = 00000000B2F7
    Back Pointer LSN = 0000 0947 01BC
    Originator Code = 02 = Index Manager
      Function ID = 65 = 101 : ADDKEY_DP
      Pool ID = 2 Object ID = 4
      Parent Pool ID = 2 Parent Object ID = 4
    Index Root Page = 00000002 = 2

```



解説:

■実際のログレコード(DB区画 1)

```

Record 8      Log Page Offset = 0240 = 576
  Log File Offset = 00002240 = 8768
    Record LSN = 0000 0947 0240
    Record Size = 007C = 124
    Record Type = 4E = Normal
      lrecflags = 0002
      Record TID = 00000000B2F7
    Back Pointer LSN = 0000 0947 0204
    Originator Code = 01 = Data File Manager
      Function ID = 78 = 120 : UPDREC_DP
      Pool ID = 2 Object ID = 4
        RID = 00000005 Pool Page = 0 Slot = 5
        reclen = 34 free_sp = 0
        recoff = 0
        oRecType = NORMAL
        oTotRecLen = 34
      TRHEAD Rec Type = FIXEDVAR
# Bytes In Fix Length Data = 26

Old Data:
00 02 00 02 00 74 68 69 73 20 69 73 20 72 6F 77 *....this is row*
32 20 20 20 20 20 20 20 20 00 *2 . *

```



解説:

■実際のログレコード(DB区画 1)

```

Function ID = 78 = 120 : UPDREC_DP
Pool ID = 2 Object ID = 4
RID = 00000005 Pool Page = 0 Slot = 5
reclen = 34 free_sp = 2804
recoff = 2816
oRecType = NORMAL
oTotRecLen = 34
TRHEAD Rec Type = FIXEDVAR
# Bytes In Fix Length Data = 26

New Data:
00 02 00 03 00 74 68 69 73 20 69 73 20 72 6F 77 *....this is row*
32 20 20 20 20 20 20 20 20 00 *2 . *

```

```

Record 9      Log Page Offset = 02BC = 700
  Log File Offset = 000022BC = 8892
    Record LSN = 0000 0947 02BC
    Record Size = 0038 = 56
    Record Type = 4C = Lock
      lrecflags = 0020
      Record TID = 00000000B2F7
    Back Pointer LSN = 0000 0947 0240
      Size = 36 = 2 locks
      First Lock = 00020004000000050000000000 Type R Mode 5 Flags 32

```



解説:

■実際のログレコード(DB区画 1)

```

Record 10      Log Page Offset = 02F4 = 756
               Log File Offset = 000022F4 = 8948
               Record LSN = 0000 0947 02F4
               Record Size = 00B8 = 184
               Record Type = 81 = PE Prepare
               lrecflags = 0002
               Record TID = 00000000B2F7
               Back Pointer LSN = 0000 0947 02BC
               Time Stamp = 3EAD1017 = 1051529239 2003-04-28-11.27.19 GMT
               Nanoseconds = 5
               logspace = 0
               logspace2 = 288
               coorNextLsn = 0000 0697 8090
               gxit data = 00000000000000BF0 (Node 0, TID 00000000BF0)

Record 11      Log Page Offset = 03AC = 940
               Log File Offset = 000023AC = 9132
               Record LSN = 0000 0947 03AC
               Record Size = 003C = 60
               Record Type = 85 = Commit PE Subordinator
               lrecflags = 0002
               Record TID = 00000000B2F7
               Back Pointer LSN = 0000 0947 02F4
               Time Stamp = 3EAD1017 = 1051529239 2003-04-28-11.27.19 GMT
               Nanoseconds = 8
               gxit data = 00000000000000BF0 (Node 0, TID 00000000BF0)
               nodeInfo = 0
               Authid Length = 8
               Authid = 4

```

解説:

■実際のログレコード(DB区画 1)

```

Record 12      Log Page Offset = 03E8 = 1000
               Log File Offset = 000023E8 = 9192
               Record LSN = 0000 0947 03E8
               Record Size = 0028 = 40
               Record Type = 80 = LogSync
               lrecflags = 0000
               Record TID = 000000000000
               Back Pointer LSN = 0000 0000 0000
               VTS(seconds) = 3EAD1017 = 1051529239
               Nanoseconds = 8
               nodenum = 0
               last_ext = (26, 1051528344)

```

データベース区画の追加

■ ADD_PARTITION=N/Yパラメーター

- 追加後、COLDスタートであればADD_PARTITION=Nでも追加された区画を認識
- 追加後、WARMスタートであればADD_PARTITION=Yを指定しないと成功しない

```
$ db2start nodenum 3 addnode hostname host4 port 0
```



```
$ db2stop  
$ db2start
```

0	host1	0
1	host2	0
2	host3	0
3	host4	0

区画追加後、ADD_PARTITION=NでCaptureをWARM Startした場合

ASN0183E CAPTURE "ASN". The Capture program detected an inconsistency between the IBMSNAP PARTITIONINFO table and DB2 partition information.
 ASN0122E CAPTURE "ASN". An error occurred while reading the restart information or DB2 log. The Capture program will terminate.
 ASN0008I CAPTURE "ASN". The Capture program was stopped.



解説:

- Manualまたはdb2startで区画を追加後、追加された区画で下記を実行する必要がある
 - Set logretain to logging
 - Backup the database.
- ADD_PARTITION=Yパラメータを使用する場合は
 データベース区画を追加後、CaptureをWARMにて立ち上げなおした場合のみ
 asnccmdコマンド、CAPPARMSでは発行不可



- ▶ *Replica (Update Anywhere)*
- ▶ レプリケーション対象データの操作
- ▶ レプリケーション処理の操作
- ▶ データ共用・区分化対応
- ▶ 特殊な表、データタイプのレプリケーション
- ▶ マルチベンダー・レプリケーション
- ▶ その他



内容

■ MQT

■ LOBデータ

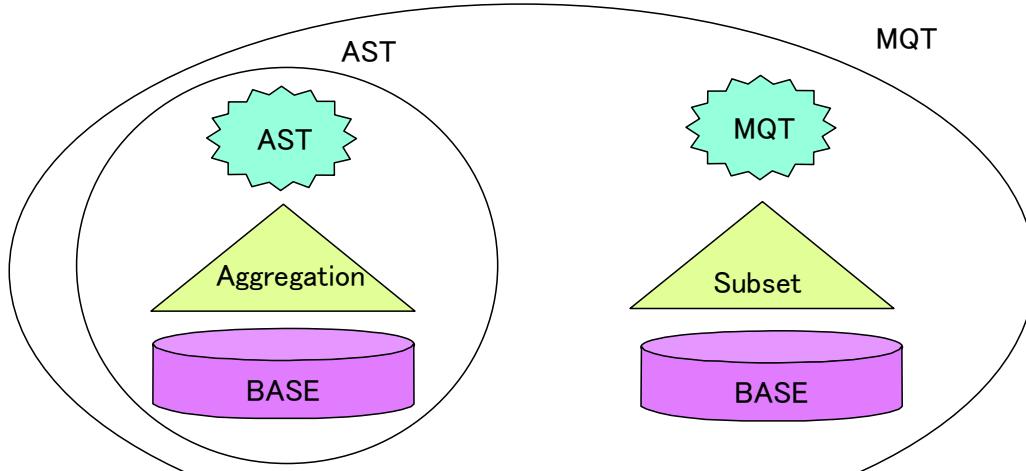
■ DATALINK



MQT (AST)

■ MQT (AST) の複製

- MQTをソースとしてレプリケーション
 - ▶ Refresh Immediate/Refresh Deferred MQTをソース表としてCapture利用可能
 - ▶ DB2 for z/OS V8からUDB 同様にソースに指定可能
- MQTをターゲットとしてレプリケーション (キャッシュ表)
 - ▶ Maintained By User MQTの差分メインテナスとしてDPropRを利用する
 - ▶ Nicknameを利用し、他社DBからも使用可能



解説:

- BASE表としてUDB V8.1からAggregation(Group by等)なしのAST使用可能
UDB V7

```
CREATE SUMMARY TABLE AST1 AS ( SELECT I1, I2 FROM BASE1 WHERE I1 > 1 ) DATA INITIALLY DEFERRED REFRESH IMMEDIATE
```

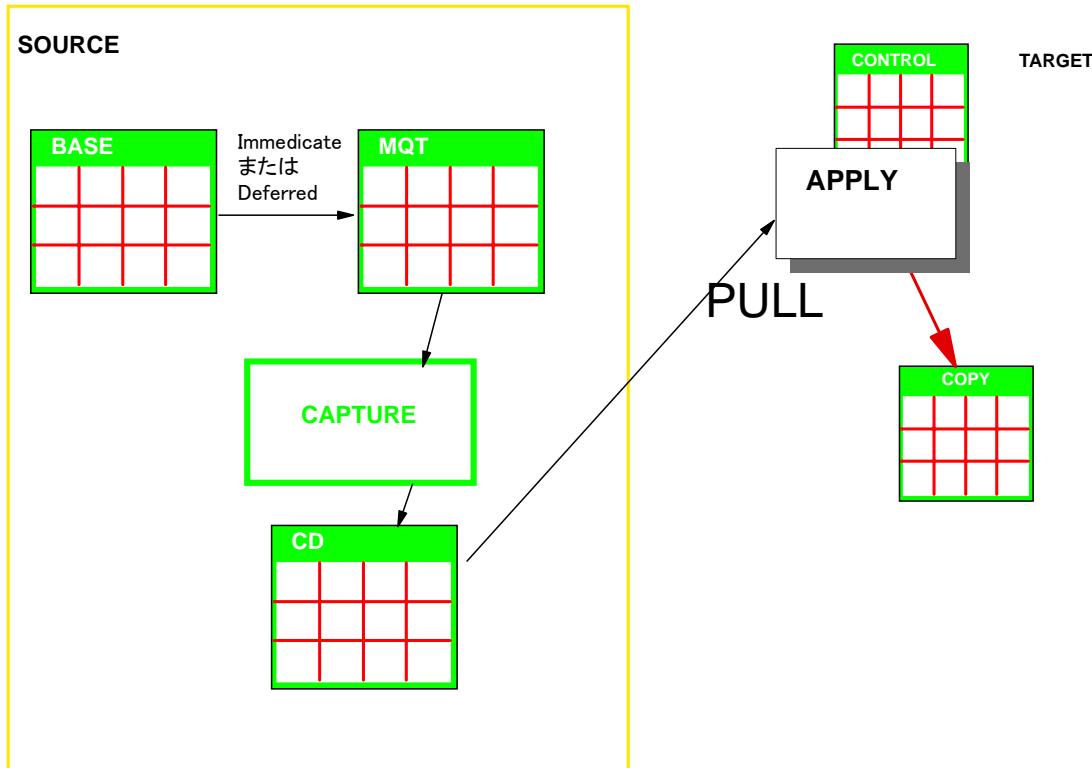
```
SQL20058N The full select specified for the summary table "DB2V7.AST1" is not valid. SQLSTATE=428EC
```

UDB V8

```
CREATE SUMMARY TABLE AST1 AS ( SELECT I1, I2 FROM BASE1 WHERE I1 > 1 ) DATA INITIALLY DEFERRED REFRESH IMMEDIATE
DB200001 The SQL command completed successfully.
```

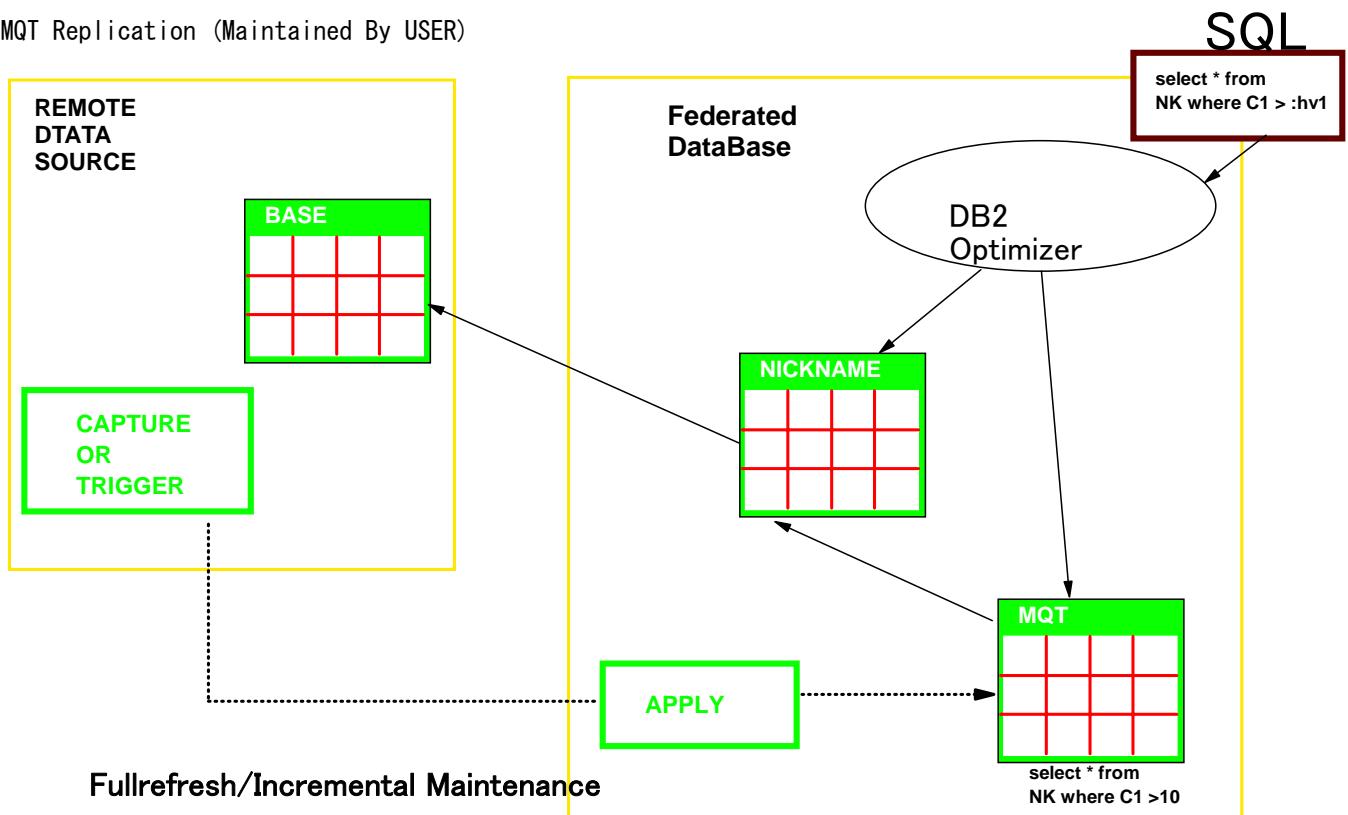
解説:

■MQT Replication (Maintained By SYSTEM)



解説:

■MQT Replication (Maintained By USER)

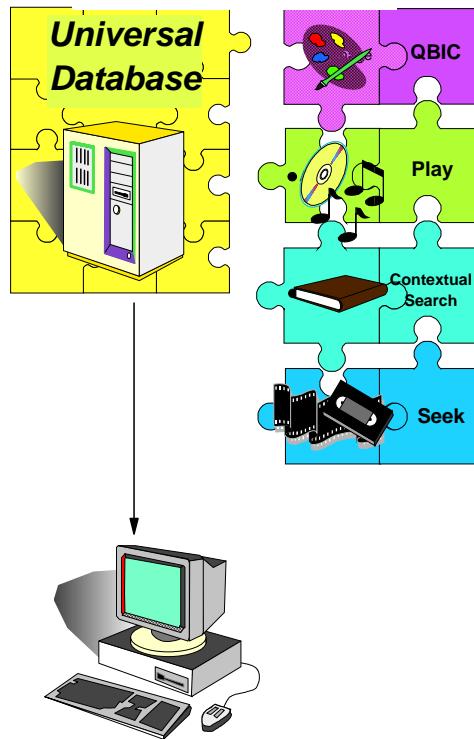


LOBデータ

■LOBデータのレプリケーション

- DB2 for OS/390, UDBとともにサポート
 - ▶ BLOB
Image, Audio, Videoデータなどを格納
 - ▶ CLOB
SBCS文字およびMixed CHARを格納
 - ▶ DBCLOB
DBCS文字を格納

UDB for OS/390, UDB



■CD表には変更後のデータは格納されない

■LOB列は1つの表あたり10列まで

解説:

■LOB表の作成例 (DB2 For OS/390)

```
create tablespace basets;
create table lobtest
(c1 smallint not null,
c2 rowid generated always,
c3 clob(1k),
c4 smallint) in basets;
create unique index ix1 on lobtest (c1);

create lob tablespace lobts log no;
create aux table auxtab in lobts stores lobtest column c3;
create unique index ixa on auxtab;
insert into lobtest (c1,c3,c4) values (1,CLOB('AZUMA'),1);
```

■CD表の作成例

```
CREATE TABLE AZUMA.LOBTEST(
IBMSNAP_COMMITSEQ CHAR(10) FOR BIT DATA NOT NULL,
IBMSNAP_INTENTSEQ CHAR(10) FOR BIT DATA NOT NULL,
IBMSNAP_OPERATION CHAR(1) NOT NULL, C1 SMALLINT NOT NULL, XC1 SMALLINT,
C2 VARCHAR(17) FOR BIT DATA NOT NULL, C3 CHAR(1), C4 SMALLINT) ;
```

LOB列はCD表にはキャプチャーされません。その代わりに、LOB列データの変更を意味するものがキャプチャーされます。実際のLOBデータは、APPLEYにより直接ベース表からターゲット表へ、変更されたLOB列に関して複写されます。このために、ユーザー表はターゲット表と同じ基本キー列を持つ必要があります。LOB変更インディケーターは、空白値可能なCHAR(1)の列で、この列名は、ベース表のLOB列と同じ列名です。ベース表には1つ以上のLOB列が存在すれば、CD表にも1つ以上のLOGインディケーター列があります。この指示フラグ列の値は、更新される行についてのみ重要です。UPDATE処理でLOB列が変更された場合、列の値は'U'を含みます。それ以外の場合には、NULLが設定されています。

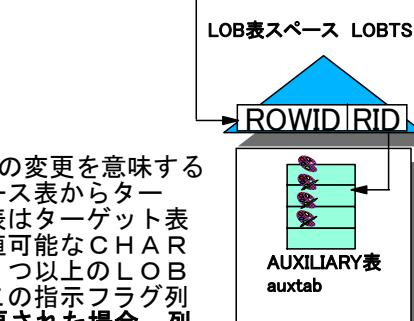
CD表には変更されたLOBデータを収集せずLOB Locatorを格納する。

BEFORE-IMAGEがCD表にない

→READ Only Copyのみ, Update Anywhereはサポートされない

表スペース BASETS

LOBTEST表		ベース表			
ROWID	CLOB	C1	C2	C3	C4
			X'010...		



AUXILIARY表の索引は、ベース表のROWID列とLOBデータへのRIDから構成される

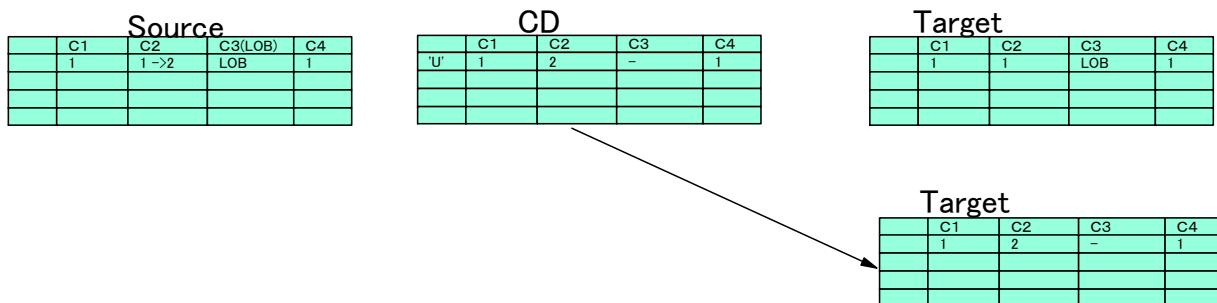
解説:

■下記の条件では既知の問題あり:

- Target表に行が存在しないためUpdate→Insertの変換が発生
- かつ
- そのMemberに少なくとも1つのLOBカラムが存在
- かつ
- その更新自身はLOBを更新していない

■結果

- Different errors might be received by Apply depending on LOB definitions on source/target table
- If LOB column in source table is defined as NOT NULL:
Apply fails with sqlcode -452
- If LOB column in target table is defined as NOT NULL:
Apply fails with sqlcode -407
- If the LOB column is nullable in both source and target:
Apply inserts a NULL into LOB column. Target table is out of synch



解説:

■Apply FP10よりエラー (STATUS=-1, ACTIVATE=0になる)

```

PSET: RBA translation for member 0
PSET: Fetch answer set for member 0
Compiled(F) at 06:43:52 on Aug 12 2005 (Level 80124m_FP10)
CDSET: spill_file(0) = /home/db2v8/dpr/lob/QUAL1.000
FETSET: The number of rows fetched is 1.
PSET: Commit3 ok
PSET: connect to V8DB; rc = 1
PSET: CURRTSRVRTYPE is SQL, CURRTSRVRVERSION is 08

```

```

Compiled(A) at 06:43:52 on Aug 12 2005 (Level 80124m_FP10)
UTGP: Commit1 ok
fopen filename /home/db2v8/dpr/lob/QUAL1.000

```

```

2005-10-31-17.36.45.924989 ASN0999E "Apply" : "QUAL1" : "WorkerThread" : Error condition "Update is turned to Insert, LOB value is not fetched", error code(s): "*", "*", "*".

```

```

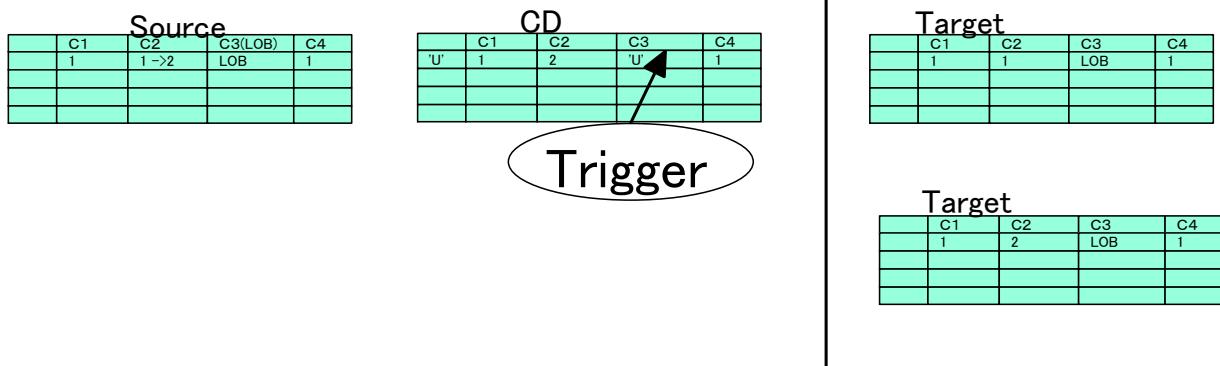
PSET: ROLLBACK
PSET: success is 0
PSET: connect to V8DB; rc = 1
CLOS: setRepeatCopy is 0
CPCLOS: success is 0. retcode is 0.
CLOS: activate = 0
CLOS: status = -1

```

解説:

- Triggerによるエラー回避策

```
CREATE TRIGGER FILCD
AFTER INSERT ON DB2V8.CD01
REFERENCING NEW AS N FOR EACH ROW MODE DB2SQL
BEGIN ATOMIC
IF IBMSNAP_OPERATION='U' THEN
UPDATE DB2V8.CD01 SET DESCRIPTION='U';
END IF;
END$
```



解説:

- このページはブランクです

DATALINK

■ DATALINKデータ・タイプ

- データベースによって外部ファイルのアクセス、保全性、および回復を制御できるデータタイプ

■ DATALINKデータ・タイプのレプリケーション

- DATALINK 列は複写可能
- ただしDATALINK 列が指す外部ファイルは複写されない。
 - ▶ ファイルを複写する場合、変更適用プログラムはユーザー出口ルーチンを使用

解説:

■ Datalink列サポート

リモート・ネットワークを介して大きなファイル（マルチメディア・データなど）にアクセスするのは、非効率的で費用がかかります。これらのファイルが不变であるか、変更される回数がごくわずかである場合、リモート・サイトに複製することによってこれらのファイルに短時間でアクセスできるだけでなく、ネットワーク・トラフィックも軽減できます。DB2 ユニバーサル・データベースには DATALINK データ・タイプが用意されており、これを使うことによってデータベースにこのような種類のファイルに対するアクセス、保全性、回復などを制御させることができます。DB2 ユニバーサル・データベースは、OS/390 を除くすべてのプラットフォームで DATALINK 値をサポートしています。

DB2 は DATALINK 列を複製すると同時に、ASNDLCOPY ユーザー出口ルーチンを使ってその DATALINK 列が参照している外部ファイルを複製します。このルーチンはそれぞれのソース・リンク参照をターゲット・リンク参照に変換した後、該当する外部ファイルをソース・システムからターゲット・システムにコピーします。sqlllib/samples/repl ディレクトリーには、ファイルの転送に FTP やファイル・コピー用デーモン (ASNDLCOPYD.SMP) を使用できるサンプルのルーチン (ASNDLCOPY.SMP) があります。AS/400 の場合、サンプル・プログラムはライブラリー QDPR 内のファイル QCSRC, QCBLLESRC、および QRPGLESRC にあります。ASNDLCOPY 出口ルーチンの使用および ASNDLCOPYD ファイル・コピー・デーモンの使用を参照してください。

外部ファイルは非常に大きい場合があるため、変更適用プログラムと、これらのファイルをコピーするのに使うファイル転送方式の両方にとて十分なネットワーク帯域幅を確保してください。同じように、ターゲット・システムにもこれらのファイルが入るだけのディスク・スペースを確保しなければなりません。

推奨事項:

DATALINK 値にはそれぞれ別個のサブスクリプション・セットを使用してください。それは、変更適用プログラムは ASNDLCOPY ルーチンがその複製を完了するのを待ってからサブスクリプション・セットの複製を完了させるからです。外部ファイルのコピー時にエラーが生じると、サブスクリプション・セット全体の複製が失敗します。

制約事項:

DB2 が DATALINK 値をサポートしている方法の関係上、DB2 データベース間で DATALINK 値の複製を行えるのは AIX , AS/400, Windows NT であり、DATALINK 値をサポートしていないプラットフォームには、使用できません。

DATALINK 列で随時更新複製を使用する場合は、対立検出レベルに None を指定する必要があります。

DB2 は DATALINK 列が参照する外部ファイルの更新対立は検査しません。DATALINK 列における変更前イメージはサポートしていません。DB2 は、常に DATALINK 列が参照する最新バージョンの外部ファイルを複製します。ターゲット表が基礎集約表または変更集約表である場合は、DATALINK 列がサポートされません。

- ▶ *Replica (Update Anywhere)*
- ▶ レプリケーション対象データの操作
- ▶ レプリケーション処理の操作
- ▶ データ共用・区分化対応
- ▶ 特殊な表、データタイプのレプリケーション
- ▶ マルチベンダー・レプリケーション
- ▶ その他

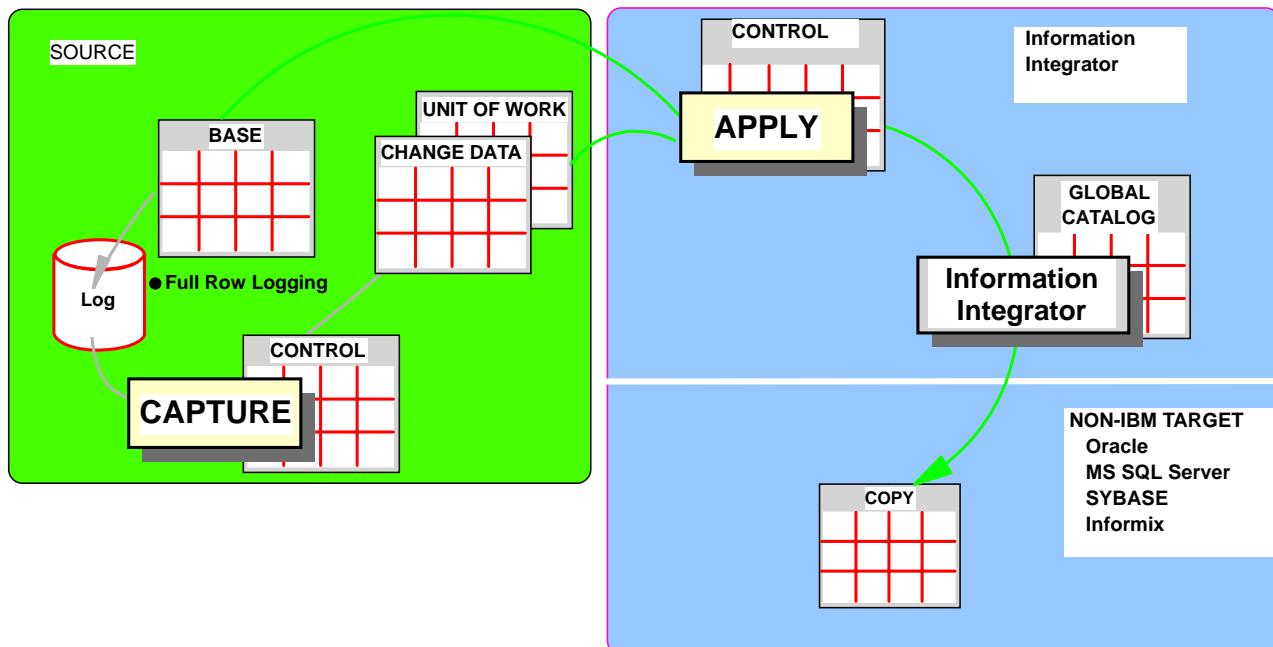


blank page



マルチベンダー・レプリケーション

- DpropRとInformation Integratorを組み合わせて使用することにより、DB2からDB2以外のRDBへのレプリケーションが可能

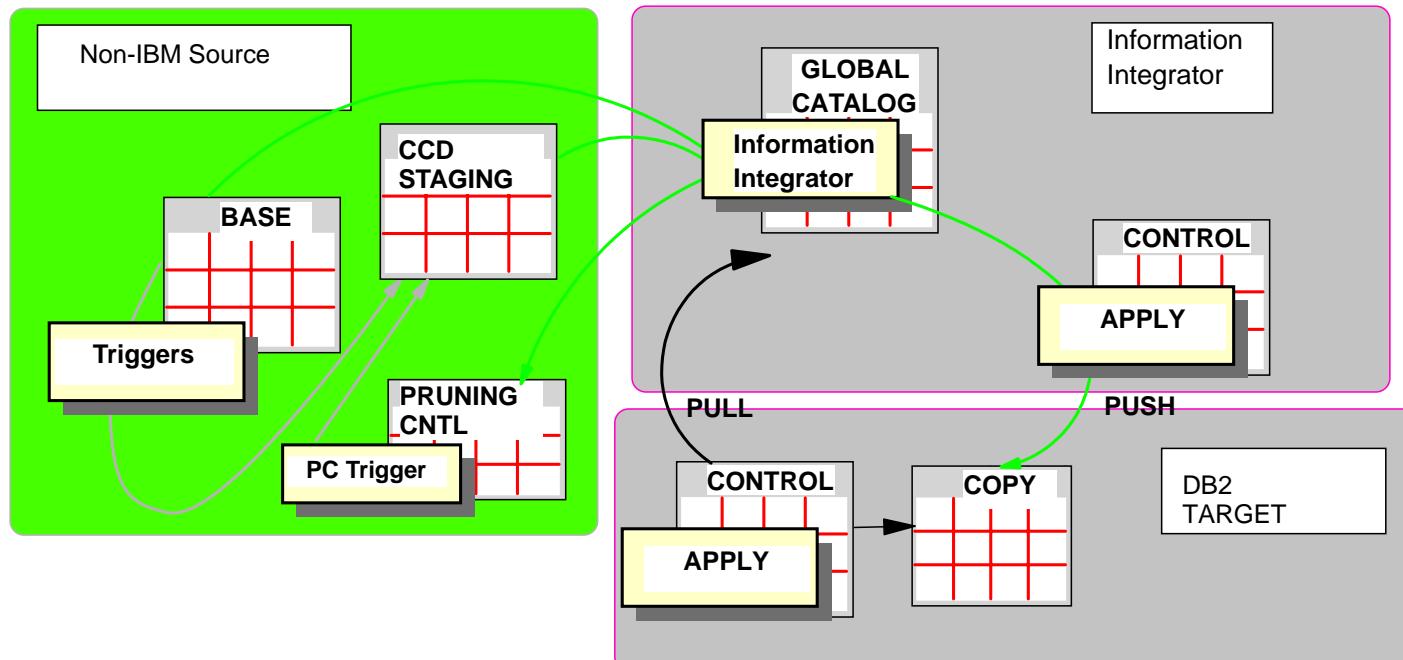


解説:

- Websphere Information Integratorを使用したマルチベンダーレプリケーション
DB2 UDB V8あるいはWebsphere Information Integratorを使用した連合サーバー機能は、データがリレーションナルと非リレーションナル、ローカルとリモート、IBM プラットフォームからのものと非 IBM プラットフォームからのもののいずれであっても、データがローカルであるかのように、すべてのデータを単一サイト・イメージで提供します。
連合サーバー機能 は、SQL 言語、データ・アクセス方式、ネットワーク・プロトコル、オペレーティング・システムなどの違いをマスクします。
多くの業種において、複製は異種環境におけるデータへのアクセスを向上させるための優れたメカニズムです。IBM レプリケーション・テクノロジーは、連合サーバー機能に統合されており、データへの同期と非同期の両方のアクセスを提供します。

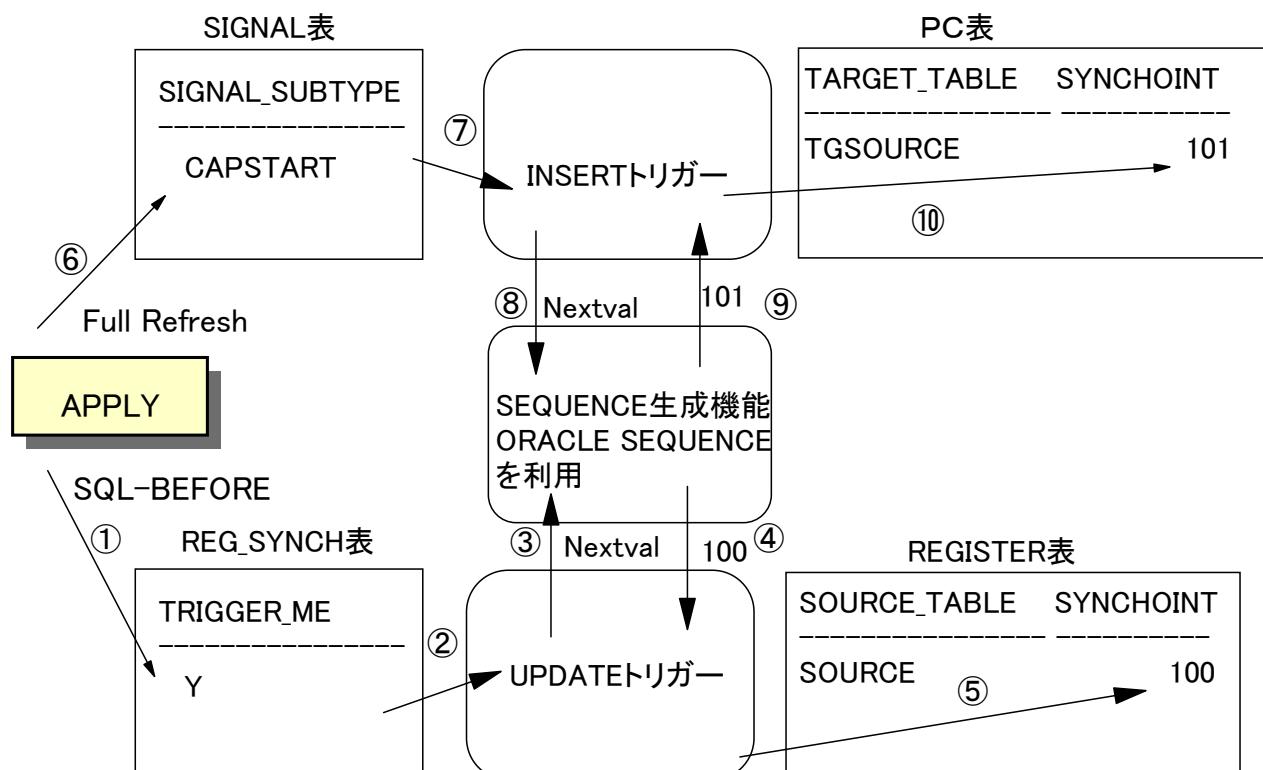
マルチベンダー・レプリケーション

- DpropRとWebsphere Information Integratorを組み合わせて使用することにより、DB2以外のRDBからDB2へのレプリケーションが可能。
- トリガーの利用



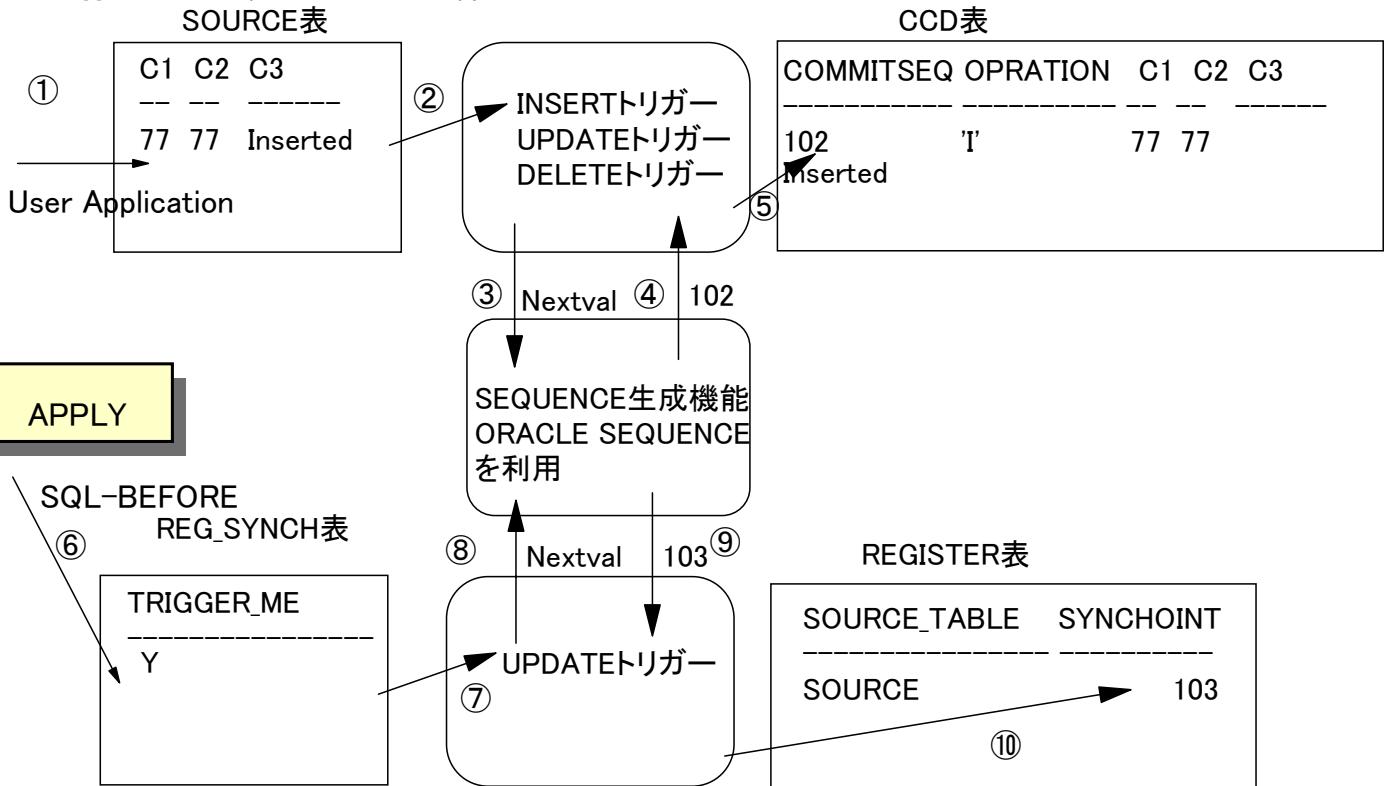
解説:

- Trigger Based Replication (全件copy)



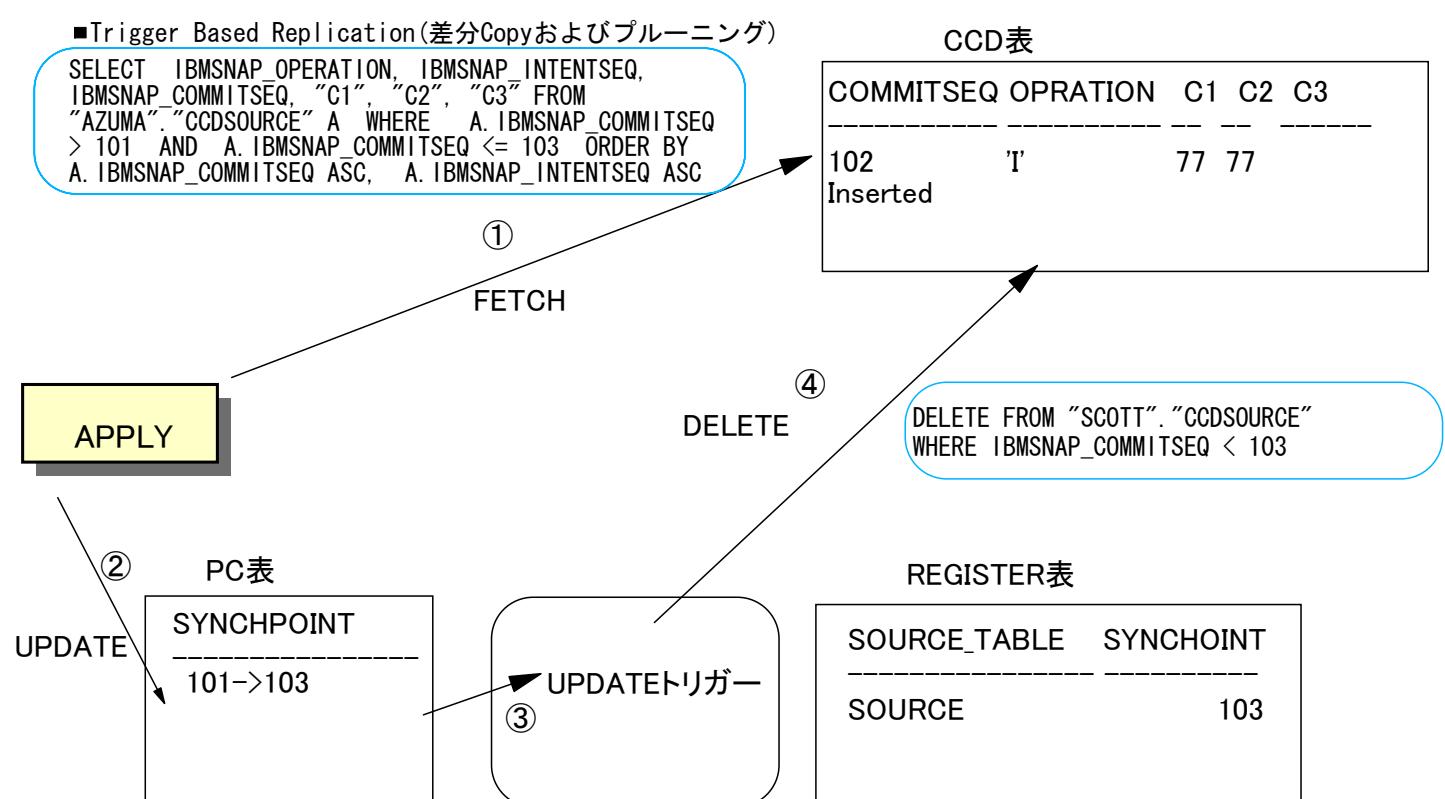
解説:

■Trigger Based Replication(差分Copy)



解説:

■Trigger Based Replication(差分Copyおよびプルーニング)



解説:

■キャプチャ・トリガー

レプリケーション・ソースが非 DB2 リレーショナル・データベースに存在する場合、レプリケーション・センターはキャプチャ・プログラムよりも キャプチャ・トリガーを使用して、そのソースへの変更をキャプチャします。作成されたトリガーは、レプリケーション・ソースごとの 1 つの変更キャプチャのセットで、プルーニング・トリガー、REG_SYNCH トリガー、シグナル・トリガーです。

■変更キャプチャ・トリガー

これらのトリガーは、非 DB2 リレーショナル・データベースの表に生じた変更を記録します。トリガーはソース表について定義した CCD 表での変更を記録します。更新、削除、および挿入には、3 つの変更キャプチャがあります。

■プルーニング・トリガー

このトリガーはサブスクリプション・セットごとの SQL ステートメントのセットで、その中の各メンバーごとに 1 つのステートメントを含みます。これらのステートメントは、アプライ・プログラムがサブスクリプション・セットの処理を完了したときにソース・サーバーで実行されます。各ステートメントは、ある特定のメンバーのソースである CCD 表を整理します。また、独自に別の SQL ステートメントを提供することによって、同時にシグナル・コントロール表を整理できます。

■REG_SYNCH トリガー

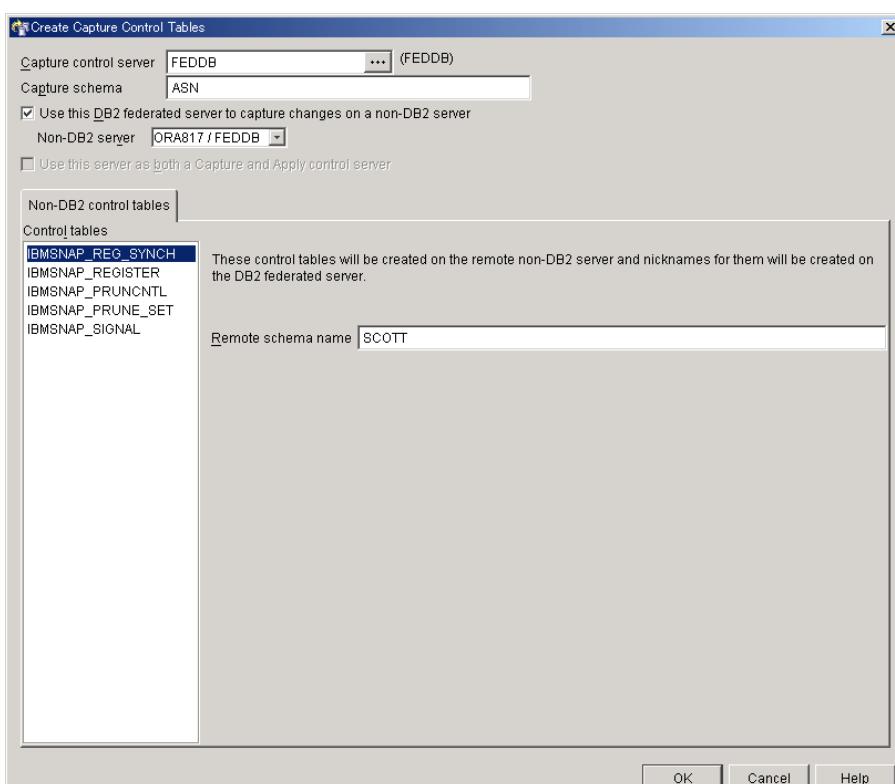
このトリガーは、非 DB2 リレーショナル・データベースにある登録表 (IBMSNAP_REGISTER) を更新します。このコントロール表を更新すると、アプライ・プログラムは CCD 表のためのニックネームに変更があるかどうか探します。CCD 表は、非 DB2 リレーショナル・データベースにマップされる連合サーバーに常駐しています。

■シグナル・トリガー

このトリガーは、DB2 ソースから DB2 ターゲットへのレプリケーションで発生する、アプライ・プログラムおよびキャプチャ・プログラム間のハンドシェークのロジックをエミュレートします。これは、整理コントロール (IBMSNAP_PRUNCNTL) 表の同期点の値を順序的に次の値で更新する、挿入後トリガーです。アプライ・プログラムは、シグナル (IBMSNAP_SIGNAL) 表に行を挿入してからフル・リフレッシュを開始し、その結果このトリガーが起動することになります。

解説:

■定義例



解説:

- レプリケーションセンターが作成するコントロール表作成SQL

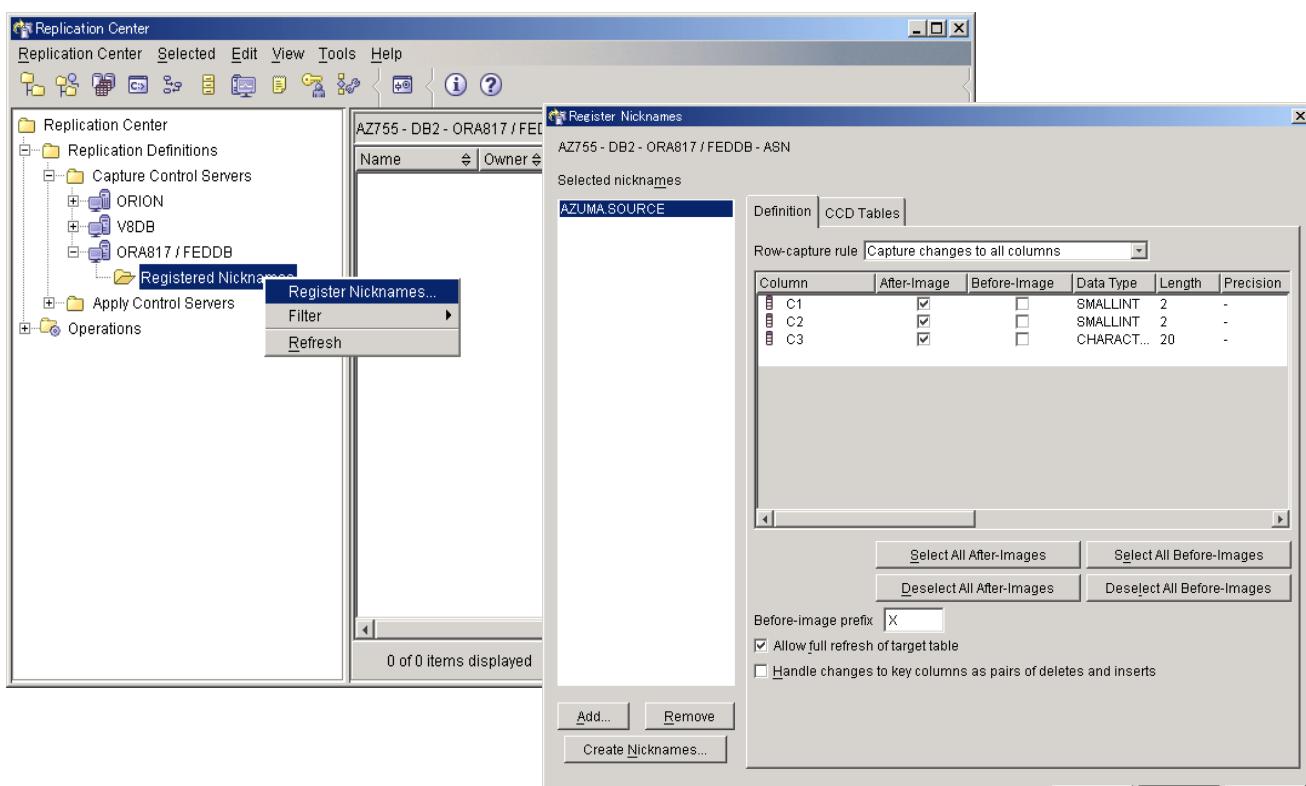
```

SET PASSTHRU ORA817#
CREATE TABLE "SCOTT"."IBMSNAP_REGISTER" (....)#
CREATE SEQUENCE "SCOTT"."SGENERATOR001"MINVALUE 100 INCREMENT BY 1#
CREATE TABLE "SCOTT"."IBMSNAP_PRUNCNTL" ....#
CREATE TABLE "SCOTT"."IBMSNAP_PRUNE_SET"....#
CREATE TABLE "SCOTT"."IBMSNAP_SIGNAL"....#
CREATE TRIGGER "SCOTT"."SIGNAL_TRIGGER" AFTER INSERT ON
"SCOTT"."IBMSNAP_SIGNAL" FOR EACH ROW
BEGIN UPDATE "SCOTT"."IBMSNAP_PRUNCNTL" SET SYNCHPOINT =
LPAD(TO_CHAR("SCOTT"."SGENERATOR001".NEXTVAL), 20, '0'), SYNCHTIME=SYSDATE
WHERE RTRIM(MAP_ID) = RTRIM(:NEW.SIGNAL_INPUT_IN) AND SYNCHPOINT=HEXTORAW('000000000000000000000000') ;
END;#
CREATE TABLE "SCOTT"."IBMSNAP_REG_SYNCH"#
CREATE TRIGGER "SCOTT"."REG_SYNCH_TRIGGER"
AFTER UPDATE ON "SCOTT"."IBMSNAP_REG_SYNCH"
BEGIN
UPDATE "SCOTT"."IBMSNAP_REGISTER"
SET SYNCHPOINT=LPAD(TO_CHAR("SCOTT"."SGENERATOR001".NEXTVAL), 20, '0'),
SYNCHTIME=SYSDATE;END;#
SET PASSTHRU RESET#
CREATE NICKNAME ASN.IBMSNAP_REGISTER FOR ORA817."SCOTT"."IBMSNAP_REGISTER"#
CREATE NICKNAME ASN.IBMSNAP_PRUNCNTL FOR ORA817."SCOTT"."IBMSNAP_PRUNCNTL"#
CREATE NICKNAME ASN.IBMSNAP_PRUNE_SET FOR ORA817."SCOTT"."IBMSNAP_PRUNE_SET"#
CREATE NICKNAME ASN.IBMSNAP_SIGNAL FOR ORA817."SCOTT"."IBMSNAP_SIGNAL"#
CREATE NICKNAME ASN.IBMSNAP_REG_SYNCH FOR ORA817."SCOTT"."IBMSNAP_REG_SYNCH"#
CREATE TABLE ASN.IBMSNAP_CAPSCHEMAS(CAP_SCHEMA_NAME VARCHAR(30))#
CREATE UNIQUE INDEX ASN.IBMSNAP_CAPSCHEMAX ON ASN.IBMSNAP_CAPSCHEMAS(CAP_SCHEMA_NAME ASC)#
INSERT INTO ASN.IBMSNAP_CAPSCHEMAS(CAP_SCHEMA_NAME) VALUES ('ASN')#

```

解説:

- コントロール表が作成後のレプリケーションセンターでのRegistration



解説:

■ Registration SQL

```

CONNECT TO FEDDB #
SET PASSTHRU ORA817#
CREATE TABLE "SCOTT"."CCDSOURCE"(IBMSNAP_COMMITSEQ  RAW(10) NOT NULL, IBMSNAP_INTENTSEQ  RAW(10)
NOT NULL, IBMSNAP_OPERATION  CHAR(1) NOT NULL, IBMSNAP_LOGMARKER  DATE NOT NULL,
C1  NUMBER (4 , 0), C2  NUMBER (4 , 0), C3  CHAR ( 20 ) )#


CREATE TRIGGER "SCOTT"."ICCDSOURCE"
AFTER INSERT ON "SCOTT"."SOURCE"
FOR EACH ROW BEGIN INSERT INTO "SCOTT"."CCDSOURCE"
(C1, C2, C3, IBMSNAP_COMMITSEQ, IBMSNAP_INTENTSEQ, IBMSNAP_OPERATION, IBMSNAP_LOGMARKER )
VALUES (:NEW.C1, :NEW.C2, :NEW.C3, LPAD(TO_CHAR("SCOTT"."SGENERATOR001".NEXTVAL), 20, '0'),
LPAD(TO_CHAR("SCOTT"."SGENERATOR001".NEXTVAL), 20, '0'), 'I', SYSDATE);END;#


CREATE TRIGGER "SCOTT"."UCCDSOURCE"
AFTER UPDATE ON "SCOTT"."SOURCE"
FOR EACH ROW BEGIN INSERT INTO "SCOTT"."CCDSOURCE"
(C1, C2, C3, IBMSNAP_COMMITSEQ, IBMSNAP_INTENTSEQ, IBMSNAP_OPERATION, IBMSNAP_LOGMARKER)
VALUES (:NEW.C1, :NEW.C2, :NEW.C3, LPAD(TO_CHAR("SCOTT"."SGENERATOR001".NEXTVAL), 20, '0'),
LPAD(TO_CHAR("SCOTT"."SGENERATOR001".NEXTVAL), 20, '0'), 'U', SYSDATE);END;#

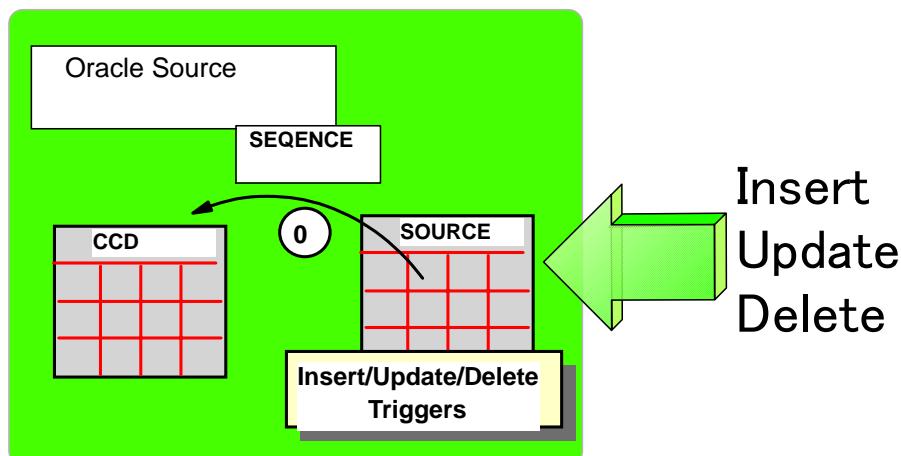

CREATE TRIGGER "SCOTT"."DCCDSOURCE"
AFTER DELETE ON "SCOTT"."SOURCE"
FOR EACH ROW BEGIN INSERT INTO "SCOTT"."CCDSOURCE"
(C1, C2, C3, IBMSNAP_COMMITSEQ, IBMSNAP_INTENTSEQ, IBMSNAP_OPERATION, IBMSNAP_LOGMARKER)
VALUES (:OLD.C1, :OLD.C2, :OLD.C3, LPAD(TO_CHAR("SCOTT"."SGENERATOR001".NEXTVAL), 20, '0'),
LPAD(TO_CHAR("SCOTT"."SGENERATOR001".NEXTVAL), 20, '0'), 'D', SYSDATE);END;#


COMMIT#
SET PASSTHRU RESET#

```

解説:

■ トリガーデザインの例



■ 生成されるSQLの例

```

CONNECT TO FEDDB #
SET PASSTHRU ORA817#
CREATE TABLE "SCOTT"."CCDSOURCE"(IBMSNAP_COMMITSEQ  RAW(10) NOT NULL, IBMSNAP_INTENTSEQ  RAW(10)
NOT NULL, IBMSNAP_OPERATION  CHAR(1) NOT NULL, IBMSNAP_LOGMARKER  DATE NOT NULL,
C1  NUMBER (4 , 0), C2  NUMBER (4 , 0), C3  CHAR ( 20 ) )#


CREATE TRIGGER "SCOTT"."ICCDSOURCE"
AFTER INSERT ON "SCOTT"."SOURCE"
FOR EACH ROW BEGIN INSERT INTO "SCOTT"."CCDSOURCE"
(C1, C2, C3, IBMSNAP_COMMITSEQ, IBMSNAP_INTENTSEQ, IBMSNAP_OPERATION, IBMSNAP_LOGMARKER )
VALUES (:NEW.C1, :NEW.C2, :NEW.C3, LPAD(TO_CHAR("SCOTT"."SGENERATOR001".NEXTVAL), 20, '0'),
LPAD(TO_CHAR("SCOTT"."SGENERATOR001".NEXTVAL), 20, '0'), 'I', SYSDATE);END;#


CREATE TRIGGER "SCOTT"."UCCDSOURCE"
AFTER UPDATE ON "SCOTT"."SOURCE"
FOR EACH ROW BEGIN INSERT INTO "SCOTT"."CCDSOURCE"
(C1, C2, C3, IBMSNAP_COMMITSEQ, IBMSNAP_INTENTSEQ, IBMSNAP_OPERATION, IBMSNAP_LOGMARKER)
VALUES (:NEW.C1, :NEW.C2, :NEW.C3, LPAD(TO_CHAR("SCOTT"."SGENERATOR001".NEXTVAL), 20, '0'),
LPAD(TO_CHAR("SCOTT"."SGENERATOR001".NEXTVAL), 20, '0'), 'U', SYSDATE);END;#


CREATE TRIGGER "SCOTT"."DCCDSOURCE"
AFTER DELETE ON "SCOTT"."SOURCE"
FOR EACH ROW BEGIN INSERT INTO "SCOTT"."CCDSOURCE"
(C1, C2, C3, IBMSNAP_COMMITSEQ, IBMSNAP_INTENTSEQ, IBMSNAP_OPERATION, IBMSNAP_LOGMARKER)
VALUES (:OLD.C1, :OLD.C2, :OLD.C3, LPAD(TO_CHAR("SCOTT"."SGENERATOR001".NEXTVAL), 20, '0'),
LPAD(TO_CHAR("SCOTT"."SGENERATOR001".NEXTVAL), 20, '0'), 'D', SYSDATE);END;#


COMMIT#
SET PASSTHRU RESET#

```

上記Insert トリガー以外にUPDATE/DELETE トリガーが作成

解説:

```

CREATE NICKNAME AZUMA.CCDSOURCE FOR ORA817."SCOTT"."CCDSOURCE"#
COMMIT#
ALTER NICKNAME AZUMA.SOURCE ALTER COLUMN C1 LOCAL TYPE DECIMAL(4, 0)#
ALTER NICKNAME AZUMA.CCDSOURCE ALTER COLUMN C1 LOCAL TYPE DECIMAL(4, 0)#
ALTER NICKNAME AZUMA.SOURCE ALTER COLUMN C2 LOCAL TYPE DECIMAL(4, 0)#
ALTER NICKNAME AZUMA.CCDSOURCE ALTER COLUMN C2 LOCAL TYPE DECIMAL(4, 0)#
COMMIT#
INSERT INTO ASN.IBMSNAP_REGISTER (SOURCE_OWNER, SOURCE_TABLE,
SOURCE_VIEW_QUAL, GLOBAL_RECORD, SOURCE_STRUCTURE, SOURCE_CONDENSED,
SOURCE_COMPLETE, CD_OWNER, CD_TABLE, PHYS_CHANGE_OWNER,
PHYS_CHANGE_TABLE, CD_OLD_SYNCHPOINT, CD_NEW_SYNCHPOINT,
DISABLE_REFRESH, CCD_OWNER, CCD_TABLE, CCD_OLD_SYNCHPOINT,
SYNCHPOINT, SYNCHTIME, CCD_CONDENSED, CCD_COMPLETE, ARCH_LEVEL,
DESCRIPTION, BEFORE_IMG_PREFIX, CONFLICT_LEVEL,
CHG_UPD_TO_DEL_INS, CHGONLY, RECAPTURE, OPTION_FLAGS,
STOP_ON_ERROR, STATE, STATE_INFO ) VALUES(
'AZUMA','SOURCE',0,'N',1,'Y','Y','NONE',
'000000000000000001','AZUMA','CCDSOURCE',
null,null,0,'AZUMA','CCDSOURCE',X'000000000000000000000001',null,null,'N','N','0801',
null,'X','0','N','Y','NNNN','Y','I',null)#
COMMIT#

```

DB2 Universal Database

解説:

```

SET PASSTHRU ORA817#
CREATE TRIGGER "SCOTT"."PRUNCNTL_TRIGGER" AFTER UPDATE ON "SCOTT"."IBMSNAP_PRUNCNTL"
DECLAREMIN_SYNCHPOINT RAW (10);
CURSOR C1 IS SELECT DISTINCT SOURCE_OWNER, SOURCE_TABLE, PHYS_CHANGE_OWNER, PHYS_CHANGE_TABLE
FROM "SCOTT"."IBMSNAP_PRUNCNTL" WHERE SYNCHPOINT IS NOT NULL;C1_REC C1%ROWTYPE;
MUTATING EXCEPTION;PRAGMA EXCEPTION_INIT(MUTATING, -4091);
BEGIN
OPEN C1;
LOOP FETCH C1 INTO C1_REC;
EXIT WHEN C1%NOTFOUND;
BEGIN
SELECT MIN(SYNCHPOINT) INTO MIN_SYNCHPOINT FROM "SCOTT"."IBMSNAP_PRUNCNTL"
WHERE SOURCE_OWNER = C1_REC.SOURCE_OWNER AND SOURCE_TABLE = C1_REC.SOURCE_TABLE;
EXCEPTION WHEN NO_DATA_FOUND THEN NULL;
END;
BEGIN
IF C1_REC.PHYS_CHANGE_OWNER IS NULL AND C1_REC.PHYS_CHANGE_TABLE IS NULL THEN
C1_REC.PHYS_CHANGE_TABLE:=NULL;
ELSIF C1_REC.PHYS_CHANGE_OWNER = 'AZUMA' AND C1_REC.PHYS_CHANGE_TABLE= 'CCDSOURCE' THEN
DELETE FROM "SCOTT"."CCDSOURCE" WHERE IBMSNAP_COMMITSEQ < MIN_SYNCHPOINT;
END IF;
END;
END LOOP;
CLOSE C1;
EXCEPTION WHEN MUTATING THEN NULL;
END:#
```

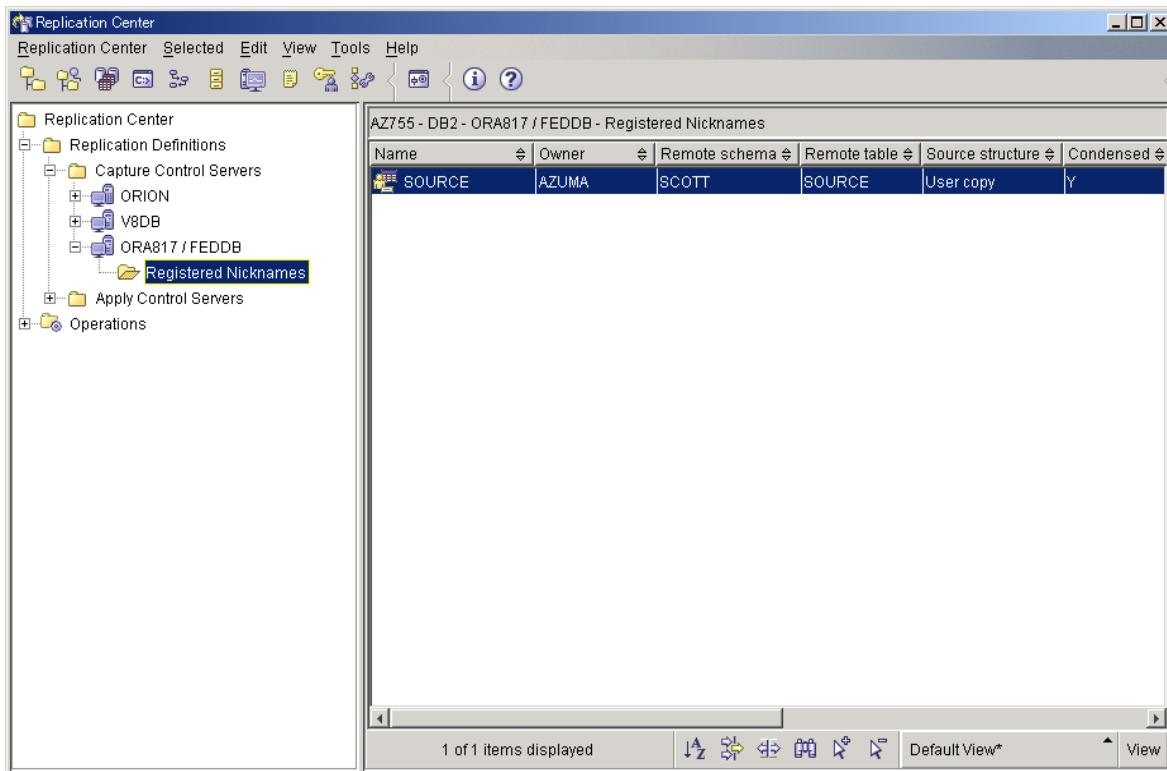
```

COMMIT#
SET PASSTHRU RESET#
COMMIT#
```

DB2 Universal Database

解説:

■Registration後のRC



解説:

■Subscription作成SQL(抜粋)

```

INSERT INTO ASN. IBMSNAP_SUBS_STMTS (
    APPLY_QUAL, SET_NAME, WHOS_ON_FIRST, BEFORE_OR_AFTER, STMT_NUMBER, EI_OR_CALL, SQL_STMT, ACCEPT_SQLSTATES )
SELECT 'QUALF', 'SETF', 'S', 'X', COALESCE (max(stmt_number+5), 1), 'E',
    'UPDATE ASN. IBMSNAP_REG_SYNCH SET TRIGGER_ME = ''Y''',
    '02000' FROM ASN. IBMSNAP_SUBS_STMTS
    WHERE APPLY_QUAL= 'QUALF' AND SET_NAME='SETF' AND WHOS_ON_FIRST='S' AND BEFORE_OR_AFTER='X';

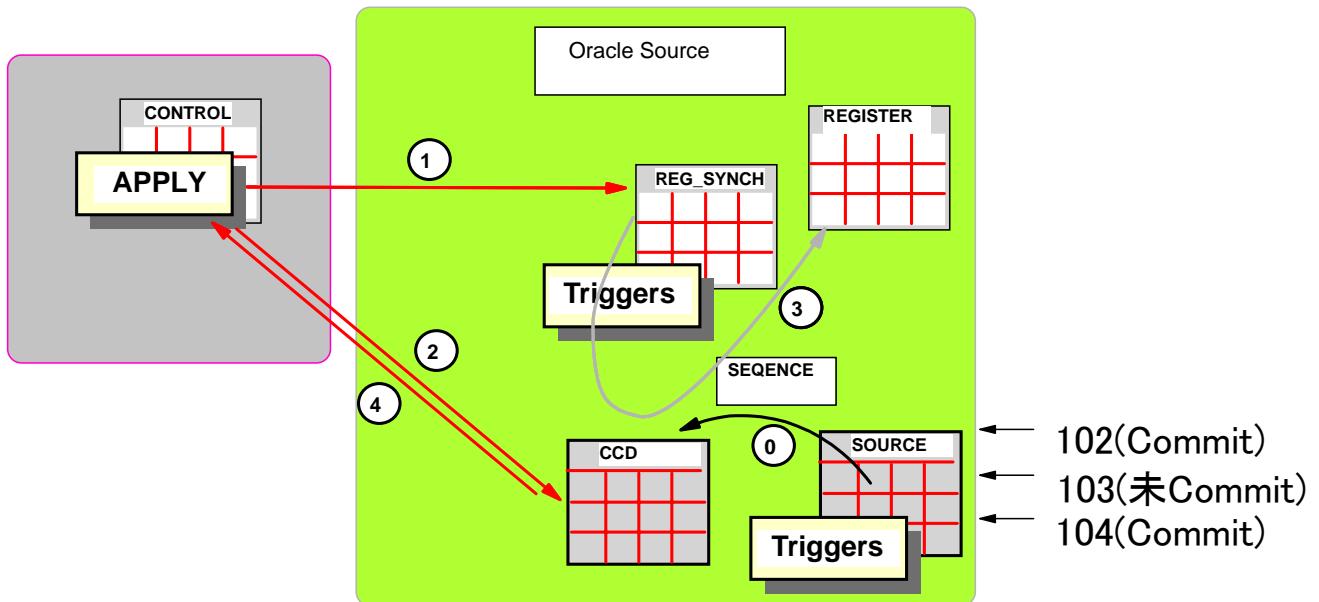
UPDATE ASN. IBMSNAP_SUBS_SET
SET AUX_STMTS = (SELECT COUNT(*) FROM ASN. IBMSNAP_SUBS_STMTS
    WHERE APPLY_QUAL = 'QUALF'
    AND SET_NAME = 'SETF' AND WHOS_ON_FIRST = 'S')
    WHERE APPLY_QUAL = 'QUALF' AND SET_NAME = 'SETF' AND WHOS_ON_FIRST = 'S';

INSERT INTO ASN. IBMSNAP_SUBS_STMTS (
    APPLY_QUAL, SET_NAME, WHOS_ON_FIRST, BEFORE_OR_AFTER, STMT_NUMBER, EI_OR_CALL, SQL_STMT)
SELECT DISTINCT 'QUALF', 'SETF', 'S', 'G', -1, 'E', 'LOCK TABLE AZUMA.CCDSOURCE IN SHARE MODE'
FROM ASN. IBMSNAP_SUBS_STMTS
    WHERE APPLY_QUAL = 'QUALF'
        AND SET_NAME = 'SETF'
        AND WHOS_ON_FIRST = 'S'
        AND NOT EXISTS (SELECT SET_NAME
            FROM ASN. IBMSNAP_SUBS_STMTS
            WHERE APPLY_QUAL = 'QUALF'
                AND SET_NAME = 'SETF'
                AND WHOS_ON_FIRST = 'S'
                AND BEFORE_OR_AFTER = 'G'
                AND SQL_STMT LIKE 'LOCK TABLE AZUMA.CCDSOURCE IN SHARE MODE');

```

解説:

- ApplyはCCD表からFetchする前にSQL-BEFOREでREG_SYNCH表を更新する
ApplyはREGISTER_SYNCHPOINT値より低いCOMMITSEQ値しかCCD表からFetchしない為、事前にUpper Boundをセット
この更新はREG_SYNCH表上のトリガーにて行われる (BEFORE_OR_AFTER='X')
- ApplyはCCD表をLOCKする
SEQ Numberはコミット順序を意味しない、あくまでRequest Order順ソース更新とApplyが同時稼動すると
Applyモレのデータが発生することを防止する為 (BEFORE_OR_AFTER='G') → UDB Fixpak4でLOCKなしに改善



解説:

- Fixpak4以降の場合、ORACLEがソースの場合全件COPY回避を実行するSQLは以下のようになります
FP4より、CCD表のトリガーによりCCD表のIBMSNAP_COMMITSEQに番号を振り直します。
この番号を利用してターゲット表へのレプリケーションが実行されます。
また、フルリフレッシュ時には、この番号を元にフルリフレッシュ以前にCCD表に入って来たデータの消し込みが行われます。しかし、既存の手順では、CCD表の更新がなくIBMSNAP_COMMITSEQに番号が振られません。
そのため、フルリフレッシュ以前にCCD表に蓄えられたデータの消し込みが行われず、
次回の差分レプリケーション時にAPPLYにより、そのデータの反映を行ってしまいます。

キヤブチャー・コントロール・サーバーで実行

```
UPDATE ASN. IBMSNAP_REGISTER SET DISABLE_REFRESH = 1 WHERE SOURCE_OWNER = 'E503230' AND SOURCE_TABLE = 'EMP'
AND SOURCE_VIEW_QUAL = 0;
UPDATE ASN. IBMSNAP_PRUNCNTL SET SYNCHPOINT = NULL, SYNCHTIME = NULL
WHERE SOURCE_OWNER = 'E503230' AND SOURCE_TABLE = 'EMP'
AND SOURCE_VIEW_QUAL = 0 AND APPLY_QUAL = 'QUAL1' AND SET_NAME = 'SET1'
AND TARGET_OWNER = 'E503230' AND TARGET_TABLE = 'TGEMP';
-- CCD表への更新を追加
```

```
UPDATE E503230. CCDEMP SET IBMSNAP_OPERATION = 'G' WHERE IBMSNAP_OPERATION = 'G'
UPDATE ASN. IBMSNAP_PRUNCNTL SET SYNCHPOINT = X'00000000000000000000000000000000', SYNCHTIME = CURRENT_TIMESTAMP
WHERE SOURCE_OWNER = 'E503230' AND SOURCE_TABLE = 'EMP'
AND SOURCE_VIEW_QUAL = 0 AND APPLY_QUAL = 'QUAL1' AND SET_NAME = 'SET1'
AND TARGET_OWNER = 'E503230' AND TARGET_TABLE = 'TGEMP';
INSERT INTO ASN. IBMSNAP_SIGNAL
(SIGNAL_TIME, SIGNAL_TYPE, SIGNAL_SUBTYPE, SIGNAL_INPUT_IN, SIGNAL_STATE) VALUES
(CURRENT_TIMESTAMP, 'CMD', 'CAPSTART', '1', 'P');
```

アプライ・コントロール・サーバーで実行

```
UPDATE ASN. IBMSNAP_SUBS_SET SET ACTIVATE = 0 WHERE
SET_NAME = 'SET1' AND APPLY_QUAL = 'QUAL1';
---- 手動フルリフレッシュ----
```

アプライ・コントロール・サーバーで実行

```
UPDATE ASN. IBMSNAP_SUBS_MEMBR SET MEMBER_STATE = 'L' WHERE
SET_NAME = 'SET1' AND APPLY_QUAL = 'QUAL1' AND WHOS_ON_FIRST = 'S';
UPDATE ASN. IBMSNAP_SUBS_SET SET ACTIVATE = 1, LASTRUN = CURRENT_TIMESTAMP,
LASTSUCCESS = CURRENT_TIMESTAMP, SYNCHTIME = CURRENT_TIMESTAMP, SYNCHPOINT = NULL
WHERE SET_NAME = 'SET1' AND APPLY_QUAL = 'QUAL1'
```

- ▶ *Replica (Update Anywhere)*
- ▶ レプリケーション対象データの操作
- ▶ レプリケーション処理の操作
- ▶ データ共用・区分化対応
- ▶ 特殊な表、データタイプのレプリケーション
- ▶ 文字コード・データタイプ変換
- ▶ マルチベンダー・レプリケーション
- ▶ **その他**



内容

- Apply実行時のコードページ考慮点
- Apply BIND時のISOLATION UR注意点
- Apply(z/OS, OS/390) 30090N問題
- Apply(LUW) SQL1145N 問題
- Capture/Applyのアプリケーションからの使用
- V8でサポートされなくなった機能



Apply実行時のコードページ考慮点

- UDBのUnicodeまたはEUCデータベースを使用し、ソースのデータベースコードページが異なる場合、Apply環境はソースサーバー側のDB2CODEPAGEを指定する

- DB2 UDB間のReplicationでターゲットデータベースがUnicodeまたはEUCを使用している場合、コード変換によって文字の長さが長くなる場合に、Target tableのカラムの長さを十分にとってもデータのトランケーションが発生してしまう。（具体的にはEUCデータベースの場合には半角カタカナや特殊記号の場合、Unicodeデータベースの場合は英数小文字以外の文字）
- 【発生条件】
 - ▶ コード変換によって文字の長さが長くなり、ソース列定義の長さより長くなる場合に発生する。

解説:

■ 【対応】

Applyの実行環境をSource側のコードページにあわせることでトランケーションを抑止することでき回避可能です。（具体的にはAPPLYの実行環境のDB2CODEPAGEを設定）

レプリケーションにおけるコード変換の発生で関係するのは、ソース・データベース、ターゲット・データベースおよびAPPLYが稼動する環境のDB2CODEPAGE
これらを仮に

ソース・データベース : コードページA
APPLY : コードページB

ターゲット・データベース : コードページC
とすると、コード変換は以下の箇所で発生する可能性があります。 （各コードページは全て異なるとします）
Source -->Apply-->Target

CP-A CP-B CP-C : CP=codepage

①APPLYがソース表（CD表）からデータを受け取った時点で、CP-AからCP-Bへの変換が発生

②APPLYはデータを受け取るホスト変数の領域をソース表（CD表）の列定義から決定し、受け取ったデータをSPILLファイルに書き出す

③SPILLファイルからデータを読み込み、ターゲット表を更新、この時点でCP-BからCP-Cへの変換が発生

上記でCP-AからCP-Bへの変換で文字の長さが長くなる場合、ソース表（CD表）の列定義の長さのホスト変数では桁あふれが発生する場合があります。

この場合、SPILLファイルに書き出されたデータは既にトランケートされたデータであるため、ターゲット表にデータが挿入/更新されても、あふれたデータ部分がなくなってしまうことになります。

通常APPLYはターゲット・データベースと同じインスタンスで稼動させることが多く、

この場合はCP-B=CP-Cとなります。特にLinuxなどEUCをプライマリーのコードページで使用する環境でAPPLYを稼動させる場合は、特に当問題に合致しやすいといえます。

上記の場合にはCP-B=CP-AとなるようにApplyの実行環境のdb2codepage環境変数をSET/Exportで指定して実行してください

Apply BIND時のISOLATION UR注意点

■ Apply V8より曖昧なカーサーがURでバインドしてもCSに変更される

● 影響

- ▶ Applyがソースサーバーに接続し、REGISTER表を読み取り時、COMMIT_INTERVAL値を超えるまでLOCK Waitしてしまう

Applyは動的SQLを使用して登録情報を読み取る

	Cursor = Ambiguous Prep = nothing specified Bind = ISOLATION UR	Cursor = Ambiguous Prep = BLOCKING ALL Bind = ISOLATION UR	Cursor = Ambiguous Prep = nothing specified Bind = ISOLATION UR + BLOCKING ALL
V8 Static SQL	CS was used	CS was used	CS was used
V8 Dynamic SQL	CS was used	UR was used	UR was used
V7	UR	UR	UR

■ ISOLATION URでBINDしたPackageがCSへエスカレーションされる

● 明示的にFOR FETCH ONLYを指定していないカーサーのみ

- ▶ 動的、静的SQLの両方に影響する
- ▶ V8 Static SQLでBlocking ALL指定にも関わらずCSにISOLATIONが選択されることはトロントへ報告済み

● 曖昧なカーサーを使用した場合のみ問題が発生する

```
EXEC SQL DECLARE cr1 CURSOR FOR
  SELECT ROW_NUMBER,TEST_DATA,DESCRIPTION
  FROM TEST_SOURCE
```

解説:

- 明示的バインドを実施する場合は必ずBLOCKING ALLオプションを使用する
- MVSから明示的バインドを実施する場合は必ずCURRENTDATA (NO) ALLオプションを使用する

- ▶ Administration Guide: Performanceより

Note: Cursors that are updatable operating under the Uncommitted Read isolation level will behave as if the isolation level was cursor stability. When it runs a program using isolation level UR, an application can use isolation level CS. This happens because the cursors used in the application program are ambiguous. The ambiguous cursors can be escalated to isolation level CS because of a BLOCKING option. The default for the BLOCKING option is UNAMBIG. This means that ambiguous cursors are treated as updatable and the escalation of the isolation level to CS occurs. To prevent this escalation, you have the following two choices:

- Modify the cursors in the application program so that they are unambiguous.
- Change the SELECT statements to include the FOR READ ONLY clause.

Leave cursors ambiguous in the application program, but precompile the program or bind it with the BLOCKING ALL option to allow any ambiguous cursors to be treated as read-only when the program is run.

Apply (z/OS, OS/390) SQL30090N 問題

■ Apply (z/OS, OS/390) からNickname経由（他社DB Replicationなど）実行時、SQL30090で失敗する

- 原因

- ▶ UDB V8よりTCPIP DRDA 2PC AS機能が提供された

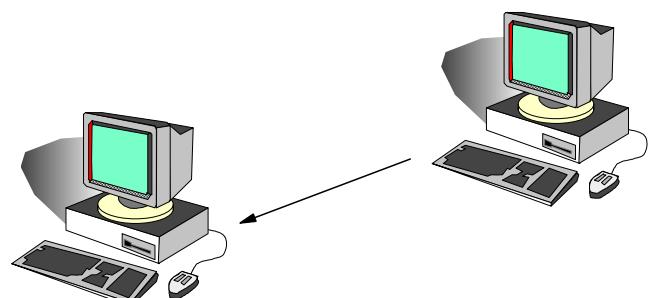
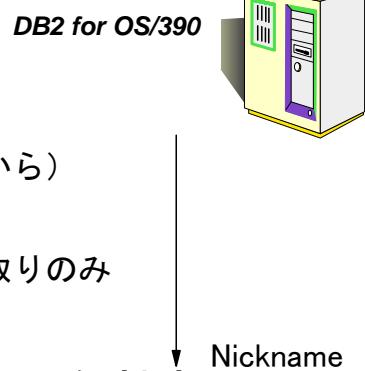
- 解決策

- ▶ FEDERATED=YESをUDB側に設定し DRDA 1PC ASとなる(Fixpak2から)

- 影響

- ▶ WAS/390, CICS/ESA, IMS/ESAからはローカル表であっても読み取りのみ
- ▶ Unix&WindowsからのXA applicationは影響なし

■ Nickname 2PC機能が提供された場合に変更される可能性あり



解説:

■ Apply/MVSなしでSQL30090N RC=18の再現方法(3partnameでNicknameを更新する)

```
=> db2 connect to kusatsu user azuma using azuma
```

```
Database Connection Information
```

```
Database server      = DB2 OS/390 7.1.1
SQL authorization ID = AZUMA
Local database alias = KUSATSU
```

```
[db2v8@marron(Ja_JP)/home/db2v8/fed]
=> db2 'update v8db.db2v8.source set c2=c2+1'
```

```
SQL30090N Operation invalid for application execution environment. Reason
code = "18". SQLSTATE=25000
```

Apply (LUW) SQL1145N 問題

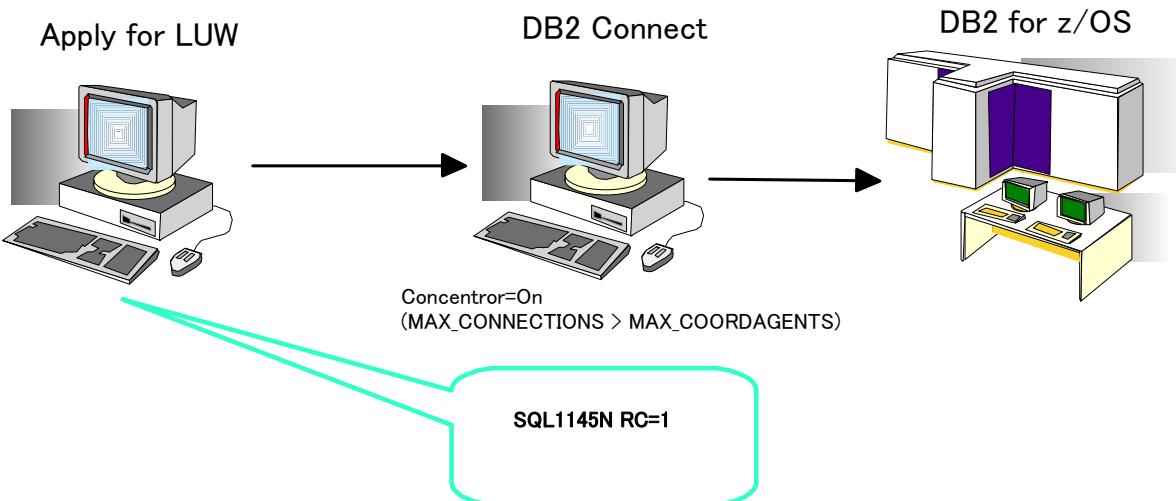
■ Apply/LUWをDB2 UDBコンセントレーターと組み合わせて起動不可

● 原因

- ▶ コンセントレーターがDynamic SQLをサポートしないため

● 回避策

- ▶ DB2 Connect コンセントレーター=OFF



解説:

■ ApplyでSQL1145N RC=1

```

PSET: Commit1 ok
PSET: connect to KUSATSU; rc = 1
PSET: CURRTSRVRTYPE is DSN, CURRTSRVRVERSION is 07
ABIND: KUSATSU: sqlca.sqlcode is -1145
2005-10-31-14.02.17.248439 ASN0552E "Apply" : "QUAL1" : "WorkerThread" : The program encountered an SQL
error. The server name is "KUSATSU". Th
e SQL request is "SELECT/VALUES". The table name is "asnmsgt.SQC". The SQLCODE is "-1145". The SQLSTATE
is "560AF". The SQLERRMC is "1~". The SQ
LERRP is "SQLJRIIF".

```

```

2005-10-31-14.02.17.248439 SQL1145N "Apply" : "QUAL1" : "WorkerThread" : PREPARE statement is not
supported when using a gateway concentrator.
Reason code: "1". SQLSTATE=560AF

```

```

*** SQL ERROR ***: SQL1145N PREPARE statement is not supported when using a gateway concentrator.
Reason code: "1". SQLSTATE=
560AF

```

Capture/Applyのアプリケーションからの使用

■ Capture/Apply API Support

- Capture および Apply アプリケーション・プログラミング・インターフェースを使用すると、収集プログラムと変更適用プログラムを両方とも、アプリケーションの中から開始できます。
- 収集プログラム(CAPTURE)はautostop オプションが必須
 - ▶ 新しい AUTOSTOP 呼び出しパラメーターを使うことによって、収集プログラムをバッチ・モードで実行することができます。 収集プログラムは、ログからすべての変更を収集して終了するまで実行されます。
- 変更適用プログラム(APPLY)のcopyonce オプションが必須
 - ▶ 新しい COPYONCE 呼び出しパラメーターを使うことによって、変更適用プログラムをバッチ・モードで実行することができます。 変更適用プログラムは、適格なすべてのサブスクリプション・セットの実行を試行してから終了します。

解説:

■ Capture/Apply API Support

asnccp コマンドを使用して収集プログラムを開始したり、ashapply コマンドを使用して変更適用プログラムを開始したりする代わりに、アプリケーションの中からルーチンを呼び出して、収集および変更適用プログラムを開始することができます。

これらのルーチンを使用するには、収集プログラムの場合は AUTOSTOP オプション、変更適用プログラムの場合は COPYONCE オプションを指定しなければなりません。これは、この API が同期実行しかサポートしていないためです。

解説:

■Capture_APIを使用した例

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "asn.h" /* replication API parameters */

/*****************/
/* This is a helper function to print out parameter contents as */
/* invoked by user. */
/*****************/
int printParms(const struct asnParms parms)
{
    int count = 0;
    printf("Start Capture from API: asncap ");
    if (parms.parmCount > 0)
    {
        for( count = 0; count < parms.parmCount; count++ )
    {
        printf("%s ", parms.parms[count]->val);
    }
        printf("\n\n");
        return (0);
    }
    else
    {
        printf("No parameter specified!\n\n");
        return(-1);
    }
}
```

解説:

```
/*****************/
/* This main routine sets the parameters required by Capture and */
/* invokes asnCapture. The variable rc indicates if Capture ran */
/* successfully or not. */
/*****************/
int main(int argc, char** argv)
{
    struct asnParms captureParms;
    struct asnParm *currParm;
    int rc = 0;
    int count = 0;

    /* This example has 4 parameters passed to Capture */
    /* Note: If you desire to eliminate some of the */
    /* parameters, make sure you change the parmCount and */
    /* captureParms.parms array appropriately. */
    /* captureParms.parms = (struct asnParm **)malloc (captureParms.parmCount */
    /* sizeof(struct asnParm*)); */

    captureParms.parmCount = 4; /* number of parameters */

    captureParms.parms = (struct asnParm **)malloc (captureParms.parmCount *
        sizeof(struct asnParm*));
```

解説:

```
*****
/* modify the string "CAPSRV" to reflect your own */
/* Capture Server. */
*****
currParm = (struct asnParm *)malloc(sizeof(struct asnParm));
strcpy(currParm->val, "CAPTURE_SERVER=V8DB");
currParm->byteCount = strlen(currParm->val);
captureParms.parms[0]=currParm; /* first Capture parameter */

*****
/* modify the string "WARMSI" to reflect your start */
/* mode for capture. */
*****
currParm = (struct asnParm *)malloc(sizeof(struct asnParm));
strcpy(currParm->val, "STARTMODE=WARMSI");
currParm->byteCount = strlen(currParm->val);
captureParms.parms[1]=currParm; /* second Capture parameter */

*****
/* add this parameter to send all messages to both the */
/* standard output and the log file */
*****
currParm = (struct asnParm *)malloc(sizeof(struct asnParm));
strcpy(currParm->val, "LOGSTOUT");
currParm->byteCount = strlen(currParm->val);
captureParms.parms[2]=currParm; /* third Capture parameter */
```

解説:

```
*****
/* required: AUTOSTOP */
*****
currParm = (struct asnParm *)malloc(sizeof(struct asnParm));
strcpy(currParm->val, "AUTOSTOP");
currParm->byteCount = strlen(currParm->val);
captureParms.parms[3]=currParm; /* fourth Capture parameter */

*****
/* other parameters you can add: */
/* (Please refer to the Replication Guide and */
/* Reference for parameter details) */
*/
/*-----*/
/* CAPTURE_SCHEMA=schema */
/* CAPTURE_PATH=path */
/* AUTOPRUNE */
/* COMMIT_INTERVAL */
/* LAG_LIMIT */
/* LOGREUSE */
/* MEMORY_LIMIT=n */
/* MONITOR_INTERVAL=n */
/* MONITOR_LIMIT=n */
/* PRUNE_INTERVAL=n */
/* RETENTION_LIMIT=n */
/* SLEEP_INTERVAL=n */
/* TERM */
/* TRACE_LIMIT=n */
*****
```

解説:

```

rc = printParms(captureParms);           /* print parameters out to verify */
if ( rc == -1 )                         /* no parameters specified */
    return(rc);

rc = asnCapture (&captureParms);      /* invoke Capture API */

if ( rc == 0 )                          /* check rc from Capture */
    printf("Capture completed sucessfully.\n");
else
    printf("Capture failed with rc = %d\n", rc);

for(count=0; count < captureParms.parmCount; count++)
    free(captureParms.parms[count]);
free(captureParms.parms);               /* free parameter list */

return(rc);
}

```

解説:

■コンパイル、リンク (Windows環境の場合)

```

C:\repl\sv8\test>nmake /f capture_api_nt.mak

Microsoft (R) Program Maintenance Utility Version 6.00.8168.0
Copyright (C) Microsoft Corp 1988-1998. All rights reserved.

cl /c /I"c:\sql\lib\include" capture_api.c
Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 12.00.8168 for 80x86
Copyright (C) Microsoft Corp 1984-1998. All rights reserved.

capture_api.c
link /libpath:"c:\sql\lib\lib" db2repl.lib capture_api.obj
Microsoft (R) Incremental Linker Version 6.00.8168
Copyright (C) Microsoft Corp 1992-1998. All rights reserved.

```

解説:

■実行結果

```
C:\$rep\$\$v8test>capture_api
Start Capture from API: asncap CAPTURE_SERVER=V8DB STARTMODE=WARMSI LOGSTDOUT AU
TOSTOP

2002-09-16-15.44.02.284000 ASN0529I "Capture" : "ASN" : The value of "CAPTURE_S
ERVER" was set to "V8DB" at startup by the following method: "COMMANDLINE".
2002-09-16-15.44.02.294000 ASN0529I "Capture" : "ASN" : The value of "CAPTURE_S

2002-09-16-15.44.02.574000 ASN8008D "Capture" : "ASN" : "Created" IPC queue wit
h key(s) "(OSSEIPC0tempDB2.V8DB ASN.CAP IPC, OSSEIPC1tempDB2.V8DB ASN.CAP IPC, 0
SSEIPC2tempDB2.V8DB ASN.CAP IPC)".
2002-09-16-15.44.03.626000 ASN0100I CAPTURE "ASN". The Capture program initiali
zation is successful.
2002-09-16-15.44.03.636000 ASN0109I CAPTURE "ASN". The Capture program has succ
cessfully initialized and is capturing data changes for "1" registrations. "0" re
gistrations are in a stopped state. "0" registrations are in an inactive state.

2002-09-16-15.44.03.686000 ASN0133I CAPTURE "ASN". The Capture program has reac
hed the end of the active log and will terminate because the AUTOSTOP feature is
specified.
2002-09-16-15.44.03.706000 ASN0123I CAPTURE "ASN". At program termination, the
highest log sequence number of a successfully captured log record is "853D:BA72:
0000:0000:0000" and the lowest log sequence number of a record still to be commi
tted is "0000:0000:0000:6E07:5AFA".
2002-09-16-15.44.14.621000 ASN8008D "Capture" : "ASN" : "Destroyed" IPC queue w
ith key(s) "(OSSEIPC0tempDB2.V8DB ASN.CAP IPC, OSSEIPC1tempDB2.V8DB ASN.CAP IPC,
OSSEIPC2tempDB2.V8DB ASN.CAP IPC)".
2002-09-16-15.44.14.621000 ASN0008I CAPTURE "ASN". The Capture program was stop
ped.
```

解説:

■このページはブランクです

V8でサポートされなくなった機能

- LOGRETAIN=CAPTUREのサポート廃止
- RUNSTATSの自動使用廃止
- DB2 DataJoiner レプリケーション管理(DJRA) ツールの廃止
 - V8でのレプリケーション環境では、レプリケーション・センターに統合

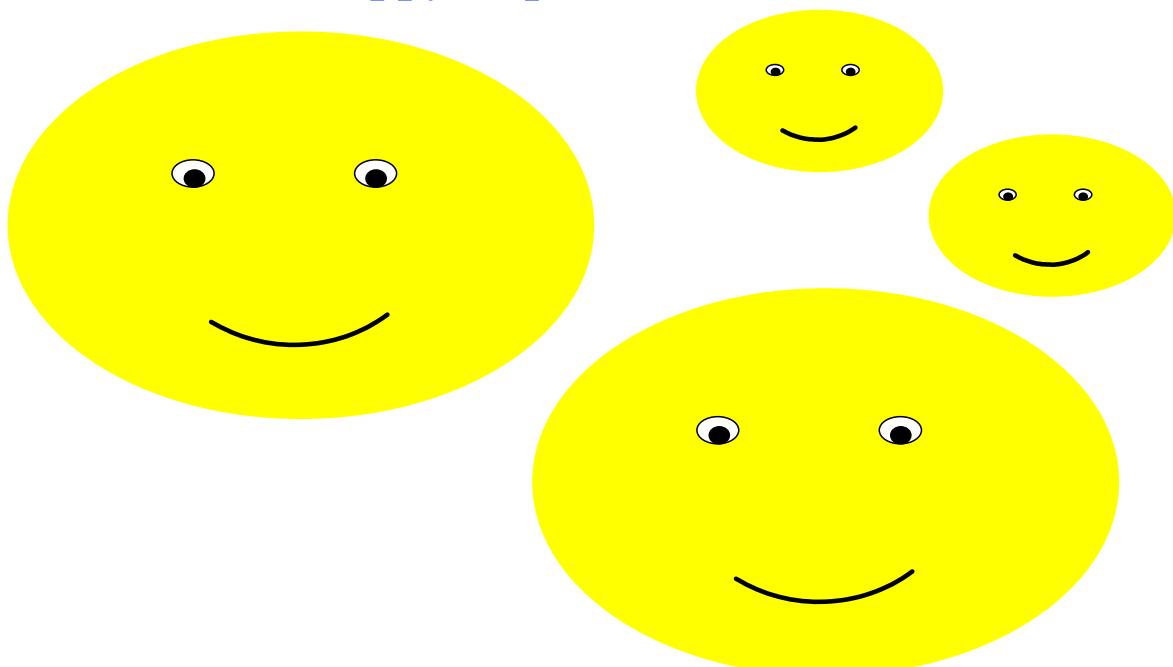


blank page





Happy Replication



おしまい

