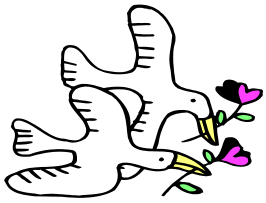
DB2 Information Management Software

第2章
SQLの機能強化

＜第2．00版　2005年　3月＞

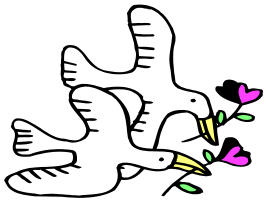
お断り：当資料は、DB2 Universal Database for Linux, UNIX and Windows V8.2 をベースに作成されています。



ブランク・ページです。

目次

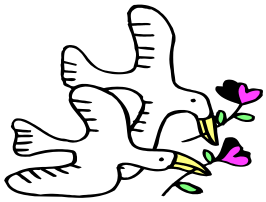
- Native PSM
- トリガー本体内のプロシージャーの呼び出し
- SAVEPOINT
- Larger SQL
- SET LOCKTIMEOUT
- 生成列の定義変更
- REOPTオプション
- USE AND KEEP LOCKS



ブランク･ページです。

Native PSM

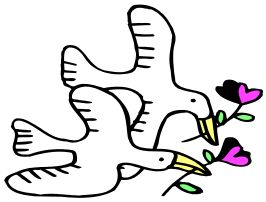
お断り：当資料は、DB2 Universal Database for Linux, UNIX and Windows V8.2 をベースに作成されています。



ブランク・ページです。

目次

- SQL Routineとは
- SQL Routineの種類とインプリメンテーション
- SQL PL , Inline SQLとは
- SQL Procedureの利点と新機能
- DB2 UDB V8.2での使用制約の変更
- SQL PL V8.2での使用制限の緩和

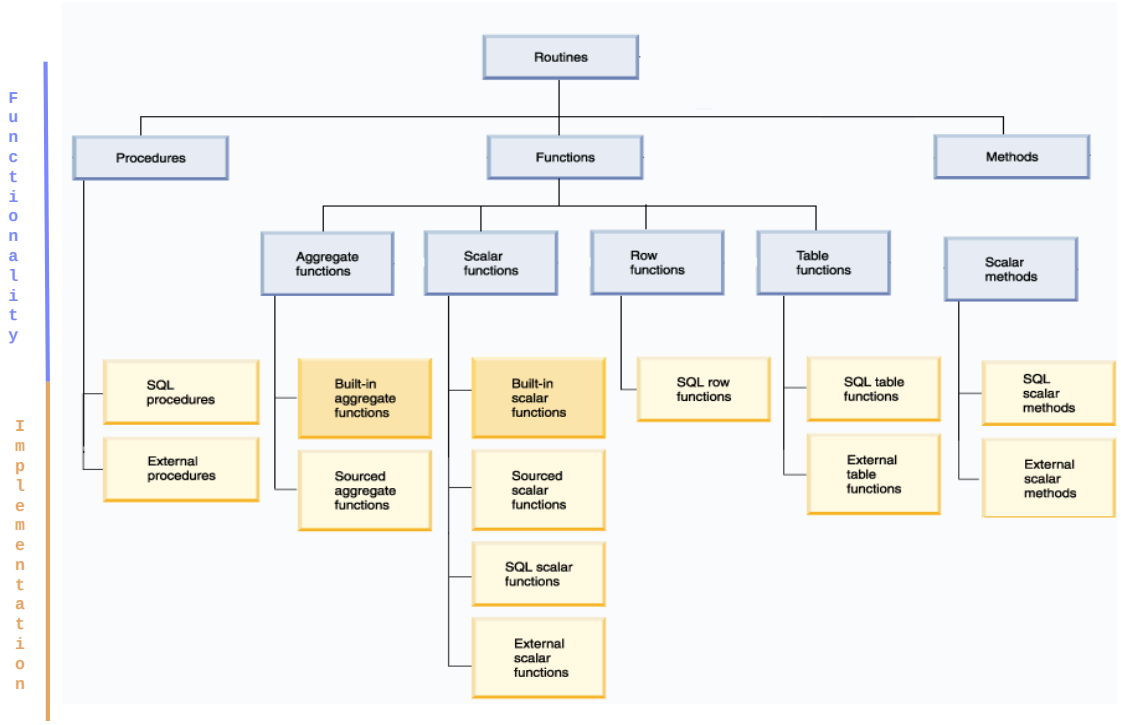


ブランク･ページです。

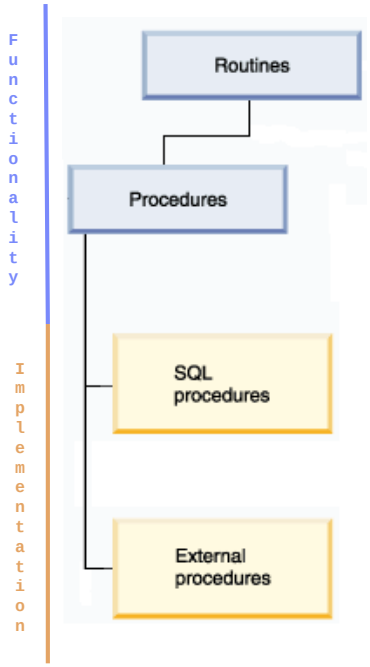
SQL Routineとは

- **プロシジャー**
  - クライアント側のアプリケーションLogicをデータベースにカプセル化
  - 多くのインターフェースを使用して呼ばれるサブルーチン
    - `CALL PROC(a_in,b_out);`
- **Functions**
  - Scalar,Table,Row Function
  - SQLステートメントを使用しカラムリストとして呼び出し可能
  - テーブルやデータの加工を実施可能
    - `SELECT foo_scalar(c1) FROM T;`
    - `SELECT a FROM T, TABLE(foo_table(T.c1)) F;`
- **Methods**
  - UDFの機能を提供
    - `SELECT C..foo_method() FROM T;`

SQL Routineの種類とインプリメンテーション



SQL Routine: ストアードプロシージャー



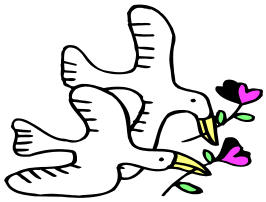
- **External Procedures**
  - CREATE PROCEDUREステートメントで EXTERNAL指定で作成
  - Logicはclassやlibraryでデータベース外に おかれている
    - C, C++, Java (JDBC/SQLJ), OLE, COBOL,
    - NET CLR languages: C#, VB
- **SQL Procedures （ UDB V7.1から ）**
  - CREATE PROCEDURE Lang SQLで作成
  - LogicはCREATE PROCEDUREステートメント内のBodyに格納
  - **SQL PL言語でコーディング**
    - **SQL** static and dynamic statement (s)

SQL PL , Inline SQLとは

- **SQL PL**
  - SQL PL = SQL + Control flow
  - 他社のT-SQL (Microsoft/Sybase), PL/SQL (Oracle) と同等
  - C言語に変換される (DB2 UDB Version 8.1まで)
  - SQL Procedureでのみ使用可能
- **Inline SQL (Marco PSM)**
  - SQL PLのViewのようなもの、SQL PLのSubset
  - 制限された (macro) 機能
    - No condition handling, save points, …
    - Triggers, SQL Functions, SQL Methods, dynamic compound内で使用可能
    - Use CALL in V8.2 for complex logic （V8.2 から）

解説：T-SQL vs SQL PL

TSQL	SQL Procedures
DECLARE @varname datatype =defaultvalue	DECLARE varname datatype DEFAULT defaultvalue;
SELECT @var1=value	SET var1 = value;
SELECT @var1=colname from table where…	SET var1 = (SELECT colname from table where …);
SELECT @v1=col1,@v2=col2,@v3=col3 from table...	SELECT col1,col2,col3 into v1,v2,v3 from table...
WHILE expression BEGIN … END	WHILE expression DO … END WHILE;
CONTINUE	ITERATE
BREAK	LEAVE loop_label
IF (….) BEGIN … END ELSE ….	IF (….) THEN … ELSE …. END IF;
EXECUTE procname(parm1,parm2,...)	CALL procname(parm1,parm2,…);
EXECUTE @retval=procname(parm1,parm2,…)	CALL procname(parm1,parm2,…); GET DIAGNOSTICS retval = RETURN_STATUS;
RETURN <int_value>	RETURN < int_expr>;
@@rowcount	GET DIAGNOSTICS <var> = ROWCOUNT
GOTO <label>	GOTO <label>
RAISERROR <error>,"msg"	SIGNAL <sqlstate> SET MESSAGE_TEXT='msg' (The semantic isslightly different because of the SQLSTATE type (CHAR5) and RAISERROR does not interrupt control flow.)

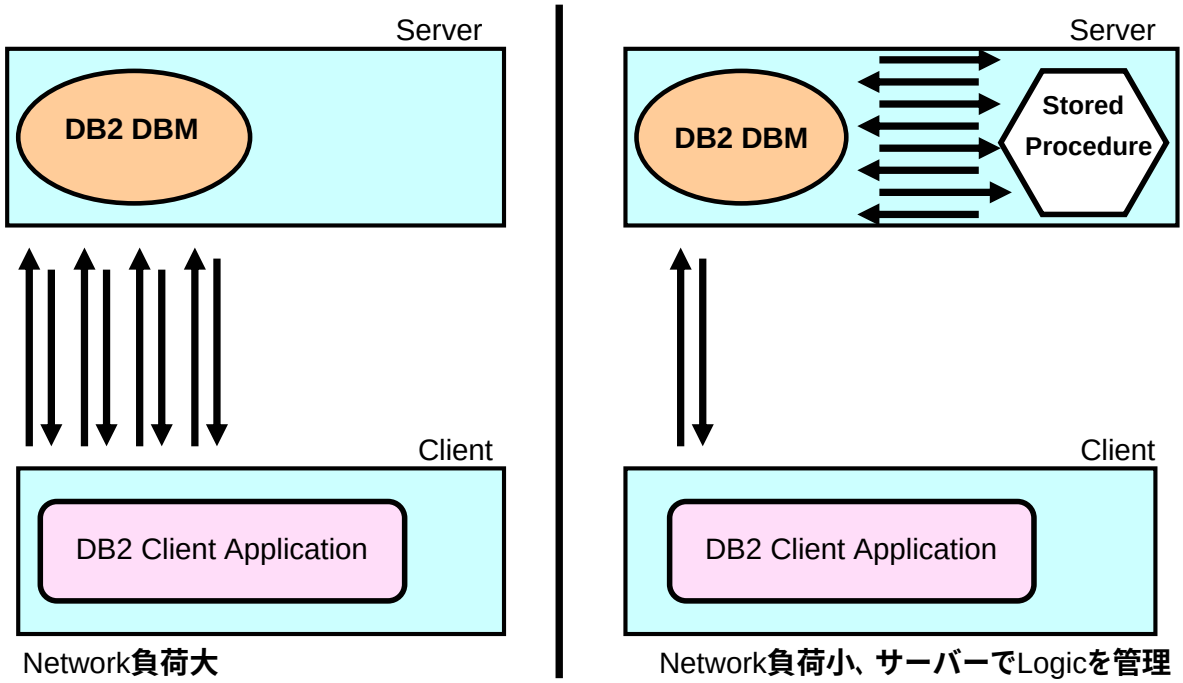


ブランク・ページです。

SQL Procedureの利点と新機能

- DB2 UDB familyでサポート (UNIX, NT, AS400, OS390, zOS)
- SQL/PSM のSubset(SQL99 ANSI/ISO/IEC standard – SQL/'PSM)
- パラメーターの受け渡し: IN, OUT, INOUT
- 強力condition / error-handling モデル
- DB2独自の拡張 (extensions to standard): GOTO, RETURN statements, Result Sets, SQLSTATE,…
- データベースのObjectとして格納
  - 自動的にバックアップ、リストアー時に対象になる
- CALLステートメントから呼び出し可能
- ネスト (16階層まで) また、リカーシブ (recursive) サポート
  - より高いSQL DATAアクセスレベルを呼び出すことはできない例
  - NO SQL < CONTAINS SQL < READS SQL DATA < MODIFIES SQL DATA
- Savepointを使用しTransactionコントロールも可能
- **DB2 UDB (LUW) Version 8.2新機能:**
  - Cコンパイラーが不要
  - SQL proceduresのMax Sizeが64KBから2 MB!
  - トリガーからの呼び出し可能

解説：Stored Procedureの利点



解説：Savepointを使用しTransactionコントロールも可能 (1/3)

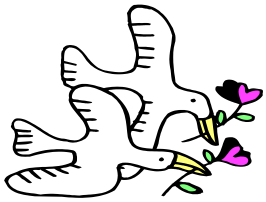
```
[db2v8@dbdcserv(Ja_JP)/home/db2v8/nativePSM]
==> db2 -vtd% -f savepoint.sql
DROP TABLE TEST_SOURCE
DB20000I  The SQL command completed successfully.

CREATE TABLE TEST_SOURCE (ROW_NUMBER INT NOT NULL, TEST_DATA  INT, DESCRIPTION  CHAR(20))
DB20000I  The SQL command completed successfully.

CREATE UNIQUE INDEX TEST_SOURCE_X ON TEST_SOURCE (ROW_NUMBER)
DB20000I  The SQL command completed successfully.

INSERT INTO TEST_SOURCE VALUES (1,100,'this is row1')
DB20000I  The SQL command completed successfully.

INSERT INTO TEST_SOURCE VALUES (2,200,'this is row2')
DB20000I  The SQL command completed successfully.

INSERT INTO TEST_SOURCE VALUES (3,300,'this is row3')
DB20000I  The SQL command completed successfully.

INSERT INTO TEST_SOURCE VALUES (4,400,'this is row4')
DB20000I  The SQL command completed successfully.

INSERT INTO TEST_SOURCE VALUES (5,500,'this is row5')
DB20000I  The SQL command completed successfully.

INSERT INTO TEST_SOURCE VALUES (6,600,'this is row6')
DB20000I  The SQL command completed successfully.

DROP PROCEDURE savepttest
DB20000I  The SQL command completed successfully.
```


解説：Savepointを使用しTransactionコントロールも可能 (2/3)

```
CREATE PROCEDURE savepttest ()     LANGUAGE SQL
BEGIN
   DECLARE COL1,COL2 SMALLINT ;    DECLARE atend SMALLINT DEFAULT 0;
   DECLARE CUR1 CURSOR WITH HOLD FOR
      SELECT ROW_NUMBER,TEST_DATA FROM TEST_SOURCE ORDER BY ROW_NUMBER;

   DECLARE CONTINUE HANDLER FOR NOT FOUND SET atend=1;
   DECLARE EXIT HANDLER FOR SQLEXCEPTION SET atend=1;
   OPEN CUR1;
   FETCH CUR1 INTO COL1,COL2;
   nextrow:
      IF atend = 0 THEN

         SAVEPOINT svpt ON ROLLBACK RETAIN CURSORS;
         WHILE  ( atend = 0 ) DO
               SET COL2 =  COL2 + 100 ;
               UPDATE TEST_SOURCE SET TEST_DATA = COL2 WHERE row_number = COL1;
               FETCH CUR1 INTO COL1,COL2;
         END WHILE;
         IF COL2 > 400 THEN
               COMMIT;
         ELSE
               ROLLBACK TO SAVEPOINT svpt;
               RELEASE SAVEPOINT svpt;
         END IF;
         GOTO nextrow;
      END IF;

END
DB20000I  The SQL command completed successfully.
```

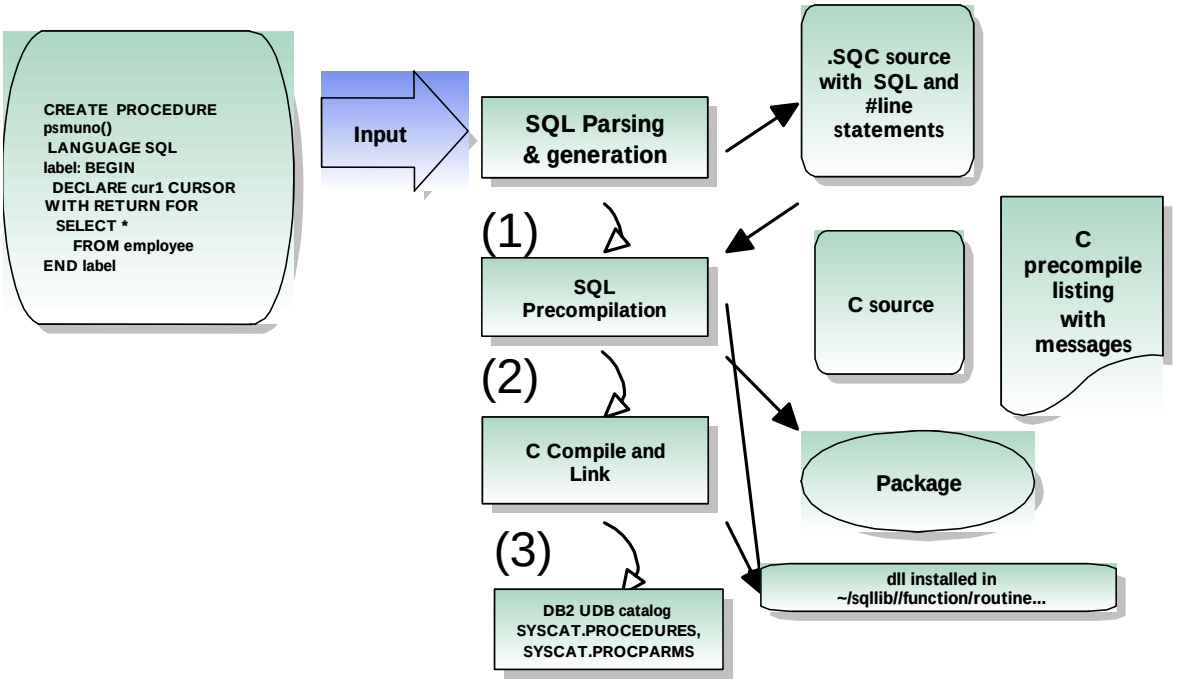
解説：Savepointを使用しTransactionコントロールも可能 (3/3)

```
call savepttest()

  Return Status = 0


select * from TEST_SOURCE

ROW_NUMBER  TEST_DATA   DESCRIPTION
----------- ----------- --------------------
          1         200 this is row1
          2         300 this is row2
          3         400 this is row3
          4         500 this is row4
          5         600 this is row5
          6         700 this is row6

  6 record(s) selected.
```

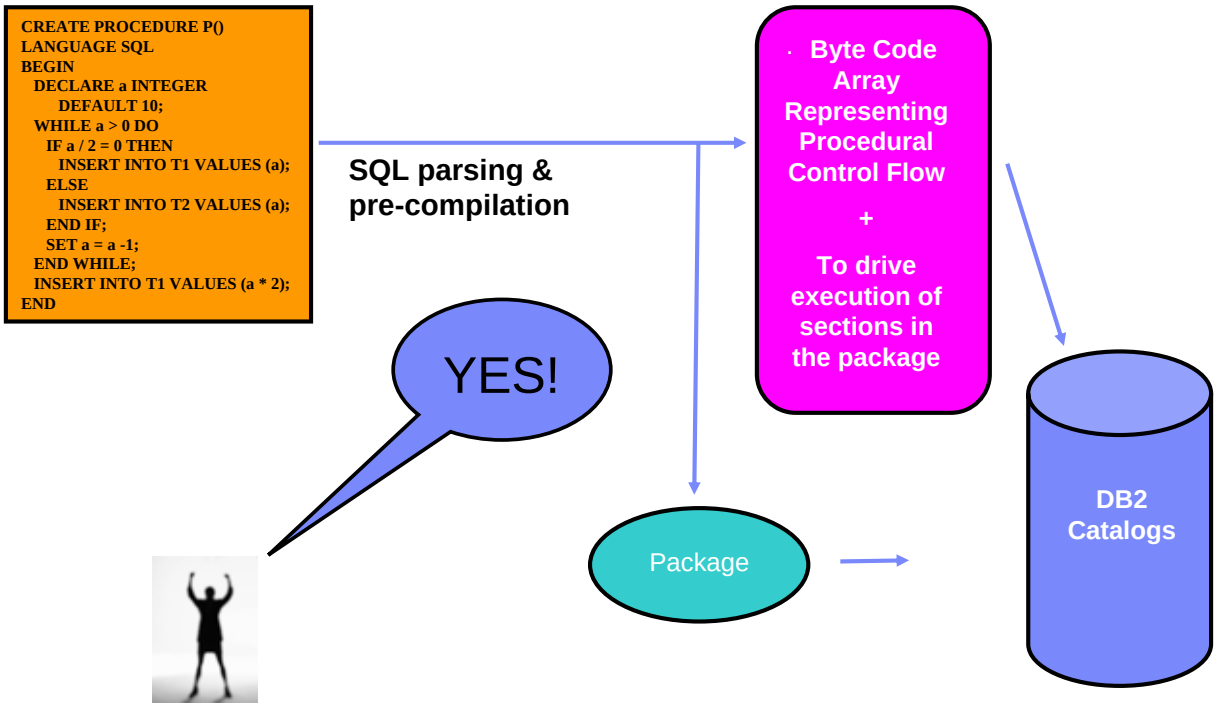



ブランク・ページです。

解説：C言語への変換 (V8.1まで)



解説：V8.2 Native PSM （C言語への変換なし）



解説：V8.2 Native PSM

```
CREATE PROCEDURE P()
LANGUAGE SQL
BEGIN
  DECLARE a INTEGER DEFAULT 10;
  WHILE a > 0 DO
    IF a / 2 = 0 THEN
      INSERT INTO T1 VALUES (a);
    ELSE
      INSERT INTO T2 VALUES (a);
    END IF;
    SET a = a -1;
  END WHILE;
  INSERT INTO T1 VALUES (a * 2);
END
```

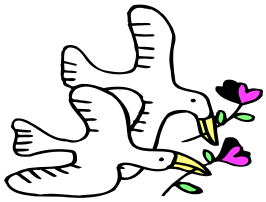
Bytecodeプログラム

(SYSIBM.SYSCODEPROPERTIES.SQL_COMPILED_CODEへ格納)

```
Begin: EvalQuery 0
While: EvalQuery 1
       IfFalseGoto End
       EvalQuery 2
       IfFalseGoto Else
       EvalQuery 3
       Goto Endif
Else:  EvalQuery 4
Endif: EvalQuery 5
       Goto While
End:   EvalQuery 6
```

Package with sections for these queries

```
VALUES (10) INTO :a;                                                    /* Query 0 */
VALUES (CASE WHEN :a > 0 THEN 1 ELSE 0) INTO :boolresult;       /* Query 1 */
VALUES (CASE WHEN :a  / 2 = 0 THEN 1 ELSE 0) INTO :boolresult; /* Query 2 */
INSERT INTO T1 VALUES (:a);                                             /* Query 3 */
INSERT INTO T2 VALUES (:a);                                             /* Query 4 */
VALUES (:a - 1) INTO :a;                                                /* Query 5 */
INSERT INTO T1 VALUES (:a * 2);
```

ブランク・ページです。

DB2 UDB V8.2での使用制約の変更(1)

DB2_SQLROUTINE_PREPOPTSレジストリー変数 (V7.1より)

(プリコンパイルおよびバインド・オプションのカスタマイズ)

- プリコンパイルおよびバインド・オプションは、DB2_SQLROUTINE_PREPOPTS DB2 レジストリー変数を設定することによってカスタマイズできます。
- db2set DB2_SQLROUTINE_PREPOPTS=options
- ここで、options は、DB2 プリコンパイラーで使用するプリコンパイル・オプションのリストを示します。使用できるオプションは、次のものだけです。

```
BLOCKING {UNAMBIG | ALL | NO}
DATETIME {DEF | USA | EUR | ISO | JIS | LOC}
DEGREE {1 | degree-of-parallelism | ANY}
DYNAMICRULES {BIND | RUN}
EXPLAIN {NO | YES | ALL}
EXPLAINSNAP {NO | YES | ALL}
INSERT {DEF | BUF}
ISOLATION {CS |RR |UR |RS |NC}
QUERYOPT optimization-level
VALIDATE {RUN | BIND}
```


DB2 UDB V8.2での使用制約の変更(2)

SET_ROUTINE_OPTS (V8.2)

(プリコンパイルおよびバインド・オプションのカスタマイズ)

- プリコンパイルおよびバインド・オプションは、DB2_SQLROUTINE_PREPOPTS DB2 レジストリー変数を設定することによってカスタマイズできます。これらのオプションは、SET_ROUTINE_OPTS ストアード・プロシージャーを使用して、プロシージャー・レベルで変更できます。現行セッションで SQL プロシージャーを作成するために設定されているオプションの値は、GET_ROUTINE_OPTS 関数を使用して取得できます。
- プロシージャーは、現行セッションで SQL プロシージャーを作成するのに使用されるオプションを設定します。この設定は、DB2_SQLROUTINE_PREPOPTS レジストリー変数で指定された、インスタンス全体の設定をオーバーライドします。

REBIND_ROUTINE_PACKAGE (V8.1)

- REBIND_ROUTINE_PACKAGE プロシージャーは、SQL プロシージャーに関連付けられたパッケージを再バインドします。これは機能的には REBIND コマンドと同じですが、これは、パッケージ名の代わりにプロシージャー名を引き数として使用します。REBIND_ROUTINE_PACKAGE プロシージャーは、コマンド行またはアプリケーションから呼び出すことができます。

解説：SET_ROUTINE_OPTS()使用例

```
[db2v8@dbdcserv(Ja_JP)/home/db2v8]
==> db2 "call set_routine_opts('ISOLATION UR')"

  Return Status = 0

[db2v8@dbdcserv(Ja_JP)/home/db2v8]
==> db2 -vtd$ -f rset.db2

CREATE PROCEDURE RSET( )
LANGUAGE SQL
SPECIFIC RSETSP
READS SQL DATA
BEGIN
DECLARE C1 CURSOR WITH RETURN TO CALLER FOR
SELECT * FROM TEST_SOURCE ORDER BY ROW_NUMBER;
OPEN C1;
END
DB20000I  The SQL command completed successfully.

[db2v8@dbdcserv(Ja_JP)/home/db2v8]
==> db2  'call rset()'

  Result set 1
  --------------
  ROW_NUMBER  TEST_DATA   DESCRIPTION
  ----------- ----------- --------------------
          100         100 Committed
          101         101 In-Flight

  2 record(s) selected.
  Return Status = 0
```

```
[db2v8@dbdcserv(Ja_JP)/home/db2v8]
==> db2 +c "insert into test_source values (101,101,'In-Flight')"
DB20000I  The SQL command completed successfully.
```


解説：REBIND_ROUTINE_PACKAGE()使用例

```
==> db2 -vtd$ -f rset.db2
drop procedure RSET
DB20000I  The SQL command completed successfully.

CREATE PROCEDURE RSET
( )
LANGUAGE SQL
SPECIFIC RSETSP
READS SQL DATA
BEGIN
DECLARE C1 CURSOR WITH RETURN TO CALLER FOR
SELECT * FROM TEST_SOURCE ORDER BY ROW_NUMBER;
OPEN C1;
END
DB20000I  The SQL command completed successfully.


[db2v8@dbdcserv(Ja_JP)/home/db2v8]
==> db2 "call sysproc.rebind_routine_package('P','RSET','ANY')"

  Return Status = 0

[db2v8@dbdcserv(Ja_JP)/home/db2v8]
==> db2 "call sysproc.rebind_routine_package('SP','RSETSP','ANY')"

  Return Status = 0
```

‘P’ →プロシージャー名
‘SP’ → Specific名

DB2 UDB V8.2での使用制約の変更(3)

- 下記2つのCLPコマンドはV7.2, V8.1同様に使用可能
  - GET ROUTINE
    - データベースから.SARファイルへの抽出
  - PUT ROUTINE
    - .SARファイルからデータベースへの格納
- 2つの"Built-in"ストアードプロシージャー
  - SYSFUN.GET_ROUTINE_SAR
  - SYSFUN.PUT_ROUTINE_SAR

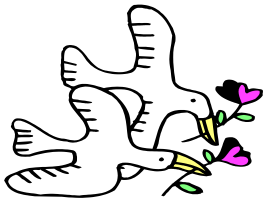
解説：GET ROUTINE/PUT ROUTINE使用例

```
[db2v8@dbdcserv(Ja_JP)/home/db2v8]
==> db2 connect to sample

   Database Connection Information

 Database server        = DB2/6000 8.2.0
 SQL authorization ID   = DB2V8
 Local database alias   = SAMPLE

[db2v8@dbdcserv(Ja_JP)/home/db2v8]
==> db2 get routine into rset.sar from procedure rset
DB20000I  The GET ROUTINE command completed successfully.
[db2v8@dbdcserv(Ja_JP)/home/db2v8]
==> ls -l *.sar
-rw-rw-rw-   1 db2v8    db2adm         1933 Oct 04 23:33 rset.sar
==> db2 connect to v8db

  Database Connection Information

 Database server        = DB2/6000 8.2.0
 SQL authorization ID   = DB2V8
 Local database alias   = V8DB

[db2v8@dbdcserv(Ja_JP)/home/db2v8]
==> db2 put routine from rset.sar owner db2v8
DB20000I  The PUT ROUTINE command completed successfully.
[db2v8@dbdcserv(Ja_JP)/home/db2v8]
```

解説：GET ROUTINE/PUT ROUTINE使用例

```
[db2v8@dbdcserv(Ja_JP)/home/db2v8]
==> db2 connect to sample

   Database Connection Information

 Database server        = DB2/6000 8.2.0
 SQL authorization ID   = DB2V8
 Local database alias   = SAMPLE

[db2v8@dbdcserv(Ja_JP)/home/db2v8]
==> db2 get routine into rset.sar from procedure rset
DB20000I  The GET ROUTINE command completed successfully.
[db2v8@dbdcserv(Ja_JP)/home/db2v8]
==> db2 connect to windb user azuma using x080598x

   Database Connection Information

 Database server        = DB2/NT 8.2.0
 SQL authorization ID   = AZUMA
 Local database alias   = WINDB

[db2v8@dbdcserv(Ja_JP)/home/db2v8]
==> db2 put routine from rset.sar owner azuma
SQL0443N  Routine "SYSFUN.PUT_ROUTINE_SAR" (specific name "PUT_SAR") has
returned an error SQLSTATE with diagnostic text "-20135, 55046, 1".
SQLSTATE=38000
```

異なるOS間の移動はV8.2でも不可



ブランク・ページです。

SQL PL V8.2での使用制限の緩和

- FOR Loop内でのWITH HOLD指定可能 (V8.2)
  - FOR Loop内で大量の挿入などを実施している場合、COMMITを発行し獲得しているLOCKを解除したいが、COMMITを発行するとカーサーが破壊されFOR Loopの継続ができなかった。このような問題を解決するためFOR Loop内でWITH HOLDを指定し FOR Loopカーサーを宣言可能にした
- 変数からのSIGNAL(V8.2)
  - いままでのシグナルステートメントは特定のSQLSTATEを静的に指定しなければならなかったが、CHAR(5)の変数で指定可能になった

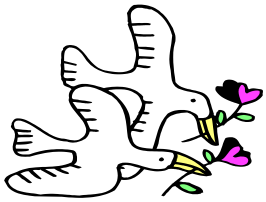
解説：FOR Loop内でのWITH HOLD指定可能

```
(Script)
drop table scratch%
create table scratch (c1 int)%
insert into scratch values 1,2,3%
drop table tletter%
create table tletter (c1 int)%
drop procedure fortest%

echo -- Nested Compound  within FOR   ------------%

create procedure fortest ()
language sql
begin not atomic
 declare data int default 0;
l: for row as cur1 CURSOR with hold for
    select c1 from scratch
    do
    set data = c1;
    insert into tletter values (data);
    commit;
 end for l;
end%

call fortest()%

select * from tletter%
```

```
 (実行結果)
.....
-- Nested Compound  within FOR   --------------------
create procedure fortest ()
language sql
begin not atomic
 declare data int default 0;
l: for row as cur1 CURSOR with hold for
    select c1 from scratch
    do
    set data = c1;
    insert into tletter values (data);
    commit;
 end for l;
end
DB20000I  The SQL command completed successfully.

call fortest()

  Return Status = 0


select * from tletter

C1
-----------
          1
          2
          3

  3 record(s) selected.
```

トリガー本体内のプロシージャーの呼び出し

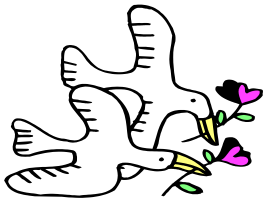
お断り：当資料は、DB2 Universal Database for Linux, UNIX and Windows V8.2 をベースに作成されています。



ブランク・ページです。

目次

- CALL in Triggerとは
- 制約事項



ブランク・ページです。

CALL in Triggerとは

■ トリガー本体内のプロシージャーの呼び出しが可能

●今後は、トリガー・アクション内のプロシージャーを参照する CALL ステートメントを実行すれば、単一パーティション環境においてトリガーからか、または他の動的コンパウンド・ステートメントからプロシージャーを呼び出すことができます。 CALL ステートメントは、SQL プロシージャーと外部プロシージャーの実行に使用されます。

```
create trigger TR1 after insert on T1
 referencing new as new
  for each row mode db2sql
  begin atomic
  declare row int;
  call p1(row);
  insert into t2 values (row);
  end
```


解説：トリガー内でのCALL例

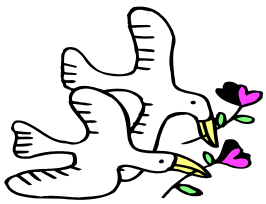
```
create table rt1 (c1 int)
DB20000I  SQL コマンドが正常に終了しました。

insert into rt1 values (1),(2),(3)
DB20000I  SQL コマンドが正常に終了しました。

CREATE PROCEDURE P1(OUT result int)
MODIFIES SQL DATA
LANGUAGE SQL
OLD SAVEPOINT LEVEL
BEGIN NOT ATOMIC
  SET result = ( select count(*) from rt1) ;
  END
DB20000I  SQL コマンドが正常に終了しました。

call p1(?)

  出力パラメーターの値
  -------------------------
  パラメーター名: RESULT
  パラメーター値: 3

  リターン状況 = 0
```

```
create table t1 (c1 int)
DB20000I  SQL コマンドが正常に終了しました。

create table t2 (c1 int)
DB20000I  SQL コマンドが正常に終了しました。

create trigger TR1 after insert on T1
 referencing new as new
  for each row mode db2sql
  begin atomic
  declare row int;
  call p1(row);
  insert into t2 values (row);
  end
DB20000I  SQL コマンドが正常に終了しました。

insert into t1 values (1),(2)
DB20000I  SQL コマンドが正常に終了しました。

select * from t2

C1
-----------
          3
          3

  2 レコードが選択されました。
```

解説：INSTEAD OF トリガー内でのCALL例

```
create table rt1 (c1 int)
DB20000I  The SQL command completed
successfully.

insert into rt1 values (1),(2),(3)
DB20000I  The SQL command completed
successfully.

CREATE PROCEDURE P1(OUT result int)
MODIFIES SQL DATA
LANGUAGE SQL
OLD SAVEPOINT LEVEL
BEGIN NOT ATOMIC
  SET result = ( select count(*) from rt1) ;
  END
DB20000I  The SQL command completed
successfully.

call p1(?)

  Value of output parameters
  --------------------------
  Parameter Name  : RESULT
  Parameter Value : 3

  Return Status = 0
```

```
create table t1 (c1 int)
DB20000I  The SQL command completed successfully.

create view v1 as select c1 from t1
DB20000I  The SQL command completed successfully.

create table t2 (c1 int)
DB20000I  The SQL command completed successfully.

create trigger TR1 instead of insert on v1
 referencing new as new
  for each row mode db2sql
  begin atomic
  declare row int;
  call p1(row);
  insert into t2 values (row);
  end
DB20000I  The SQL command completed successfully.

insert into v1 values (1)
DB20000I  The SQL command completed successfully.

select * from t2

C1
-----------
          3

  1 record(s) selected.
```

ブランク・ページです。

制約事項

- SQL トリガー、SQL ルーチン、動的コンパウンド・ステートメントのいずれかからプロシージャーを呼び出すときには、 以下の制約事項が適用されます
  - パーティション・データベース環境では、トリガーまたはSQL UDF からプロシージャーを呼び出せません。
  - 対称マルチプロセッサー (SMP) マシンでは、トリガーからのプロシージャー呼び出しが 1 つのプロセッサーで実行されます。
  - トリガーから呼び出すプロシージャーには、COMMIT ステートメント、または作業単位のロールバックを試行する ROLLBACK ステートメントを組み込めません。 ROLLBACK TO SAVEPOINT ステートメントは使用できますが、 指定のセーブポイントがプロシージャー内に存在している必要があります。
  - **(ランタイム・エラー)** トリガーが読み取り/書き込みを行う表に対してプロシージャーも読み取り/書き込みを行おうとすると、読み取り/書き込みの競合が検出された場合にエラーが発生します。
  - アクセス・レベル MODIFIES SQL DATA で作成したプロシージャーを参照する CALL ステートメントを含んだ BEFORE トリガーは作成できません。
  - プロシージャーによってフェデレーテッド表を変更 (参照も) してはなりません。つまり、ニックネームの検索 UPDATE、ニックネームの検索 DELETE、ニックネームの検索 INSERT をプロシージャーに含めてはなりません。
  - プロシージャーに指定した結果セットにはアクセスできません。

解説：パーティション・データベース環境でのトリガー内でのCALL例

```
create trigger TR1 after insert on T1
 referencing new as new
  for each row mode db2sql
  begin atomic
  declare row int;
  call p1(row);
  insert into t2 values (row);
  end

DB21034E  コマンドが、有効なコマンド行プロセッサー・コマンドでないため、 SQL
ステートメントとして処理されました。  SQL
処理中に、そのコマンドが返されました。

SQL0270N  関数をサポートしていません (理由コード = "70")。  LINE NUMBER=6.
SQLSTATE=429970
```

```
db2 '? SQL0270N'


SQL0270N 関数をサポートしていません (理由コード =
        "<reason-code>")。

説明:

以下の理由コードによって示されている制限に違反しているため、ステートメントを処理できません。

 70 照会最適化に使用可能になっているビューを使用不可にした後、基本表の列長を変更してから、
それらのビューを照会最適化用に使用可能にしてください。

 71 並列環境においては、トリガー、SQL 関数、SQL メソッド、または動的コンパウンド・ステートメントの中で CALL
ステートメントを使用しないでください。
```

解説：CALLされたStored Procedure内にCOMMITのある場合

```
create trigger TR1 after insert on T1
 referencing new as new
  for each row mode db2sql
  begin atomic
  declare row int;
  call p1(row);
  insert into t2 values (row);
  end
DB20000I  The SQL command completed
successfully.

insert into t1 values (1),(2)
DB21034E  The command was processed as an SQL
statement because it was not a
valid Command Line Processor command.  During
SQL processing it returned:
SQL0723N  An error occurred in a triggered SQL
statement in trigger
"DB2V8.TR1".  Information returned for the
error includes SQLCODE "-751",
SQLSTATE "     " and message tokens
"DB2V8.P1|SQL041005203116110".
SQLSTATE=09000
```

db2 '? SQL0751N'

SQL0751N ルーチン "<routine-name>"
(特定名 "<specific-name>")
が、許可されていないステートメントを実行しようと
しました。

説明:

ルーチンの本体をインプリメントするために使用される
プログラムは、接続ステートメントの発行を許可されていません。ルーチンが関数またはメソッドの場合は、
COMMIT および ROLLBACK (SAVEPOINT オプションなし)
も許可されません。ルーチンがプロシージャーであり、
それがトリガー、関数、メソッド、または動的コンパウンド・
ステートメントの中で呼び出される場合、そのプロシージャーの中で COMMIT または ROLLBACKステートメントは実行できません。

解説：MODIFIED SQL DATAプロシージャーを参照するCALLを含んだBEFORE トリガー使用不可

```
create table rt1 (c1 int)
DB20000I  The SQL command completed
successfully.

insert into rt1 values (1),(2),(3)
DB20000I  The SQL command completed
successfully.

CREATE PROCEDURE P1()
MODIFIES SQL DATA
LANGUAGE SQL
OLD SAVEPOINT LEVEL
BEGIN NOT ATOMIC
  insert into rt1 values (10),(20),(30);
  END
DB20000I  The SQL command completed
successfully.

create table t1 (c1 int)
DB20000I  The SQL command completed
successfully.
```

```
create trigger TR1 no cascade before insert on T1
 referencing new as new
  for each row mode db2sql
  begin atomic
  call p1();
  end
SQL0797N  The trigger "DB2V8.TR1" is defined
with an unsupported triggered SQL  statement.
LINE NUMBER=6.  SQLSTATE=42987
。
```

解説：プロシージャーによってフェデレーテッド表をアクセス

```
create wrapper DRDA
DB20000I  SQL コマンドが正常に終了しました。

create server  udb  type DB2/NT VERSION 8.2
WRAPPER DRDA AUTHORIZATION "azuma" PASSWORD "x"
ptions (Node 'LOCAL',DBNAME 'SAMPLE',FOLD_ID 'L',
FOLD_PW 'L')
DB20000I  SQL コマンドが正常に終了しました。

create user mapping for USER server udb options
(Remote_authid ‘ azuma ’ , Remote_password
‘ x080598x ’ )
DB20000I  SQL コマンドが正常に終了しました。

set passthru udb
DB20000I  SQL コマンドが正常に終了しました。

drop table rt1
DB20000I  SQL コマンドが正常に終了しました。

create table rt1 (c1 int)
DB20000I  SQL コマンドが正常に終了しました。

set passthru reset
DB20000I  SQL コマンドが正常に終了しました。

create nickname nk1 for udb.azuma.rt1
DB20000I  SQL コマンドが正常に終了しました。

insert into nk1 values (1),(2),(3)
DB20000I  SQL コマンドが正常に終了しました。
```

```
CREATE PROCEDURE P1(OUT result int)
READS SQL DATA
LANGUAGE SQL
BEGIN
  SET result = ( select count(*) from nk1) ;
END
DB20000I  SQL コマンドが正常に終了しました。

create table t1 (c1 int)
DB20000I  SQL コマンドが正常に終了しました。

create table t2 (c1 int)
DB20000I  SQL コマンドが正常に終了しました。

create trigger TR1 after insert on T1
 referencing new as new
  for each row mode db2sql
  begin atomic
  declare row int;
  call p1(row);
  insert into t2 values (row);
  end
DB20000I  SQL コマンドが正常に終了しました。

insert into t1 values (1)
SQL0723N  トリガー "AZUMA.TR1" のトリガー SQL
ステートメントでエラーが発生しました。
 SQLCODE "-20136"、SQLSTATE "55047"
およびメッセージ・トークン
"AZUMA.P1|SQL041005211901017"
を含むエラーの情報が戻りました。  SQLSTATE=09000
```


解説：表の競合によるランタイムエラー

```
CREATE PROCEDURE RSET
( )
LANGUAGE SQL
SPECIFIC RSETSP
BEGIN
UPDATE TEST_SOURCE SET TEST_DATA=TEST_DATA+1;--
END
DB20000I  The SQL command completed
successfully.

begin atomic
declare a smallint;--
update test_source set test_data=test_data+1;--
call rset();--
end

SQL0746N  Routine "DB2V8.RSET" (specific name
"RSETSP") violated nested SQL
statement rules when attempting to perform
operation "MODIFY" on table
"DB2V8.TEST_SOURCE".  SQLSTATE=57053
```

db2 '? SQL0746N'

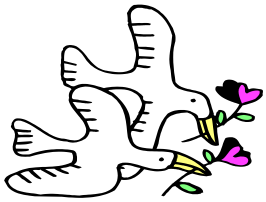
SQL0746N **ルーチン** "<routine-name>“
(**特定名** "<specific-name>")
が表 "<table-name>" **に対して操作** “<operation>"
の実行を試みたときに、ネストされた SQL
ステートメント規則に違反しました。

説明:

ルーチン “<routine-name>” (**特定名** “<specific-name>”) **は、**
表"<table-name>" **に対して操作** "<operation>"
を実行しようとしました。
この操作は、アプリケーション、あるいはそのアプリケーションから直接または間接的に呼び出されるルーチンによる表の使用と競合します。

SAVEPOINT

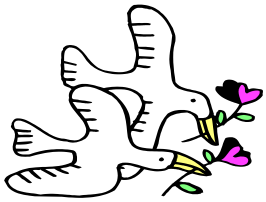
お断り：当資料は、DB2 Universal Database for Linux, UNIX and Windows V8.2 をベースに作成されています。



ブランク・ページです。

目次

- SAVEPOINTとは
- DB2 UDB V8.1 での使用制約
- DB2 UDB V8.2での使用制約の変更
- SAVEPOINT使用上の考慮点
- まとめ



ブランク・ページです。

SAVEPOINTとは

■トランザクションまたは作業単位においてSQLステートメントが失敗した場合にロールバックすることができるメカニズム

```
SAVEPOINT S1 ON ROLLBACK RETAIN CURSORS ;
INSERT INTO MYTAB (C1, C2) VALUES (2,3);
INSERT INTO MYTAB (C1, C2) VALUES (2,1);
INSERT INTO MYTAB (C1, C2) VALUES (245, 5);
ROLLBACK TO SAVEPOINT S1;

SAVEPOINT S1 ON ROLLBACK RETAIN CURSORS ;
INSERT INTO MYTAB (C1, C2) VALUES (2, 300) ;
INSERT INTO MYTAB (C1, C2) VALUES (200,3);
COMMIT;

SELECT * FROM MYTAB ;
```

実行結果

```
C1          C2
----------- -----------
          2         300
        200           3
```


解説

■ Syntax

```
>>-SAVEPOINT-- savepoint-name --+--------+----------------------->
                               '-UNIQUE-' .

                              .-ON ROLLBACK RETAIN LOCKS-.
>--ON ROLLBACK RETAIN CURSORS--+--------------------------+----><
```

SAVEPOINTステートメントは、トランザクション処理の中で現在の位置に名前を与えマークします。ROLLBACK TO ステートメントでマークされたSAVEPOINT名を指定すると、トランザクション全体ではなく一部のみを取り消すことが可能です。

・SAVEPOINT：

・UNIQUE：
アプリケーションでSAVEPOINTがアクティブの間、このSAVEPOINTの名前がアプリケーションによって再使用されないことを指定します。そのSAVEPOINTが既に存在する場合、エラー(SQLSTATE 3B501)が出力されます。

・ON ROLLBACK RETAIN CURSORS：
SAVEPOINTを発行した後にOPENされたCURSORに関してROLLBACK TO SAVEPOINT によるシステムの動作を指定します。CURSORは ROLLBACK TO SAVEPOINT 発行後の影響を受けず、位置を維持します。ただし、SAVEPOINT後、DDLで変更された従属に関してのCURSORは無効 (SQLCODE 910) になります。

・ON ROLLBACK RETAIN LOCKS：
SAVEPOINTを発行した後にロックされたオブジェクトに関してROLLBACK TO SAVEPOINTによるシステムの動作を指定します。ロックはROLLBACK TO SAVEPOINT 発行後も影響を受けず、保持されます。

・RELEASE SAVEPOINT：
マークされたSAVEPOINTを開放します。明示的にコマンドを発行しなければ、トランザクション終了時に開放されます。開放されたSAVEPOINTをROLLBACKすることはできません。

・ROLLBACK TO SAVEPOINT：
SAVEPOINTまでROLLBACKします。
・SAVEPOINT内のDDL (CREATE、ALTERなど) で作成されたオブジェクトを参照するCURSORはROLLBACK後使用できません (SQLCODE -910, SQLSTATE=57007)。
・SAVEPOINT内で参照されているCURSORのOPEN状態は保たれFETCH時は整合性を持った値を返します
・LOB LocatorはROLLBACK TO SAVEPOINT後も使用できます

DB2 UDB V8.1 での使用制約

- ネストされたSAVEPOINTはサポートされない
- SAVEPOINT内でAtomic Compound SQLは使用不可
- Atomic Compound SQL内でSAVEPOINTは使用不可

- トリガー内でSAVEPOINTは使用不可
- トランクション内で使用できるSAVEPOINTの数には制限はない
- SAVEPOINT内でのSET INTEGRITYステートメントはDDLと同様に扱われる
- On rollback release locksやOn rollback close cursorsはサポートされない
- 3part nameやNicknameに関して使用できない

DB2 UDB V8.2での使用制約の変更

- ネストされたSAVEPOINTは使用可能
- SAVEPOINT内でAtomic Compound SQLは使用可能
- Atomic Compound SQL内でSAVEPOINTは使用可能

- トリガー内でSAVEPOINTは使用不可
- トランクション内で使用できるSAVEPOINTの数には制限はない
- SAVEPOINT内でのSET INTEGRITYステートメントはDDLと同様に扱われる
- On rollback release locksやOn rollback close cursorsはサポートされない
- 3part nameやNicknameに関して使用できない

解説：SAVEPOINT内でのAtomic Compound SQL

```
EXEC SQL CREATE TABLE SAVE2(COL1 INT NOT NULL) ;
EXEC SQL COMMIT ;

EXEC SQL INSERT INTO SAVE2 VALUES(101),(102),(103) ;
EXEC SQL SAVEPOINT SAV1 ON ROLLBACK RETAIN CURSORS ;
EXEC SQL BEGIN COMPOUND ATOMIC STATIC
         DELETE FROM SAVE2;
         INSERT INTO SAVE2 VALUES (201),(202),(203);
         END COMPOUND;
```

実行結果(V8.1)

実行時にエラー　：　SQL20112N SQLSTATE=3B002
SAVEPOINTが既に存在し、ネストされたSAVEPOINTはサポートされないため
SAVEPOINTを設定することができません。

実行結果(V8.2)

```
SAVE2 col1 = 201
SAVE2 col1 = 202
SAVE2 col1 = 203
```


解説：Atomic Compound SQL内でのSAVEPOINT

```
EXEC SQL CREATE TABLE SAVE2(COL1 INT NOT NULL) ;
EXEC SQL COMMIT ;

EXEC SQL BEGIN COMPOUND ATOMIC STATIC
         INSERT INTO SAVE2 VALUES (201),(202),(203);
         SAVEPOINT SAV2 ON ROLLBACK RETAIN CURSORS ;
         INSERT INTO SAVE2 VALUES (301),(302),(303);
         ROLLBACK TO SAVEPOINT SAV2 ;
         END COMPOUND;
```

実行結果(V8.1)

プリコンパイル時にエラー　：　SQL4011N
SQL4011N 複合SQLステートメントに、無効なSQLサブステートメントがあります
SQL4011N 複合SQLステートメントに、無効なSQLサブステートメントがあります
SQL0095N　前のエラーにより、バインド･ファイルが作成されませんでした、

実行結果(V8.2)

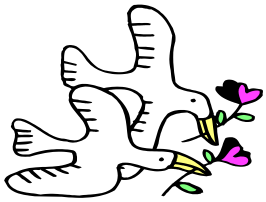
```
SAVE2 col1 = 201
SAVE2 col1 = 202
SAVE2 col1 = 203
```

解説：ネストされたSAVEPOINT

```
CREATE TABLE save2 (col1 int not null)
  DB20000I  SQL コマンドが正常に終了しました。
INSERT INTO save2 VALUES (101), (102), (103)
  DB20000I  SQL コマンドが正常に終了しました。
SAVEPOINT S1 ON ROLLBACK RETAIN CURSORS
  DB20000I  SQL コマンドが正常に終了しました。
DELETE FROM save2
  DB20000I  SQL コマンドが正常に終了しました。
INSERT INTO save2 VALUES (201)
  DB20000I  SQL コマンドが正常に終了しました。
SAVEPOINT S2 ON ROLLBACK RETAIN CURSORS
  DB20000I  SQL コマンドが正常に終了しました。
ROLLBACK TO SAVEPOINT S1
  DB20000I  SQL コマンドが正常に終了しました。
SELECT * FROM save2
COL1
-----------
        101
        102
        103
  3 レコードが選択されました。
RELEASE SAVEPOINT S1
  DB20000I  SQL コマンドが正常に終了しました。
COMMIT
  DB20000I  SQL コマンドが正常に終了しました。
```

V8.1 では SQL20112Nエラーが出力される
SAVEPOINTが既に存在し、ネストされたSAVEPOINTはサポートされないため
SAVEPOINTを設定することができません。SQLSTATE=3B002

V8.2 では ネストされたSAVEPOINTが
サポートされるようになったため実行可能



ブランク・ページです。

DB2 UDB V8.2での使用制約

- ネストされたSAVEPOINTは使用可能
- SAVEPOINT内でAtomic Compound SQLは使用可能
- Atomic Compound SQL内でSAVEPOINTは使用可能

- トリガー内でSAVEPOINTは使用不可
- トランクション内で使用できるSAVEPOINTの数には制限はない
- SAVEPOINT内でのSET INTEGRITYステートメントはDDLと同様に扱われる
- On rollback release locksやOn rollback close cursorsはサポートされない
- 3part nameやNicknameに関して使用できない

解説：トリガー内でのSAVEPOINT (1/2)

トリガー内でSAVEPOINTを使用*しない*場合

```
CREATE TABLE T1 (C11 INT)
DB20000I  SQL コマンドが正常に終了しました。

CREATE TABLE T2 (C21 INT, C22 INT)
DB20000I  SQL コマンドが正常に終了しました。

INSERT INTO T2 VALUES (1,0)
DB20000I  SQL コマンドが正常に終了しました。

CREATE TRIGGER TR1 AFTER INSERT ON T1
       REFERENCING NEW AS MODIFIED
       FOR EACH ROW
       MODE DB2SQL
       WHEN (MODIFIED.C11 < 20)
       BEGIN ATOMIC
       UPDATE T2 SET C22=C22+20 ;--
       END
DB20000I  SQL コマンドが正常に終了しました。
```

```
INSERT INTO T1 VALUES (10)
DB20000I  SQL コマンドが正常に終了しました。

SELECT * FROM T1

C11
-----------
         10
  1 レコードが選択されました。

SELECT * FROM T2
C21         C22
----------- -----------
          1          20
  1 レコードが選択されました。

COMMIT
DB20000I  SQL コマンドが正常に終了しました。
```

解説：トリガー内でのSAVEPOINT (2/2)

トリガー内でSAVEPOINTを使用*した*場合

```
CREATE TRIGGER TR1 AFTER INSERT ON T1
       REFERENCING NEW AS MODIFIED
       FOR EACH ROW
       MODE DB2SQL
       WHEN (MODIFIED.C11 < 20)
       BEGIN ATOMIC
       SAVEPOINT SAV1 ON ROLLBACK RETAIN CURSORS ;--
       UPDATE T2 SET C22=C22+20 ;--
       ROLLBACK TO SAVEPOINT SAV1 ;--
       RELEASE SAVEPOINT SAV1 ;--
       END

DB21034E  コマンドが、有効なコマンド行プロセッサー・
コマンドでないため、 SQL
ステートメントとして処理されました。  SQL
処理中に、そのコマンドが返されました。
SQL0104N  "20)  BEGIN ATOMIC   " に続いて予期しない
トークン "SAVEPOINT SAV1 ON
ROLLBACK RE" が見つかりました。入力が予想されるトーク
ンには
"<compound_SQL_stmt>" が含まれている可能性があります。
LINE NUMBER=7.
SQLSTATE=42601
```

```
INSERT INTO T1 VALUES (10)
DB20000I  SQL コマンドが正常に終了しました。

SELECT * FROM T1
C11
-----------
         10

  1 レコードが選択されました。

SELECT * FROM T2

C21         C22
----------- -----------
          1           0

  1 レコードが選択されました。


COMMIT
DB20000I  SQL コマンドが正常に終了しました。
```


解説：SET INTEGRITYはDDLと同様に扱われる

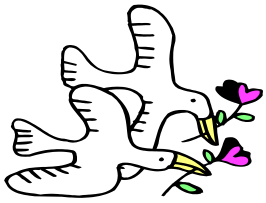
```
EXEC SQL CREATE TABLE SAVE2(COL1 INT NOT NULL) ;
EXEC SQL COMMIT ;

EXEC SQL INSERT INTO SAVE2 VALUES(101),(102),(103) ;
EXEC SQL SAVEPOINT SAV1 ON ROLLBACK RETAIN CURSORS ;
EXEC SQL DELETE FROM SAVE2;
EXEC SQL INSERT INTO SAVE2 VALUES (201),(202),(203);

EXEC SQL SET INTEGRITY FOR SAVE2 OFF ;
EXEC SQL SET INTEGRITY FOR SAVE2 IMMEDIATE CHECKED ;

strcpy(st, "SELECT * FROM SAVE2 ");
EXEC SQL PREPARE S1 FROM :st;
EXEC SQL DECLARE C1 CURSOR FOR S1 ;
EXEC SQL OPEN C1 ;

EXEC SQL ROLLBACK TO SAVEPOINT SAV1 ;

do  {
EXEC SQL FETCH C1 INTO :col1 ;
if (SQLCODE!=0) break;
   printf("SAVE2 col1 = %d \n", col1);
}while(1);

EXEC SQL CLOSE C1 ;
EXEC SQL COMMIT ;
```

実行結果
```
Error: SQLCODE = -910.
```

SQL0910N
SQL ステートメントが、変更がペンディングになっている オブジェクトをアクセスできません。
sqlcode : -910
sqlstate : 57007

解説：その他の制約

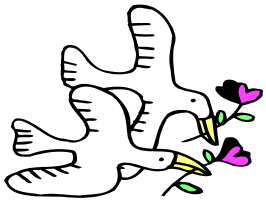
- 以下のいずれかの状態になると新規のSAVEPOINTレベルが開始される
  - 新規UOWの開始
  - NEW SAVEPOINT LEVEL文節で定義されたProcedureの呼び出し
  - Atomic Compound SQLステートメントの開始
- SAVEPOINTが発行されたイベントが終了/削除されるとSAVEPOINTは開放される
  - CURSOR、DDL、データ変更などは親SAVEPOINTによって継承され、親SAVEPOINTに対して出されたSAVEPOINT関連のステートメントが適用される
    - 親SAVEPOINT：ネストされたSAVEPOINTにおいて事前に発行されたSAVEPOINT
- その他
  - 現行のSAVEPOINT外で設定されたSAVEPOINTを開放、破棄、Rollbackすることはできない



ブランク・ページです。

SAVEPOINT使用上の考慮点

- SAVEPOINT内にDDL文がありかつROLLBACK TO SAVEPOINTが発行された場合、そのDDLに関連するCURSORは使用不可になる
- NOT LOGGED INITIALLY表がSAVEPOINT内にある場合、ROLLBACK TO SAVEPOINT はUOW全体に影響する
- SAVEPOINT内に一時表がある場合、ROLLBACK TO SAVEPOINTは一時表をDropする
- SAVEPOINT、RELEASE SAVEPOINT、ROLLBACK TO SAVEPOINTが発行された場合、DB2 Buffered Insert用のBufferをFlushする
- CURSOR BLOCKING NOを使用しない場合、ROLLBACK TO SAVEPOINT後のFETCHはROLLBACKされていない値を受け取ってしまう
- xa_endは暗黙的にRELEASE SAVEPOINTを発行する



ブランク･ページです。

解説：DDLステートメントとSAVEPOINT

```
CREATE TABLE T1
CREATE TABLE T2
CREATE TABLE T3
INSERT INTO T1
INSERT INTO T2
INSERT INTO T3

SAVEPOINT S1 ON ROLLBACK RETAIN CURSORS
PREPARE S1 (select from t1)
ALTER TABLE T1 ADD COLUMN…
PREPARE S2 (select from t2)
ALTER TABLE T3 ADD COLUMN…
PREPARE S3 (select from t3)
OPEN C1 USING S1
OPEN C2 USING S2
OPEN C3 USING S3
ROLLBACK TO SAVEPOINT S1

FETCH C1
FETCH C2
FETCH C3
```

T1表に対してDDLを発行

T3表に対してDDLを発行

C2は成功。
C1及びC3はSQL0910Nになる

SQL0910N SQL ステートメントが、変更がペンディングになっているオブジェクトをアクセスできません。

解説：SAVEPOINTとNOT LOGGED INITIALLY

```
CREATE TABLE save2 (col1 int not null)
  DB20000I  SQL コマンドが正常に終了しました。

INSERT INTO save2 VALUES (101), (102), (103)
  DB20000I  SQL コマンドが正常に終了しました。

DECLARE C1 CURSOR FOR SELECT * FROM SAVE2
  DB20000I  SQL コマンドが正常に終了しました。

OPEN C1
  DB20000I  SQL コマンドが正常に終了しました。

SAVEPOINT S1 ON ROLLBACK RETAIN CURSORS
  DB20000I  SQL コマンドが正常に終了しました。

FETCH FROM C1
COL1
-----------
        101
  1 レコードが選択されました。

CREATE TABLE SAVET2 (COL1 INT) NOT LOGGED INITIALLY
  DB20000I  SQL コマンドが正常に終了しました。

INSERT INTO SAVET2 values 201
  DB20000I  SQL コマンドが正常に終了しました。
```

```
FETCH FROM C1
COL1
-----------
        102
  1 レコードが選択されました。

ROLLBACK TO SAVEPOINT S1
DB21034E  コマンドが、有効なコマンド行プロセッサー・
コマンドでないため、 SQL
ステートメントとして処理されました。  SQL
処理中に、そのコマンドが返されました。
SQL1476N  現在のトランザクションはエラー "0" のため
ロールバックされました。
SQLSTATE=40506

SELECT * FROM save2
SQL0204N  "TAKAYA.SAVE2" は未定義の名前です。
SQLSTATE=42704
```

UOWがRollbackされる

表が存在しない

解説：SAVEPOINTと一時表　(1/2)

```
CREATE TABLE REGULAR (COL1 INT NOT NULL)
  DB20000I  SQL コマンドが正常に終了しました。
INSERT INTO REGULAR VALUES (123)
  DB20000I  SQL コマンドが正常に終了しました。
DECLARE GLOBAL TEMPORARY TABLE T2 (COL1 INT )
NOT LOGGED
  DB20000I  SQL コマンドが正常に終了しました。
INSERT INTO SESSION.T2 VALUES(7)
  DB20000I  SQL コマンドが正常に終了しました。
DECLARE GLOBAL TEMPORARY TABLE T22 (COL1 INT)
NOT LOGGED
  DB20000I  SQL コマンドが正常に終了しました。
INSERT INTO SESSION.T22 VALUES (22)
  DB20000I  SQL コマンドが正常に終了しました。
SELECT * FROM REGULAR
COL1
----------
         123
  1 レコードが選択されました。
SELECT * FROM SESSION.T2
COL1
-----------
           7
  1 レコードが選択されました。
```

```
SELECT * FROM SESSION.T22
COL1
-----------
          22
 1 レコードが選択されました。
SAVEPOINT S1 ON ROLLBACK RETAIN CURSORS
  DB20000I  SQL コマンドが正常に終了しました。
INSERT INTO REGULAR VALUES (456)
  DB20000I  SQL コマンドが正常に終了しました。
INSERT INTO SESSION.T2 VALUES (1), (3), (5)
  DB20000I  SQL コマンドが正常に終了しました。
DECLARE GLOBAL TEMPORARY TABLE T3 (COL1 INT)
NOT LOGGED
  DB20000I  SQL コマンドが正常に終了しました。
INSERT INTO SESSION.T3 VALUES (2), (4), (6)
  DB20000I  SQL コマンドが正常に終了しました。
SELECT * FROM REGULAR
COL1
-----------
         123
         456
  2 レコードが選択されました。
```


解説：SAVEPOINTと一時表　(2/2)

```
SELECT * FROM SESSION.T2
COL1
-----------
           7
           1
           3
           5
  4 レコードが選択されました。
SELECT * FROM SESSION.T22
COL1
-----------
          22
  1 レコードが選択されました。
SELECT * FROM SESSION.T3
COL1
-----------
           2
           4
           6
 3 レコードが選択されました。
ROLLBACK TO SAVEPOINT S1
 DB20000I  SQL コマンドが正常に終了しました。
```

```
SELECT * FROM REGULAR
COL1
-----------
         123
  1 レコードが選択されました。
SELECT * FROM SESSION.T2
COL1
-----------
  0 レコードが選択されました。
SELECT * FROM SESSION.T22
COL1
-----------
          22
  1 レコードが選択されました。
SELECT * FROM SESSION.T3
 SQL0204N  "SESSION.T3" は未定義の名前です。  SQLSTATE=42704
```

SAVEPOINT内で変更がある一時表は空になる

SAVEPOINT内で作成された一時表はdropされる

解説：Blocking OptionとSAVEPOINT (1/2)

```
EXEC SQL CREATE TABLE SAVE2(COL1 INT NOT NULL) ;
EXEC SQL COMMIT ;
EXEC SQL INSERT INTO SAVE2 VALUES(101),(102) ;

EXEC SQL SAVEPOINT SAV1 ON ROLLBACK RETAIN CURSORS ;
EXEC SQL INSERT INTO SAVE2 VALUES (201),(202),(203);

strcpy(st, "SELECT * FROM SAVE2 ");
EXEC SQL PREPARE S1 FROM :st;
EXEC SQL DECLARE C1 CURSOR FOR S1 ;
EXEC SQL OPEN C1 ;
EXEC SQL FETCH C1 INTO :col1 ;
printf("SAVE2 col1 = %d \n", col1);
printf("FETCH SQLCODE = %d \n", SQLCODE);

EXEC SQL CREATE ALIAS SAVEA2 FOR TAKAYA.SAVE2 ;
printf("CREATE ALIAS SQLCODE = %d \n", SQLCODE);
EXEC SQL ROLLBACK TO SAVEPOINT SAV1 ;
printf("Rollbacked SAVEPOINT \n");

EXEC SQL FETCH C1 INTO :col1 ;
printf("SAVE2 col1 = %d \n", col1);
printf("FETCH SQLCODE = %d \n", SQLCODE);
EXEC SQL FETCH C1 INTO :col1 ;
printf("SAVE2 col1 = %d \n", col1);
printf("FETCH SQLCODE = %d \n", SQLCODE);

EXEC SQL CLOSE C1 ;
```

101、102 をINSERT

201、202、203 をINSERT

Blocking OptionとSAVEPOINT (2/2)

Blocking **NO** でプリプロセスした場合

```
SAVE2 col1 = 101
FETCH SQLCODE = 0
CREATE ALIAS SQLCODE = 0
Rollbacked SAVEPOINT
SAVE2 col1 = 102
FETCH SQLCODE = 0
SAVE2 col1 = 102
FETCH SQLCODE = 100
```

Blocking **ALL** でプリプロセスした場合

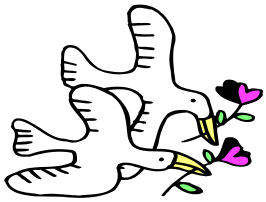
```
SAVE2 col1 = 101
FETCH SQLCODE = 0
CREATE ALIAS SQLCODE = 0
Rollbacked SAVEPOINT
SAVE2 col1 = 102
FETCH SQLCODE = 0
SAVE2 col1 = 201
FETCH SQLCODE = 0
```

201はSAVEPOINT後にINSERTされた値
で既にDB2によってDELETEされた値

Blocking NO のオプションを使用しない限り、ROLLBACK TO SAVEPOINT後のOPEN/FETCHの値は
ROLLBACKされていない値を読み取ってしまう可能性がある。

まとめ

- DB2 UDB V8.2よりSAVEPOINTのネストが登場
  - SAVEPOINT内でAtomic Compound SQLが使用可能
  - Atomic Compound SQL内でSAVEPOINTが使用可能
- 制約
  - トリガー内でSAVEPOINTは使用不可
  - トランクション内で使用できるSAVEPOINTの数には制限はない
  - SAVEPOINT内でのSET INTEGRITYステートメントはDDLと同様に扱われる
  - On rollback release locksやOn rollback close cursorsはサポートされない
  - 3part nameやNicknameに関して使用できない
- 考慮点
  - SAVEPOINT内でのDDL
  - SAVEPOINT内での NOT LOGGED INITIALLY 表　など



ブランク・ページです。

Larger SQL

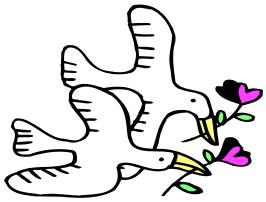
お断り：当資料は、DB2 Universal Database for Linux, UNIX and Windows V8.2 をベースに作成されています。

概要

- SQLステートメントサイズが64Kから2MBまでに拡張
  - ステートメント･サイズの拡張によって、64K以上のトリガーまたはストアード･プロシージャーが作成できる
  - 64K以上のトリガーやストアードプロシージャーをもつ他のRDBMからDB2 UDB へそれらを移行することが可能になった
- V8.2 GAではCLPおよびコマンドセンターからの実行はサポートされていない
  - Technote(2004.10.05)
    http://www-1.ibm.com/support/docview.wss?rs=71&context=SSEPGG&uid=swg21181258&loc=en_US&cs=utf-8&lang=en

SET LOCKTIMEOUT

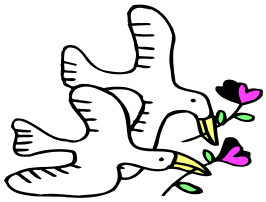
お断り：当資料は、DB2 Universal Database for Linux, UNIX and Windows V8.2 をベースに作成されています。



ブランク・ページです。

目次

- V8.1でのロックタイムアウト設定
- V8.2　SET LOCK TIMEOUT
- ロックタイムアウト設定と考慮点



ブランク･ページです。

V8.1でのロックタイムアウト設定

- DB2 UDB V8.1までのロックタイムアウト
  - セッションベルではロックタイムアウトの設定ができなかった
  - LOCKTIMEOUT DB構成パラメーターでの秒数指定
  - データベース単位の設定
- 他のベンダーはアプリケーション内での設定が可能
  - Informix
    - SET LOCK MODE TO WAIT (秒数指定可)
  - Oracle
    - LOCK TABLEステートメントのNOWAITオプション (待機しない、秒数指定不可)
  - Sybase
    - SET LOCK WAIT | NO WAIT (秒数指定可)
  - Microsoft
    - SET LOCK_TIMEOUT ステートメント(秒数指定可)

V8.2 SET LOCK TIMEOUT

- 以下の設定方法が可能 (優先度順)
  1. CURRET LOCK TIMEOUT特殊レジスターでの設定
     - SET CURRENT LOCK TIMEOUTステートメントで設定
  2. LOCKTIMEOUT DB構成パラメーター
- 利点
  - アプリケーション内でのロック･タイムアウト設定が可能に
  - IBM Informixとの互換性
    - SET CURRENT LOCK MODE

V8.2 SET LOCK TIMEOUT

SET CURRENT LOCK TIMEOUT

```
        .-CURRENT-.                .-=-.
>>-SET--+---------+--LOCK TIMEOUT--+---+----------------------->

>--+-WAIT----------------------+-----------------------------><
   +-NOT WAIT------------------+
   +-NULL----------------------+
   | .-WAIT-.                  |
   +-+------+--integer-constant-+
   '-host-variable-------------'
```

- WAIT　　CURRENT LOCK TIMEOUT=-1 既存のロックが解除されるまでロック待機する
- NOT WAIT　　CURRENT LOCK TIMEOUT=0　既存のロックがある場合には、ロック待機しない
- NULL　　値を設定しない。LOCKTIMEOUT DB構成パラメーターの値が有効。
- [WAIT] 数値　　待機時間を、-1から32767までの秒数で指定
- ホスト変数　　ホスト変数で秒数を設定

infomix互換

- SET CURRENT LOCK MODE TO
  - オプション指定方法はLOCK TIMEOUTコマンドと同様

解説：

V8．2では、ロック競合が発生し得る場合の待機時間を、アプリケーションごとにLOCKTIMEOUT値として設定することができます。
この値は、優先順位順に、1．CURRENT LOCKTIMEOUT特殊レジスターの設定、2．これまでと同様のDB構成パラメーターのLOCKTIMEOUT値により設定することが可能です。
1．CURRENT LOCKTIMEOUT特殊レジスターは、SET CURRENT LOCK TIMEOUTステートメントにより、設定を行います。
　また、指定できる時間の最大値は、V8.1の30000秒に対し、V8.2では32767秒になりました。

db2 SET LOCK TIMEOUT WAIT	-1に設定、既存のロックが解放されるか、デッドロックが検出されるまでロック待機
db2 SET LOCK TIMEOUT NOT WAIT	ロック待機時間を0秒に設定、ロック待機せずにSQL0911を戻す
db2 SET LOCK TIMEOUT NULL	DB構成パラメータのLOCKTIMEOUT値を使用
db2 SET LOCK TIMEOUT *integer*	LOCKTOMEOUT秒数を指定する、指定可能範囲は-1～32767秒
db2 SET LOCK TIMEOUT {ホスト変数}	ホスト変数でLOCKTIMEOUT秒数を指定する、師弟可能範囲は-1～32767秒を指定

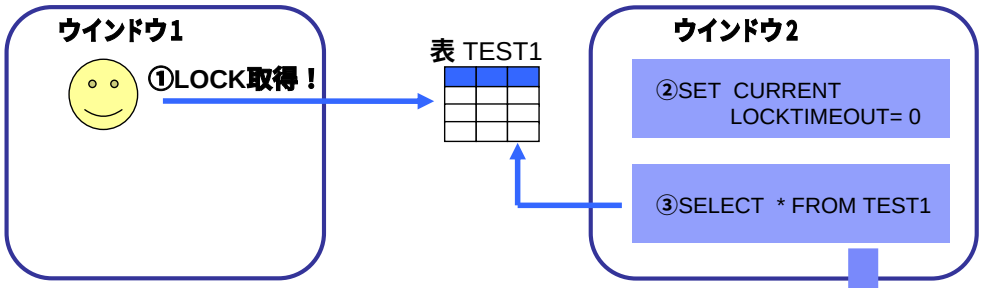
2．特殊レジスターが設定されていない場合（NULLの場合）は、DB構成パラメータの値を使用します。

InformixとDB2の互換性
Informixでは、これまでもSET CURRENT LOCK MODE TOによるセッションレベルでのロックタイムアウト設定が可能でした。
DB2 UDB V8.2では、SET CURRENT LOCK TIMEOUTステートメントと同様に、 Informix と互換性のあるSET CURRENT LOCK MODE TOステートメントも使用することができます。

```
SET LOCK MODE TO --+-- NOT WAIT----------------------+---|
                  |                                  |
                  +-- WAIT ------+------------------+
                                 |                  |
                                 +-integer-constant--+
```

例）db2 SET LOCK TIMEOUT WAIT　　-1に設定、既存のロックが解除されるまでロック待機する
　　db2 SET LOCK TIMEOUT NOT WAIT　　ロック待機時間を0秒に設定
　　db2 SET LOCK TIMEOUT WAIT 30　　ロック待機時間を30秒に設定

解説：SET LOCK TIMEOUTステートメント使用例



①ウインドウ1で、表test1にLOCKを取得し、コミットを出さずにおく

```
db2 +C LOCK TABLE TEST1 IN EXCLUSIVE MODE
```

②ウインドウ2で、CURRENT LOCK TIMEOUT特殊レジスターをゼロに設定する

```
db2 SET CURRENT LOCK TIMEOUT 0
```

③ウインドウ2でLOCK中の表TEST1に対し、SELECTを実行する。

```
db2 "SELECT * FROM TEST1
```

待機せずに SQL0911N で タイムアウト！

```
SQL0911N デッドロックまたはタイムアウトのため、現在のトランザクションがロールバックされました。
理由コード "68"。 SQLSTATE=40001
```


解説：LOCK MODEステートメント使用例

•INFORMIX互換のSET LOCK MODEステートメントを使用し、CURRENT LOCK TIMEOUT特殊レジスターが変更されることを確認する。

```
$ db2 GET DB CFG FOR SAMPLE | FIND "LOCKTIMEOUT

ロック・タイムアウト (秒)                    (LOCKTIMEOUT) = -1
```

構成パラメーターの LOCKTIMEOUT値は -1

```
$ db2 SET LOCK MODE TO WAIT 10

$ db2 VALUES CURRENT LOCK TIMEOUT

1
-----------
         10
  1 record(s) selected.
```

INFORMIX互換のステートメントを用いて、LOCKTIMEOUT値を10秒に変更

•特殊レジスターをNULLに設定したとき、DB構成パラメータの値が使用されることを確認する

```
$ db2 SET CURRENT LOCK TIMEOUT NULL
$ db2 VALUES CURRENT LOCK TIMEOUT

1
-----------
          -1
  1 record(s) selected.
```

特殊レジスターの値がNULLなので、DB構成パラメーターの値が使用される

解説：アプリケーションからの 使用例

LOCKTIMEOUTはセッションレベル、および、データベースレベルで設定可能ですが、db2cli.iniの中での設定も可能です。

•ＤＢ構成パラメーターのLOCKTIMEOUT値を-1に設定

```
$ db2 GET DB CFG FOR SAMPLE | FIND "LOCKTIMEOUT"
 ロック・タイムアウト (秒)                          (LOCKTIMEOUT) = -1
```

•表EMPLOYEEにロックを取得

```
$ db2 +c LOCK TABLE EMPLOYEE IN EXCLUSIVE MODE
```

•Javaのアプリケーションを実行。アプリケーション内で、①LOCKTIMEOUT値を5秒に設定し、②EMPLOYEE表を参照させる。
③アプリケーションは5秒経過後、SQL0911を戻す

```
Execute Statement:
   set lock mode to 5
 Execute statement values(current lock timeout)

 Results:
   special register values
   ----------------------
   5    sec : locktimeout

SELECT * FROM EMPLOYEE WHERE EMPNO < '000100'
***SQL Exception***
[IBM][CLI Driver][DB2/NT] SQL0911N  デッドロックまたはタイムアウトのため、現在のトランザクションがロールバックされました。理由コード "68"。  SQLSTATE=40001
```

①特殊レジスターの値を5秒に設定し、VALUEを確認している

②SELECT文を実行

③5秒間待機した後、SQL0911Nでエラー

解説：アプリケーションからの 使用例

ソースコードの例を示す。

```
//set current lock timeout 5
    stmt.executeUpdate("set current lock timeout 5");

    System.out.println("  Execute statement values(current lock timeout) ");
    ResultSet rs = stmt.executeQuery(" values(current lock timeout) ");


    System.out.println();
    System.out.println("  Results:¥n" + "    special register values¥n" +  "    ----------------------");
    int locktimeout = 0;
    while (rs.next())
    {
      locktimeout = rs.getInt(1);
      System.out.println("    " + locktimeout + "    sec : locktimeout¥n¥n");
      //System.out.println("    " + Data.format(locktimeout, 8));
    }
```

解説：アプリケーションからの 使用例

タイプ4ドライバーの例。properties put を使用している

```
System.out.println("About to connect to "+url);
properties.put("currentLockTimeout", "-1");
System.out.println("CurrentLockTimeout property is -1");
connect_to_db(con,properties,url);

properties.put("currentLockTimeout", "32676");
System.out.println("CurrentLockTimeout property is 32676");
connect_to_db(con,properties,url);

properties.put("currentLockTimeout", "40");
System.out.println("CurrentLockTimeout property is 40");
connect_to_db(con,properties,url);

properties.put("currentLockTimeout", "0");
System.out.println("CurrentLockTimeout property is 0");
```


ロックタイムアウト設定と考慮点

CLI/ODBCサポート

- LOCKTIMEOUT DB構成パラメーターの省略時値を設定可能
  - db2cli.ini に記述

例) LOCKTIMEOUT={ -1 | 0 | 正数<=32767 }

```
UPDATE CLI CFG FOR SECTION SAMPLE USING LOCKTIMEOUT 5
```

```
; Comment lines start with a semi-colon.

[SAMPLE]
LOCKTIMEOUT=5
```

ロック・タイプ

- SET LOCK TIMEOUTステートメントがサポートされるロックタイプ
  - 行
  - 表
  - 索引キー
  - MDCブロック

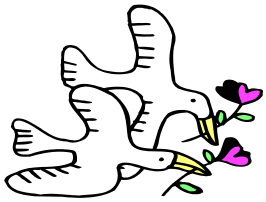
Federation上の制約

- ニックネームを使用したJOIN処理の場合、ニックネームを経由したSQLはCURRENT LOCK TIMEOUT特殊レジスターの値を引き継がない

解説：

ロックタイプ
ここで紹介したSET LOCK TIMEOUTステートメントは、以下のロックタイプでのみ有効です。
・行
・表
・索引キー
・ＭＤＣブロック
※ ＭＤＣブロックとは、多次元クラスタリング（ＭＤＣ）で管理される単位で，ディスク上の連続したページです。具体的には、表スペースのひとつひとつのエクステントがＭＤＣにおけるブロックとして扱われます。

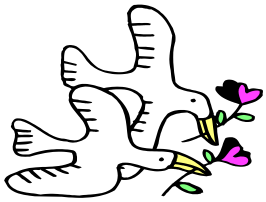
Federated
・Nick Nameを使用したJOIN処理の場合、Nick Nameを経由したSQLは、CURRENT LOCK TIMEOUT 特殊レジスターの値を引き継ぎません



ブランク・ページです。

生成列の定義変更

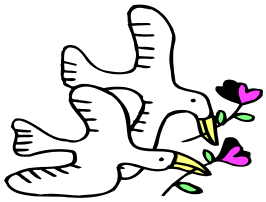
お断り：当資料は、DB2 Universal Database for Linux, UNIX and Windows V8.2 をベースに作成されています。



ブランク・ページです。

目次

- 生成列
- V8.2 ALTER CULUMN 機能強化
- 使用例



ブランク・ページです。

生成列 (Generated Columns)

- 生成列
  - 生成列に格納される値は、式を使って計算された列
- ID列
  - IDを値に格納する列 (NOT NULL属性必要)

```
CREATE TABLE T1
        (C1 INT NOT NULL
        GENERATED ALWAYS AS IDENTITY (START WITH 1, INCREMENT BY 1),
        C2 DOUBLE,
        C3 DOUBLE GENERATED AS (C1+C2),
        C4 INT GENERATED AS (CASE WHEN C1>C2 THEN 1 ELSE NULL END));
```

```
INSERT INTO T1 VALUES (DEFAULT, 2, DEFAULT, DEFAULT)
```

```
C1          C2                      C3                      C4
----------- ----------------------- ----------------------- -----------
          1   +2.00000000000000E+000  +3.00000000000000E+000           -
```


解説:

生成列は基礎表に定義されます。生成列に格納される値は、式を使って計算された値です。挿入操作や更新操作で指定された値ではありません。作成する表で、特定の式や述部を頻繁に使用することがわかっている場合、その上に1つまたは複数の生成列を追加することができます。生成列を使用すると、表データを照会する際のパフォーマンスを向上させることができます。

たとえば、パフォーマンスが重要な場合、式の評価のコストを高める恐れのある要因が2つあります。
照会中に式の評価を何度も行わなければならないこと。
計算が複雑であること。

照会のパフォーマンスを向上させるに、式の結果を入れる列をもう1つ定義することができます (生成列) 。そうすれば、同じ式を含んだ照会をするときに、生成列を直接使用できます。また、オプティマイザーの照会書き直し(re-write)時に、その式を生成列に置き換えることもできます。

オプティマイザーによる re-writeの例

```
SELECT *
FROM T1
WHERE C1+C2 = 5
```

```
SELECT Q1.C1 AS "C1", Q1.C2 AS
"C2", Q1.C3 AS "C3",
  Q1.C4 AS "C4"
FROM DB2.T1 AS Q1
WHERE (Q1.C3 =
+5.00000000000000E+000)
```

生成列への挿入・変更

- INSERTの方法はWITH DEFAULT句で作成されたカラムと同様
  - INSERT INTO t1 VALUES(1,2,DEFAULT,DEFAULT)
  - INSERT INTO t1 (C1, C2) VALUES(1,2)
  - INSERT INTO t1 (C1, C1) SELECT C1,C2 FROM t2
- 生成列で定義されるgeneration-expressionの値が変わるような変更では、生成列が再計算される
  - UPDATE t1 SET C2 = 2 WHERE C1 = 3
- 生成列に対して明示的に値を指定することはできない
  - INSERT INTO t1 VALUES(1,2,3,NULL)
  - UPDATE t1 SET C3 = 3.0 WHERE C1 = 1
  - いずれもSQL0798N(428C9)のエラーになる

解説：

•INSERTの方法はWITH DEFAULT句で作成されたカラムと同様に行います。
•生成列を除いたカラム をカラム・リストに指定するか、VALUESリストで生成列にあたるカラムにDEFAULTキーワードを指定することにより行います。

```
INSERT INTO  t1 VALUES(DEFAULT,2,DEFAULT,DEFAULT)
INSERT INTO  t1 (C2) VALUES(3)
```

```
C1          C2                      C3                      C4
----------- ----------------------- ----------------------- -----------
          1   +2.00000000000000E+000  +3.00000000000000E+000           -
          2   +3.00000000000000E+000  +5.00000000000000E+000           -
```

•生成列で定義されるgeneration-expressionの値が変わるような変更では、生成列が再計算される

```
UPDATE T1 SET C2=5 WHERE C2=3
```

```
C1          C2                      C3                      C4
----------- ----------------------- ----------------------- -----------
          1   +2.00000000000000E+000  +3.00000000000000E+000           -
          2   +5.00000000000000E+000  +7.00000000000000E+000           -
```

C2=3 の行のC2の列を変更。
生成列が再計算され、C3に新しい値が入っている。

•生成列に対して明示的に値を指定することはできません。 たとえ、指定した値がDB2により生成される値と同じであったとしてもエラーとなります。

```
INSERT INTO t1 VALUES(1,2,3,NULL)
UPDATE t1 SET C3 = 3.0 WHERE C1 = 1
```

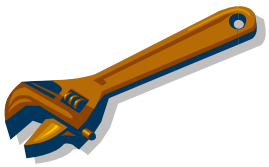
```
SQL0798N  GENERATED ALWAYS として定義されている列 "C3"
に値を指定することはできません。  SQLSTATE=428C9
```

いずれもSQL0798N(428C9)の
エラーになる！

V8.2 ALTER CULUMN 機能強化

■ALTER TABLE ステートメント ALTER CULUMN 強化

- 1．生成列ではない列に、generated-expression属性を追加可能
- 2．generated-expression属性を削除可能
- 3．ID属性を追加可能
- 4．GENERATED ALWAYS と GENERATED BY DEFAULTを置換可能
- 5．ID属性を削除可能
- 6．デフォルト属性を削除可能
- 7．1つのALTER文で、デフォルト、ID列、式のそれぞれを変更可能
- 8．GENERATED ALWAYS AS = GENERATED AS



解説： ケース1 生成列ではない列に、generated-expression属性を追加

•生成列をもつ表を作成する

```
CREATE TABLE GENE_TEST
    (C1 INT NOT NULL,
     C2 DOUBLE DEFAULT 1.1,
     C3 DOUBLE NOT NULL,
     C4 INT)
```

•2行挿入する

```
INSERT INTO GENE_TEST (C1,C3,C4)VALUES(1,1.5,9)
INSERT INTO GENE_TEST VALUES(5, DEFAULT, 1.3, 5)
```

```
C1          C2                        C3                        C4
----------- ------------------------- ------------------------- -----------
          1   +1.10000000000000E+000    +1.50000000000000E+000           9
          5   +1.10000000000000E+000    +1.30000000000000E+000           5
```

•SET INTEGRITY OFF を発行し、表をチェックペンディング状態にする。これにより、T1表への参照、更新ができなくなる。また、SET INTEGRITY OFF を行わない状態での、generated-expression属性の変更はできない。(SQL20054Nとなりエラーが発生する。)

```
SET INTEGRITY FOR GENE_TEST OFF
```

解説： ケース1 生成列ではない列に、generated-expression属性を追加

•列C3の値が、列C1+C2の値になるように、generated-expression属性を追加する

```
ALTER TABLE GENE_TEST ALTER COLUMN C3 SET GENERATED ALWAYS AS(C1+C2)
```

•次に、列C4のALTERを行う。C1>C2ならば1、そうでなければnullとなるように、generated-expression属性を追加する。

```
ALTER TABLE GENE_TEST
    ALTER COLUMN C4 SET GENERATED ALWAYS AS(CASE WHEN C1>C2 THEN 1 ELSE NULL END)
```

•表のチェック・ペンディングを解除せずにSELECTしようとすると、SQL0668Nとなる。

```
SQL0668N  操作は、理由コード "1" のため、表 "MAKIKO.GENE_TEST"に対して許可されません。  SQLSTATE=57016
```

•チェック・ペンディング を解除する。ここで、表の保全性検査をONにし、値の再計算を行う。

```
SET INTEGRITY FOR GENE_TEST IMMEDIATE CHECKED FORCE GENERATED
```

```
C1          C2                        C3                        C4
----------- ------------------------- ------------------------- -----------
          1   +1.10000000000000E+000  +2.10000000000000E+000           -
          5   +1.10000000000000E+000  +6.10000000000000E+000           1
```

再計算された値が、C3、C4のGENERATED列に入る！

解説： ケース2 generated-expression属性を削除

•1．で追加した、列C3のgenerated-expression属性を削除する。なお、ケース2以降は、チェック・ペンディングを発行しても発行しなくても同じ結果となる。（チェック・ペンディングを必要とするのは、generated-expression属性の変更の場合のみ）

```
ALTER TABLE GENE_TEST ALTER COLUMN C3 DROP EXPRESSION
```

•1行INSERTを行う。

```
INSERT INTO GENE_TEST (C1)VALUES(10)
```

```
C1          C2                        C3                        C4
----------- ------------------------- ------------------------- -----------
          1   +1.10000000000000E+000  +2.10000000000000E+000           -
          5   +1.10000000000000E+000  +6.10000000000000E+000           1
         10   +1.10000000000000E+000                          -         1
```

C3のgenerated-expression属性が削除され、NULLが挿入されている。既に、挿入済みの生成レコードは変化がないのがわかる。

解説： ケース3 ID属性を追加

•ID属性を追加するためには、対象列に予めNOT NULLが指定されていることが必要です。ここでは、列C1にNOT NULLを指定して作成済みです。

•列C1にID属性を追加し、ID=20からスタートさせるよう指定する。また、ID列の追加の場合、表のチェック・ペンディングを発行しても発行しなくても同じ結果となる。

```
ALTER TABLE GENE_TEST ALTER COLUMN C1 SET GENERATED ALWAYS AS IDENTITY(START WITH 20)
```

2行挿入する

```
INSERT INTO GENE_TEST (C2)VALUES(5.1);
INSERT INTO GENE_TEST (C1,C2)VALUES(DEFAULT, 5.1);
```

C1	C2	C3	C4
1	+1.10000000000000E+000	+2.10000000000000E+000	-
5	+1.10000000000000E+000	+6.10000000000000E+000	1
10	+1.10000000000000E+000	-	1
20	+5.10000000000000E+000	-	1
21	+5.10000000000000E+000	-	1

C1の値には、IDが挿入されている。このとき、これ以前に挿入された行の値は変更されない。

解説： ケース4 GENERATED ALWAYS と GENERATED BY DEFAULTを置換

•列に指定されたGENERATED ALWAYS 属性とGENERATED DEFAULT属性は、置き換えることが可能。
GENERATED ALWAYS：常にDB2により値がセットされるモード
GENERATED BY DEFAULT：ユーザーが値を入力しなかった場合のみ、DB2がデフォルト値をセットするモード

•3．でID列C1につけたGENERATED ALWAYS AS IDENTITY を GENERATED BY DEFAULT AS IDENTITY に変更し、ID=30からスタートさせるよう指定する。つまり、C1の値は、値の指定が無かったときのみ、IDが自動的に入力される。

```
ALTER TABLE GENE_TEST ALTER COLUMN C1 SET GENERATED BY DEFAULT RESTART WITH 30
```

•デフォルト属性を指定した列C1にも値を指定し、1行挿入する。ここでは、指定した値がC1に挿入される。

```
INSERT INTO GENE_TEST (C1,C2)VALUES(25,6.1)
```

C1	C2	C3	C4
1	+1.10000000000000E+000	+2.10000000000000E+000	-
5	+1.10000000000000E+000	+6.10000000000000E+000	1
10	+1.10000000000000E+000	-	1
20	+5.10000000000000E+000	-	1
21	+5.10000000000000E+000	-	1
25	+6.10000000000000E+000	-	1

IDは30から始まるように指定したが、明示的に値を入力したのでC1の値には、25が入っている。

解説： ケース4 GENERATED ALWAYS と GENERATED BY DEFAULTを置換

•次に、C1には値を入力せずに、1行挿入する。C1列には、まず30を用いてID列が挿入される。次にINSERTすると、C1の値は31となる。

```
INSERT INTO GENE_TEST (C2)VALUES(7.1),(7.1)
```

C1	C2	C3	C4
1	+1.10000000000000E+000	+2.10000000000000E+000	-
5	+1.10000000000000E+000	+6.10000000000000E+000	1
10	+1.10000000000000E+000	-	1
20	+5.10000000000000E+000	-	1
21	+5.10000000000000E+000	-	1
25	+6.10000000000000E+000	-	1
30	+7.10000000000000E+000	-	1
31	+7.10000000000000E+000	-	1

デフォルト値が適用され、C1のIDは30から始まる。

解説： ケース5 ID属性を削除

•列C1からID属性を削除する。この時点で、C1は、NOT NULLつきのINT列となる。

```
ALTER TABLE GENE_TEST ALTER COLUMN C1 DROP IDENTITY
```

•列C1はデフォルト値を使うようにして、1行挿入する。

```
INSERT INTO GENE_TEST (C2)VALUES(8.1)
```

C1	C2	C3	C4
1	+1.10000000000000E+000	+2.10000000000000E+000	-
5	+1.10000000000000E+000	+6.10000000000000E+000	1
10	+1.10000000000000E+000	-	1
20	+5.10000000000000E+000	-	1
21	+5.10000000000000E+000	-	1
25	+6.10000000000000E+000	-	1
30	+7.10000000000000E+000	-	1
31	+7.10000000000000E+000	-	1
0	+8.10000000000000E+000	-	-

C1にもともと、つけていたWITH DEFAULT 100 属性が消え、NOT NULLがついているので、デフォルトで0が入る。

解説：　ケース6　デフォルト属性を削除

•列C2のデフォルト属性を削除する

```
ALTER TABLE GENE_TEST ALTER COLUMN C2 DROP DEFAULT
```

•C2のデフォルト属性を削除したので、デフォルトを指定し1行挿入すると、C2にはNULLが入る。

```
INSERT INTO GENE_TEST (C1,C2)VALUES(40,DEFAULT)
```

C1	C2	C3	C4
1	+1.10000000000000E+000	+2.10000000000000E+000	-
5	+1.10000000000000E+000	+6.10000000000000E+000	1
10	+1.10000000000000E+000	-	1
20	+5.10000000000000E+000	-	1
21	+5.10000000000000E+000	-	1
25	+6.10000000000000E+000	-	1
30	+7.10000000000000E+000	-	1
31	+7.10000000000000E+000	-	1
0	+8.10000000000000E+000	-	-
40	-	-	-

C2のデフォルト属性は削除されたため、NULLが入る。

解説：　ケース7　1つのALTER文で複数の変更を行う

•列C1とC3にGENERATEDを指定した表を作成する。列C1には、必ず10が入り、列C3には、必ず、C2*2 の計算結果が入る。

```
CREATE TABLE GENE_TEST2 ( C1 INTEGER NOT NULL GENERATED ALWAYS AS (10) ,
                          C2 DOUBLE ,
                          C3 INTEGER GENERATED ALWAYS AS (INTEGER(C2)*2) NOT NULL,
                        C4 INTEGER )
```

•2行挿入する

```
INSERT INTO GENE_TEST2 (C2)VALUES(1.1),(5.1)
```

generated-expression属性の削除と、ID属性の追加が両方同時にできる！

•C3のgenerated-expression属性を削除すると同時に、C3にID属性を追加する

```
ALTER TABLE GENE_TEST2 ALTER COLUMN C3
    DROP EXPRESSION SET GENERATED BY DEFAULT AS IDENTITY (START WITH 1)
```

•2行挿入する

```
INSERT INTO GENE_TEST2 (C2)VALUES(10.1),(15.1)"
```

C1	C2	C3	C4
10	+1.10000000000000E+000	2	-
10	+5.10000000000000E+000	10	-
10	+1.01000000000000E+001	1	
10	+1.51000000000000E+001	2	-

1行目、2行目
C3には、generated-expression属性がついているため、C2*2 の計算結果が挿入される。

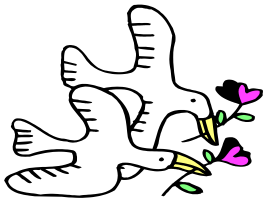
3行目、4行目
C3の属性が、ID属性に変わったため、IDが挿入されている。

REOPTオプション

(※) DB2 UDB V8.2 GA及びFP8では使用できません。
(2005年01月現在)

FP9より使用可能となりましたので、そちらの資料をご参考ください

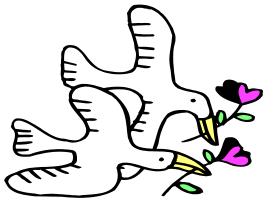
お断り：当資料は、DB2 Universal Database for Linux, UNIX and Windows V8.2 をベースに作成されています。



ブランク・ページです。

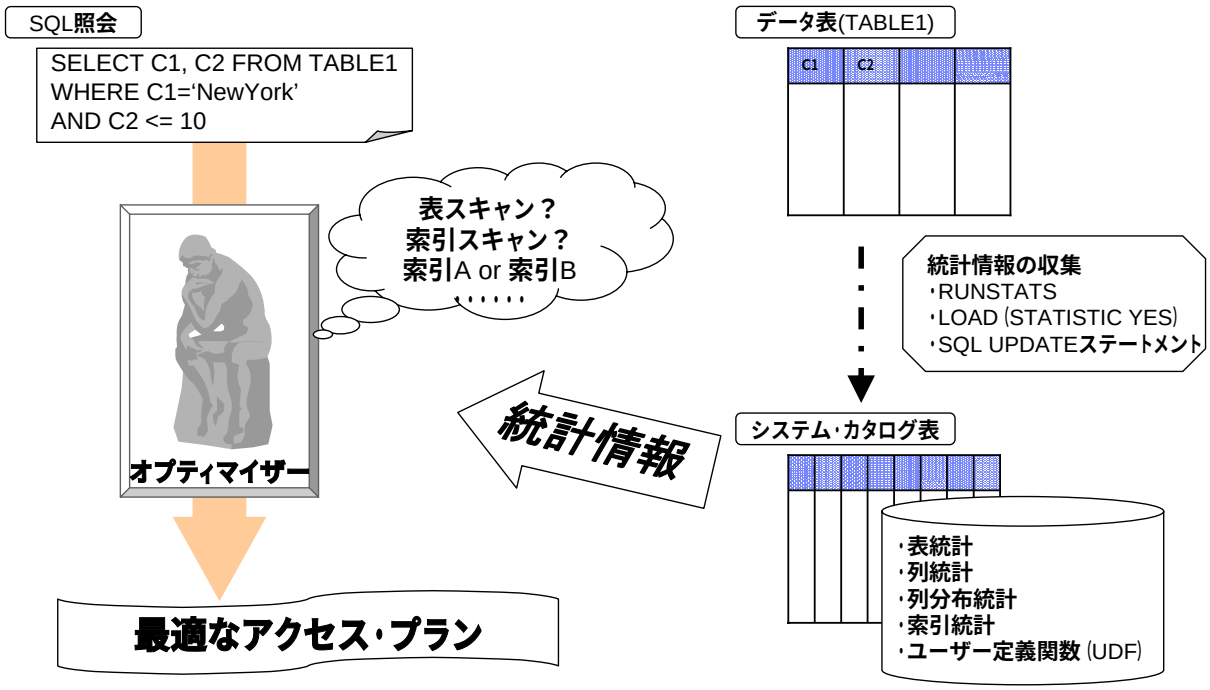
目次

- 統計情報とアクセス・プラン
- アクセス・プランはいつ作成される？
- パラメーター・マーカー
- フィルター・ファクターとアクセス・プラン
- REOPTオプション
- EXPLAINステートメント
- まとめ
- 最後に…



ブランク・ページです。

統計情報とアクセス・プラン



アクセス・プランはいつ作成される？

■静的SQL

- プリコンパイル、またはバインド時に作成される
- 実行時は、アクセスパスを検討するPREPAREは必要なし

■動的SQL

- 実行時(PREPARE)に作成される
- アクセス・プラン作成の負荷を軽減するためにパッケージ・キャッシュが使用される
  - 再利用されるためにはSQLの形式が全く同じでなければならない
  - パラメータ・マーカーを使用することで有効利用

パラメーター・マーカー

■パラメーター・マーカー

- 動的SQLで使用される
- クエスチョン・マーク(?)で表記
- 実行時に(?)に値がセットされる
- アクセス・プラン作成時には実データ値は決まっていない

➢ 例)

```
strcpy (str,"SELECT COL2 FROM T1 WHERE C1 BETWEEN ? AND ?");
EXEC SQL PREPARE st1 FROM :str;
EXEC SQL DECLARE cur1 CURSOR FOR st1;
val1=100;
val2=200;
EXEC SQL OPEN cur1 using :val1, :val2;
EXEC SQL FETCH cur1 INTO :val3;
```

⇒ 実データ値に基づくアクセス・プランではない

解説: パラメーター・マーカー

- 動的SQLの環境では、一度コンパイル・実行されたSQLは、同じSQLが再び実行される場合に備えてステートメントキャッシュというところにおかれます。
- 再度同じSQLが実行される場合には、このキャッシュされたステートメントが実行されますので、コンパイルにかかる時間やCPUなどのリソースが節約できます。
- ところが、再利用がなされるためには、whereで指定される条件の値まで含めて、全く同じSQLでなければなりません。そこで活用すべきなのが、パラメーターマーカーです。
- パラメーターマーカーは、具体的な値を割り当てる代わりに「?」を使用してステートメントのPrepareを行うことで活用します。

例1:

```
String selectString = "SELECT COL1 FROM TABLE1 WHERE COL2=10 and COL3='AAA'";
Stmt.executeQuery(selectString);
```

例2:

```
PreparedStatement selectVal = con.prepareStatement (
        "SELECT COL1 FROM TABLE1 WHERE COL2=? and COL3=?");
selectVal.setInt(1, 10);
selectVal.setString(2, "AAA");
selectVal.executeQuery();
```

- つまり、「SELECT COL1 FROM TABLE1 WHERE COL2=10 and COL3='AAA'」などというSQLをPrepareしたり、直接executeQueryで実行したりする代わりに「SELECT COL1 FROM TABLE1 WHERE COL2=？ and COL3=？」というかたちでPrepareしておき、その後で？にあてはまるような値をセットしてから実行する、という風にして使います。このようにすると、ステートメントキャッシュ上には「SELECT COL1 FROM TABLE1 WHERE COL2=？ and COL3=？」というかたちでのりますので、COL2やCOL3で指定する値がいくつであってもキャッシュ上のステートメントの再利用ができます。

アクセス・プラン作成時に実データ値が不明なケース

■パラメータ・マーカー以外に次のものがあります
- ホスト変数
- 特殊レジスター

■ホスト変数
- 組み込みSQLステートメントが参照する変数
- アプリとSQLとの間でデータのやり取りをするための変数
- DECLARE SECTIONにて宣言
- 名前の接頭部にコロン (:) をつける

```
EXEC SQL BEGIN DECLARE SECTION;
  short var1;
  short var2;
EXEC SQL END DECLARE SECTION;
val1=100;
・・・・・・・
EXEC SQL SELECT COL2 INTO :val2 FROM T1 WHERE COL1< :var1;
```


アクセス・プラン作成時に実データ値が不明なケース

■特殊レジスター
- データベース・マネージャーによって定義された記憶域
- SQLステートメントで参照可能な情報が保管されている
- 例)
  - CURRENT DATE
  - CURRENT TIME
  - CURRENT TIMESTAMP
  - CURRENT SCHEMA
  - CURRENT SERVER
  - ⋮
- 使用例)

```
SELECT COUNT(*),CURRENT TIMESTAMP
  FROM SYSIBM.SYSDUMMY1

1           2
----------- --------------------------
          1 2004-10-04-21.23.50.362001
```

パラメーター・マーカー、ホスト変数、特殊レジスターの課題点

■アクセス・プラン作成時に実データ値が不明

- 省略値のフィルター・ファクターが使用される
- 省略値のフィルター・ファクターはCOLCARD によって決定
  - COLCARDはその列に割り当てられたユニークな件数
  - COLCARDの大きい列の方が小さいフィルター・ファクターとなる
  - 例) T1表には10000行
    - C1には1～1000の整数値が各１０行　(COLCARD=1000)
    - C2には1～10000の整数値が各1行　(COLCARD=10000) つまりユニーク列
    - SELECT * FROM T1 WHERE C1 <= ? AND C2 <= ?

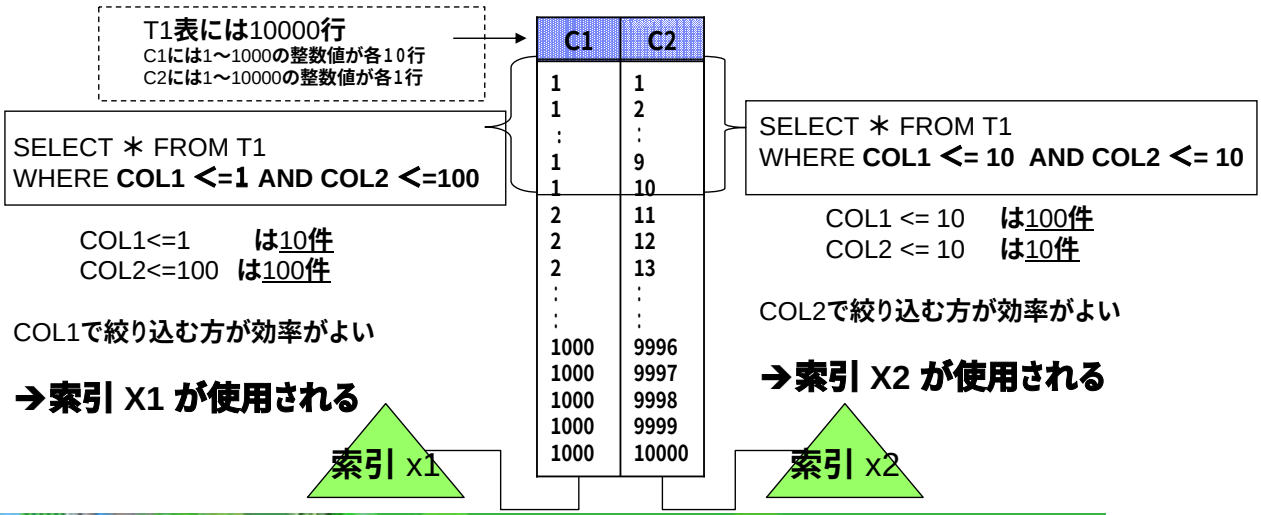
⇒ 「C１のCOLCARD < C2のCOLCARD」
“C2”が最初の条件評価に選ばれる

- **最適なアクセスプランにならないケースがある**
  - 条件節で指定されるケース
  - 実データ値によるフィルター・ファクターが省略時のフィルター・ファクターとかけ離れるケース
    - 一様に分布していない列を条件指定するケース
    - BETWEENや不等号による条件指定のケース

フィルター・ファクターとアクセス・プラン

■フィルター・ファクターとは

- その条件節を満たす行がどの程度あるかを示す指標
- 統計情報をもとにオプティマイザーが計算する
- フィルターファクターが低い
  - **その条件による絞込み効率が良い (最初に適用される条件の候補となる)**



参考) 実行例

- 目的
  - パラメータ・マーカーを使用する場合に最適なアクセスプランが作成されないケースを検証
- 表定義
  - 表TEST:　　全10000行
  - C1列:　1～1000の整数値が各10行　(COLCARD=1,000)
  - C2列:　1～10000の整数値が各1行　(COLCARD=10,000)　(ユニーク列)
- 索引定義
  - 索引X1:　C1列に定義
  - 索引X2:　C2列に定義
- 実行SQL
  - ケース1) SELECT * FROM TEST WHERE **C1 <=5 AND C2<=10000**
  - ケース2) SELECT * FROM T1 WHERE **C1<=? AND C2<=?**
    - C1に 5、C2に10000をセット
  - どちらのケースも同じ結果50行が戻される

参考) ケース1　C1<=5 and C2<=10000

```
Type      : Dynamic
  Operation: Close
  Section   : 201
  Creator   : NULLID
  Package   : SQLC2E06
  Consistency Token   : AAAAAcEU
  Package Version ID  :
  Cursor    : SQLCUR201
  Cursor was blocking: TRUE
 Text      : SELECT * FROM TEST WHERE C1 <= 5 AND C2 <= 10000
 ------------------------------------------------
  Start Time: 2004-12-13 07:32:35.242307
  Stop Time:  2004-12-13 07:32:35.259939
  Exec Time:  0.017632 seconds
  Number of Agents created: 1
  User CPU: 0.000000 seconds
  System CPU: 0.000000 seconds
  Fetch Count: 50
  Sorts: 1
  Total sort time: 0
  Sort overflows: 0
  Rows read: 50
  Rows written: 0
```

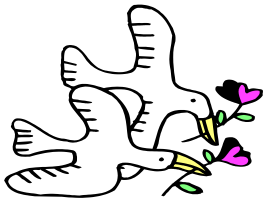
C1における索引を使用して条件を絞り込んだため、50行の読み取りが発生した

参考) ケース2　C1<= ? and C2<= ?

```
Type     : Dynamic
Operation: Close
Section  : 4
Creator  : NULLID
Package  : SYSSH200
Consistency Token  : SYSLVL01
Package Version ID  :
Cursor   : SQL_CURSH200C4
Cursor was blocking: TRUE
Text     : SELECT * FROM TEST WHERE C1 <= ? AND C2 <= ?
-----------------------------------------
Start Time: 2004-12-13 07:27:41.232574
Stop Time:  2004-12-13 07:27:41.256232
Exec Time:  0.023658 seconds
Number of Agents created: 1
User CPU: 0.020029 seconds
System CPU: 0.000000 seconds
Fetch Count: 50
Sorts: 0
Total sort time: 0
Sort overflows: 0
Rows read: 10000
Rows written: 0
```

パラメータ・マーカーを使用している
実際には、C1 =5、C2=10000が入る

省略値のフィルター・ファクターが使用され、
C2における索引が使用された為10.000件
の読み取りが発生した



ブランク・ページです。

REOPTオプション

- 実行時に最適化を行う
  - ホスト変数、パラメーターマーカー、特殊レジスターが実際に値が設定された段階でアクセスプランの決定を行う
  - 静的SQLの場合
    - ステートメント実行時
  - 動的SQLの場合
    - PREPAREの後のEXECUTE、OPEN時
- PREP、BIND、REBINDの新しいオプション
  - ホスト変数、パラメータマーカー、特殊レジスターが含まれているステートメントにのみ有効
  - **REOPT NONE　(省略値)**
    - 実行時の最適化を行わない
  - **REOPT ONCE**
    - 実行時に一度だけ最適化を行う
    - 以降のアプリはキャッシュされてステートメントを再利用
    - FLUSH PACKAGE CACHEにて再度最適化
  - **REOPT ALWAYS**
    - ステートメント実行の度に最適化を行う

解説

- **静的SQLにおけるREOPTオプション**
- BIND オプション REOPT は、ホスト変数や特殊レジスターを含む静的 SQL ステートメントを、増分バインド・ステートメントのように動作させることができます。これは、これらのステートメントが、バインド時ではなく、EXECUTE または OPEN の時にコンパイルされることを意味します。このコンパイル時に、これらの変数の実際の値に基づいて、アクセス・プランが選択されます。
- REOPT ONCE の場合は、最初の OPEN または EXECUTE 要求の後にアクセス・プランがキャッシュされ、このステートメントの後の実行で使用されます。REOPT ALWAYS の場合は、OPEN および EXECUTE 要求のたびにアクセス・プランが再生成され、現行のホスト変数、パラメーター・マーカー、および特殊レジスターの値のセットでこのプランが作成されます。
- **動的 SQL における REOPTオプション**
- REOPT ALWAYS オプションを指定している場合、DB2は、OPEN または EXECUTE ステートメントを検出するまで、つまり、これらの変数の値が分かるときまで、ホスト変数、パラメーター・マーカー、または特殊レジスターを含むステートメントの準備を延期します。その時点で、これらの値を使用してアクセス・プランが生成されます。7 同じステートメントに対するその後の OPEN または EXECUTE 要求では、ステートメントが再コンパイルされ、 変数の現行の値のセットを使用して照会プランが再最適化され、新しく生成された照会プランが実行されます。
- REOPT ONCE オプションにも同様の効果がありますが、異なる点として、プランがホスト変数、パラメーター・マーカー、特殊レジスターの値で 1 回だけ最適化されます。7このプランはキャッシュされ、その後の要求で使用されます

解説

•REOPTオプション

DB2 がホスト変数、パラメーター・マーカー、および特殊レジスターの値を使用して ランタイムにアクセス・パスを判別するようにするかどうかを指定します。有効な値は以下のとおりです。

【新しいオプション】

NONE

ホスト変数、パラメーター・マーカー、または特殊レジスターを含む SQL ステートメントのアクセス・パスは、実際の値によって最適化されません。これらの変数のデフォルトの推定値が使用され、このプランがキャッシュされて使用されます。これがデフォルト値です

ONCE

最初に照会が実行されるときに、ホスト変数、パラメーター・マーカー、7 または特殊レジスターの実際の値によって、SQL ステートメントのアクセス・パスが最適化されます。7 このプランがキャッシュされて使用されます。

ALWAYS

照会が実行されるたびに、ホスト変数、パラメーター・マーカー、7 または特殊レジスターの既知の値によって、SQL ステートメントのアクセス・パスが必ずコンパイルおよび再最適化されます。

【既存のオプション】

REOPT / NOREOPT VARS

これらのオプションは、REOPT ALWAYS および REOPT NONE によって置き換えられましたが、バックレベルの互換性のために引き続きサポートされています。DB2 がホスト変数、パラメーター・マーカー、および特殊レジスターの値を使用してランタイムにアクセス・パスを判別するようにするかどうかを指定します。**サポートしているのは DB2 for OS/390 だけです。**サポートされているオプション値のリストについては、DB2 for OS/390 の資料を参照してください。

解説

•SYSCAT.PACKAGESのREOPTVAR列で確認できます

```
>db2 bind test01.bnd reopt always

LINE    MESSAGES FOR test01.bnd
------  ------------------------------------------------------------------------
      SQL0061W  バインド・プログラムが処理中です。
      SQL0091N  バインドが、エラー "0" と 警告 "0" で終了しました。

>db2 select substr(pkgname,1,10),REOPTVAR from syscat.packages where pkgname='TEST01'

1           REOPTVAR
----------- --------
TEST01      A

  1 レコードが選択されました。
```

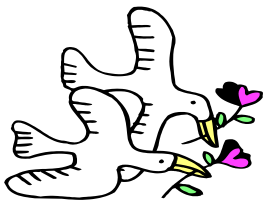
SYSCAT.PACKAGESの
REOPTVAR列

REOPT NONE　= 'N'
REOPT ONCE　='O'
REOPT ALWAYS　='A'

z/OS、OS/390におけるREOPTサポート

■z/OS, OS/390におけるREOPTオプション

- V5よりREOPTのサポート有り
  - NOREOPT(VARS) = REOPT NONE
  - REOPT(VARS) = REOPT ALWAYS
- ONCEオプションはV8からのサポート
  - REOPT(NONE)
  - REOPT(ALWAYS)
  - REOPT(ONCE)



ブランク・ページです。

CLIアプリケーションにおけるREOPTサポート

- CLIのパッケージのバインド時にREOPTを指定する
  - この時COLLECTIONオプションも指定する
    - 既存のNULLIDスキーマのCLIパッケージはBINDコマンドによって上書きできないため
    - 例) REOPT ONCEオプションで、REOPTSETスキーマのパッケージ作成
    - db2 bind @db2cli.lst **collection REOPTSET reopt once**
- COLLECTIONオプションで指定したパッケージ・スキーマを使用するように設定します
  - CLIアプリケーションの場合
    - ODBC / OLE DB / .NET / JDBC(Legacy Type 2 , 3) など
    - db2cli.iniに設定
    - 例) db2 update cli cfg for section sample
      using **CurrentPackageSet REOPTSET**
  - Universal JDBC ドライバーを使う場合
    - JDBCCollectionプロパティーにパッケージ・スキーマを設定

解説

- CLIのパッケージをバインドする時にREOPTオプションを指定する事で、CLIアプリケーションにおいてREOPTオプションを有効にすることが可能です。ただしNULLIDスキーマのCLIパッケージが、省略値のREOPT NONEで既に存在しています。(データベース作成時に自動でバインドされるため)
- BINDコマンドでは既存のパッケージは上書きされないため、COLLECTIONオプションによって任意のパッケージ・スキーマを指定して新しくパッケージを作成して下さい。
- 作成したパッケージ・スキーマを、省略値のパッケージ・スキーマとしてdb2cli.ini初期設定ファイルに設定してください。
- db2cli.ini ファイルを読み込まないUniversal JDBCドライバーを使ったアプリケーションではJDBCCollectionプロパティで使用するパッケージ・スキーマを設定します。

例)

```
DB2SimpleDataSource db2ds = new DB2SimpleDataSource();
db2ds.setDatabaseName("Sample");
db2ds.setUser("user82");
db2ds.setPassword("stingerws");
db2ds.setJdbcCollection("REOPTSET");
con = db2ds.getConnection();
```

← パッケージのスキーマ名

参考）　実行例

- 目的
  - CLIアプリケーション(JDBC type2)におけるREOPTオプションの確認
- 表定義
  - 表TEST：　　全10000行
  - C1列：1～1000の整数値　（各値は10件ずつ）
  - C2列：1～10000の整数値（各値は1件ずつ）
- 索引定義
  - 索引X1：C1列に定義
  - 索引X2：C2列に定義
- 検証ケース

	REOPTオプション	実行SQL	結果	目的
ケース1	REOPT ALWAYS	C1<=5 AND C2<=1000	50	実行する度に最適化されているか？
ケース2	REOPT ALWAYS	C1<=1000 AND C2<=50	50	
ケース3	REOPT ONCE	C1<=5 AND C2<=1000	50	2回目は最適化されず、1回目のアクセスパスを使用するか？
ケース4	REOPT ONCE	C1<=1000 AND C2<=50	50	
ケース5	REOPT ONCE	FLUSH PACKAGE CACHE後 C1<=1000 AND C2<=50	50	FLUSH PACKAGE CACHE後、 再度最適化されるか？

ケース1～2　REOPT ALWAYSに設定

```
>db2 bind @db2cli.lst collection REOPTA REOPT ALWAYS

LINE    MESSAGES FOR db2cli.lst
------  ------------------------------------------------------------------------
      SQL0061W  バインド・プログラムが処理中です。
      SQL0091N  バインドが、エラー "0" と 警告 "0" で終了しました。

>db2 select SUBSTR(PKGSCHEMA,1,10),SUBSTR(PKGNAME   where pkgschema='REOPTA'

      1         2         REOPTVAR
      ---------- ---------- --------
      REOPTA     SYSLH100   A
      REOPTA     SYSLH101   A
      REOPTA     SYSLH102   A
      REOPTA     SYSLH200   A
      REOPTA     SYSLH201   A
      REOPTA     SYSLH202   A
      REOPTA     SYSLH300   A
      REOPTA     SYSLH301   A
                 :
                 :
```

REOPT ALWAYSと指定した
パッケージを使用するように設定する

```
>UPDATE CLI CFG FOR SECTION V8DB USING CurrentPackageSet REOPTA
```

ケース1 REOPT ALWAYS　1回目：C1＜=5、C2＜=1000

(イベントモニター出力の一部)

```
Type     : Dynamic
  Section  : 4
Operation: Close
Creator  : REOPTA
  Package  : SYSSH200
  Consistency Token  : SYSLVL01
  Package Version ID  :
  Cursor   : SQL_CURSH200C4
  Cursor was blocking: TRUE
  Text     : SELECT * FROM TEST WHERE C1 <= ? AND C2 <= ?
  ------------------------------------------
  Start Time: 2004-10-05 17:13:37.205302
  Stop Time:  2004-10-05 17:13:37.210360
  Exec Time:  0.005058 seconds
  Number of Agents created: 1
  User CPU: 0.000000 seconds
  System CPU: 0.000000 seconds
  Fetch Count: 50
  Sorts: 0
  Total sort time: 0
  Sort overflows: 0
  Rows read: 50
  Rows written: 0
```

C1<=5で先に条件を絞り込んだため、
５０行の読み取りが発生した

ケース2　REOPT ALWAYS　2回目：C1＜=1000、C2＜=50

(イベントモニター出力の一部)

```
Type     : Dynamic
  Operation: Close
  Section  : 4
  Creator  : REOPTA
  Package  : SYSSH200
  Consistency Token  : SYSLVL01
  Package Version ID  :
  Cursor   : SQL_CURSH200C4
  Cursor was blocking: TRUE
  Text     : SELECT * FROM TEST WHERE C1 <= ? AND C2 <= ?
  ------------------------------------------
  Start Time: 2004-10-05 17:13:37.205302
  Stop Time:  2004-10-05 17:13:37.210360
  Exec Time:  0.005058 seconds
  Number of Agents created: 1
  User CPU: 0.000000 seconds
  System CPU: 0.000000 seconds
  Fetch Count: 50
  Sorts: 0
  Total sort time: 0
  Sort overflows: 0
  Rows read: 50
  Rows written: 0
```

C2<=50で先に条件を絞り込んだため、
50行の読み取りが発生した
つまりケース1とは異なるアクセス・パスである

ケース3～5 REOPT ONCEに設定

```
>db2 bind @db2cli.lst collection REOPTO REOPT ONCE
LINE    MESSAGES FOR db2cli.lst
------  ------------------------------------------------------------------------
        SQL0061W  バインド・プログラムが処理中です。
        SQL0091N  バインドが、エラー "0" と 警告 "0" で終了しました。

>db2 select SUBSTR(PKGSCHEMA,1,10),SUBSTR(PKGNAME   where pkgschema='REOPTO'

        1         2           REOPTVAR
        ---------- ---------- --------
        REOPTO     SYSLH100   O
        REOPTO     SYSLH101   O
        REOPTO     SYSLH102   O
        REOPTO     SYSLH200   O
        REOPTO     SYSLH201   O
        REOPTO     SYSLH202   O
        REOPTO     SYSLH300   O
                   :
                   :
```

REOPT ONCEと指定した
パッケージを使用するように設定する

```
>UPDATE CLI CFG FOR SECTION V8DB USING CurrentPackageSet REOPT0
```


ケース3 REOPT ONCE 1回目　C1＜=5、C2＜=1000

(イベントモニター出力の一部)

```
Type     : Dynamic
  Operation: Close
  Section  : 4
  Creator  : REOPTO
  Package  : SYSSH200
  Consistency Token  : SYSLVL01
  Package Version ID  :
  Cursor   : SQL_CURSH200C4
  Cursor was blocking: TRUE
  Text     : SELECT * FROM TEST WHERE C1 <= ? AND C2 <= ?
  ------------------------------------------
  Start Time: 2004-10-05 10:19:37.220115
  Stop Time:  2004-10-05 10:19:37.225347
  Exec Time:  0.005232 seconds
  Number of Agents created: 1
  User CPU: 0.000000 seconds
  System CPU: 0.000000 seconds
  Fetch Count: 50
  Sorts: 0
  Total sort time: 0
  Sort overflows: 0
  Rows read: 50
  Rows written: 0
```

C1<=5で先に条件を絞り込んだため、
５０行の読み取りが発生した

ケース4 REOPT ONCE 2回目　C1<=1000、C2<=50

(イベントモニター出力の一部)

```
Type     : Dynamic
  Operation: Close
  Section  : 4
  Creator  : REOPTO
  Package  : SYSSH200
  Consistency Token  : SYSLVL01
  Package Version ID  :
  Cursor   : SQL_CURSH200C4
  Cursor was blocking: TRUE
  Text     : SELECT * FROM TEST WHERE C1 <= ? AND C2 <= ?
  ------------------------------------------
  Start Time: 2004-10-05 10:20:18.435587
  Stop Time:  2004-10-05 10:20:18.459490
  Exec Time:  0.023903 seconds
  Number of Agents created: 1
  User CPU: 0.020028 seconds
  System CPU: 0.000000 seconds
  Fetch Count: 50
  Sorts: 0
  Total sort time: 0
  Sort overflows: 0
  Rows read: 10000
  Rows written: 0
```

C1<=1000で先に条件を絞り込んだため、
10000行の読み取りが発生した

ケース5 REOPT ONCE 3回目　C1<=1000、C2<=50

(イベントモニター出力の一部)

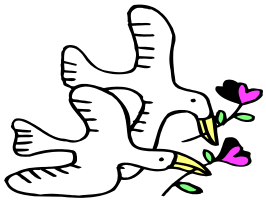
3回目の前に....
FLUSH PACKAGE CACHE DYNMICを実行

```
Type     : Dynamic
  Operation: Close
  Section  : 4
  Creator  : REOPTO
  Package  : SYSSH200
  Consistency Token  : SYSLVL01
  Package Version ID  :
  Cursor   : SQL_CURSH200C4
  Cursor was blocking: TRUE
  Text     : SELECT * FROM TEST WHERE C1 <= ? AND C2 <= ?
  ------------------------------------------
  Start Time: 2004-10-05 10:27:14.803557
  Stop Time:  2004-10-05 10:27:14.809264
  Exec Time:  0.005707 seconds
  Number of Agents created: 1
  User CPU: 0.000000 seconds
  System CPU: 0.000000 seconds
  Fetch Count: 50
  Sorts: 0
  Total sort time: 0
  Sort overflows: 0
  Rows read: 50
  Rows written: 0
```

C2<=50で先に条件を絞り込んだため、
50行の読み取りが発生した
つまり再度最適化されたことが分かる

SQL ProcedureにおけるREOPTサポート

- DB2_ROUTINE_PREPOPTSレジストリ変数
  - レジストリー変数DB2_ROUTINE_PREPOPTSにバインドパラメーターを設定した環境でCREATE PROCEDUREを実行
  - db2set DB2_ROUTINE_PREPOPTS=“REOPT ALWAYS”
- 現在はレジストリ変数が存在しない
  - FP9までには直る予定



ブランク・ページです。

REOPTオプションの考慮点

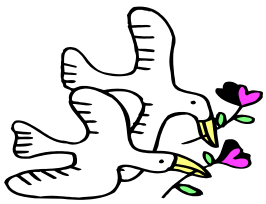
■REOPT ALWAYS

- ステートメント実行時のオーバーヘッド
  - アクセスプラン決定のためのオーバーヘッドがかかる
- LOOPの中で繰り返しSQLを実行するアプリでREOPT ALWAYSを使用すると負荷は著しく大きくなるので注意

■REOPT ONCE

- 最初に実行されキャッシュにのったステートメントが稀なケースだった場合は、後の実行へ影響がある
  - EXPLAINステートメントのWITH REOPT ONCEオプションによる分析
  - FLUSH PACKAGE CACHEステートメントの実行

NEW



ブランク・ページです。

EXPLAINステートメント

■EXPLAINステートメント

- 特定のステートメントのEXPLAIN情報をEXPLAIN表へ挿入
  - 例)SELECTステートメントのEXPLAIN情報をQUERYNO=100のタグを付けて取得
    - EXPLAIN ALL SET QUERY=100
      FOR SELECT * FROM STAFF WHERE ID=100

- パラメーター・マーカーを指定することもできる
  - 例)条件節の値 idをパラメータ・マーカーとする
    - EXPLAIN ALL FOR SELECT * FROM STAFF WHERE ID = ?
  - 条件節に指定した場合、デフォルトのフィルター・ファクターを使用

EXPLAIN機能により提供される情報
- 照会を処理する操作の順序
- コスト情報
- 述部および選択可能性の見積もり
- EXPAIN機能の実行時点のSQLステートメントで参照されているオブジェクトに関する統計

EXPLAINステートメントのWITH REOPT ONCE

■WITH REOPT ONCEオプション

- REOPT ONCE指定のBINDで生成されたパッケージ実行によって決定されたアクセス・プランの情報を得る
  - 例)
    - EXPLAIN ALL **WITH REOPT ONCE**
      FOR SELECT * FROM STAFF ID = ?

- 現在パッケージ・キャッシュ上に載っているステートメントのEXPLAIN情報をEXPLAIN表に生成する

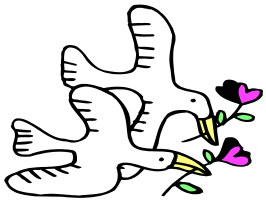
- 該当のSQLがパッケージ・キャッシュ上に見つからない場合にはエラー
  - SQL1169N　ステートメントの Explain 処理中にエラーが発生しました。
    理由コード ="1"。　SQLSTATE=560C9

解説

```
>db2 "explain all with reopt once for SELECT * FROM TEST WHERE C1 <= ? AND C2 <= ? "

DB21034E  コマンドが、有効なコマンド行プロセッサー・コマンドでないため、 SQL
ステートメントとして処理されました。 SQL
処理中に、そのコマンドが返されました。
SQL1169N  ステートメントの Explain 処理中にエラーが発生しました。理由コード =
"1"。 SQLSTATE=560C9

(プログラム実行)
>java test
Row count=5

>db2 “explain all with reopt once for SELECT * FROM TEST WHERE C1 <= ? AND C2 <=? “

SQL20287W  指定されたキャッシュ・ステートメントの環境は、現在の環境とは違うもの
です。現在の環境を使用して、指定された SQL
ステートメントを最適化しなおします。 SQLSTATE=01671
```

キャッシュに載っていないため失敗する

一度実行されたためキャッシュにのり成功



ブランク・ページです。

EXPLAINステートメントのWITH REOPT ONCE

■最適化に使った値はEXPLAIN表に生成

- そのステートメント実行時にセットされ、最適化に使った値（ホスト変数、パラメーター・マーカー、特殊レジスター）はEXPLAIN_PREDICATE表のPREDICATE_TEXT列にセット
- 例：explain all with reopt once
  for select c3 from t1 where c1<=? and c2<=?
- Explain表へのデータ生成

```
Select substr(predicate_text,1,25) as predicate from explain_predicate

PPREDICATE
     -------------------------
     (Q1.C2 <= :? [1000000])
     (Q1.C1 <= :? [5])
```


db2exfmtの出力（部分）

WITH REOPT ONCEオプション有

```
Predicates:
----------
2) Sargable Predicate
        Relational Operator:        Less Than or Equal (<=)
        Subquery Input Required:    No
        Filter Factor:              1

        Predicate Text:
        --------------
       (Q1.C2 <= :? [1000000])
```

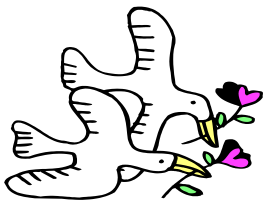
これまで… （WITH REOPT ONCEオプション無）

```
Predicates:
----------
2) Stop Key Predicate
         Relational Operator:       Less Than or Equal (<=)
         Subquery Input Required:   No
         Filter Factor:             0.0562139

         Predicate Text:
         --------------
         (Q1.C2 <= :?)
```

EXPLAIN WITH REOPT ONCE　(続き)

■EXPLAN. . . WITH REOPT ONCEの実行

- DBADM権限が必要
- 新しいレジストリー変数 DB2_VIEW_REOPT_VALUES
  - この変数にYESが設定されている環境ではDBADMは不要



ブランク･ページです。

まとめ

- 新しいBINDオプション（REOPT）を使用することで、常に最適なアクセス・プランを使用することができる
  - ホスト変数、パラメーター・マーカー、特殊レジスターを使っているケースで、DB2のデフォルトのフィルター・ファクターと実際の値が大きく異なってしまうようなケースで有効
  - 実行時の最適化のためのオーバーヘッドに注意

最後に...

- V8.2 GAおよびFP8では使用できません。
  - FP9で使用可能になる予定
  - REOPTオプションを指定すると以下のようなエラーになります

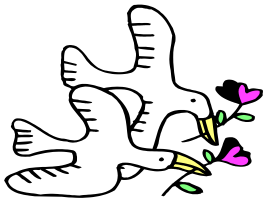
SQL0020W BIND またはプリコンパイル・オプション (名前または値)
"REOPT"は、ターゲット・データベースでサポートされていないため、
無視されま す。

マニュアルには既に新機能として記述されておりますが、
まだ使用出来ませんのでご注意ください。

FP9より使用可能となりましたので、そちらの資料をご参考ください

USE AND KEEP LOCKS

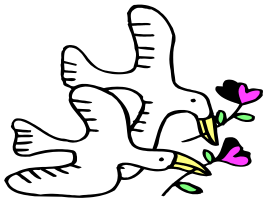
お断り：当資料は、DB2 Universal Database for Linux, UNIX and Windows V8.2 をベースに作成されています。



ブランク・ページです。

目次

- デッドロックの防止 (FOR UPDATEを使う)
- FOR UPDATEの問題
- Lock Request Clause
- 指定オプションとロック・モード
- Lockの明示指定を行う場合の考慮点
- SQL Function, SQL Methodの挙動
- まとめ

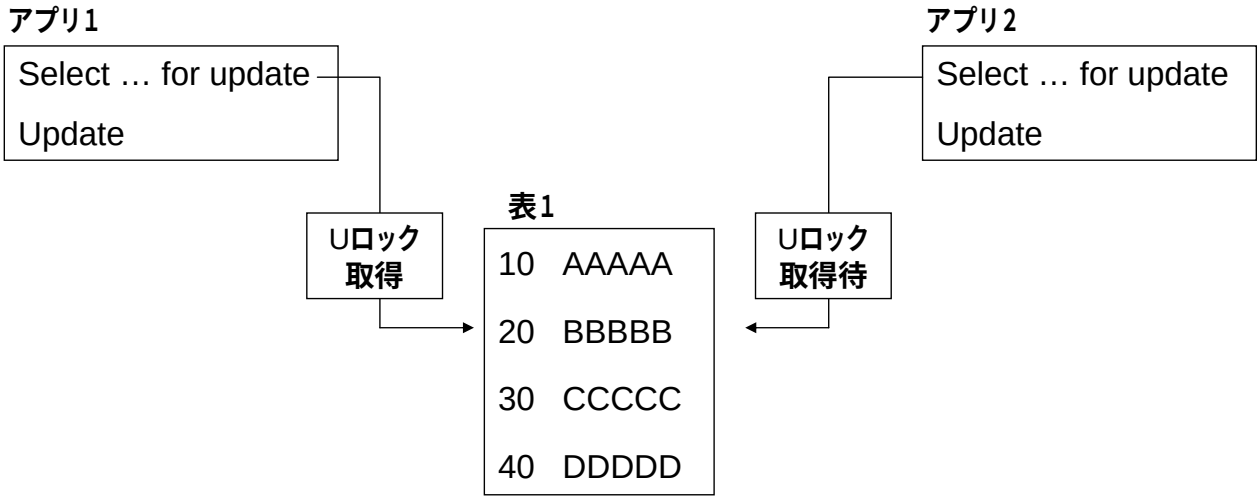


ブランク・ページです。

デッドロックの防止（FOR UPDATEを使う）

■Sロックの代わりにUロックを取得させる

- Uロック同士に互換性が無いためにロック待ちとなり、デッドロックを避けることができる



- 新機能の解説に入る前に、まずデッドロックを防止するためのロッキングの仕組みについておさらいしておきましょう。
- ひとつのトランザクションの中である行を参照、その後更新する、というような処理は、ごく一般的に行われる処理です。このとき、ある行を参照してから更新するまでの間、その行が他のトランザクションから更新されてしまうのを防止するため、DB2ではロックの機能が用いられます。例えば処理対象の行にSロックを保持しておけば、その間他のトランザクションからの更新を待たせることができます。これは更新処理に先立って取得されるXロックとSロックが競合関係にあり、Xロック取得が待たされるからです。
- このような方法によって、参照中のデータが別の処理によって更新されてしまうことは防止できますが、デッドロックの問題があります。先ほどの例のように、まず最初にSELECTでSロックを取得、次にその行にUpdateのためのXロックを取得、という風にしますと、同時に同じ行をSELECTする可能性が出てきます。Sロック同士は競合しませんので、双方のSELECTとも成功します。しかしその次のUPDATEはお互いに相手のSELECT処理によって取得されているSロックの開放を待つこととなり、デッドロック状態となります。
- このようなデッドロックを防止するためには、更新の意図があるSELECTでは、FOR UPDATEをつけたSELECTを実行するのが一般的です。これにより、SELECTで取得されるロックがSではなく、Uロックとなります。Uロック同士は競合しますので、FOR UPDATEをつけたSELECT同士は、同時に同じ行を参照できません。これによりデッドロックの防止が可能となるわけです。

FOR UPDATEの問題

- ブロッキングが使用できない
  - 結果行は1行1行送信
- JOINした表へのアクセスで使用不可
  - FOR UPDATEオプションでカーソルを定義、またはSELECTを実行しようとするとSQL0511のエラー

- このように、更新の意図があることをFOR UPDATEで宣言し、Uロックを取得させることでデッドロックの防止ができるわけですが、一方でいくつかの考慮点もありました。

①ブロッキングが使用できない
特に複数行に渡る結果行をクライアント･アプリケーションに返す場合には、ブロッキング機能を活用することがパフォーマンス上有利ですが、For Updateを指定したSelectでは、結果がブロッキングされない、という考慮点がありました。

② JOINした表へのアクセスで使用不可
SELECT文が結合された表や更新不能なView、ユーザー定義関数などに対する実行の場合、いずれもFOR UPDATEをつけて実行することは認められませんでした。そのようなケースでFOR UPDATEをつけると、SQL0511のエラーとなります。

```
SQL0511N カーソルで指定された表が変更できないので、FOR UPDATE文節は使用できません。
```

- Uロックを使用する際のこれらの考慮点は、Uロックを必要とするものの、update …where current of cursorという形式のUPDATEは無いような業務においてパフォーマンス上不利になっていました。従来のFOR UPDATEカーソルを多用した実装方法では、1行1行データが処理され、多くのデータフローを発生してしまっていました。

Lock Request Clause

- SELECTステートメントに明示的に取得するロック・モードを指定可能
  - Isolation Clauseとの組み合わせで使用する
    - WITH 分離レベル USE AND KEEP ロックタイプ LOCKS
  - 例:

```
select col1 from t1 where col2=100
      with rs use and keep update locks

select col1,col2 from t1,t2 where t1.col1=t2.col2
      with rr use and keep exclusive locks
```

- FOR UPDATE無しでUロック、Xロックの取得可能
- デッドロック防止のためのロックを取りつつ、Blockingによるパフォーマンス向上も得られる
- 結合した表へのSELECTでも任意のロック取得可

- これらの考慮点を解消するため、V8.2より、SELECTステートメントで取得するロック・モードを明示的に指定できるようになりました。これは分離レベルを指定するWITHオプションと一緒に指定します。
- なお、ロック・モードを明示的に指定できるのは、分離レベルRRとRSのときだけです。
- 上記の例の最初のSELECTでは、分離レベルRSでUロックを参照行に取得します。また2番目のSELECTでは、分離レベルRRでXロックを参照行に取得します。
- このように、FOR UPDATEをつけなくてもUロックやXロックを所得、保持することが可能になりました。この機能により、より具体的に取りたいモードのロックをアプリケーションから明示的に指定できるようになりました。そのため、デッドロック防止のためのロックを取らせつつ、ブロッキングによるによるパフォーマンス向上が得られる、またはジョインした表へのSELECTでも任意のロック取得が可能、というように前のページで挙げた課題の解決を可能とします。
- 開発元でのベンチマークテストにおいて、従来はfor updateをつけてSELECTを実行していたテストシナリオをUSE AND KEEP UDPATE LOCKSに置き換えたところ、6%のパフォーマンス向上が得られました。

指定オプションとロック・モード

	Read-only cursor	Updatable cursor
WITH RR (or default isolation RR)	S	U
WITH RR USE AND KEEP SHARE LOCKS	S	S
WITH RR USE AND KEEP UPDATE LOCKS	U	U
WITH RR USE AND KEEP EXCLUSIVE LOCKS	X	X
WITH RS (or default isolation RS)	S	U
WITH RS USE AND KEEP SHARE LOCKS	S	S
WITH RS USE AND KEEP UPDATE LOCKS	U	U
WITH RS USE AND KEEP EXCLUSIVE LOCKS	X	X

- こちらの表は分離レベルを指定するIsolation Clause、およびロックタイプを指定するLock Request Clauseに指定した値の組み合わせで、どのようなロックが取得されるかを示した表です。
- 右側が更新可能カーソル、すなわちfor updateを指定して定義したカーソルによる表アクセスの際に取得されるロックを示します。

実行例 (1)

```
create table test01 (a char(1), b char(1), c char(12))
DB20000I  SQL コマンドが正常に終了しました。

create index x1 on test01 (a)
DB20000I  SQL コマンドが正常に終了しました。

insert into test01 values
('A','a','data1'),('A','b','data2'),('A','c','data3'),
 ('B','a','data4'),('B','b','data5'),('B','c','data6')
DB20000I  SQL コマンドが正常に終了しました。

commit
DB20000I  SQL コマンドが正常に終了しました。

declare c1 cursor for select * from test01 where
test01.a='A'
  with rr use and keep update locks
DB20000I  SQL コマンドが正常に終了しました。

open c1
DB20000I  SQL コマンドが正常に終了しました。
```

```
select * from lock where table_name = 'TEST01'

TABLE_NAME LOCK_OBJECT_TYPE LOCK_MODE OBJECT_NAME
---------- ---------------- --------- --------------------
-
TEST01     ROW              U         7
TEST01     ROW              U         6
TEST01     ROW              U         5
TEST01     ROW              U         4
TEST01     TABLE            IX        TEST01

  5 レコードが選択されました。
```

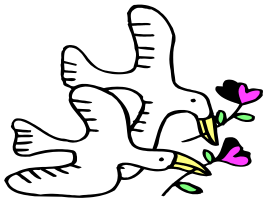

- 実際にLock Request Clauseを指定してカーソルを定義した例です。
- この例では、with rr use and keep update locksと指定がなされておりますので、分離レベルRRで索引スキャンが実行されています。そして各行に取得されるロックはUロックになっています。
- 取得されたロックを確認するため、この例では、ロック･スナップショットを実行する表関数を含むビューを定義、参照しています。ビューの定義は以下のとおりです。ロック･スナップショットの出力から、表の名前、ロックの取られた対象、ロックモード、ロック･オブジェクト名を抜き出しています。

```
create view lock (TABLE_NAME, LOCK_OBJECT_TYPE, LOCK_MODE, OBJECT_NAME)
as
select substr(TABLE_NAME,1,10) as TABLE_NAME,
case LOCK_OBJECT_TYPE
  when 1 then 'TABLE' when 2 then 'ROW' when 4 then 'TABLESPACE'
  when 5 then 'END OF TABLE' when 6 then 'KEY VALUE' when 19 then 'BLOCK'
  else 'OTHER' end,
case LOCK_MODE
  when 1 then 'IS' when 2 then 'IX' when 3 then 'S' when 4 then 'SIX' when 5 then 'X'
  when 6 then 'IN' when 7 then 'Z' when 8 then 'U' when 9 then 'NS' when 10 then 'NX'
  when 11 then 'W' when 12 then 'NW' end,
case LOCK_OBJECT_TYPE
  when 1 then TABLE_NAME
  else char(LOCK_OBJECT_NAME) end
from table (snapshot_lock('SAMPLE',-2)) as t;
```

実行例 (2)

```
create table test01 (a char(1), b char(1), c char(12))
DB20000I  SQL コマンドが正常に終了しました。

create index x1 on test01 (a)
DB20000I  SQL コマンドが正常に終了しました。

insert into test01 values
('A','a','data1'),('A','b','data2'),('A','c','data3'),
 ('B','a','data4'),('B','b','data5'),('B','c','data6')
DB20000I  SQL コマンドが正常に終了しました。

commit
DB20000I  SQL コマンドが正常に終了しました。

declare c1 cursor for select * from test01 where
test01.a='A'
  with rs use and keep exclusive locks
DB20000I  SQL コマンドが正常に終了しました。

open c1
DB20000I  SQL コマンドが正常に終了しました。
```

```
select * from lock where table_name = 'TEST01'

TABLE_NAME LOCK_OBJECT_TYPE LOCK_MODE OBJECT_NAME
---------- ---------------- --------- --------------------
-
TEST01     ROW              X         6
TEST01     ROW              X         5
TEST01     ROW              X         4
TEST01     TABLE            IX        TEST01
  4 レコードが選択されました。
```

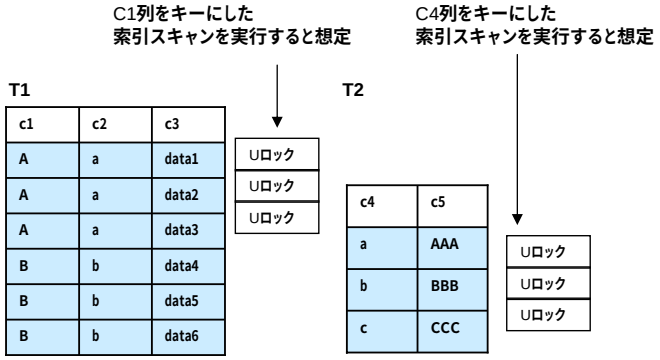



ブランク・ページです。

Lockの明示指定を行う場合の考慮点

- 分離レベルRS、またはRRで使用される
  - 大量のロック取得による並行稼動性低下に注意
    - どこにロックが取られるのか考慮すること

```
select c3,c5 from t1,t2 where t1.c1=A and t1.c2=t2.c4
  with rs use and keep update locks
```



- Lock Request Clauseを使って取得するロックを明示的に指定する場合に考慮すべき点は、分離レベルRS、またはRRで使用される、という点です。
- 分離レベルRSとは、実際に取り出された行に対してロックが保持されるような分離レベルで、分離レベルRRとは、実際に取り出されたかどうかとは関係なく、条件に合うかどうか評価された行にはすべて保持をするような分離レベルです。一方デフォルトの分離レベルであるCSでは、現在参照中の行のみ（カーソルが置かれている行にのみ）ロックが保持されます。
- 従って、分離レベル RSやRRを使用することは、分離レベルCSを使用することに比べ、より多くのロックが取得される可能性があることを意味します。効率の良い索引チューニングがなされていないような環境でRSやRRの分離レベルを用いると、大量に取得、保持されるロックが原因でトランザクションの並行稼動性が低下してしまう恐れがあります。
- 上記の最初の例では、結果としてアプリケーションに返されるのはT1表、T2表の先頭の1行目のみですが、トランザクション完了まで、T1表に関しては1～3行目にロックが保持されることになります。

SQL Function, SQL Methodの挙動

■create function / create methodの新しいオプション

- 関数/メソッドを呼び出すステートメントのLock Request Clauseで指定されたロックモードをFunction / Methodにも継承するか？
  - INHERIT ISOLATION LEVEL WITHOUT LOCK REQUEST
  - INHERIT ISOLATION LEVEL WITH LOCK REQUEST

- SQL Function/SQL Methodでのみ有効

- 取得されるロックのモードを明示的に指定できるようになったことに伴い、ユーザー定義の関数/メソッドについても新しいオプションが追加になりました。
- 追加された新しいオプションは、その関数やメソッドを呼び出すステートメントのLock Request Clauseで指定されたロックモードを、その関数やメソッド内にも継承するかどうかを指定するものです。
  - 継承しない: INHERIT ISOLATION LEVEL WITHOUT LOCK REQUEST
  - 継承する: INHERIT ISOLATION LEVEL WITH LOCK REQUEST
- このオプションは、SQL関数、およびSQLメソッドすなわちCREATE FUNCTION、CREATE METHODステートメントの中でSQLを使ってロジックの記述がなされているようなタイプの関数やメソッドのみで有効です。

Functionの例

```
create table test1 (c1 int, c2 int)
DB20000I  The SQL command completed successfully.

create unique index idx1 on test1 (c1)
DB20000I  The SQL command completed successfully.

insert into test1 values (1,100),(2,200),(3,300),(4,400),(5,500)
DB20000I  The SQL command completed successfully.

create function func1 (x int) returns int
 language sql reads sql data no external action
 deterministic
 inherit isolation level with lock request
 return select c2 from test1 where c1=x
DB20000I  The SQL command completed successfully.
```


Functionの例 (続き)

```
values (func1(3)) with rr use and keep update locks

1
-----------
        300

  1 record(s) selected.

select * from lock where table_name='TEST1'

TABLE_NAME LOCK_OBJECT_TYPE LOCK_MODE OBJECT_NAME
---------- --------------- --------- -----------
TEST1      ROW             U         6
TEST1      TABLE           IX        TEST1

  2 record(s) selected.
```

Methodの例

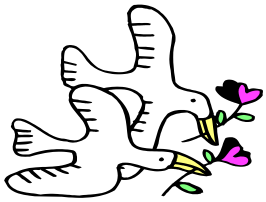
```
create type tp1 as (i1 int, i2 int)
  mode db2sql
  ref using int
  method mt1 (p1 int)
  returns int
DB20000I  The SQL command completed successfully.

create method mt1 (p1 int)
  returns int for tp1
 inherit isolation level with lock request
 return  select c2 from test1 where c1=p1;
DB20000I  The SQL command completed successfully.
```


Methodの例 (続き)

```
select tp1()..mt1(T.c1) from table( values(2) ) as T(c1)
  with rs use and keep exclusive locks

1
-----------
        200

  1 record(s) selected.


select * from lock where table_name='TEST1'

TABLE_NAME LOCK_OBJECT_TYPE LOCK_MODE OBJECT_NAME
---------- --------------- --------- -----------
TEST1      ROW             X         5
TEST1      TABLE           IX        TEST1

  2 record(s) selected.
```

まとめ

- FOR UPDATEを指定しなくとも、Lock Request Clauseにより任意のロックを取得できる
  - Blockingの効果によるパフォーマンス向上（対For Update）
  - Joinした表への任意のロック取得



ブランク・ページです。