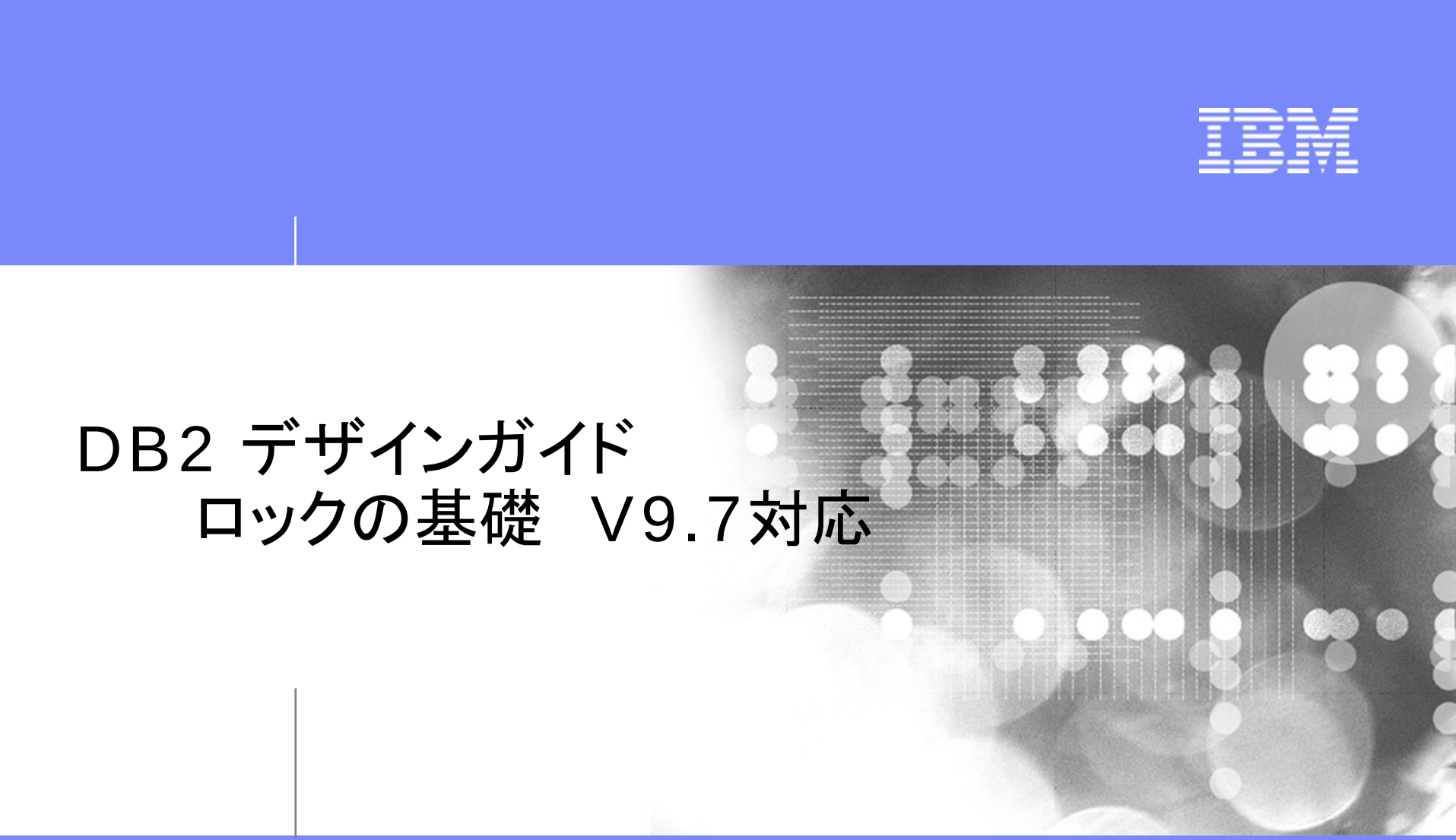


DB2 デザインガイド
ロックの基礎　V9.7対応

内容

- １．ロックの基礎
  - １．１．ロックと排他制御が必要な理由
  - １．２．排他制御とロック
  - １．３．ロックの対象と範囲
  - １．４．ロックの保持期間
  - １．５．ロック待機とタイムアウト
  - １．６．デッドロック
  - １．７．ロックエスカレーション
  - １．８．カーソルとロック
- ２．分離レベルとロック
  - ２．１．分離レベルとは
  - ２．２．分離レベルとロック
  - ２．３．分離レベルの設定と確認
- ３．ロックのモニタリング
  - ３．１．　今取得されているロックを確認するには？
  - ３．２．　データベース全体のロック情報を確認するには？
  - ３．３．　デッドロック発生時の詳細情報を確認するには？
  - ３．４．　ロックタイムアウト発生時の詳細情報を確認するには？
  - ３．５．　障害ログ（db2diag.log)に含まれる情報
  - ３．６．　V9.7 新機能：　ロックイベントモニターの紹介
- ４．オプティミスティック・ロッキング（V9.5～）
- ５．Currently Committed （V9.7～）

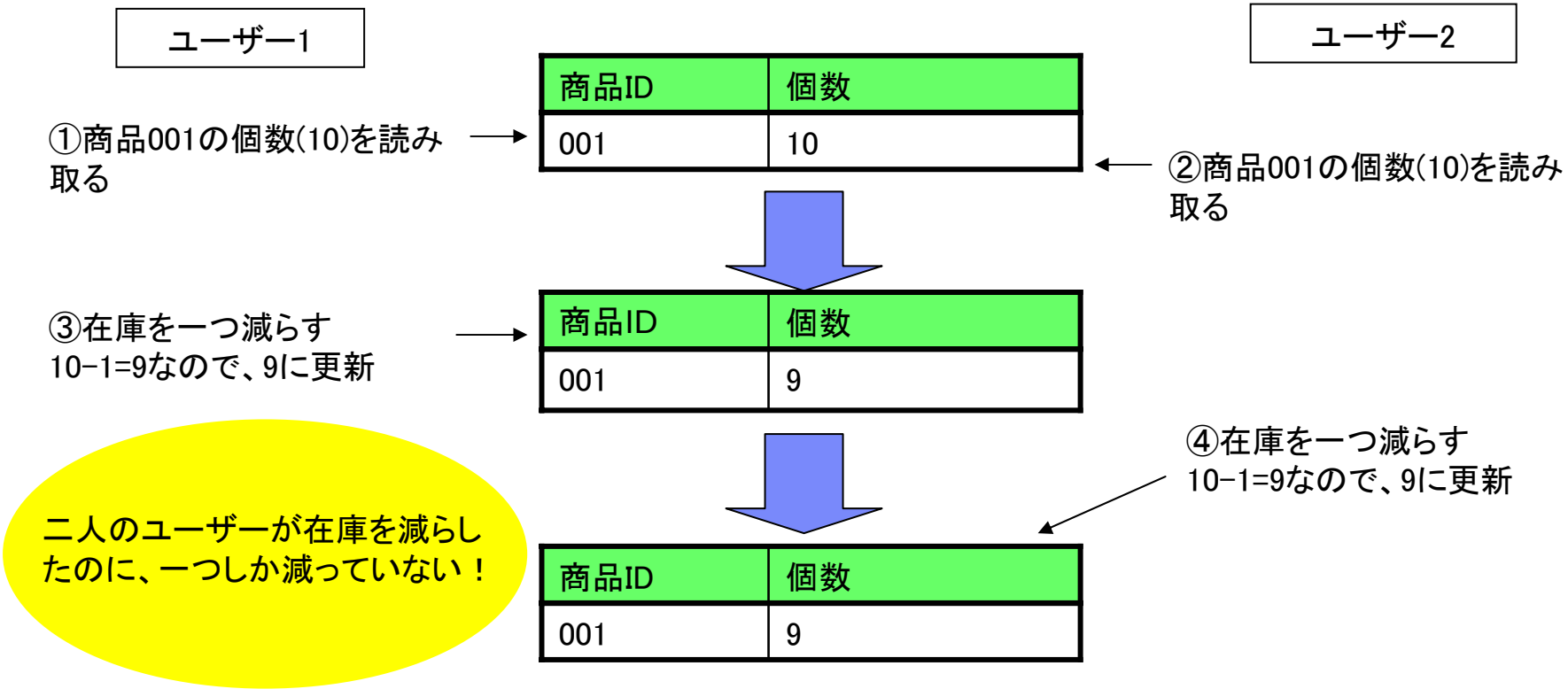
1. ロックの基礎

内容

- １．ロックの基礎
  - １．１．ロックと排他制御が必要な理由
  - １．２．排他制御とロック
  - １．３．ロックの対象と範囲
  - １．４．ロックの保持期間
  - １．５．ロック待機とタイムアウト
  - １．６．デッドロック
  - １．７．ロックエスカレーション
  - １．８．カーソルとロック

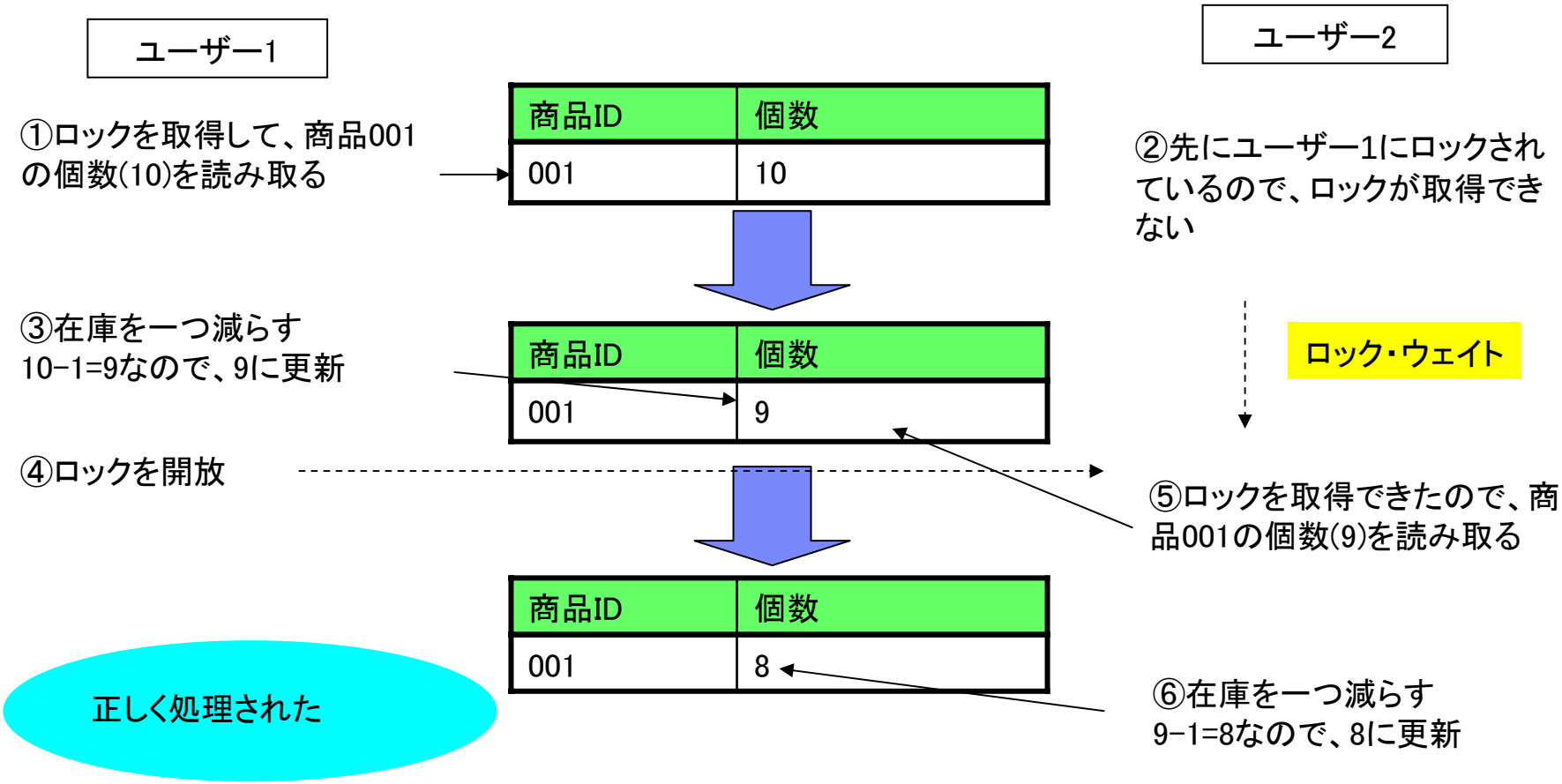
1．1．ロックによる排他制御が必要な理由

- ロックがないと・・・・・・
  - 複数ユーザー(複数トランザクション)で同じデータにアクセスした場合に、データの整合性が取れなくなる
  - 例）在庫の引き当て



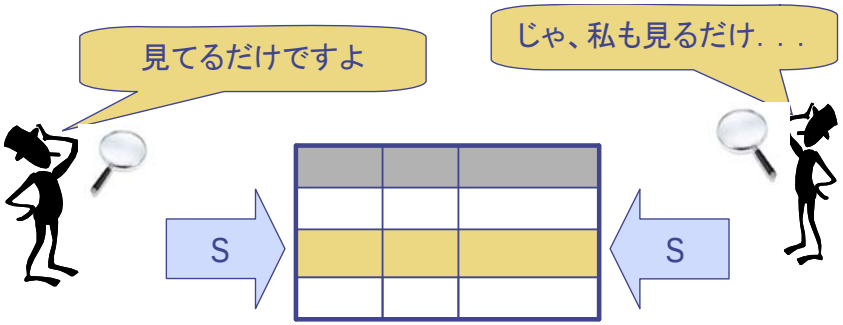
1．1．ロックによる排他制御が必要な理由

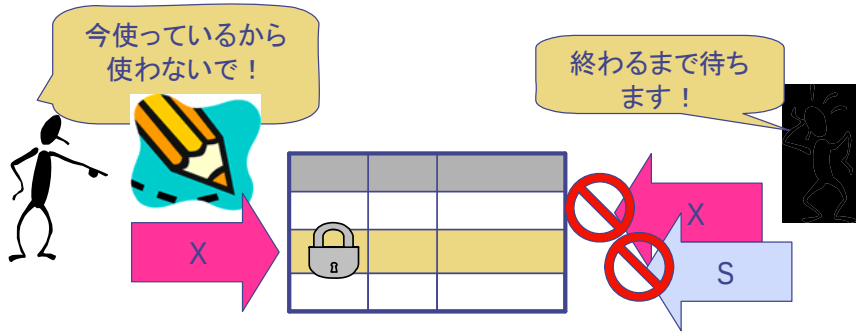
- ロックがあると・・・・
  - 複数のトランザクションから同時にアクセスされた場合でも整合性を保つことができる



1．2．排他制御とロック　-　ロックの基本的なモード

- ロックの基本的なモード
  - ロックのモード　（基本的には3種類）
    - 排他ロック（e**X**clusive）：更新処理時に取得されるロック
    - 更新ロック(**U**pdate)：更新を意図した参照を実施するときに取得されるロック
      - デッドロックの回避目的
    - 共用ロック（**S**hared）：照会処理時に取得されるロック





1．2．排他制御とロック　－　ロックの互換性

- ロックの互換性
  - ロックのモード同士で互換性が異なる
  - 既に取得されているロックが解放されるのを待たなければ、別アプリケーションからのロックの取得が不可能な場合 、それらのロック・モードは「互換性がない」
    - 互換性がある　→　互いにロックを取得しデータ操作ができる
    - 互換性がない　→　一方がロックを取得している間は、対象データに対して要求するロックを取れない

		保持されているロック		
		S（共有）	U（更新）	X（排他）
要求されているロック	S（共有）	○	○	×
	U（更新）	○	×	×
	X（排他）	×	×	×

○：互換性がある
→　要求ロックを取ることができる

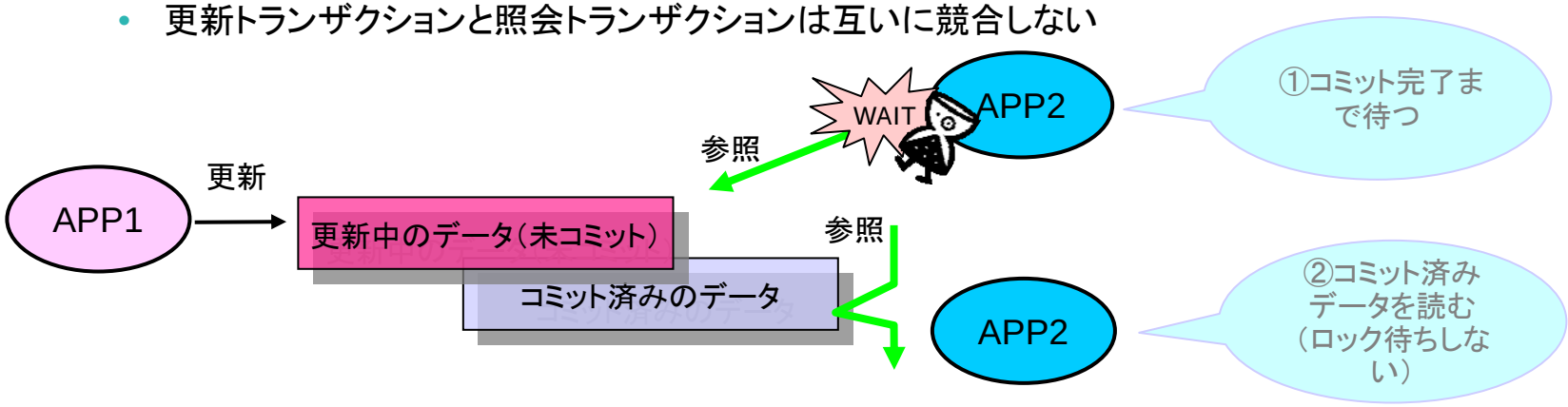
×：互換性がない
→　要求ロックを取ることができない

1．2．排他制御とロック - ロック競合時のデータの見せ方

- 更新中のデータを参照しようとした場合に、そのデータの見せ方に対して2つのポリシーがあり、V9.7からデフォルトの動作が変わっている。
  - ① コミットが完了するまで待つ　（V9.5までの動き）
    - 照会対象のデータが別のトランザクションにより更新中である場合、対象データがコミットまたはロールバックされるまでアクセスさせない
    - 照会トランザクションは、常に最新状態のデータを参照できる
  - **② 最後にコミットされた（更新前の）状態を見せる （V9.7　分離レベルCSでのデフォルト動作）**

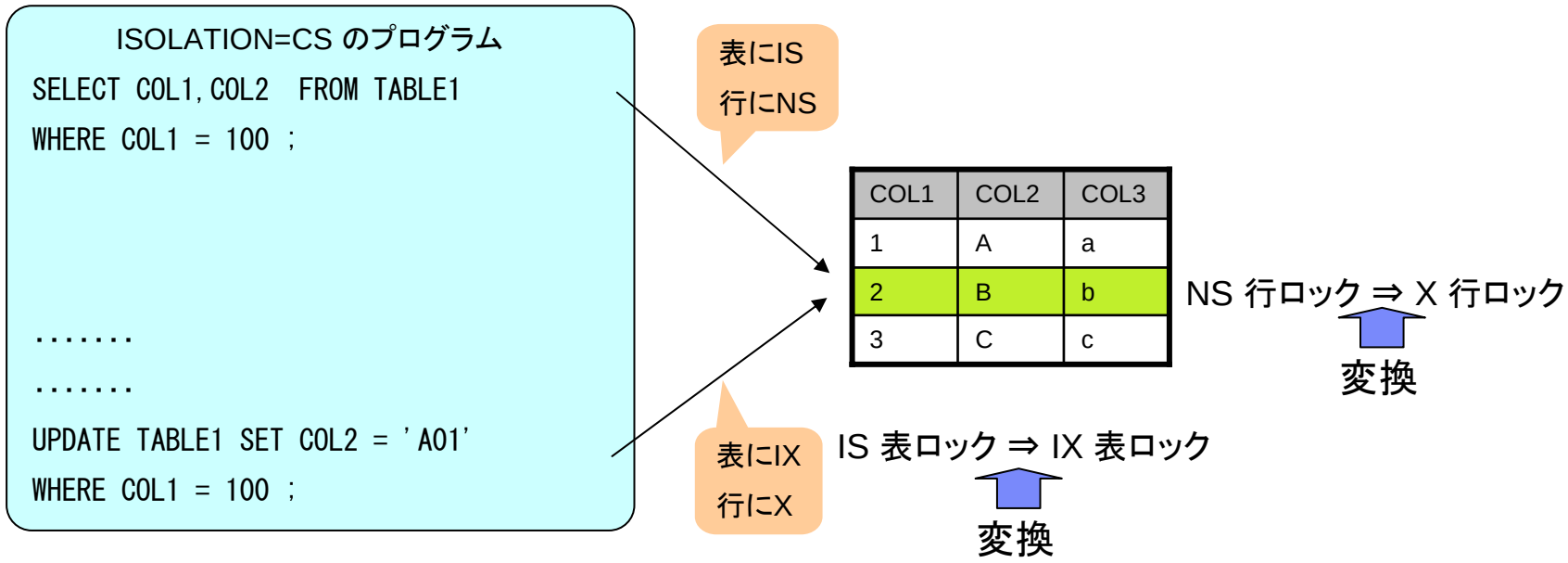
V9.7 NEW

    - 照会対象のデータが別のトランザクションにより更新され未コミット状態である場合、更新前の状態を参照させる
    - 更新トランザクションと照会トランザクションは互いに競合しない



1．2．排他制御とロック – ロックの変換

- ロックの変換
  - ロックは、処理に必要なモードに変換されていく
    - 1 アプリケーションから各 DB オブジェクトに取得可能なロックはひとつ
      - ひとつのアプリケーションが、ひとつの行に複数のロックを取得することは出来ない
  - ロック変換の例
    - 照会した行を更新する場合



（参考） 1．2．排他制御とロック – ロックのモード詳細(1/3)

- ロックのモード（詳細）
  - 11 のロック・モード (* 排他性の低い順に列挙)
    - 照会系のロック
      - IN (Intent None) 、IS (Intent Share) 、NS (Next Key Share) 、S (Share)
    - 更新系のロック
      - IX (Intent Exclusive) 、SIX (Share with Intent Exclusive) 、U (Update) 、

        NW (Next Key Weak Exclusive)、X (Exclusive) 、W (Weak Exclusive) 、

        Z (Super Exclusive)

実際にはより多くのロックのモードが存在する。

（参考） 1．2．排他制御とロック - ロックのモード詳細(2/3)

- ロックのモード（詳細）
  - 厳密には、さらに ISOLATION とアクセス・パスによりロックの対象となるモードが変化する
    - 表や行のみに取られるロックモードや、両方に取られるロックモードがある。
      - 表スペースにとられるロック　　: IN、IS、IX、Z

モード	対象	このモードで表ロックが取得されたときの、同一トランザクションからの行ロック	このモードでロックが取得されるケース
IN	表	行ロックは取得しない	UR で照会時の表ロック
IS	表	行ロックは取得しない	RR、RS、CS で照会時の表ロック
NS	行		RS、CS で照会時の行ロック
S	行 表	行ロックは取得しない	RR で照会時の行ロック 表単位でロックが取得される場合の、照会時の表ロック
IX	表	照会行に S、NS、U 更新行に X	更新時または FOR UPDATE カーソルでの照会時の表ロック
SIX	表	照会行にロックなし 更新行に X 行ロック	同一トランザクション内で、すでに S を取得しているときに IX ロック、またはすでに IX を取得している表に S ロックの取得要求が発生した場合の表ロック
U	行 表	行ロックは取得しない	FOR UPDATE カーソルでの照会時の行または表ロック更新時には X ロックに変わる 表単位でロックが取得される場合の、FOR UPDATE カーソルで照会処理時の表ロック
NW	行		索引のある表に INSERT した行の、次の行に保持される、瞬間的なロック タイプ2索引では、次の行が RR スキャンのアプリケーションにロックを保持されている場合に、NW を取得する
X	行 表	行ロックは取得しない	更新時の行ロック 表単位でロックが取得される場合の、更新時の表ロック
W	行		タイプ2索引が作成されていない表の索引にINSERTした時の行ロック
Z	表		特定の状況下で取得　(表の ALTER、DROP、REORG、索引の CREATE、DROP 等)

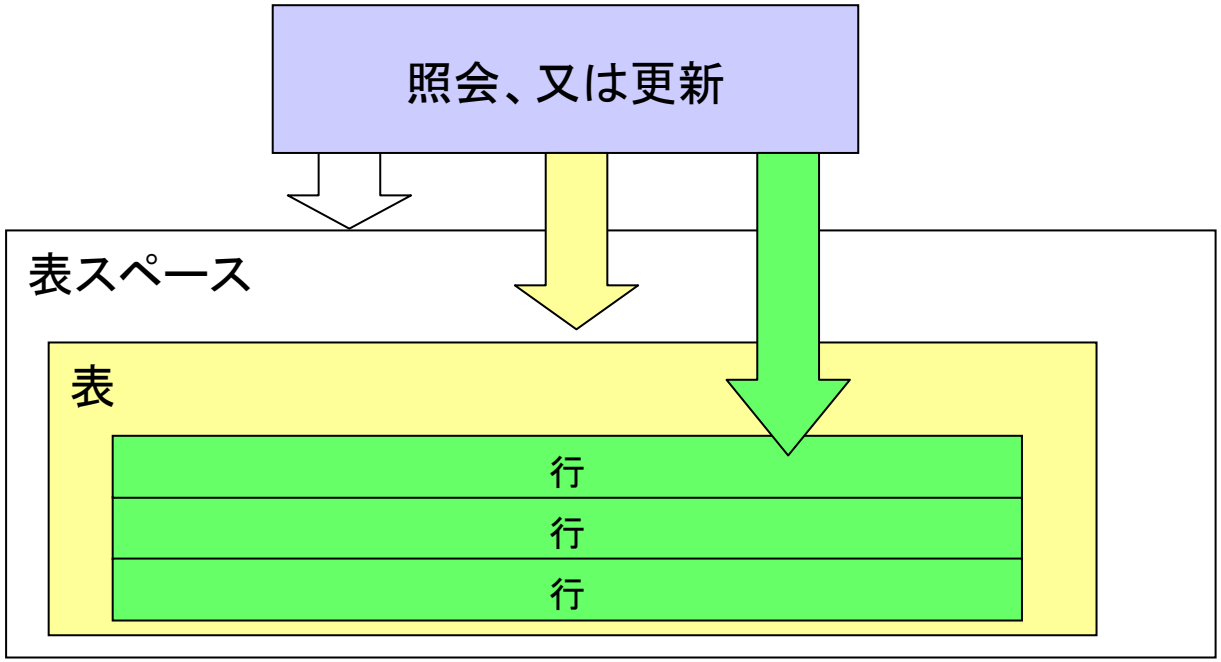
（参考） 1．2．排他制御とロック - ロックのモード詳細(3/3)

- ロックのモード詳細 （互換性一覧）
  - ○ ：ロック取得可能
  - × ：ロック取得不可能となり、ロック待ちとなる

	取得されているロック											
取得要求されているロック	モード	None	IN	IS	NS	S	IX	SIX	U	X	Z	NW	W
	None	○	○	○	○	○	○	○	○	○	○	○	○
	IN	○	○	○	○	○	○	○	○	○	×	○	○
	IS	○	○	○	○	○	○	○	○	×	×	×	×
	NS	○	○	○	○	○	×	×	○	×	×	○	×
	S	○	○	○	○	○	×	×	○	×	×	×	×
	IX	○	○	○	×	×	○	×	×	×	×	×	×
	SIX	○	○	○	×	×	×	×	×	×	×	×	×
	U	○	○	○	○	○	×	×	×	×	×	×	×
	X	○	○	×	×	×	×	×	×	×	×	×	×
	Z	○	×	×	×	×	×	×	×	×	×	×	×
	NW	○	○	×	○	×	×	×	×	×	×	×	○
	W	○	○	×	×	×	×	×	×	×	×	○	×

1．3．ロックの対象と範囲

- ロックの対象範囲となるものには、行、表、表スペースなどがある。（※）
  - 照会・更新のアプリケーション実行時には行または表単位のロックが取得される。
  - ユーティリティー実行時には行、表に加えて表スペース単位でのロックが取得される場合もある。
  - ロックの範囲が広くなれば、並行稼動性は低くなる（ロック待ちが発生しやすい）
- 索引にロックは取得しない



--- ※ MEMO ---

* MDC (Multi Dimension Clustering) 環境の場合には、行またはブロック、表、表スペースがロックの単位

* パーティション表の場合には、行、パーティション、表、表スペースがロックの対象

（参考）1．3．ロックの対象と範囲 - ロックの単位を制御するには？

- プログラムから明示的に表ロックの取得が可能
  - LOCK TABLE 表名 IN SHARE MODE　　：表に共用ロックを取得
  - LOCK TABLE 表名 IN EXCLUSIVE MODE　：表に排他ロックを取得
- ロックの単位を表ごとに設定可能
  - ALTER TABLEステートメントの LOCKSIZE オプション
    - ROW　　：行単位でのロック (省略時値)
    - BLOCKINSERT　　：挿入操作時のブロックロック指定 (MDC 表に対してのみ有効)
    - TABLE　　：表単位でのロック
- LOCKSIZE オプションと LOCK TABLE ステートメントの比較

ロックを制御する方法	ALTER TABLE table_name LOCKSIZE xxxx	LOCK TABLE table_name IN xxxx MODE
影響の範囲	その表にアクセスする全トランザクション	単一のトランザクション
設定の持続性	永続的	一時的 LOCK TABLE による活動化 COMMIT あるいは ROLLBACK による非活動化
考慮点	表の存在する表スペースのポイント・イン・タイム回復の回復可能な最短時間に影響を与える	回復可能最短時間に影響を与えない

（参考）1．3．ロックの対象と範囲　-　解説：ロックの単位を制御するには？(1/2)

- ALTER TABLE ステートメントの LOCKSIZE オプションにより、表にアクセスする際に使用されるロックのサイズを指定することができます。省略時では、表が作成されるときに、ロックのサイズは行レベルを取るように設定されています。
- ALTER TABLE ステートメントで表を変更し、ロックの設定を表レベルにあげることが出来ます。表レベルのロックを設定し、取得・解放されるロックの数を減らすことにより、照会処理のパフォーマンスが向上する可能性があります。しかし、表全体にロックが取得されるため、同時稼動性は落ちます。
- 一度設定を可能にした後は、その表に関する後続のトランザクションが表にアクセスするときに使用されるロックのサイズは、指定したロックモードで処理されます。表定義におけるこのオプションを使用することで、通常のロック・エスカレーションの発生を抑えることにはなりません。また、LOCK TABLE ステートメントは、表スペースのポイント・イン・タイム回復が可能なことに何も影響を与えないことに対して、ALTER TABLE ステートメントは、ポイント・イン・タイムで回復可能な時点に関連します。

（参考）1．3．ロックの対象と範囲 - 解説：ロックの単位を制御するには？(2/2)

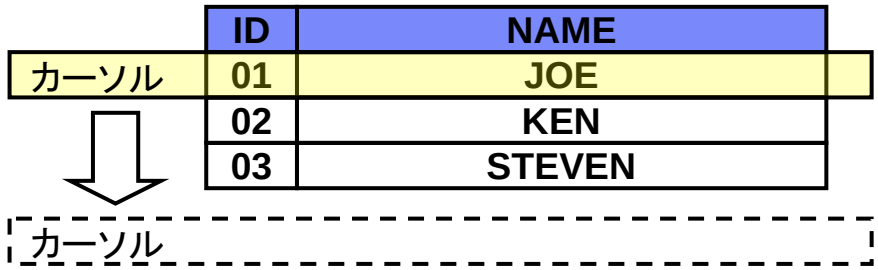
- ALTER TABLE で指定可能なロックの単位
  - ROW
    - 行ロックを取ることを意味します。表が作成されるときのロック・サイズの省略時値です。
  - TABLE
    - 表ロックを取ることを意味します。適切な、共用または排他ロックが表にとられ、意図ロックは使用されません。この値を使用し、取得が必要なロックの数を減らすことにより、照会のパフォーマンスを向上させる可能性があります。しかし、ロックは表全体に取られるので、同時稼動性は落ちます。以下の場合には、表ロックを使用するほうが良いでしょう。
      - 表が読み取りのみで、常に S ロックが必要である。他のトランザクションがその表に S ロックを取得することが可能であるため、表レベル・ロックはパフォーマンスを向上させる。
      - ある一人のユーザーが保守のために表にアクセスし、限られた時間内で X ロックを取得する。その表に ALTER TABLE により表レベル・ロックを定義することにより、表レベルの X ロックを取得可能とする。そのユーザーの作業が一旦終了すれば、再度 ALTER TABLE により、行レベルロックを取得するように、表の設定を戻すことが出来る。
  - BLOCKINSERT
    - MDC 表のみに適用可能な、ブロック・レベルのロックです。
    - INSERT 操作の間のみ、ブロック・レベルのロックが使用されます。INSERT 以外の操作では行ロックとなります。
    - 以下の効果が見込めます。
      - 同じセルに大量に INSERT を行う場合に、同時稼動性が向上
      - 取得が必要なロックの数を減らす
      - ロックを取得、解放する回数が少ないので、CPU負荷は低い

1．4．ロックの保持期間

- 作業単位(Unit of Work)が完了 (COMMIT / ROLLBACK) されるまで保持される
  - 作業単位を意識して、こまめにCOMMITを実施するようにする。
- ロックの取得
  - ブロッキングが有効な参照の場合、カーソルのOPEN時に（サーバー側でFETCHが行われ）ロックが取得される。
  - ブロッキングが無効の参照の場合、アプリケーションがカーソルからFETCHする際にロックが取得される。
- ロックの解放
  - 分離レベルCS（CUR_COMMIT=DISABLE）では、ロックはカーソルが移動すると解放される。
    - 参照時には、実際は**ブロッキング**によって、クライアントが読む前にカーソルは移動しロックが外れている。
  - 分離レベルRSでは選択行にロック（NS)が残る。分離レベルRRでは走査行にロック(S)が残る。
  - カーソルを CLOSE しても、ロックは外れない
    - ロックを外しリソースを解放するためには、必ず COMMIT を行うこと
    - RR / RS のアプリケーションによる Read Lock については、CLOSE CURSOR の WITH RELEASE オプションを使用することにより、CLOSE時に解放することが可能

V9.7
CUR_COMMIT=
ONではロックを取
得しない。

※ ISOLATION レベル (CS、RR、RS など) に関しては後述

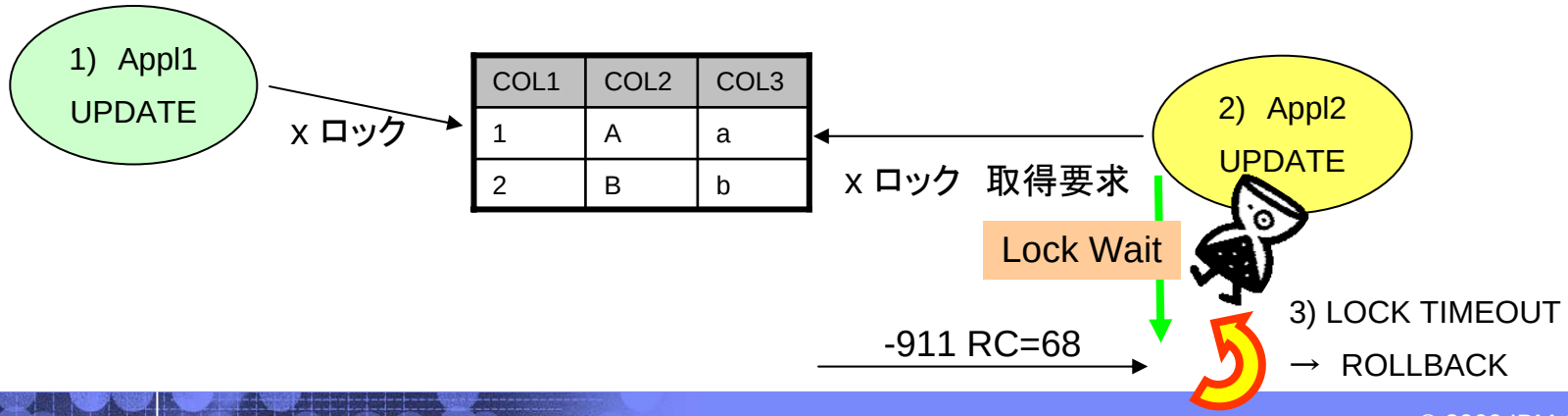


1．5．ロック待機とロック・タイムアウト

- ロック待機 (Lock Wait)
  - 要求したロックに互換性のないロックが取られていた場合、ロックが解放されるまで待つこと
- ロック・タイムアウト
  - ロック待機状態でいた時間が LOCKTIME に達した場合、待機していたアプリケーションの作業単位がロールバックする
    - SQLCODE -911 RC 68 がアプリケーションに戻される
  - LOCKTIMEOUT　（DB構成パラメータ）
    - ロック待ちを行う時間（秒）を設定する
    - 省略時値　　:-1　（無制限にロック待ちを行う）

1) Appl1 がレコード1 をUPDATE

2) Appl2 がレコード1 をUPDATEする際に、Lock Wait



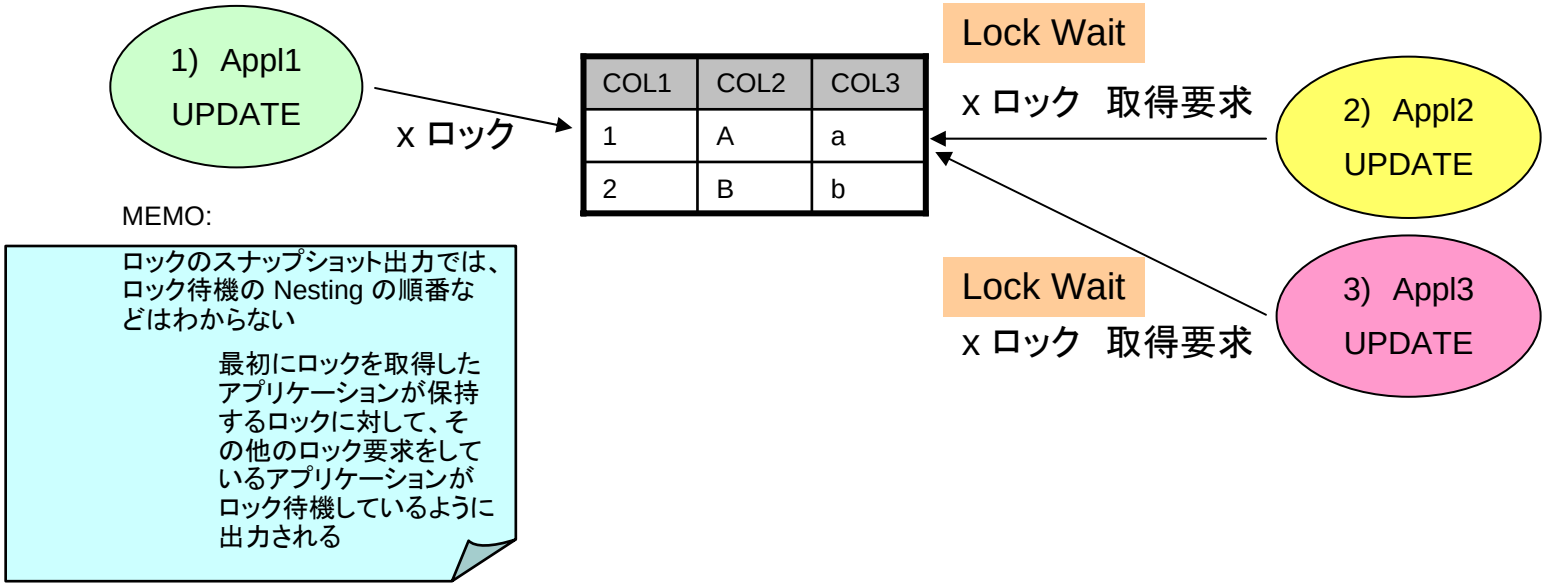
1. 5. ロック待機とロック・タイムアウト - ロック待機のNesting

- ロック待機は Nest する
  - 既にロック待機が発生している DB オブジェクトに、互換性のないロック取得の要求があった場合、その要求以前に存在したロック待機がすべて解消するまで、ロック取得は待たされる

1) Appl1 がレコード1 をUPDATEし、X ロックを取得

2) Appl2 がレコード1 をUPDATEする際に、Appl1 のロックに対して Lock Wait

3) Appl3 がレコード1 をUPDATEする際に、Appl2 のロックに対して Lock Wait

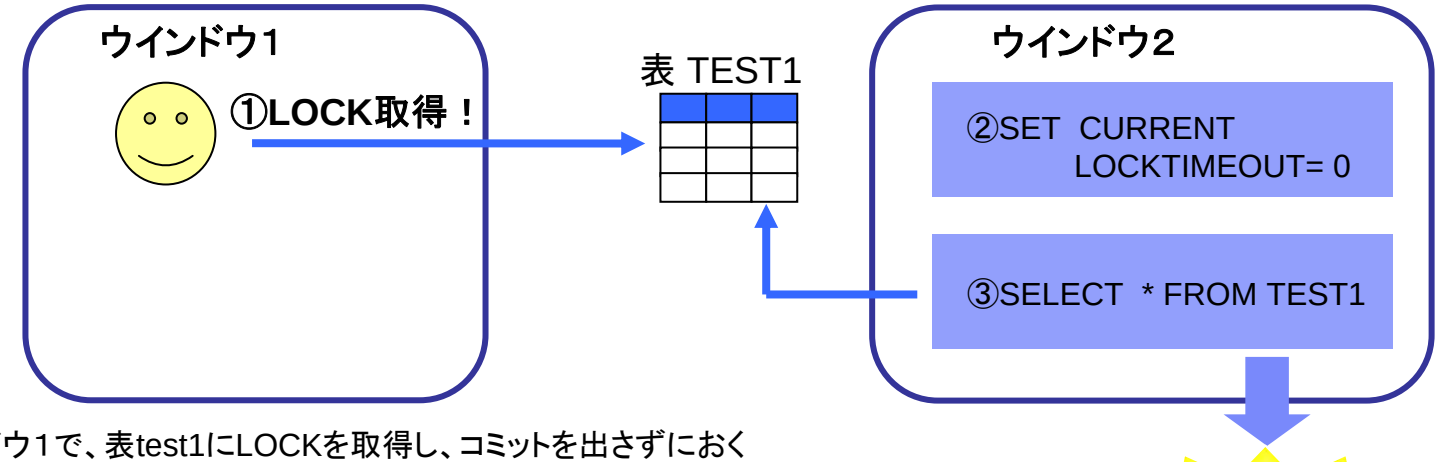


（参考）1．5．ロック待機とロック・タイムアウト - セッション単位でロックタイムアウトを設定するには？(1/3)

- SET LOCKTIMEOUTによって、セッション単位で、ロック・タイムアウトを設定
  - ロックタイムアウト設定(優先度順)
    - CURRENT LOCK TIMEOUT特殊レジスターでの設定
    - LOCKTIMEOUT DB構成パラメーターでの秒数指定
- 使用方法
  - SET CURRENT LOCK TIMEOUT [WAIT/NOT WATI/NULL/WAIT 数値]
    - 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
- 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 }
- 注意点
  - 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 LOCKTIMEOUT特殊レジスターの値を引き継がない

（参考）1．5．ロック待機とロック・タイムアウト
- セッション単位でロックタイムアウトを設定するには？(2/3)



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
```

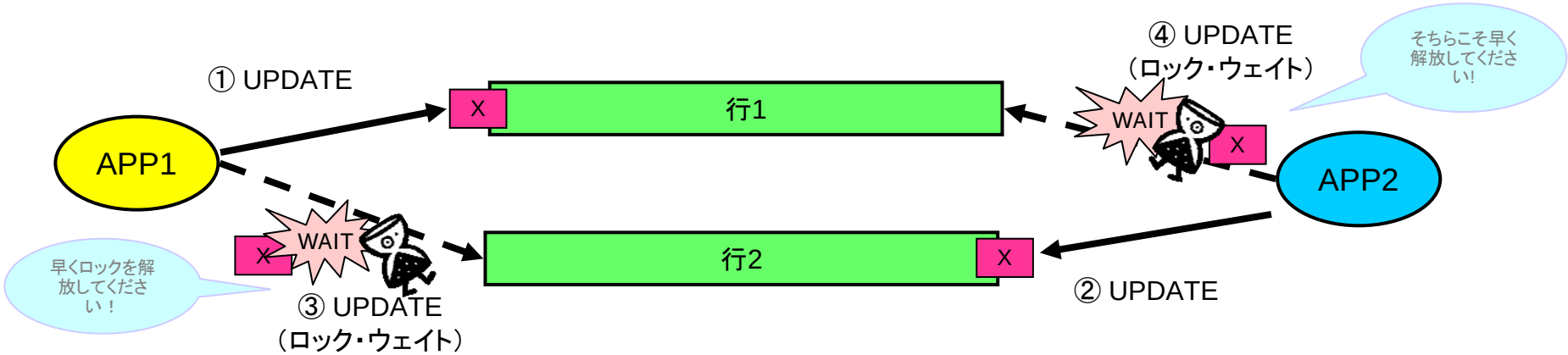
（参考）1．5．ロック待機とロック・タイムアウト - セッション単位でロックタイムアウトを設定するには？(3/3)

アプリケーションからの実行例（ソースコード・サンプル）

```
//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));
  }
```

1．6．デッドロック　– デッドロックとは

- デッドロックとは
  - 複数のトランザクションが、お互いにロックの解放待合って、双方とも動きが取れなくなる状況
  - デッドロックを解消するために、どちらかのトランザクションをROLLBACKする



時間

APP1	APP2
行1をUPDATE（行1にXロック）	
	行2をUPDATE（行2にXロック）
行2をUPDATE（ロック・ウェイト）	
	行1をUPDATE（ロック・ウェイト）

お互いがロック・ウェイトし続ける ＝ デッド・ロック

1．6．デッドロック　－　デッドロックの検出機能

- デッドロックが起こって止まったままにならないように
  - **デッドロック検出機能**が監視している
    - 監視の間隔はデータベース構成パラメーターのDLCHKTIMEで変更可能（デフォルトは10秒）

- デッドロックを検出すると…
  - デッドロックを解消するために、**どちらかのトランザクションをROLLBACK**する
  - SQL0911エラー(理由コード：2)　→　db2diag.logに記録される

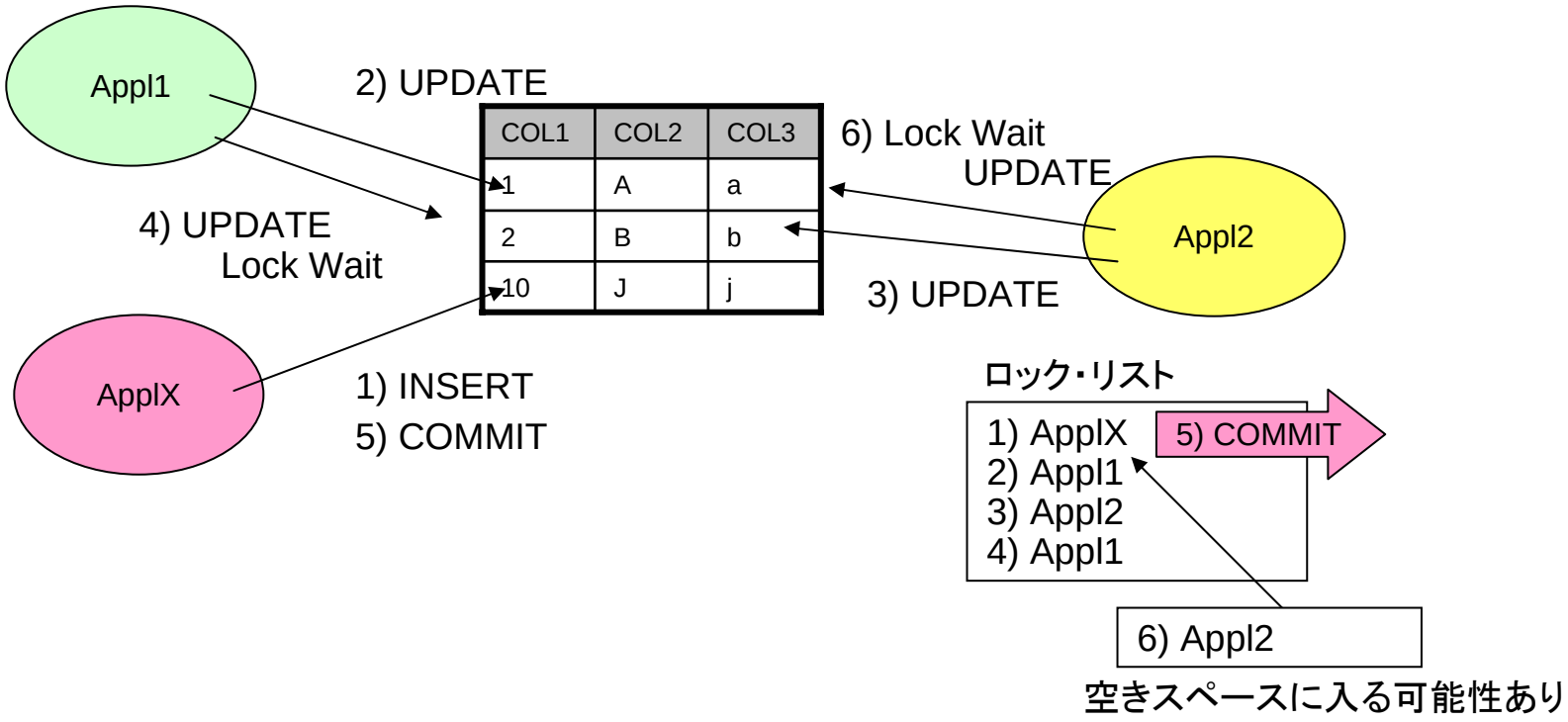
デッドロック検出機能をあてにして
アプリケーションを作成するべきではない

（参考）1．6．デッドロック　–　デッドロックの発生と回避

- デッドロック発生の要因
  - ロック・エスカレーションの発生
  - アプリケーション内での表ロックの取得
  - BIND 時の分離レベル (ISOLATION) の範囲が不必要に広い
  - 複数アプリケーションが幾つかの同じ行に、異なる順番でロックを取得しようとしている
- デッドロック発生の回避
  - 頻繁な COMMIT
  - ロックの範囲がより狭い分離レベルを使用
  - 参照結果を更新する場合、結果行に更新ロック（Uロック）を取得する（FOR UPDATE OFカーソルを利用）
  - 同一表への大量更新時には、あらかじめ表ロックを取得し、処理後速やかに COMMIT を行う
  - ロック・エスカレーションの頻度を下げる
  - WITH RELEASE オプション付きのカーソルの CLOSE (RR、RS の時のみ) を使用
- 例外処理ルーチンの組み込み
  - SQLCODE=-911、SQLERRD(2)=2

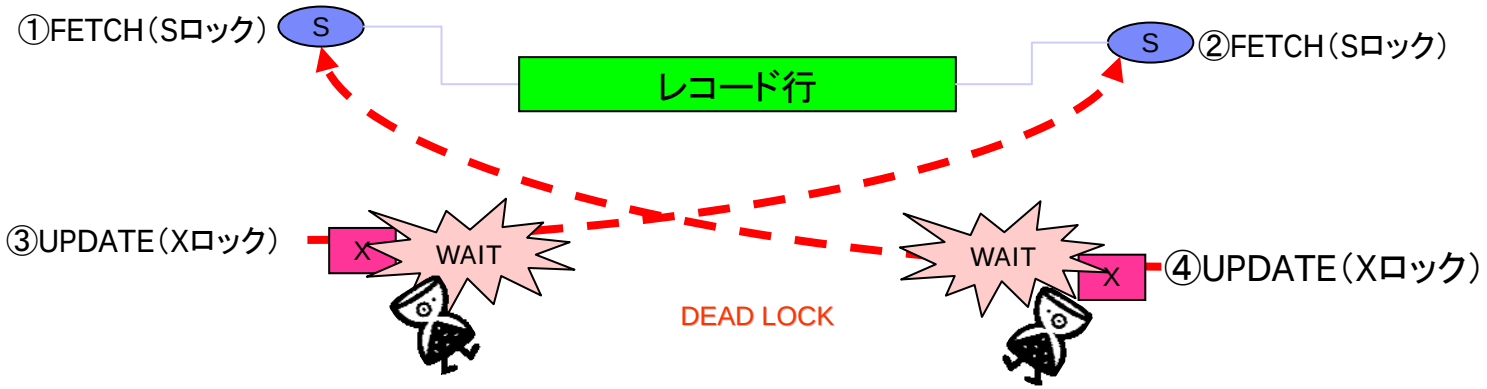
（参考）1．6．デッドロック
– どのトランザクションがデッドロックによってロールバックするか？

- デッドロック・ディテクターは、ロックリストを読み、ロックのチェインを作成
- デッドロックが検知された場合
  - デッドロック・ディテクターは、ロック・チェインへ最後に取りこまれたトランザクションを選択しロールバックする。
  - ロールバックされるのは、必ずしも最後に開始したトランザクションではない



1．6．デッドロック – デッドロックを回避するためのU（更新）ロック

- 参照後に更新を行うようなトランザクションを並行実施する場合にデッドロックが発生するケースがある。

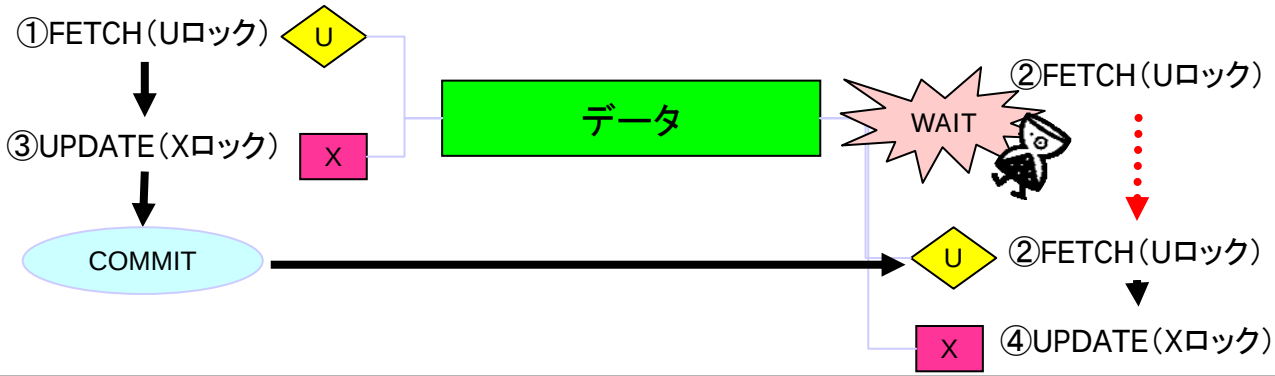


APP1	APP2
① 行1を参照（Sロック取得）	
	② 同じ行1を参照（Sロック取得できてしまう）
③ 行1を更新要求（行1をXロックに変換） → APP2が持っているSロック（②）の解放待ち	
	④ 同じ行1を更新要求（行1をXロックに変換要求） → ①のAPP1が持っているSロック（①）の解放待ち

取得済のSロックに対して、Xロックを取得する事はできない為、ロックの開放を待ってしまう

1．6．デッドロック – デッドロックを回避するためのU（更新）ロック （2.2）

- FOR UPDATE OF付きカーソルを使用して参照することで、結果行に更新ロック（Uロック）が取られる
- Uロックが取られている行に対して、他方のトランザクションはUロックを取得することができず、相手がロックを解放するまで待ちます
- お互いが保持しているロックを互いに待ち合うという状況、すなわちデッドロックにはならない

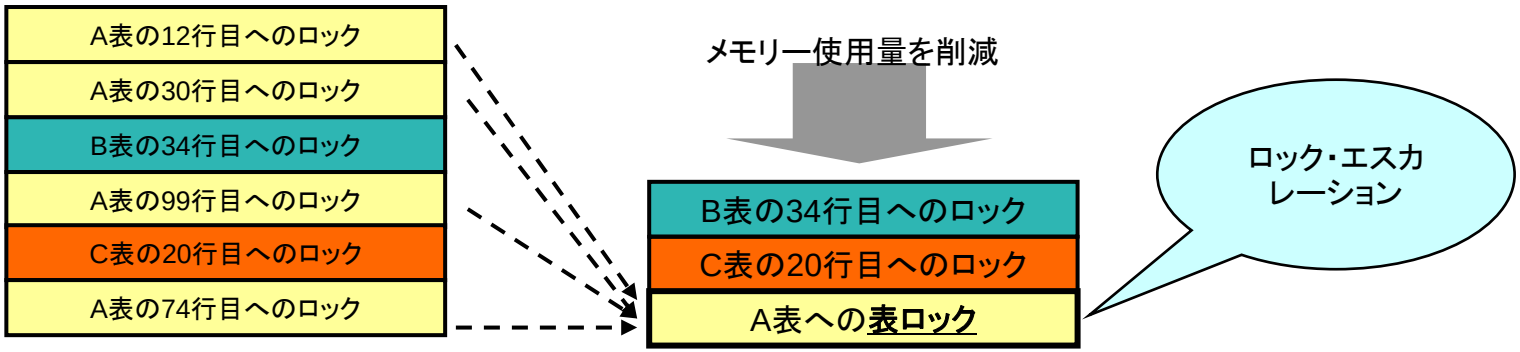


APP1	APP2
①行1をFOR UPDATEで参照（Uロック取得）	
	②行1をFOR UPDATEで参照（ロック待ち）
③行1を更新要求（U→Xロックに変換） コミット実施	
	行1にUロックを取得（ロック待ち解除）
	④行1を更新要求（U→Xロックに変換）

1．7．ロックエスカレーション- ロック情報用のメモリ

- ロック情報は、メモリ（ロック・リスト）に保存される
  - DB全体で1つ（※）。すべてのアプリケーションが共有
  - 最大量はLOCKLISTパラメーターで指定
  - ロックのリクエストは順番を維持して保存される
- DB全体でロック・リストが不足する場合や、1つのアプリケーション（接続）が使用する量が指定したパーセンテージを超えた場合に「ロック・エスカレーション」が発生
- ロック・エスカレーション
  - 多数の行 (またはブロック）ロックを1つのテーブルロックに変換することで使用メモリー使用量を削減すること。
  - ロックリストの節約になるが、ロックの範囲は広がる（表レベルとなる）ため、並行可動性は落ちる。

※ NOTE
区分化データベースではノードごとにロックリストの管理が行われる。



1．7．ロックエスカレーション – ロックエスカレーションの原因と回避

- ロック・エスカレーションの原因
  - 1 アプリケーションに許されるロックの数を超えて、ロックを保持しているアプリケーションがある場合
    - トランザクションが maxlocks を超えた場合、そのトランザクションが最も多くのロックを保持している表の行ロックを、表ロックへ変更する。
  - DB全体でロックに使用されているメモリー容量が、locklist の値に達した場合
    - ロックリストを最も多く使用しているトランザクションが選ばれ、同じ表への行ロックを表ロックへ変更する。
- ロック・エスカレーションの回避
  - ロックの範囲がより狭い分離レベルを使用する
  - 同一表に大量の更新処理がある場合には、あらかじめ表ロックを取得し、処理後速やかにCOMMITを行う
  - 頻繁なCOMMITを行う
  - locklist (DB CFG) 、maxlocks (DB CFG) を増やす
    - どちらも現在はデフォルトでAUTOMATIC設定であるため、必要に応じて調整される

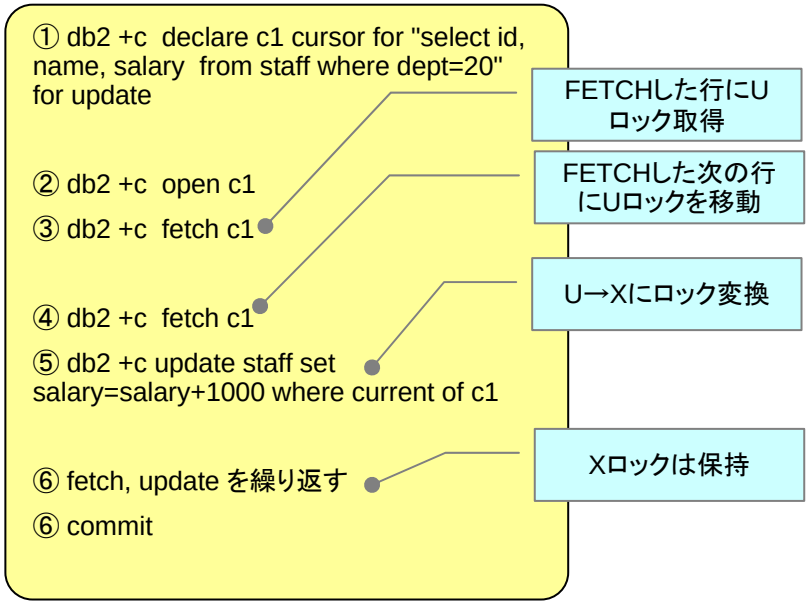
1．8．カーソルとロック - ブロッキングとロックの関係

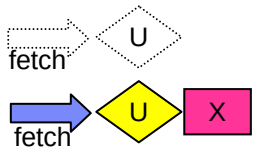
- ブロッキングとは
  - クライアントへのデータ転送効率化のため、サーバーからの結果行をブロックにまとめてクライアントに返す。
- ブロッキングが行われる場合のロック
  - ブロッキングが行われる場合、サーバーからブロックが転送された時点でロックは既に解放されているため、アプリケーションからFETCHする際にはロックが取られない。
    - アプリケーションプログラムでOPEN CURSOR した時に、サーバー側でのFETCH が次々と実行されて結果行が入ったブロックがクライアントのメモリーに返される。このときにサーバー側でのロックは既に外れている。
    - アプリケーションからのFETCHによって返される行は、クライアントに到着済みのメモリーに入っているブロックからの行が渡されるため、サーバーとの通信やロックの取得/解放は、アプリーション側でのFETCH毎に行われない。
- ブロッキングが行われる条件
  - カーソルのタイプと、パッケージのブロッキングオプションによってブロッキングするかしないかが決まる。
    - カーソルタイプ（FOR UPDATE, FOR READ ONLY/FOR FETCHONLY，指定なし）3通り
    - プリコンパイル、BINDのBLOCKINGオプション(ALL, NO, UNAMBIG)3通り

BIND/PREP オプション ＼ カーソルタイプ	読み取り専用カーソルカーソル（FOR READ ONLY/FOR FECTH ONLY)	あいまい	更新意図 (FOR UPDATE OF)
ALL	ブロッキングされる	ブロッキングされる	ブロッキングされない
UNANBIGUOUS	ブロッキングされる	ブロッキングされない	ブロッキングされない
NO	ブロッキングされない	ブロッキングされない	ブロッキングされない

1. 8. カーソルとロック - FOR UPDATEカーソル使用時の動き

- 処理例: (ISOLATION=CS、BLOCKING＝ALL の場合)
  - FETCH 時に FETCH 対象行に、U ロックが取得される
    - FETCH 後に更新処理があることを想定して、行に排他制御を行う。
  - OPEN カーソル時には行は読み込まれず、FETCH 時に 1 行づつ読み込まれる
    - ブロッキング(FOR UPDATE なしの場合行われる)が行われない。
    - FETCH 時に、照会行に U ロックを取得、カーソルが次の行に移ると、前の行へのロックは外される
  - ISOLATION=CS の場合、カーソルを CLOSE すると U ロックは外される





ISOLATION=CS、BLOCKING＝ALL の場合

ID	NAME	DEPT	SARALY
10	Sanders	20	3000
20	Pernal	20	1000
30	Smith	10	2000
40	Yamaguchi	20	4000

2. 分離レベルとロック

内容

- 2. 分離レベルとロック
  - 2. 1. 分離レベルとは
  - 2. 2. 分離レベルとロック
  - 2. 3. 分離レベルの設定と確認

2．1.「分離レベル」とは

- 分離レベルとは、トランザクションの中で、どの程度厳密にデータの一貫性を保つかを決定するレベル。
- ISOが定義する分離レベルには以下の4つがある。
- それぞれの分離レベルによって「未コミットデータの読み取り」・「反復不能読み取り」・「幻像読み取り」の事象発生を許可するか否かが決まる。（それぞれの事象については次頁で説明）

狭い

↑ ロックの範囲 ↓

広い

ISO分離レベル（DB2での対応分離レベル）	意味の要約	未コミットデータの読み取り	反復不能読み取り	幻像読み取り
Read Uncommitted（Uncommitted Read: UR）	コミットされていないデータも読めるため、一度参照したレコードが、無かったことになっても構わない。	ある	ある	ある
Read Committed（Cursor Stability: CS） DB2のデフォルト	コミット済みのレコードだけを参照したい。しかし、一度参照したレコードをもう一度参照した場合に、変更されても構わない	ない	ある	ある
Repeatable Read（Read Stability: RS）	参照したレコードは、更新されたくない。もう一度読み直したときに、同じレコードが他から更新されず、同じ状態であって欲しい。	ない	ない	ある
Serializable（ Repeatable Read : RR）	上に加えて、もう一度読み直したときに、追加のレコードが増えるのも許さない。	ない	ない	ない

2．1.「分離レベル」とは
- 未コミットデータへの読取り

- コミット前のデータを読み取ってしまい。更新データがロールバックされた場合には、読取りデータは実際とは異なるデータとなる

URでのみ発生
CS/RS/RRでは発生しない

- (例)

Reservations

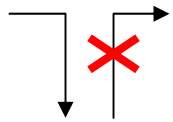
Flight	Seat	Status
512	7C	AVAILABLE
512	7B	AVAILABLE
⋮	⋮	⋮

予約していた席をキャンセルしようとして、やっぱり止めた。

APP1

(1)
Update Reservations
Set Status = ‘AVAILABLE'
Where Flight = 512
and Seat = '7C'
and Status = ‘RESERVED’

(3)
Rollback



512	7C	AVAILABLE

APP2

(2)
Select seat
From Reservations
Where Status=‘AVAILABLE’

空席だと思ったが、実は予約済みだった。（確定していない情報だった。）

(4)
正しくない結果セット

2．1.「分離レベル」とは
- 解説：未コミットデータの読取り

- 未コミットデータへのアクセスは次の状況で発生します。
  1. トランザクション 1 が、レコードを更新する
  2. トランザクション 2 が、非コミット読み取りを行う

     述部の条件に合致しないために、ステップ 1 の更新行は SELECT されない
  3. トランザクション 1 が、ロールバックを行う
  4. トランザクション 2 は結果として、正しくない結果を得ることになる

     （ロールバック後のデータは、検索条件に合致していたかもしれない）
- UNCOMMITTED READが望ましい場合もあります。

例：

- （例えば予約システムで）、大まかな空席情報概算を、すばやく検索 し、予約確定は後で行ってもかまわない
- 今の状況をとりあえず確認したい

2. 1.「分離レベル」とは
- 反復不可能読取り (Non Repeatable Reads)

- 一つの作業単位内で、同じ SELECT 文が実行された場合に、1回目の参照結果行について同じ照会結果が保証されない

UR/CSで発生
RS/RRでは発生しない

- (例)

Reservations

Flight	Seat	Status
512	7C	RESERVED
512	7B	AVAILABLE

APP2

APP1

(1)
Select seat
From Reservations
Where Status =‘AVAILABLE’
and Flight = 512
(3)
(1) の SELECT 文を再度実行すると、Seatが 7C の行は戻されない

512	7C	AVAILABLE
512	7B	AVAILABLE

未確定の情報は読まないが、一度目の参照結果が変更されることがある。

512	7B	AVAILABLE

(2)
Update Reservations
Set Status = ‘RESERVED'
Where Flight = 512
and Seat = '7C'
and Status = ‘AVAILABLE'

2．1.「分離レベル」とは
– 解説：　反復不可能読取り (Non Repeatable Reads)

- 反復不可能読取りは、「同一作業単位内においては、SQL の再実行が一度以上行われた場合、毎回正確に同一な結果セットを返すこと」を保障しません。
- 反復可能読取りでは、その結果セットを保証するために、SQL 文の結果が変わってしまう INSERT / UPDATE / DELETE が許されません。
- この例では、1回目の参照と2回目の参照の間に、512 便の席 7C が予約された場合、1回目と2回目の空席情報の参照結果は異なります。

2. 1.「分離レベル」とは
- 幻像読取り (Phantom Reads)

- 同一作業単位内で、同じ SELECT 文を実行した場合に、他の更新処理により条件に合致するようになったデータまで照会結果に入る

UR/CS/RSで発生
RRでは発生しない

- (例)

Reservations

Flight	Seat	Status
512	7A	AVAILABLE
512	7B	AVAILABLE
⋮	⋮	⋮

プログラム1

(1)
Select seat
From Reservations
Where Status='AVAILABLE'

(3)
(1) の SELECT 文を再度実行すると、7Aはアサイン可能

プログラム2

(2)
Update Reservations
Set Status = 'AVAILABLE'
Where Flight = 512
and Seat = '7A'
and Status = 'RESERVED'

512	7B	AVAILABLE

一度目の参照結果より、条件に合致するレコードが増えている。

512	7A	AVAILABLE
512	7B	AVAILABLE

2. 1.「分離レベル」とは
- 解説: 幻像読取り(Phantom Reads)

- 幻像読取りとは、反復不可能読取りの一種で、同一作業単位内において二回目の照会の際に、一回目と同じデータ + 追加のデータを戻します。
- 例:
  - トランザクション 1 が、空席を照会
  - トランザクション 2 が、別レコードの予約をキャンセル
  - 再び、トランザクション 1 のステートメントを実行すると、追加のレコードも返ってくる
- *アプリケーションの要件によっては、こういう結果が望ましい場合もあります。*

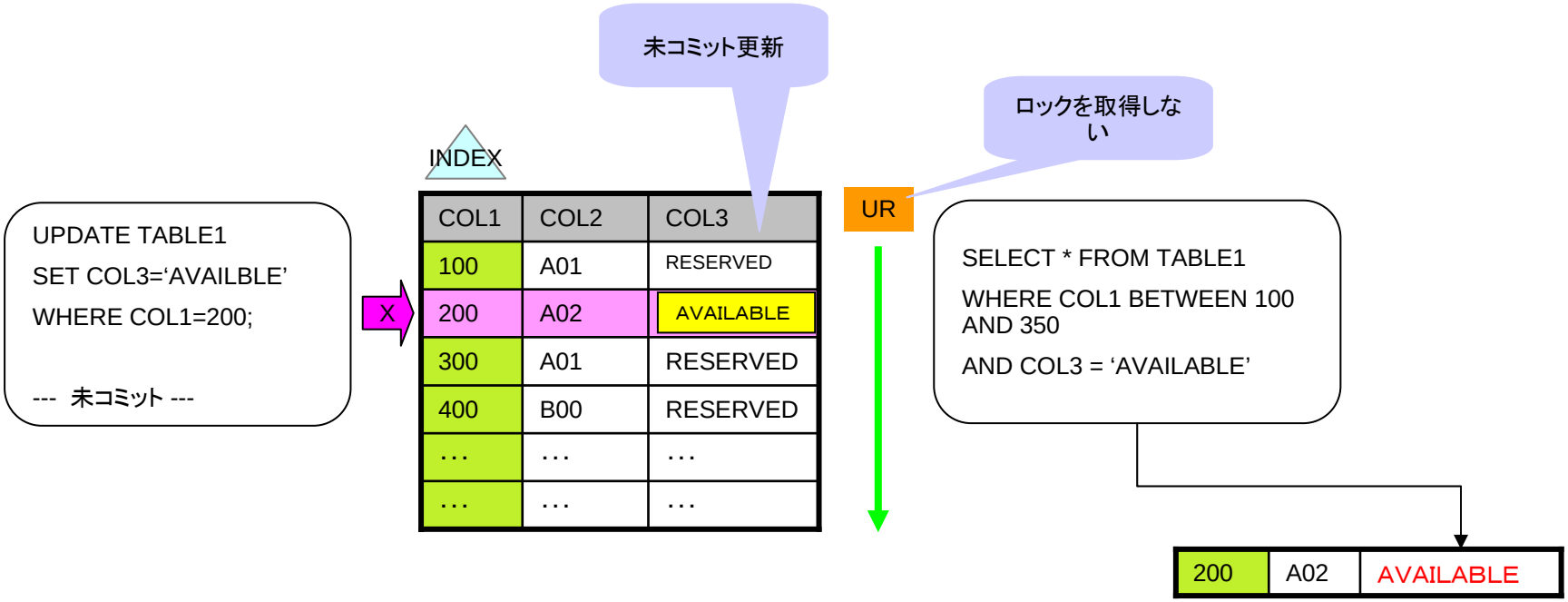
2．1.「分離レベル」とは
分離レベルとロック

- 利用するロックのモードやロックを保持する期間を変えて、それぞれの分離レベルを実現する。

分離レベル		参照時に取得するロック
RR (Repeatable Read)　反復可能読取り * Serializable		走査した行に対して、S ロックを取得 操作行の索引キーの次の High キー (索引順の次の値) を持つ行にも S ロックを取得
RS (Read Stability)　読取り固定 *Repeatable Read		結果行に対して、NS ロックを取得
CS (Cursor Stability) カーソル固定 (省略時値) *Read Committed	CURRENTLY COMMITTED無効: V9.5まで	FETCH した (読み込んだ) 行に対して、NS ロックを取得
	CURRENTLY COMMITTED有効: V9.7デフォルト	行へのロックは取得しない　（*例外あり）
UR (Uncommitted Read)　未コミット読取り *Read Uncommitted		行へのロックは取得しない

2. 2. 分離レベルとロック - UR

- UR (Uncommitted Read)：未コミット読取り
  - 行へのロックは取得しない
  - コミットが完了していない更新データを読み込んでしまう。



2．2．分離レベルとロック - CS

- CS (Cursor Stability) :カーソル固定
  - コミットされたデータのみを読む (デフォルトの分離レベル）
  - V9.7より、未コミット更新中のデータを参照する時の動作が選択できる。

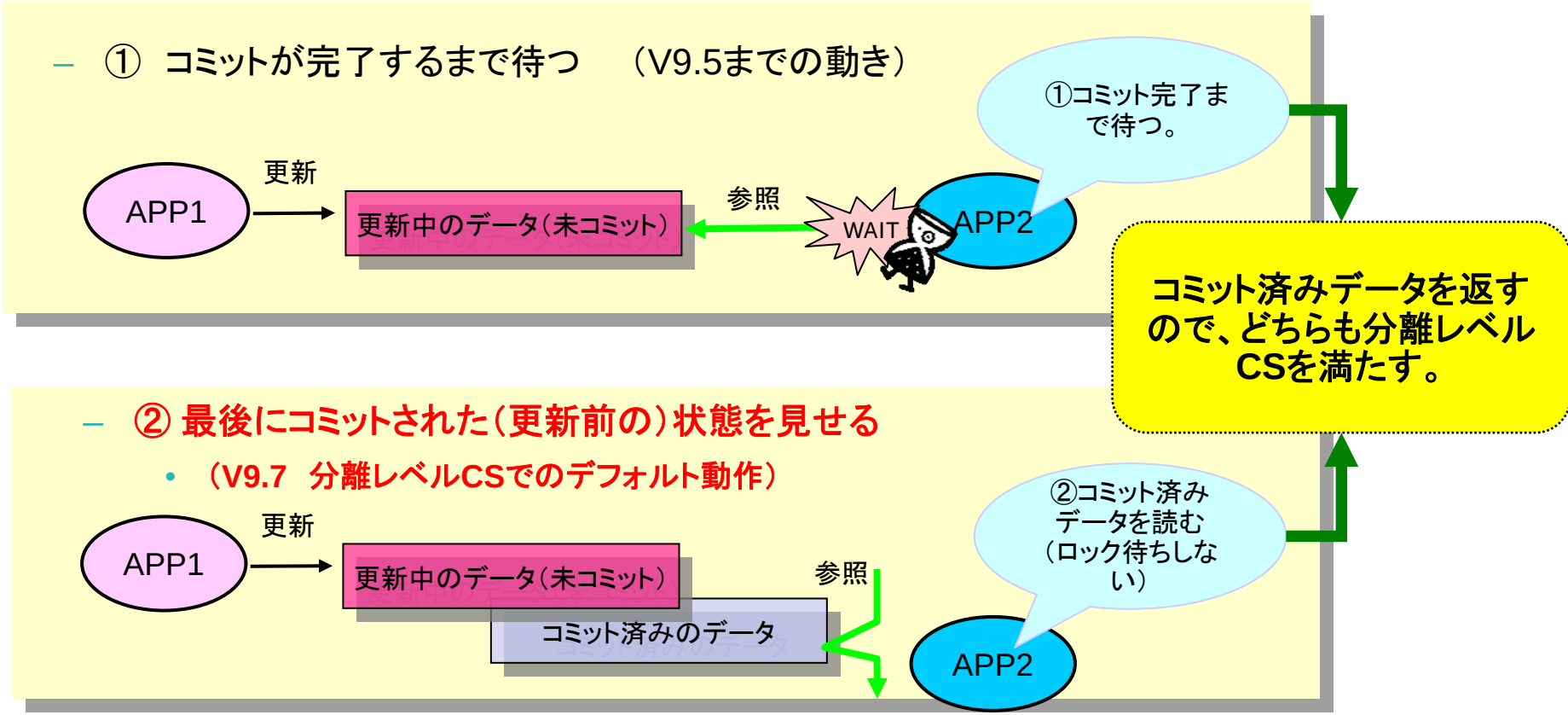
- ① 未コミット更新中データがコミットされるまで待つ。（ロック待ちが発生する）
- V9.5までの動きまたは、V9.7でCUR_COMMIT=DISABLEDの時

V9.7
NEW

- ② 未コミット更新中のデータについては、そのときにコミット済みのデータを返す（ロック待ちが発生しない）。
  - V9.7 CUR_COMMITTED=ON の場合 （V9.7デフォルト）

分離レベルCSでロック待ちしなくても良いのうか？

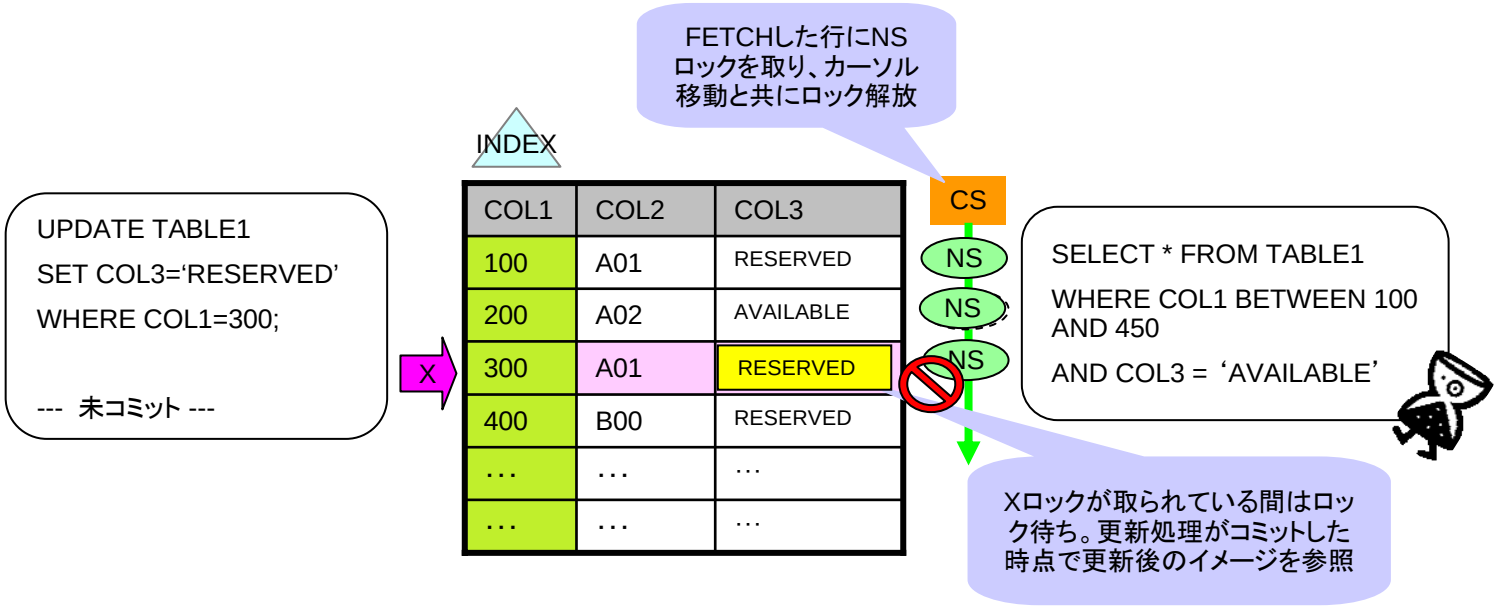
- 分離レベルCSが満たすべき要件とは・・・
  - **コミット済みのレコードだけを参照したい。**しかし、一度参照したレコードをもう一度参照した場合に、そのレコードの内容が変更されても構わない



2. 2. 分離レベルとロック - CS (CUR_COMMIT無効)

～V9.5

- CS (Cursor Stability)　CUR_COMMIT無効 （～V9.5）
  - コミットされたデータのみを読む
    - 読み終わった行に対しては新たに別の更新がかかるかもしれない。(もう一度読んだときには結果が変っているかもしれない)
  - FETCH した (読み込んだ) 行に対して、NS ロックを取得。カーソルが移動した時点でロックを解放(※)
  - 未コミット更新中データを参照しようとした場合、更新処理がコミットされるまで待つ。
    - ロック待ちが発生する （V9.5までの動きまたは、V9.7でCUR_COMMIT=DISABLEDの時の動き）



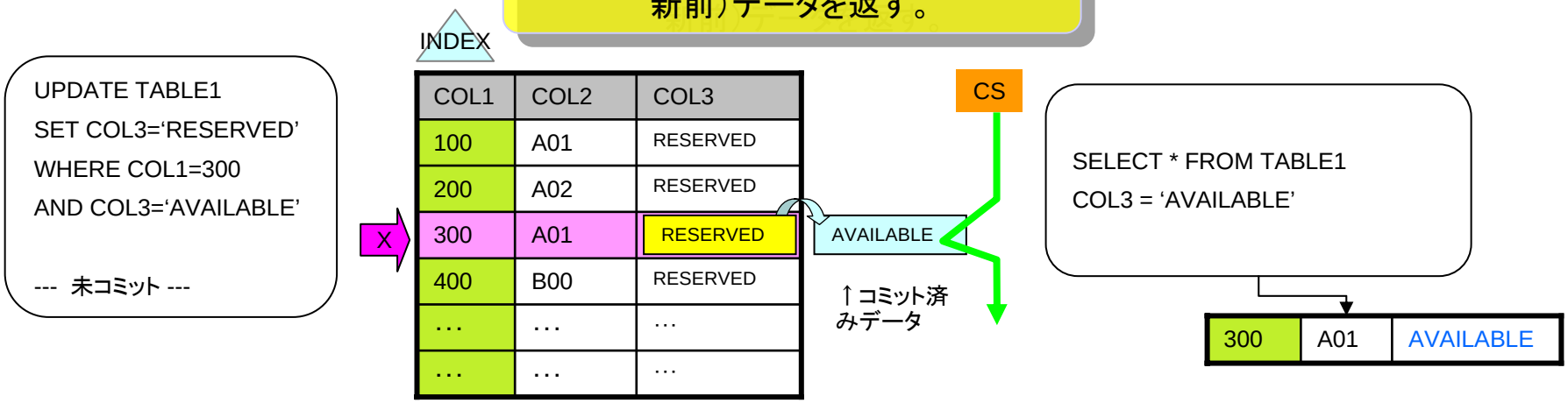
V9.7 NEW

2. 2. 分離レベルとロック - CS (CUR_COMMIT有効 1/3)

- CS (Cursor Stability) CUR_COMMIT有効 (V9.7 デフォルト)
  - コミットされたデータのみを読む
    - 読み終わった行に対しては新たに別の更新がかかるかもしれない。(もう一度読んだときには結果が変っているかもしれない)
  - FETCHする行にロックを取らない
  - 未コミット更新中データを参照しようとした場合、最後に(参照しようとしたそのときに)コミットされていたデータを参照する(※)
    - ロック待ちが発生しない
    - V9.7でCUR_COMMIT=ON (デフォルト)時の動き
    - 最後にコミットされたデータはログバッファーまたはログファイルから読む

--- ※ MEMO ---
このような動きを「CURRENTLY COMMITED」と呼んでいます。

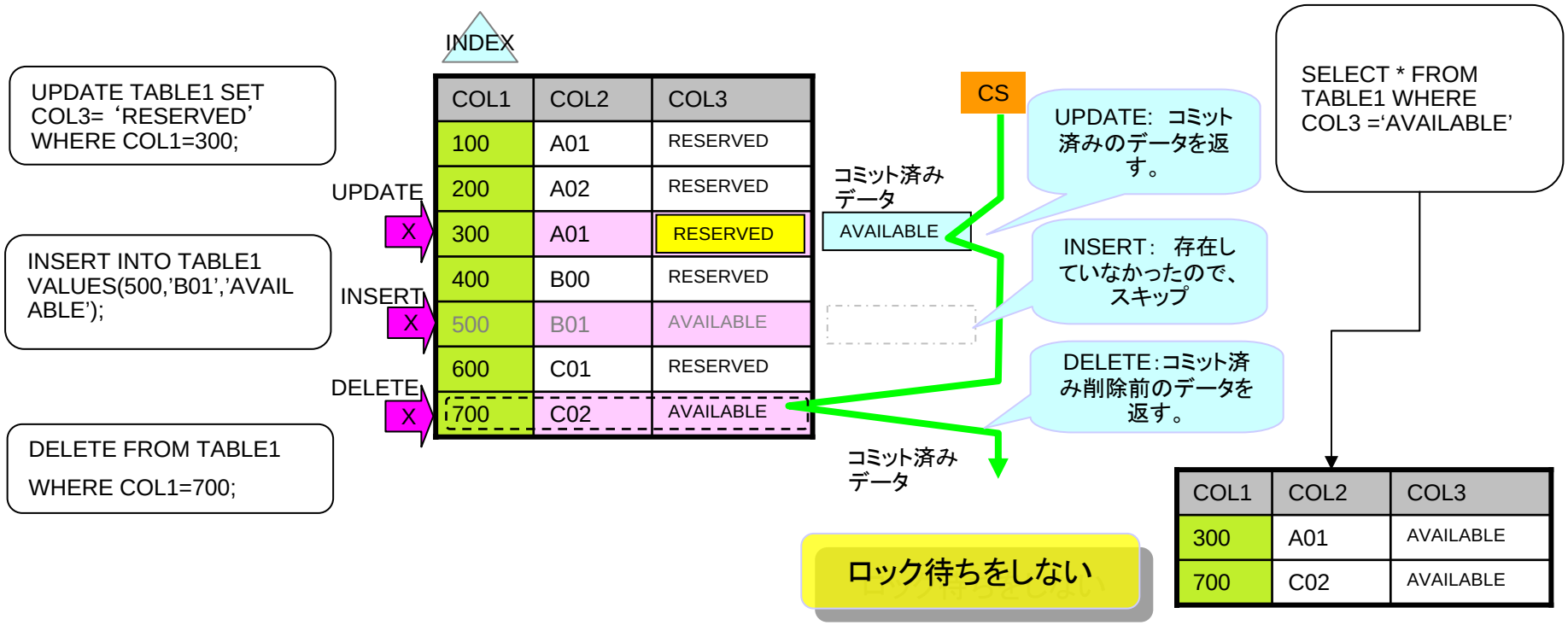
ロック待ちは発生せず、コミット済み(更新前)データを返す。



V9.7 NEW

2. 2. 分離レベルとロック - CS (CUR_COMMIT有効 2/3)

- CS (Cursor Stability) CUR_COMMIT有効
  - 未コミット更新に対する参照処理の動き
    - 未コミットUPDATE: ロック待ちはせず、最後にコミットされたデータが返される
    - 未コミットINSERT: ロック待ちはせず、INSERT行をスキップする
    - 未コミットDELETE: ロック待ちはせず、DELETE前のデータが返される

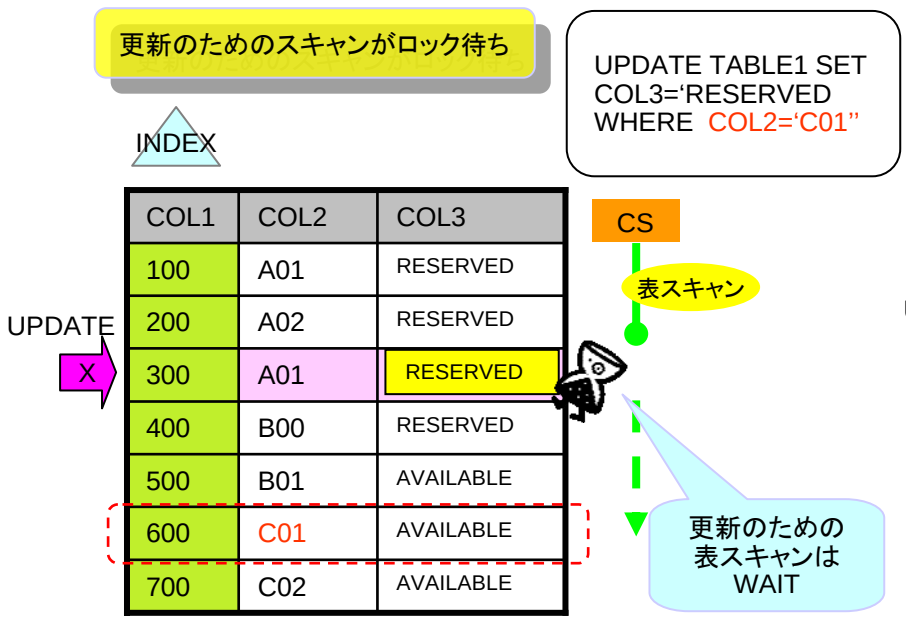


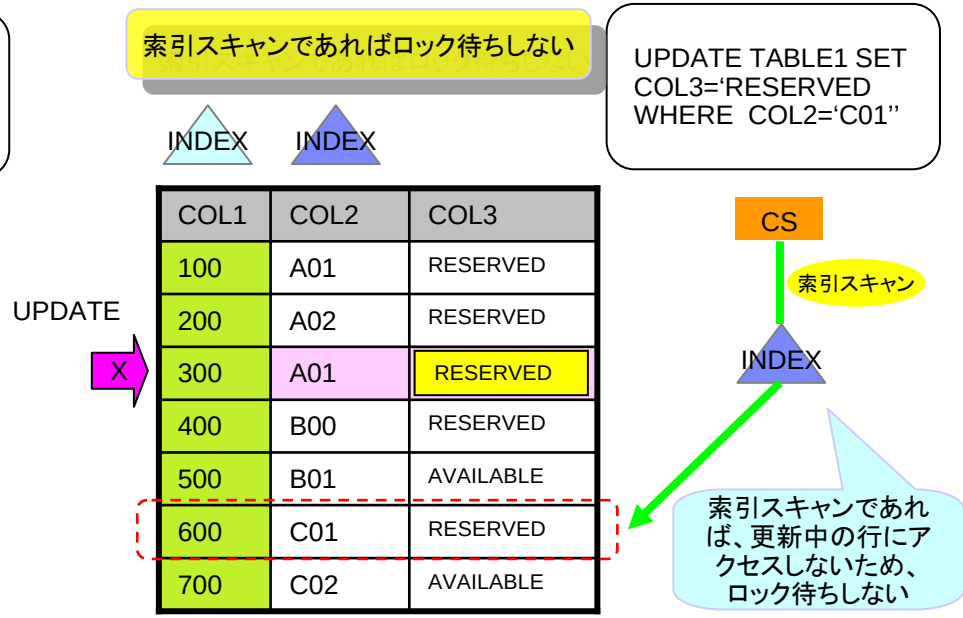
V9.7 NEW

2. 2. 分離レベルとロック - CS (CUR_COMMIT有効　3/3)

- CS (Cursor Stability)　CUR_COMMIT有効
  - CURRENTLY COMMITEDが有効にならないケース
    - FOR UPDATEカーソルではロック待ちとなる
      - UロックとXロックは互換がないのでもちろんロック待ちとなる
    - UPDATE/DELETEに伴うスキャンでは、未コミット更新に対してロック待ちとなる
      - 更新対象には該当しないが、表スキャンによりアクセスされる場合など （下図参照）
    - 「未コミットInsert行のスキップ」は更新のためのスキャンでも実施される。

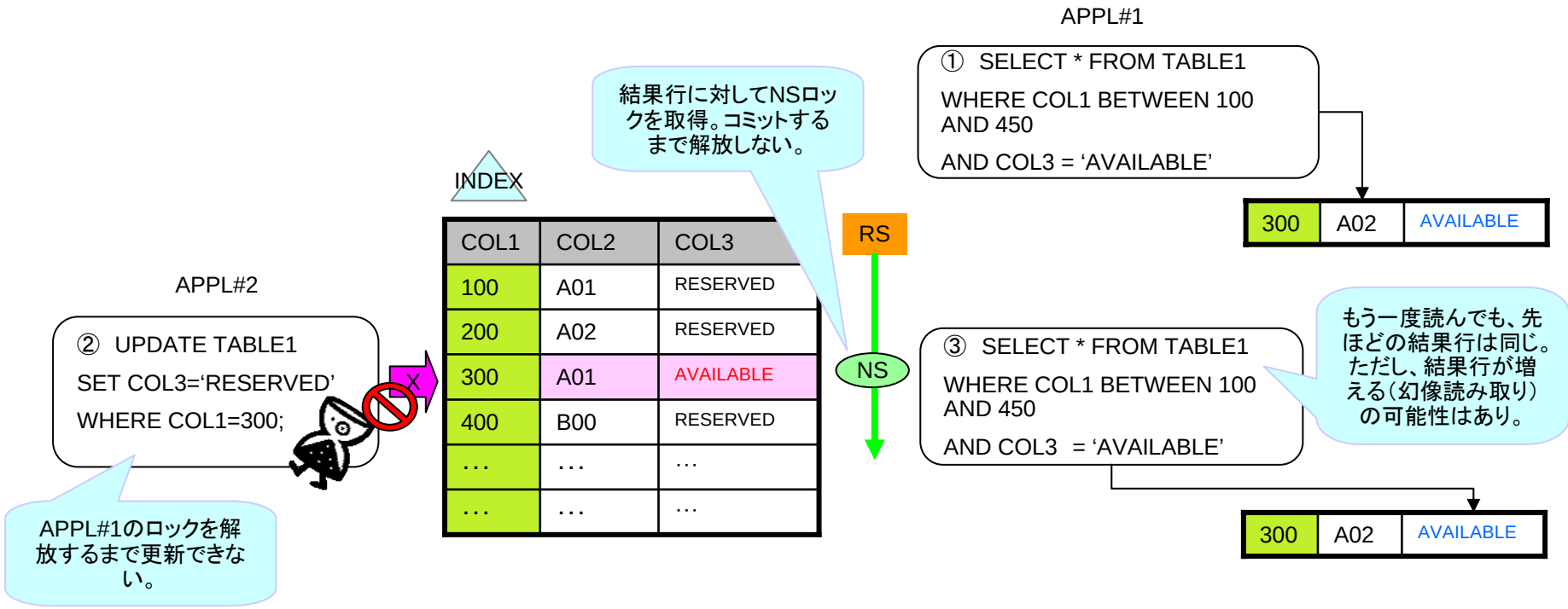
--- ※ MEMO ---
CURRENTLYCOMMITED が有効にならないそのほか のケースについては後述





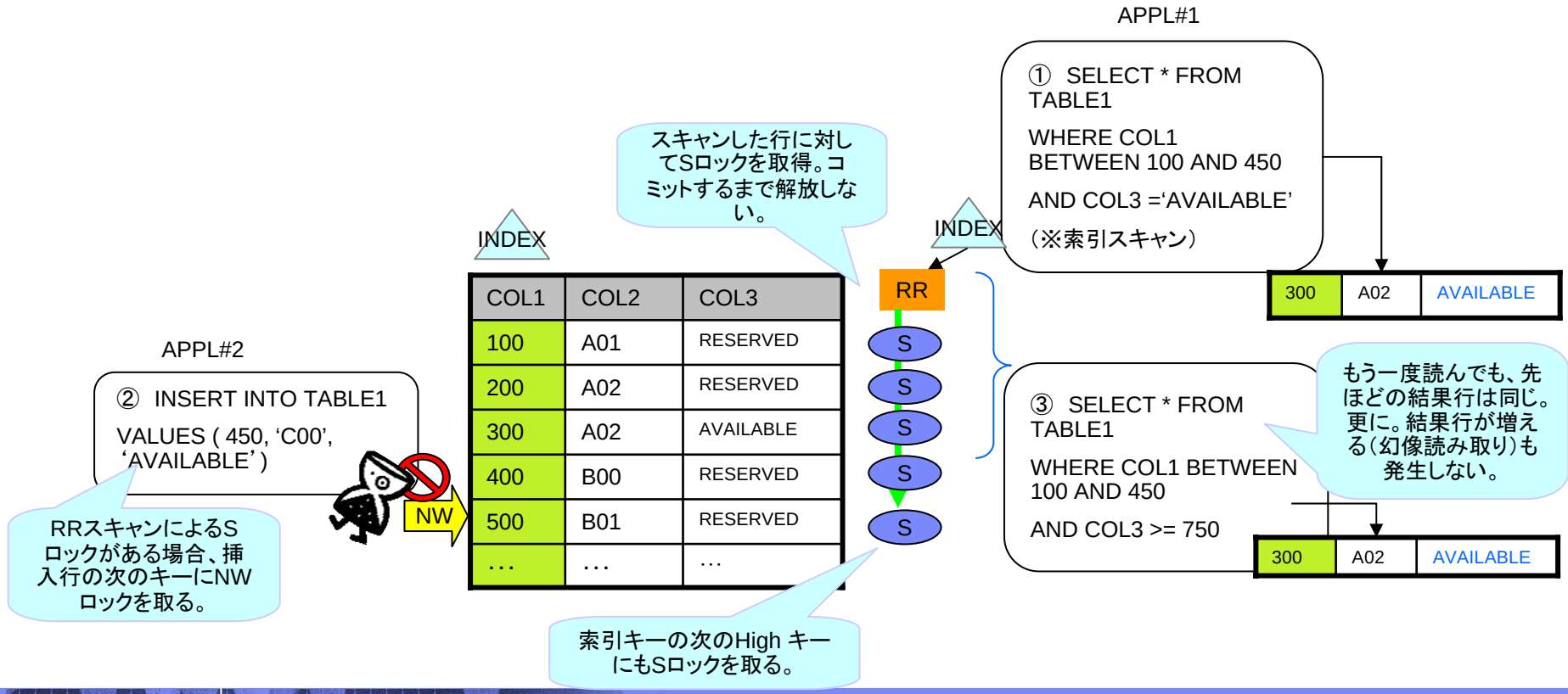
2. 2. 分離レベルとロック - RS

- RS (Read Stability) ：読取り固定
  - 参照したレコードは、更新されたくない。もう一度読み直したときに、同じレコードが他から更新されず、同じ状態であって欲しい。
    - 結果行に対して、NS ロックを取得し、コミットされるまで保持する。



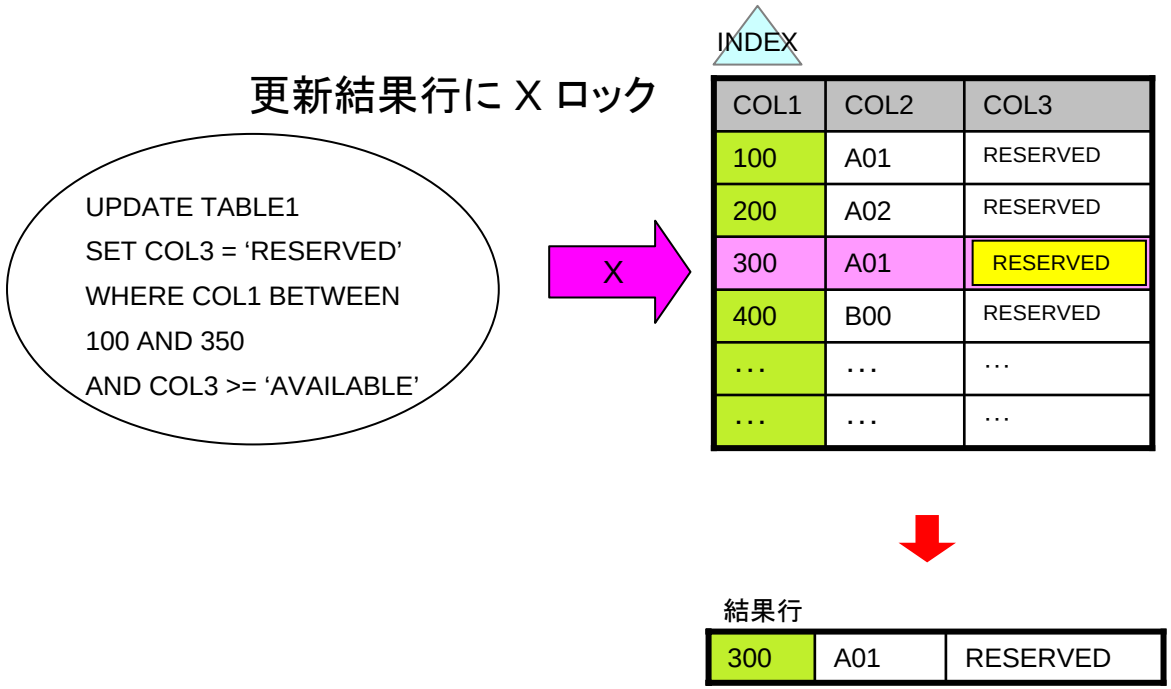
2. 2. 分離レベルとロック - RR

- RR (Repeatable Read)　: 反復可能読取り
  - もう一度読み直したときに、追加のレコードが増えるのも許さない。
    - 走査した行のすべてに対して、S ロックを取得、保持
    - 更に操作行の索引キーの次の High キー (索引順の次の値) を持つ行にも S ロックを取得



2. 2. 分離レベルとロック – 更新時のロック

- 分離レベルの違いによる、更新処理時のロック取得の違いはない
  - 更新処理時には、分離 レベルの設定値にかかわらず、必ず、更新行に排他ロック (X ロック) が取得され、COMMIT まで保持される



2．2．分離レベルとロック - 分離レベルの比較

- 照会処理時のロック取得
  - 更新された行については、作業単位完了 (COMMIT / ROLLBACK) までロックを取得する
  - 照会行については、ISOLATIONによりロックの取得範囲と保持期間が異なる

ISOLATION	RR	RS	CS (CUR_COMITT= DISABLED)	CS (CUR_COMMIT= ON)	UR
ロックの範囲	作業単位内で走査した全ての行、および索引キーの次の行	作業単位内で参照した結果行	カーソルがおかれた行のみ	ロックを取得しない	ロックを取得しない
未コミットデータへのアクセス	×	×	×	○（参照時点でコミット済み情報）	○
反復不可能読取り	×	×	○	○	○
幻像読取り	×	○	○	○	○
照会行へのロック (Read Lock) の保持期間	作業単位内	作業単位内	カーソルが次の行に進むまで	ロックは保持しない	ロックは保持しない
他のアプリケーションへの影響	作業単位内で走査した全ての行への更新処理が不可能	作業単位内で参照した結果行への更新処理が不可能	カーソルがおかれた行への更新処理は不可能	参照時のロックは保持しない	参照時ロックは保持しない
長所	同一作業単位内で実行された SELECT 文には必ず同じ結果行が戻される	高い同時稼動性と結果行の保持	コミット行のみを照会する場合	コミット行のみを紹介し、未コミット更新に対してもコミット済みデータ参照可能	ロックの負荷がない同時稼動性が高い
短所	多くの行ロックを取得し、LOCKLIST の不足から、ロック・エスカレーションが発生する可能性あり	幻像読取りが発生する	反復不可能読取りと幻像読取りが発生する	反復不可能読取りと幻像読取りが発生すること	反復不可能読取りと幻像読取りに加えて見コミット読み取りが発生する

（参考）2. 2. 分離レベルとロック - ISOLATION=UR の考慮点

- 照会のみと特定できないカーソルの場合には、URを指定していても、CSの動作になる
- 以下の場合には、URとして動作する
  - SELECT 文にソート処理がある場合には、照会のみのカーソルと判断される
  - FOR READ ONLY または FOR FETCH ONLY オプションを指定し、明示的に照会のみの SELECT 文であることを宣言する
  - BIND または PREP コマンドで、BLOCKING オプションの指定を ALL に設定する
    - 省略時値は UNAMBIG
    - 静的 SQL の場合、BLOCKING＝ALL を指定しても、CS の動作になることが確認されている

< ISOLATION=UR で、PREP または BIND した場合>

		PREP = BLOCKING UNAMBIG	PREP または BIND = BLOCKING ALL
DB2 9	静的 SQL	CS の動作になる	CS の動作になる
	動的 SQL	CS の動作になる	UR の動作になる
DB2 UDB V8	静的 SQL	CS の動作になる	CS の動作になる
	動的 SQL	CS の動作になる	UR の動作になる
DB2 UDB V7		UR の動作になる	UR の動作になる

2．3．分離レベルの設定と確認 – 設定方法

- 分離レベルはSQLステートメントに直接設定したり、バインド時にパッケージに指定したりすることができる。
  - SQL ステートメントレベルでの分離レベル指定（WITH [UR/CS/RS/RR] 分離レベル節）
    - 例） SELECT col1 FROM table1 WITH UR
  - BIND時の分離レベル指定
    - 例） BIND bindfile.bnd ISOLATION UR

- 静的SQLの場合 ： 以下の優先度で設定される
  - ステートメントに設定されたisolation-clause(分離レベル節)
  - データベースへのパッケージのバインド時に、そのパッケージに指定された分離レベル

- 動的SQLの場合 ： 以下の優先度で設定される
  - ステートメントに設定されたisolation-clause(分離レベル節)
  - SET CURRENT ISOLATION ステートメントが現行セッション内で発行されている場合は、CURRENT ISOLATION 特殊レジスターの値が使用される。
  - データベースへのパッケージのバインド時にそのパッケージ用に指定された分離レベルが使用される。

2. 3. 分離レベルの設定と確認 – 設定方法

- その他の設定方法
  - CLI、ODBC アプリケーション
    - db2cli.ini ファイルに設定する方法
      - TXNISOLATION ＝ 1 2 8 16 32
      - （ 1:UR、2:CS、8:RR、16:RS、32:NC ＊NCはAS/400版のみ ）
    - アプリケーションで設定する方法
      - SQL_SetConnectAttr() の SQL_ATTR_TXN_ISOLATION オプション または SQL_SetStmtAttr() の SQL_ATTR_TXN_ISOLATION オプション
        - UR:SQL_TXN_READ_UNCOMMITED
        - CS: SQL_TXN_READ_COMMITED
        - RS: SQL_TXN_REPEATABLE_READ
        - RR:SQL_TXN_SERIALIZABLE
        - NC:SQL_TXN_NOCOMMIT
  - JDBC アプリケーション
    - Connection.TRANSACTION_READ_UNCOMMITED = Uncommitted Read（非コミット読取り）
    - Connection.TRANSACTION_READ_COMMITTED = Cursor Stability（カーソル固定）
    - Connection.TRANSACTION_REPEATABLE_READ = Read Stability（読み取り固定）
    - Connection.TRANSACTION_SERIALIZABLE = Repeatable Read（反復可能読取り）
    - Connection.TRANSACTION_NONE = DB2 UDBでは、サポートされておらず、指定するとエラー
  - SET CURRENT ISOLATION ステートメント
    - 現行セッション内で発行される動的SQLの分離レベルを指定する。
      - SET CURRENT ISOLATION [UR/CS/RS/RR/RESET]
  - CLP (コマンド行プロセッサー)
    - CHANGE ISOLATION TO [CS | RR | RS | UR | NC]

2．3．分離レベルの設定と確認 – 確認方法

- 埋め込み SQLでの分離レベル確認方法
  - PREP / BIND コマンドの ISOLATION オプションによって指定。(BIND 時に指定しない場合には、PREP 時の設定が有効となる。)
    - 確認方法① プリコンパイルによって作成されたバインドファイルに指定された分離レベルを確認する
      - db2cfd -b オプションにより、バインド・ファイルのヘッダー部分が表示可能

```
$ db2 prep  tbsel.sqc  bindfile isolation RR
$ db2bfd -b tblsel.bnd
    .....
Name                Value
------------------  -----
Isolation Level     Repeatable Read
Creator             "BASHI197"
App Name            "TBSEL   "
```

PREP時にISOLATIONを指定できる。

バインドファイルの中で指定された分離レベルを確認

    - 確認方法② 実際にバインドされたパッケージに指定された分離レベルを確認する
      - SYSCAT.PACKAGES 表の ISOLATION 列を参照する
        (例) PKGNAME にはパッケージ名、PKGSCHEMA にはパッケージのスキーマ名を英大文字で指定

```
$ db2 bind  tbsel.bnd  isolation UR
$ db2 "select pkgname, isolation from syscat.packages where pkgname like 'TB%'"
PKGNAME          ISOLATION
---------------- ---------
TBSEL            UR
```

バインドされたパッケージの分離レベルを確認

2. 3. 分離レベルの設定と確認 – アクセス・パスでのロックのモード確認

- アクセスパスでのロック・モードの確認方法
  - 実際のアクセスパスでの分離レベル、取得されるロック・モードをEXPLAIN出力から確認する。

1. db2exfmt コマンド

2. db2expln コマンド
   - 静的 SQL (パッケージ) : db2expln ツール
     - 構文:db2expln -d データベース名 -c 作成者ID -p パッケージ名 -o 出力ファイル名
     - ヘルプの画面出力: db2expln -h
     - db2expln のみを打鍵して実行した場合には、プロンプトが画面表示され、対話式にて実行が可能

2. 3. 分離レベルの設定と確認 – アクセス・パスでのロックのモード確認

- db2exfmt によるアクセス・パスでのロックモードの出力

```
3) FETCH : (Fetch)
        Cumulative Total Cost:                  12.9107
        Cumulative CPU Cost:                    85702
        Cumulative I/O Cost:                    1
        Cumulative Re-Total Cost:    0.0263859
        Cumulative Re-CPU Cost:      37241
        Cumulative Re-I/O Cost:      0
        Cumulative First Row Cost:   12.9086
        Estimated Bufferpool Buffers:           2

        Arguments:
        ---------
        MAXPAGES: (Maximum pages for prefetch)
                  ALL
        MAXPAGES: (Maximum pages for prefetch)
                  ALL
        PREFETCH: (Type of Prefetch)
                  NONE
        ROWLOCK : (Row Lock intent)
                  SHARE
        TABLOCK : (Table Lock intent)
                  INTENT EXCLUSIVE
        TBISOLVL: (Table access Isolation Level)
                  REPEATABLE READ
```

（MEMO: db2exfmtによるアクセスパス確認方法）

① EXPLAIN 表を事前に作成 (sqllib/misc/EXPLAIN.DDL実行)

(例) db2 -tvf EXPLAIN.DDL

② SET CURRENT EXPLAIN MODE EXPLAIN

この後実行される SQL は、EXPLAIN 表に情報収集されるだけで、実際には実行されない

③ SQL の実行

④ db2exfmt コマンドで、EXPLAIN 表から EXPLAIN 情報を収集し、フォーマットする

2. 3. 分離レベルの設定と確認 – アクセス・パスでのロックのモード確認

- db2expln ツールによるアクセス・パスのロックのモードの確認

```
Package Name = "xxxxxx"."DYNEXPLN"  Version = ""
               Prep Date = 2003/08/26
               Prep Time = 17:34:16
               Bind Timestamp = 2003-08-26-17.34.16.150000
               Isolation Level          = Cursor Stability
               Blocking                 = Block Unambiguous Cursors
               Query Optimization Class = 5
               Partition Parallel       = No
               Intra-Partition Parallel = No
               SQL Path                 = "SYSIBM", "SYSFUN", "SYSPROC", "E07853"
---------------------- SECTION -------------------------------------
Section = 1
SQL Statement:
  DECLARE C1 CURSOR
  FOR
     select col1, col2
     from test03
     where col1=3

Estimated Cost        = 25.075222
Estimated Cardinality = 4.800000
Access Table Name = E07853.TEST03  ID = 2,31
|  #Columns = 2
|  Index Scan:  Name = E07853.TEST03IX  ID = 1
|  |  Regular Index (Not Clustered)
|  |  Index Columns:
|  |  |  1: COL1 (Ascending)
|  |  #Key Columns = 1
|  |  |  Start Key: Inclusive Value
|  |  |  |  1: 3
|  |  |  Stop Key: Inclusive Value
|  |  |  |  1: 3
|  |  Data Prefetch: Eligible 0
|  |  Index Prefetch: None
|  Lock Intents
|  |  Table: Intent Share
|  |  Row  : Next Key Share
|  Return Data to Application
|  |  #Columns = 2
Return Data Completion
End of section
```

BIND 時に指定された ISOLATION

ロック・モード

3. ロックのモニタリング

内容

- ３．ロックのモニタリング
  - ３．１．今取得されているロックを確認するには？
  - ３．２．データベース全体のロック情報を確認するには？
  - ３．３．デッドロック発生時の詳細情報を確認するには？
  - ３．４．ロックタイムアウト発生時の詳細情報を確認するには？
  - ３．５．障害ログ（db2diag.log)に含まれる情報
  - ３．６．V9.7 新機能： ロックイベントモニターの紹介

3．1．今取得されているロックを確認するには？

- ロック・スナップショット・モニターまたはdb2pd –locksオプションを実施
  - コマンド実行時点のスナップショット的な情報を取得
    - アプリケーションが、現在何に対して、どんなモードでロックを取得しているかを確認
    - ロック・ウェイトの状況を確認
  - ① ロック・スナップショット・モニター
    - テキスト（レポート）形式
    - モニタースイッチ（LOCK）をONにする
  - ② db2pd –locksオプション
    - テキスト（タブ区切り）形式（加工しやすい）
    - モニタースイッチONにする必要なし

3．1．今取得されているロックを確認するには？ ロック・スナップショット・モニター (1/2)

- ロック・スナップショット・モニター （LOCK WAITの例）
  - データベース全体で、現在取得されているロックやロック待機の状況を表示する
    - db2 GET SNAPSHOT FOR LOCKS ON sample

```
            Database Lock Snapshot

Database name                              = SAMPLE
Database path                              = /home/taka9/data/taka9/NODE0000/SQL00001/
Input database alias                       = SAMPLE
Locks held                                 = 6
Applications currently connected           = 3
Agents currently waiting on locks          = 1
Snapshot timestamp                         = 2007-07-17 02:38:08.647676

Application handle                         = 18
Application ID                             = *LOCAL.taka9.070716162013
Sequence number                            = 00006
Application name                           = db2bp
CONNECT Authorization ID                   = TAKA9
Application status                         = Lock-wait
Status change time                         = Not Collected
Application code page                      = 943
Locks held                                 = 3
Total wait time (ms)                       = 42685

  ID of agent holding lock                 = 17
  Application ID holding lock              = *LOCAL.taka9.070716162006
  Lock name                                = 0x0003000500000000280000452
  Lock attributes                          = 0x00000000
  Release flags                            = 0x00000001
  Lock object type                         = Row
  Lock mode                                = Exclusive Lock (X)
  Lock mode requested                      = Next Key Share (NS)
  Name of tablespace holding lock          = IBMDB2SAMPLEREL
  Schema of table holding lock             = TAKA9
  Name of table holding lock               = T1
  Data Partition Id of table holding lock  = 0
  Lock wait start timestamp                = 2007-07-17 02:37:25.962544
```

ロック待機しているアプリケーション

ロック待ち状態である

競合するロックを保持しているアプリケーションID

既に取られているロック
→ Xロックが行に取られている

このアプリケーションが取ろうとしているロック
→ Xロックが既に取られているのでNSロックが取れずロック待ちになっている。

3．1．今取得されているロックを確認するには？ ロック・スナップショット・モニター (2/2)

- 前ページのアプリケーションのロック待ちの原因となっているアプリケーション力
  - デッドロックの場合には、両方の「アプリケーション状況」が "ロック待機" となる

```
Application handle                         = 17
Application ID                             = *LOCAL.taka9.070716162006
Sequence number                            = 00004
Application name                           = db2bp
CONNECT Authorization ID                   = TAKA9
Application status                         = UOW Waiting
Status change time                         = Not Collected
Application code page                      = 943
Locks held                                 = 3
Total wait time (ms)                       = 0
List Of Locks
 Lock Name                      = 0x00030005000000000280000452
 Lock Attributes                = 0x00000000
 Release Flags                  = 0x40000000
 Lock Count                     = 1
 Hold Count                     = 0
 Lock Object Name               = 41943044
 Object Type                    = Row
 Tablespace Name                = IBMDB2SAMPLEREL
 Table Schema                   = TAKA9
 Table Name                     = T1
 Mode                           = X

 Lock Name                      = 0x00030005000000000000000054
 Lock Attributes                = 0x00000000
 Release Flags                  = 0x40000000
 Lock Count                     = 1
 Hold Count                     = 0
 Lock Object Name               = 5
 Object Type                    = Table
 Tablespace Name                = IBMDB2SAMPLEREL
 Table Schema                   = TAKA9
 Table Name                     = T1
 Mode                           = IX
```

3．1．今取得されているロックを確認するには？
db2pd -locks

- db2pd -locks を用いてロック取得状況や、ロック待ち状態を確認することができる。
  - モニタースイッチをONにすることなくロックウェイトがどのアプリケーションが保持しているロックによって引き起こされたか、解析することができます。
  - スナップショットではロックのモニタースイッチをONにする必要があります。

取得されているロックの情報を参照

(TableID 2 RecordID 0x5の行のロックを確認)

```
$ db2pd -db sample -locks show detail
Database Partition 0 -- Database SAMPLE -- Active -- Up 0 days 00:09:25
Locks:
Address        TranHdl Lockname                         Type    Mode St Owner Dur HldCnt Att    ReleaseFlg
0x402C45D8     2       00020002000000050000000052       Row     ..U  G  2     1   0      0x0000 0x00000001 TbspaceID 2 TableID 2 RecordID 0x5
0x402C4830     3       00020002000000050000000052       Row     ..U  W  2     1   0      0x0000 0x40000000 TbspaceID 2 TableID 2 RecordID 0x5
0x402C4498     2       00020002000000000000000054       Table   .IX  G  2     1   0      0x0000 0x00000001 TbspaceID 2 TableID 2
0x402C4808     3       00020002000000000000000054       Table   .IX  G  3     1   0      0x0000 0x40000000 TbspaceID 2 TableID 2
```

TranHdl列:トランザクションのハンドル

St:St ロックのステータス(G:保有している　W:Waitしている)

Owner:ロックを保持している所有者

※その他の項目はマニュアルを参照

TranHdl:3のトランザクションは、TranHdl:2のトランザクションが保持しているUロックによって、waitとなっている

3．1．今取得されているロックを確認するには？
db2pd -locks

- ロックを保持しているアプリケーションの調査
  - db2pd –transaction と付き合わせる

```
$ db2pd -db sample  -transactions

Database Partition 0 -- Database SAMPLE -- Active -- Up 0 days 00:36:25

Transactions:
Address    AppHandl [nod-index] TranHdl Locks State Tflag      Tflag2     FirstLsn       LastLsn        LogSpace SpaceReserved TID           AxRegCnt GX
0x4024D680 272      [000-00272]  2       3     READ  0x00000000 0x00000000 0x000000000000 0x000000000000 0        0             0x0000000000B2 1        0
0x4024E180 293      [000-00293]  3       3     READ  0x00000000 0x00000000 0x000000000000 0x000000000000 0        0             0x0000000000D6 1        0
```

AppHdl列：アプリケーションハンドル

※その他の項目はマニュアルを参照

AppHandl:272のアプリケーションによって、ロックは保持されていることが分かる(ウェイトしているのはAppHandl:293のアプリケーション)

3．1．今取得されているロックを確認するには？
db2pd -locks

- アプリケーションがロックウェイト時に実行していたステートメントの調査

```
$ db2pd -db sample -applications -dyn | more
Database Partition 0 -- Database SAMPLE -- Active -- Up 0 days 01:08:44
Applications:
Address    AppHandl [nod-index] NumAgents  CoorPid Status      C-AnchID C-StmtUID L-AnchID L-StmtUID  Appid
0x313297F0 293      [000-00293] 1          65942   Lock-wait   0        0         8        1         *LOCAL.minanml.060620064446
0x31328040 272      [000-00272] 1          49800   UOW-Waiting 0        0         2        1         *LOCAL.minanml.060620063607
```

```
Dynamic SQL Statements:
Address    AnchID StmtUID  NumEnv  NumVar  NumRef NumExe     Text
0x409DAB10 2      1        1       1       1      1          select * from t1 where id=2  for update with RS
0x409DB6A0 8      1        1       1       1      1          update t1 set name= ‘ddd’    where id=2
```

AnchID:アンカーID

L-AnchID:現時点から最後に実行されたアンカーID

※その他の項目はマニュアルを参照

AppHandl:293のアプリケーションは、update t1 set name=‘ddd’ where id=2
のステートメントを実行してロックウェイトとなったことが分かる

3．2．データベース全体のロック情報を確認するには？

- データベース・スナップショット
  - データベース単位に、デッドロック、ロック・エスカレーション、ロック待機などの発生回数を表示する
- アプリケーション・スナップショット
  - アプリケーション単位に、デッドロック、ロック・エスカレーション、ロック待機などの発生回数を表示する

3. 2. データベース全体のロック情報を確認するには？
データベース・スナップショット

- データベース・スナップショット
  - データベース単位に、デッドロック、ロック・エスカレーション、ロック待機などの発生回数を表示する
    - GET SNAPSHOT FOR DATABASE ON SAMPLE

```
 Database Snapshot

Database name                               = SAMPLE
Database path                               = /home/taka9/data/taka9/NODE0000/SQL00001/
Input database alias                        = SAMPLE
Database status                             = Active
Catalog database partition number           = 0
Catalog network node name                   =
Operating system running at database server= AIX 64BIT
Location of the database                    = Local
First database connect timestamp            = 2007-07-17 01:15:37.713890
Last reset timestamp                        =
Last backup timestamp                       =
Snapshot timestamp                          = 2007-07-17 02:22:22.483669

Number of automatic storage paths           = 1
Automatic storage path                      = /home/taka9/data

High water mark for connections             = 7
Application connects                        = 56
Secondary connects total                    = 4
Applications connected currently            = 3
Appls. executing in db manager currently    = 0
Agents associated with applications         = 5
Maximum agents associated with applications= 7
Maximum coordinating agents                 = 7

Locks held currently                        = 4
Lock waits                                  = 7
Time database waited on locks (ms)          = 23415
Lock list memory in use (Bytes)             = 7296
Deadlocks detected                          = 3
Lock escalations                            = 0
Exclusive lock escalations                  = 0
Agents currently waiting on locks           = 0
Lock Timeouts                               = 0
Number of indoubt transactions              = 0
```

---MEMO---

ロック待機に関する詳細情報を収集するためには、LOCK のモニタースイッチを ON に設定する。

* セッション単位で設定を変更する場合

db2 update monitor switches using LOCK on

* インスタンス単位で設定を変更する場合

db2 update db cfg for データベース名 using DFT_MON_LOCK on

3．2．データベース全体のロック情報を確認するには？
アプリケーション・スナップショット

- アプリケーション・スナップショット
  - アプリケーション単位に、デッドロックの発生回数、ロック・エスカレーションの発生回数、ロック待機の発生回数などを表示する

  - GET SNAPSHOT FOR APPLICATIONS ON SAMPLE

```
Application Snapshot

Application handle                         = 18
Application status                         = UOW Waiting
Status change time                         = Not Collected
Application code page                      = 943
Application country/region code            = 81
DUOW correlation token                     = *LOCAL.taka9.070716162013
Application name                           = db2bp
Application ID                             = *LOCAL.taka9.070716162013
Sequence number                            = 00006
TP Monitor client user ID                  =
TP Monitor client workstation name         =
TP Monitor client application name         =
TP Monitor client accounting string        =
      (省略)

Locks held by application                  = 0
Lock waits since connect                   = 4
Time application waited on locks (ms)      = 6240
Deadlocks detected                         = 1
Lock escalations                           = 0
Exclusive lock escalations                 = 0
Number of Lock Timeouts since connected    = 0
Total time UOW waited on locks (ms)        = Not Collected
```

3．2．データベース全体のロック情報を確認するには？
スナップショットの確認ポイント（ロック関連）

- ロック情報　確認ポイント
  - データの整合性確保のための重要な機能
  - しかし、特にOLTP環境では応答時間の悪化につながる恐れがある

- 並行性を上げる （OLTP環境では特に）
  - ロック待ちを少なく
  - ロックタイムアウトを少なく
  - ロック・エスカレーションを少なく
  - デッドロックを少なく

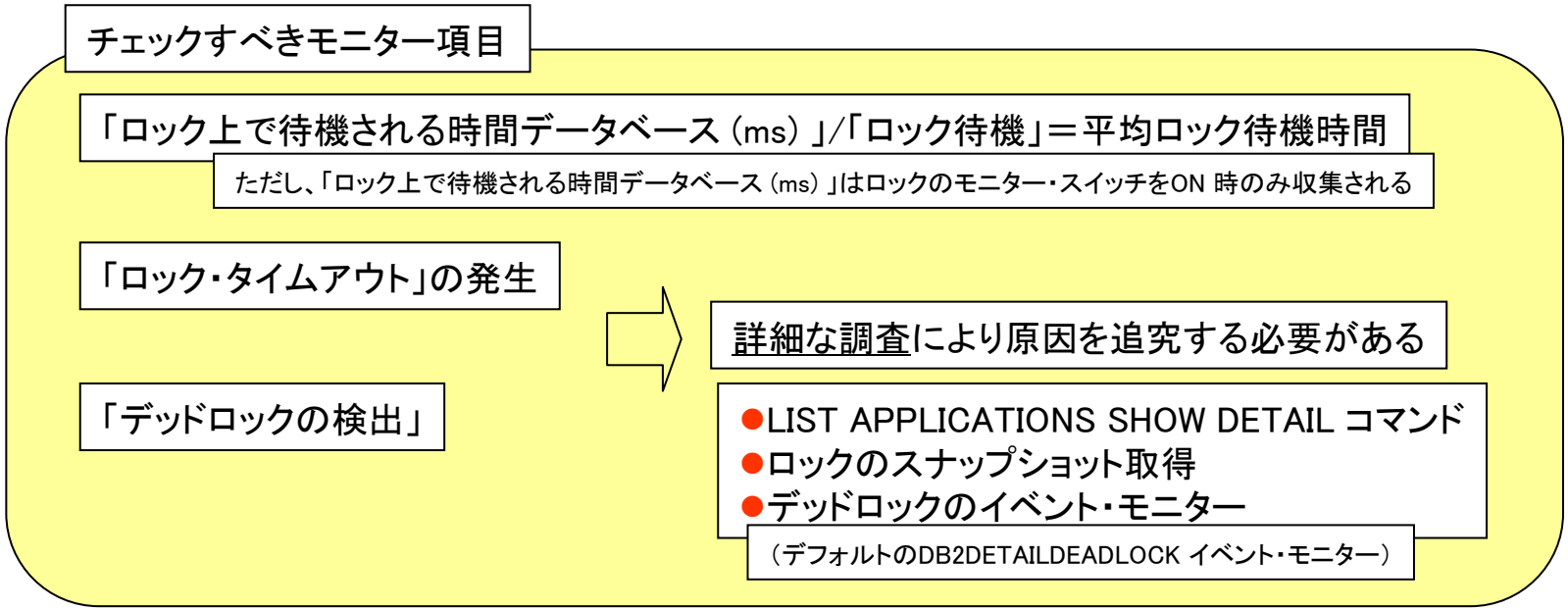
3．2．データベース全体のロック情報を確認するには？ スナップショットの確認ポイント（ロック関連）

- ロック情報関連　モニターエレメント

項目	説明	タイプ	モニタースイッチ
ロック保留 Locks held currently	現在保持されているロックの数	ゲージ	基本
ロック待機 Lock waits	アプリケーションまたは接続がロックを待機した合計回数	カウンター	基本
ロックのために待機している時間 Time database waited on locks (ms)	ロックを待った経過時間の合計	カウンター	ロック
デッドロック検出 Deadlocks detected	発生したデッドロックの合計数	カウンター	基本
ロック・エスカレーション Lock escalations	ロックが複数の行ロックから 1 つの表ロックにエスカレートされた回数	カウンター	基本
排他ロック・エスカレーション Exclusive lock escalations	ロックが複数の行ロックから 1 つの排他表ロックにエスカレートされた回数。または行に対する排他ロックにより表ロックが排他ロックになった回数	カウンター	基本
ロック上で待機中のエージェント Agents currently waiting on locks	ロック待機中のエージェントの数	ゲージ	基本
ロック・タイムアウト Lock Timeouts	オブジェクトをロックするための要求が許可されずにタイムアウトになった回数	カウンター	基本

3．2．データベース全体のロック情報を確認するには？
スナップショットの確認ポイント（ロック関連）

- ロック情報
  - 平均ロック待機時間、ロック・タイムアウト、デッドロックの検出が通常と比べて大きくなっていないか？



3．2．データベース全体のロック情報を確認するには？ スナップショットの確認ポイント（ロック関連）

- ロック情報 ― ロック・エスカレーション
  - 多数の行ロックを、一つの表ロックへ変更するプロセス
    - CPU負荷およびメモリー容量削減のためのしくみ
    - アプリケーションの並行性が落ちる
    - 特に排他ロック・エスカレーションは他の更新・照会処理の両方に影響
  - ロック・エスカレーションの原因
    - locklist（ロックに使用するメモリー容量の上限）が一杯になった時
    - Maxlocks
      （LOCKLISTの中で1アプリケーションが使用可能なメモリーの割合）の値に達した時

チェックすべきモニター項目

「ロック・エスカレーション 」

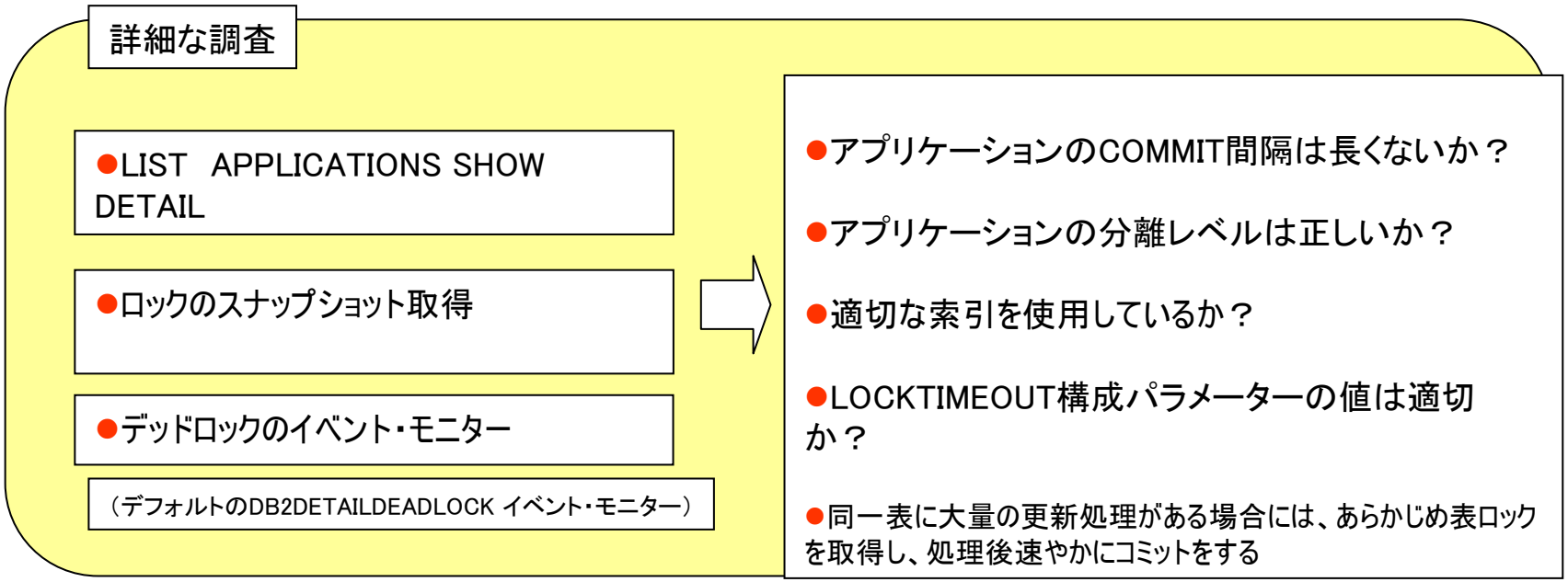
「排他ロック・エスカレーション 」

●ロック・エスカレーションの回避策
- ●頻繁なCOMMIT
- ●ロックの範囲がより狭い分離レベルを使用
- ●LOCKLIST データベース構成パラメーターを増やす
- ●MAXLOCKS データベース構成パラメーターを増やす
- ●同一表に大量の更新処理がある場合には、あらかじめ表ロックを取得し、処理後速やかにコミットをする

3．2．データベース全体のロック情報を確認するには？
スナップショットの確認ポイント（ロック関連）

- ロック待機　多発時の詳細モニター
  - ロック待機発生の原因を追究するための更なる詳細調査をし、

どのアプリケーションが問題となっているかを判別する



（参考） ロックリスト使用状況のモニタリング

- 1 データベースあたりのロックに使用されるメモリー (ページ) 量の算出
  - (1 アプリケーションあたりの平均ロック数 × 1 ロックあたりに必要なメモリー量 × MAXAPPLS / 4096) ページ
  - (例) 平均ロック数を 512 個と仮定した場合の、使用ページ数 (32bitインスタンスの場合)
    - 下限 (ページ) = (512 × 36 × MAXAPPLS) / 4096
    - 上限 (ページ) = (512 × 72 × MAXAPPLS) / 4096
    - MAXAPPLS DB CFG　　:1 データベースあたりの最大同時稼動アプリケーション数
    - この算出値を LOCKLIST DB CFG の値と比較し、ロックリストが十分か検討する

<1 ロックあたりに必要なメモリー容量>

	新規のロック (バイト)	既存のロックに対するロック(バイト)
32bit インスタンス	72	36
64bit インスタンス	112	56

- ロックリスト使用状況の確認
  - Total Lock Memory In USE (lock_list_in_use)
    - 使用中のロックリスト・メモリーのバイト数　　:モニタータイプはゲージ
    - 使用されているロックリストの割合 (%)
      = (Total Lock Memory In USE (バイト) / 4096 (バイト)) / LOCKLIST (ページ) × 100
    - この値が大きい場合には、ロックリスト・サイズの調整、またはアプリケーションのチューニングが必要となる

（参考） ロック数のモニタリング

- 現在保持されているロック数の確認
  - Locks Held
- データベース・レベルのスナップショット
  - データベース全体で保持されているロック数
    - 1 アプリケーション当たりの平均ロック保持数
      = (Lock Held Currently / Applications Connected Currently)
    - この値が大きい場合には、アプリケーションでのロック保持に関するチューニングが必要
- アプリケーション・レベルのスナップショット
  - 特定アプリケーションが保持するロック数
    - 1 アプリケーションが保持できるロック数の上限
      = (LOCKLIST × 4096 / 1 ロック当たりの使用メモリー) × (MAXLOCKS / 100)
    - この値に達すると、ロック・エスカレーションが発生する

---MEMO---

Lock Held の値は、データベース・スナップショットを取った場合と、アプリケーション・スナップショットをとった場合に、数値が示す内容が異なるので、注意が必要です。

3．3．デッドロック発生時の詳細情報を確認するには？

- デッドロック発生時の詳細状況は、デッドロック・イベントモニターで確認。
  - デフォルトでは、DB作成時に、DB2DETAILDEADLOCKという名前の、デッドロック・イベント・モニターが作成され、データベースの開始時に自動的に開始されている

■　デッドロック・イベントモニターが作成されていることの確認

```
$ db2 "select EVMONNAME, TARGET_TYPE,TARGET    from syscat.eventmonitors“

------------------------------------

DB2DETAILDEADLOCK                F          db2detaildeadlock
```

■　デッドロック・イベントモニターの内容確認

```
 $ db2evmon -db sample  -evm db2detaildeadlock
```

デッドロックの例： イベント・モニターの出力 (2)

- Deadlock Event
  - デッドロック発生に関する情報群
    - Deadlock Event : デッドロック発生情報
    - Deadlock Connection : デッドロック発生にかかわったアプリケーション情報

```
33) Deadlock Event ...
  Deadlock ID:   1
  Number of applications deadlocked: 2
  Deadlock detection time: 2007-07-16 23:42:26.285277
  Rolled back Appl participant no: 2
  Rolled back Appl Id: *LOCAL.db2inst1.070716143614
  Rolled back Appl seq number: : 0001

35) Deadlocked Connection ...
  Deadlock ID:   1
  Participant no.: 2
  Participant no. holding the lock: 1
  Appl Id: *LOCAL.db2inst1.070716143614
  Appl Seq number: 00001
  Appl Id of connection holding the lock: *LOCAL.db2inst1.070716143556
  Seq. no. of connection holding the lock: 00001
  Lock wait start time: 2007-07-16 23:42:19.802409
  Lock Name       : 0x00030005000000000280000452
  Lock Attributes : 0x00000000
  Release Flags   : 0x40000000
  Lock Count      : 1
  Hold Count      : 0
  Current Mode    : none
  Deadlock detection time: 2007-07-16 23:42:26.285397
  Table of lock waited on      : T1
  Schema of lock waited on     : DB2INST1
  Data partition id for table  : 0
  Tablespace of lock waited on : IBMDB2SAMPLEREL
  Type of lock: Row
  Mode of lock: X   - Exclusive
  Mode application requested on lock: U   - Update
  Node lock occured on: 0
  Lock object name: 41943044
  Application Handle: 12

37) Deadlocked Connection ...
  Deadlock ID:   1
  Participant no.: 1
  Participant no. holding the lock: 2
  Appl Id: *LOCAL.db2inst1.070716143556
  Appl Seq number: 00001
  Appl Id of connection holding the lock: *LOCAL.db2inst1.070716143614
  Seq. no. of connection holding the lock: 00001
  Lock wait start time: 2007-07-16 23:42:09.217108
  Lock Name       : 0x00030005000000000280000D52
  Lock Attributes : 0x00000000
  Release Flags   : 0x40000000
  Lock Count      : 1
  Hold Count      : 0
  Current Mode    : none
  Deadlock detection time: 2007-07-16 23:42:26.285564
  Table of lock waited on      : T1
  Schema of lock waited on     : DB2INST1
  Data partition id for table  : 0
  Tablespace of lock waited on : IBMDB2SAMPLEREL
  Type of lock: Row
  Mode of lock: X   - Exclusive
  Mode application requested on lock: U   - Update
  Node lock occured on: 0
  Lock object name: 41943053
  Application Handle: 8
```

解説： デッドロックの例： イベント・モニターの出力 (2)

- Deadlock Event につづいて、Deadlock Connection Statement Event の情報が続く
- Deadlock Event には、デッドロック発生に関係したアプリケーションの数、デッドロック発生時刻、ロールバックしたアプリケーション ID が入る
- Deadlock Connection には、デッドロック発生に関係したアプリケーション接続に関する情報が入る
- Statement Event には、デッドロック時にロールバックした SQL ステートメントが入る

デッドロック・イベントモニター： WITH DETAILS オプション

- デッドロックの原因を診断するのに役立つ、より詳細な情報を取得
  - デッドロックが発生したアプリケーションで実行されていたステートメント
  - デッドロックが発生したアプリケーションで獲得しているロック
  - デッドロックが発生したパーティション番号 (パラレル環境のみ)
- WITH DETAILS オプションの指定方法
  - このオプション指定をしない場合には、DB2 UDB V7 までと同様の情報を収集
  - (例)
    - CREATE EVENT MONITOR EV01 FOR DEADLOCK WITH DETAILS WRITE TO FILE '/mydir/EV01.out'
- Connection Header Event レコード発生のタイミング変更
  - 従来は、モニター開始時点に存在する全ての接続に関して情報を出力
    - デッドロックに関係なくとも情報が出力された
  - WITH DETAILS オプションを指定した場合、デッドロック発生のタイミングで Connection Header Event レコードを出力
    - デッドロックに関与した接続に関する情報のみを出力

解説： デッドロック・イベントモニター機能拡張

- DB2 UDB V8 で、デッドロック・イベントに関するモニタリング機能が拡張され、デッドロックが発生する原因を診断するのに役立つ、より詳細な情報を取得することが出来るようになりました。追加された情報は以下の通りです。
  - デッドロックが発生したアプリケーションで実行されていたステートメント
  - デッドロックが発生したアプリケーションで獲得しているロック
  - デッドロックが発生したパーティション番号 （パラレルの場合のみ）
- 機能拡張が行われる前のデッドロックイベントでは、ステートメント情報がなかったため、ユーザーは、DEADLOCK と STATEMENT の両方のイベントモニターを取得し、デッドロックの解析を行っていました。
- WITH DETAILS オプションの追加
- WITH DETAILS オプションにより、より詳細なデッドロック・イベント・モニター・レコードが生成されます。
- WITH DETAILS オプションを指定しない場合は、従来のデッドロック・イベント・モニター・レコードが生成されます。
  - (例)
  - CREATE EVENT MONITOR EV01 FOR DEADLOCKS WITH DETAILS WRITE TO FILE '/mydir/EV01.out'
- Connection Header Event レコードを生成するタイミングの変更
  - Connection Header Event レコードが生成されるタイミングが変わりました
  - 従来のイベントモニターでは、モニターが開始された時点で存在している全ての接続に対して、レコードが生成されていました。また、新しい接続が発生する毎にレコードは生成されていました。つまり、デッドロックの発生原因になっていない全ての接続情報が生成されていたことになります。
  - WITH DETAILS オプションを指定した場合は、Connection Header Event レコードは、デッドロックが発生した時点で生成され、かつデッドロックに関与している接続のみの情報だけが生成されるようになりました。不要な接続情報は生成されず、システム・パフォーマンスへの影響を軽減できます。

(参考) デッドロック・イベント・レコード

- (使用例)
- (例 1) WITH DETAILS オプションをつけたデッドロック・イベント・モニターの出力レコードの例です。
- Connection Header Event レコードは、デッドロックに関与した二つの接続だけで生成されています。
- Deadlocked Connection レコードに、Deadlocked Statement 情報と List of Locks 情報が追加されていることがわかります。
  - 作成した イベント・モニター
  - CREATE EVENT MONITOR dlmon1 FOR DEADLOCKS WITH DETAILS WRITE TO FILE '/eventmonito/deadlock/dlmon1' MAXFILES 3 MAXFILESIZE 1000

イベントレコード出力例

```
------------------------------------------------------------------------
                          EVENT LOG HEADER
  Event Monitor name: DLMON1
  Server Product ID: SQL09012
  Version of event monitor data: 8
  Byte order: BIG ENDIAN
  Number of nodes in db2 instance: 1
  Codepage of database: 943
  Territory code of database: 81
  Server instance name: taka9
------------------------------------------------------------------------

------------------------------------------------------------------------
  Database Name: SAMPLE
  Database Path: /home/taka9/data/taka9/NODE0000/SQL00001/
  First connection timestamp: 2007-07-17 01:15:37.713890
  Event Monitor Start time:   2007-07-17 01:44:57.803346
------------------------------------------------------------------------

3) Deadlock Event ...
  Deadlock ID:   2
  Number of applications deadlocked: 2
  Deadlock detection time: 2007-07-17 01:46:38.731548
  Rolled back Appl participant no: 2
  Rolled back Appl Id: *LOCAL.taka9.070716162013
  Rolled back Appl seq number: : 0003
```

デッドロックによって
ロールバックされた
アプリケーション番号

```
4) Connection Header Event ...
  Appl Handle: 18
  Appl Id: *LOCAL.taka9.070716162013
  Appl Seq number: 00003
  DRDA AS Correlation Token: *LOCAL.taka9.070716162013
  Program Name     : db2bp
  Authorization Id: TAKA9
  Execution Id     : taka9
  Codepage Id: 943
  Territory code: 81
  Client Process Id: 1171652
  Client Database Alias: SAMPLE
  Client Product Id: SQL09012
  Client Platform: Unknown
  Client Communication Protocol: Local
  Client Network Name: sato3
  Connect timestamp: 2007-07-17 01:20:13.424572

5) Deadlocked Connection ...
  Deadlock ID:   2
  Participant no.: 2
  Participant no. holding the lock: 1
  Appl Id: *LOCAL.taka9.070716162013
  Appl Seq number: 00003
  Appl Id of connection holding the lock: *LOCAL.taka9.070716162006
  Seq. no. of connection holding the lock: 00001
```

(参考) デッドロック・イベント・レコード (続き)

```
Lock wait start time: 2007-07-17 01:46:32.590502
Lock Name       : 0x00030005000000000280000452
Lock Attributes : 0x00000000
Release Flags   : 0x40000000
Lock Count      : 1
Hold Count      : 0
Current Mode    : none
Deadlock detection time: 2007-07-17 01:46:38.731612
Table of lock waited on      : T1
Schema of lock waited on     : TAKA9
Data partition id for table  : 0
Tablespace of lock waited on : IBMDB2SAMPLEREL
Type of lock: Row
Mode of lock: X   - Exclusive
Mode application requested on lock: U   - Update
Node lock occured on: 0
Lock object name: 41943044
Application Handle: 18
Deadlocked Statement:
  Type      : Dynamic
  Operation: Execute Immediate
  Section   : 203
  Creator   : NULLID
  Package   : SQLC2F0A
  Cursor    :
  Cursor was blocking: FALSE
  Text      : update T1 set COL2 ='TEST2' where COL1 = 1
List of Locks:
 Lock Name          : 0x00000001000000010001D50056
 Lock Attributes    : 0x00000000
 Release Flags      : 0x40000000
 Lock Count         : 1
 Hold Count         : 0
 Lock Object Name   : 0
 Object Type        : Internal - Variation
 Data partition id  : -1
 Mode               : S   - Share
```

デッドロック発生時に
実行されていた
ステートメントで
ロールバックされたもの

```
 Lock Name          : 0x00030005000000000280000D52
  Lock Attributes   : 0x00000000
  Release Flags     : 0x40000000
  Lock Count        : 1
  Hold Count        : 0
  Lock Object Name  : 41943053
  Object Type       : Row
  Tablespace Name   : IBMDB2SAMPLEREL
  Table Schema      : TAKA9
  Table Name        : T1
  Data partition id : 0
  Mode              : X   - Exclusive

  Lock Name         : 0x53514C4332463041E2F82CF041
  Lock Attributes   : 0x00000000
  Release Flags     : 0x40000000
  Lock Count        : 1
  Hold Count        : 0
  Lock Object Name  : 0
  Object Type       : Internal - Plan
  Data partition id : -1
  Mode              : S   - Share

  Lock Name         : 0x00030005000000000000000054
  Lock Attributes   : 0x00000000
  Release Flags     : 0x40000000
  Lock Count        : 3
  Hold Count        : 0
  Lock Object Name  : 5
  Object Type       : Table
  Tablespace Name   : IBMDB2SAMPLEREL
  Table Schema      : TAKA9
  Table Name        : T1
  Data partition id : 0
  Mode              : IX  - Intent Exclusive

Locks Held:    4
Locks in List: 4
```

```
6) Connection Header Event ...
  Appl Handle: 17
  Appl Id: *LOCAL.taka9.070716162006
  Appl Seq number: 00002
  DRDA AS Correlation Token: *LOCAL.taka9.070716162006
  Program Name     : db2bp
  Authorization Id: TAKA9
  Execution Id     : taka9
  Codepage Id: 943
  Territory code: 81
  Client Process Id: 1020088
  Client Database Alias: SAMPLE
  Client Product Id: SQL09012
  Client Platform: Unknown
  Client Communication Protocol: Local
  Client Network Name: sato3
  Connect timestamp: 2007-07-17 01:20:06.174669
```

(参考) デッドロック・イベント・レコード (続き)

```
7) Deadlocked Connection ...
 Deadlock ID:   2
 Participant no.: 1
 Participant no. holding the lock: 2
 Appl Id: *LOCAL.taka9.070716162006
 Appl Seq number: 00002
 Appl Id of connection holding the lock: *LOCAL.taka9.070716162013
 Seq. no. of connection holding the lock: 00001
 Lock wait start time: 2007-07-17 01:46:29.386231
 Lock Name       : 0x00030005000000000280000D52
 Lock Attributes : 0x00000000
 Release Flags   : 0x40000000
 Lock Count      : 1
 Hold Count      : 0
 Current Mode    : none
 Deadlock detection time: 2007-07-17 01:46:38.731719
 Table of lock waited on      : T1
 Schema of lock waited on     : TAKA9
 Data partition id for table  : 0
 Tablespace of lock waited on : IBMDB2SAMPLEREL
 Type of lock: Row
 Mode of lock: X   - Exclusive
 Mode application requested on lock: U   - Update
 Node lock occured on: 0
 Lock object name: 41943053
 Application Handle: 17
 Deadlocked Statement:
  Type     : Dynamic
  Operation: Execute Immediate
  Section  : 203
  Creator  : NULLID
  Package  : SQLC2F0A
  Cursor   :
  Cursor was blocking: FALSE
  Text     : update T1 set COL2 ='TEST2' where COL1 = 10
```

デッドロック発生時に
実行されていた
ステートメント

```
List of Locks:
 Lock Name          : 0x000000010000000010001320056
 Lock Attributes    : 0x00000000
 Release Flags      : 0x40000000
 Lock Count         : 1
 Hold Count         : 0
 Lock Object Name   : 0
 Object Type        : Internal - Variation
 Data partition id  : -1
 Mode               : S   - Share

 Lock Name          : 0x00030005000000000280000452
 Lock Attributes    : 0x00000000
 Release Flags      : 0x40000000
 Lock Count         : 1
 Hold Count         : 0
 Lock Object Name   : 41943044
 Object Type        : Row
 Tablespace Name    : IBMDB2SAMPLEREL
 Table Schema       : TAKA9
 Table Name         : T1
 Data partition id  : 0
 Mode               : X   - Exclusive

 Lock Name          : 0x53514C4332463041E2F82CF041
 Lock Attributes    : 0x00000000
 Release Flags      : 0x40000000
 Lock Count         : 1
 Hold Count         : 0
 Lock Object Name   : 0
 Object Type        : Internal - Plan
 Data partition id  : -1
 Mode               : S   - Share

 Lock Name          : 0x00030005000000000000000054
 Lock Attributes    : 0x00000000
 Release Flags      : 0x40000000
 Lock Count         : 3
 Hold Count         : 0
 Lock Object Name   : 5
 Object Type        : Table
 Tablespace Name    : IBMDB2SAMPLEREL
 Table Schema       : TAKA9
 Table Name         : T1
 Data partition id  : 0
 Mode               : IX  - Intent Exclusive

Locks Held:    4
Locks in List: 4
```

デッドロック・イベントモニター： HISTORY / VALUES オプション

- 追加された二つのオプション (HISTORY、VALUES) により、デッドロックに関する更に詳細な情報を取得可能
  - デッドロックの原因究明が容易になる
- HISTORY オプション
  - 現行作業単位における全 SQL ステートメントの履歴を出力
    - 直前の作業単位で開かれた WITH HOLD カーソルも含む
    - 非コミット読み取り (UR) 分離レベルで発行された SELECT ステートメントは、ステートメント履歴に含まれない
    - 各 SQL ステートメントのステートメント・コンパイル環境情報
  - WITH DETAILS の指定が必要
- VALUES オプション
  - 各 SQL ステートメント実行時のパラメーター・マーカーへの入力値を出力
    - LOB データ、LONG データは含まれない
  - WITH DETAILS HISTORY の指定が必要
- 既存のイベント・モニターで新しいオプションを使用するには、イベント・モニターを再作成する必要がある
  - デフォルトのデッドロック・イベントモニター (DB2DEADLOCK) では、HISTORY、VALUES オプションは設定されていない (変更できない)
  - オプションを指定しない場合は、DB2 UDB V8.1 までと同じ情報を収集
  - 使用時には他のオプションと併用する必要がある

V9.5

3．4．　ロックタイムアウト発生時の詳細情報を確認するには？

- ロックタイムアウト・イベント発生時に関連情報を出力させる。
  - レジストリにて設定する
    - db2set DB2_CAPTURE_LOCKTIMEOUT=ON
  - 出力先
    - DIAGPATHで指定されたディレクトリ
    - ファイル名
      - db2locktimeout.par.AGENTID.yyyy-mm-dd-hh:mm:ss
        - par ：データベース・パーティション番号
        - AGENTID ：エージェント ID
        - yyyy-mm-dd-hh:mm:ss ：年、月、日、時間、分、および秒から成るタイム・スタンプ
  - 出力情報
    - 内容
      - 発生日時、対象のロック、アプリケーション情報など
    - 対象
      - ロック要求側（ロック・タイムアウト・エラーを受け取ったアプリケーション）
      - 現行ロック所有者
  - 考慮点
    - 出力されたファイルをクリーンアップする必要あり

V9.5

3．4．　ロックタイムアウト発生時の詳細情報を確認するには？

- ロックタイムアウトレポート例

```
$ cat db2locktimeout.0.6456.2007-11-26-15:32:25

LOCK TIMEOUT REPORT

Date:                 26/11/2007
Time:                 15:32:25
Instance:             taira95
Database:             SAMPLE
Database Partition: 0


Lock Information:

   Lock Name:      0002000F000000000601000152
   Lock Type:      Row
   Lock Specifics: Tablespace ID=2, Table ID=15, Row ID=x0000000006010001


Lock Requestor:
   System Auth ID:           TAIRA95
   Application Handle:       [0-5172]
   Application ID:           *LOCAL.taira95.071126054456
   Application Name:         db2bp
   Requesting Agent ID:      6456
   Coordinator Agent ID:     6456
   Coordinator Partition:    0
   Lock timeout Value:       5000 milliseconds
   Lock mode requested:      .NS
   Application Status:       (SQLM_UOWEXEC)
   Current Operation:        (SQLM_FETCH)
   Lock Escalation:          No

   Context of Lock Request:
   Identification:             UOW ID (6); Activity ID (1)
   Activity Information:
      Package Schema:          (NULLID  )
      Package Name:            (SQLC2G13NULLID  )
      Package Version:         ()
      Section Entry Number:    201
      SQL Type:                Dynamic
      Statement Type:          DML, Select (blockable)
      Effective Isolation:     Cursor Stability
      Statement Unicode Flag: No
      Statement:               select * from staff where JOB='Sales'


Lock Owner (Representative):
   System Auth ID:           TAIRA95
   Application Handle:       [0-5216]
   Application ID:           *LOCAL.taira95.071126062125
   Application Name:         db2bp
   Requesting Agent ID:      5197
   Coordinator Agent ID:     5197
   Coordinator Partition:    0
   Lock mode held:           ..X

List of Active SQL Statements:  Not available

List of Inactive SQL Statements from current UOW: Not available
```

保持されているロック情報

ロックウェイト発生場所（表スペース、表、行）の特定可能

獲得しようとしているロック情報（ロックウェイト側）

全てのケースでSQLステートメントが出力されるわけではない
- ・静的SQL
- ・DPFのｻﾌﾞｴｰｼﾞｪﾝﾄ
- ・ユーティリティ（Backupなど）

などの場合には出力されない
（このケースは動的SQLだが保持側のSQLステートメントは出力されなかった）

対象のSQLステートメント

3. 5. 障害ログ(db2diag.log)に含まれる情報

- 障害ログ(db2diag.log)にもロック関連の情報が記録される。
  - NOTIFYLEVEL の設定によって記録内容が変る

  - ロック・エスカレーションの発生
  - ロック・タイムアウトの発生 （NOTIFYLEVEL=4と設定した場合）
  - デッドロックの発生（NOTIFYLEVEL=4と設定した場合）

3. 5. 障害ログ(db2diag.log)に含まれる情報
ロック・エスカレーション情報(1/2)

- NOTIFYLEVEL の設定により、出力内容が異なる
  - NOTIFYLEVEL=3 (省略時値)
    - ロック・エスカレーションの発生情報
  - NOTIFYLEVEL=4
    - ロック・エスカレーションの発生情報 + エスカレーションに起因した動的 SQL
      - 静的 SQL は記録されない
    - エスレーション時の xxxxxx.nfy の出力

<NOTIFYLEVEL=3 の表示>

```
2007-07-16-18.04.16.148163   Instance:db2inst1   Node:000
PID:1011948(db2agent (SAMPLE) 0)   TID:1   Appid:*LOCAL.db2inst1.070716085920
data management  sqldEscalateLocks Probe:2   Database:SAMPLE

ADM5500W  DB2 is performing lock escalation.  The total number of locks
currently held is "82", and the target number of locks to hold is "41".

2007-07-16-18.04.16.148437   Instance:db2inst1   Node:000
PID:1011948(db2agent (SAMPLE) 0)   TID:1   Appid:*LOCAL.db2inst1.070716085920
data management  sqldEscalateLocks Probe:3   Database:SAMPLE

ADM5502W  The escalation of "79" locks on table "DB2INST1.T1" to lock intent
"X" was successful.
```

<NOTIFYLEVEL=4 の表示>

```
2007-07-16-18.11.53.090180   Instance:db2inst1   Node:000
PID:1020150(db2agent (SAMPLE) 0)   TID:1   Appid:*LOCAL.db2inst1.070716091144
data management  sqldEscalateLocks Probe:1   Database:SAMPLE

ADM5501I  DB2 is performing lock escalation.  The total number of locks
currently held is "85", and the target number of locks to hold is "42".  The
current statement being executed is "update t1 set col2 = 'TEST2' where col1
between 1 and 100".

2007-07-16-18.11.53.090986   Instance:db2inst1   Node:000
PID:1020150(db2agent (SAMPLE) 0)   TID:1   Appid:*LOCAL.db2inst1.070716091144
data management  sqldEscalateLocks Probe:3   Database:SAMPLE

ADM5502W  The escalation of "81" locks on table "DB2INST1.T1" to lock intent
"X" was successful.
```

3. 5. 障害ログ(db2diag.log)に含まれる情報
ロック・エスカレーション情報(2/2)

- DIAGLEVEL=3、4 の場合に、ロック・エスカレーションの発生情報が出力される
  - エスカレーション時の db2diag.log の出力

```
2007-07-16-18.17.36.563285+540 E90155A470          LEVEL: Warning
PID     : 1077348              TID  : 1            PROC : db2agent (SAMPLE) 0
INSTANCE: db2inst1                NODE : 000          DB   : SAMPLE
APPHDL  : 0-7                  APPID: *LOCAL.db2inst1.070716091729
AUTHID  : DB2INST1
FUNCTION: DB2 UDB, data management, sqldEscalateLocks, probe:3
MESSAGE : ADM5502W  The escalation of "81" locks on table "DB2INST1.T1" to lock
          intent "X" was successful.
```

3. 5. 障害ログ(db2diag.log)に含まれる情報
ロック・タイムアウト情報(1/2)

- DIAGLEVEL の設定により、db2diag.log 出力内容が異なる
  - DIAGLEVEL=3 (省略時値)
    - 出力なし
  - DIAGLEVEL=4
    - ロック・タイムアウトの発生情報
    - ロック・タイムアウトに起因した動的 SQL、静的 SQL パッケージ名、オブジェクト情報、ロック・モード

<DIAGLEVEL=4 の表示>

```
2007-07-16-18.43.26.290902+540 I171681A764        LEVEL: Info
PID     : 1020094              TID  : 1           PROC : db2agent (SAMPLE) 0
INSTANCE: db2inst1                 NODE : 000         DB   : SAMPLE
APPHDL  : 0-11                 APPID: *LOCAL.db2inst1.070716094314
AUTHID  : DB2INST1
FUNCTION: DB2 UDB, lock manager, sqlplnfd, probe:80
DATA #1 : String, 130 bytes
Request for lock "TAB: (3, 5)" in mode ".IS" timed out
Application caused the lock wait is "*LOCAL.db2inst1.070716094236"
Statement:
DATA #2 : Hexdump, 44 bytes
0x0780000020B55200 : 7365 6C65 6374 202A 2066 726F 6D20 7431    select * from t1
0x0780000020B55210 : 2077 6865 7265 2063 6F6C 3120 6265 7477     where col1 betw
0x0780000020B55220 : 6565 6E20 3120 616E 6420 3130              een 1 and 10

2007-07-16-18.43.26.291331+540 I172446A766        LEVEL: Info
PID     : 1020094              TID  : 1           PROC : db2agent (SAMPLE) 0
INSTANCE: db2inst1                 NODE : 000         DB   : SAMPLE
APPHDL  : 0-11                 APPID: *LOCAL.db2inst1.070716094314
AUTHID  : DB2INST1
FUNCTION: DB2 UDB, oper system services, sqlofica, probe:10
DATA #1 : SQLCA, PD_DB2_TYPE_SQLCA, 136 bytes
 sqlcaid : SQLCA     sqlcabc: 136   sqlcode: -911   sqlerrml: 2
 sqlerrmc: 68
 sqlerrp : SQLRI2CB
 sqlerrd : (1) 0x80100044       (2) 0x00000044       (3) 0x00000000
           (4) 0x00000000       (5) 0xFFFFFE0C       (6) 0x00000000
 sqlwarn : (1)       (2)       (3)       (4)         (5)       (6)
           (7)       (8)       (9)       (10)        (11)
 sqlstate: 40001
```

ROLLBACK したアプリケーション
*LOCAL.db2inst1.070716094314

ロックWAITしたオブジェクト
"TAB: (3,5)"
取得しようとしていたロックのモード
IS

ロックを保持していたアプリケーション
*LOCAL.db2inst1.070716094236

ROLLBACK した動的 SQL
select * from t1 where col1
between 1 and 10

3. 5. 障害ログ(db2diag.log)に含まれる情報
ロック・タイムアウト情報(2/2)

- DIAGLEVEL DBM 構成パラメーターを 4 に設定することにより、ロック待機に関する詳細情報を障害ログに出力することが出来ます。
- 詳細情報には、タイムアウトに起因するオブジェクト名、ロックモード、ロックを保持していたアプリケーション、動的 SQL ステートメント、静的 SQL パッケージ名 などが出力されます。

3. 5. 障害ログ(db2diag.log)に含まれる情報
デッドロック情報(1/2)

- DIAGLEVEL の設定により、db2diag.log 出力内容が異なる
  - DIAGLEVEL=3 (省略時値)
    - 出力なし
  - DIAGLEVEL=4
    - デッドロック・タイムアウト発生情報
    - デッドロック・タイムアウトに起因した動的 SQL、静的 SQL パッケージ名、オブジェクト情報、ロック・モード
  - デッドロックによるタイムアウト時の db2diag.log の出力

<DIAGLEVEL = 4>

```
2007-07-16-23.49.53.667358+540 I260519A797        LEVEL: Info
PID     : 827576               TID  : 1           PROC : db2agent (SAMPLE) 0
INSTANCE: db2inst1                NODE : 000         DB   : SAMPLE
APPHDL  : 0-12                 APPID: *LOCAL.db2inst1.070716144819
AUTHID  : DB2INST1
FUNCTION: DB2 UDB, lock manager, sqlplnfd, probe:80
DATA #1 : String, 165 bytes
Request for lock "REC: (3, 5) RID x0000000002800004" in mode "..U" failed due to deadlock
Application caused the lock wait is "*LOCAL.db2inst1.070716144813"
Statement:
DATA #2 : Hexdump, 42 bytes
0x0780000020B53CC0 : 7570 6461 7465 2054 3120 7365 7420 636F    update T1 set co
0x0780000020B53CD0 : 6C32 203D 2754 4553 5432 2720 7768 6572    l2 ='TEST2' wher
0x0780000020B53CE0 : 6520 636F 6C31 203D 2031                   e col1 = 1
```

ROLLBACK したアプリケーション
*LOCAL.db2inst1.070716144819

ロックWAITしたオブジェクト
REC: (3, 5) RID x0000000002800004
取得しようとしていたロックのモード
U

ロックを保持していたアプリケーション
*LOCAL.db2inst1.070716144813

ROLLBACK した動的 SQL
update T1 set col2 = 'TEST2'
where col1 = 1

3. 5. 障害ログ(db2diag.log)に含まれる情報 デッドロック情報(2/2)

- 続き

<DIAGLEVEL = 4>

```
2007-07-16-23.49.53.667717+540 I261317A765        LEVEL: Info
PID     : 827576               TID  : 1           PROC : db2agent (SAMPLE) 0
INSTANCE: db2inst1                NODE : 000         DB   : SAMPLE
APPHDL  : 0-12                 APPID: *LOCAL.db2inst1.070716144819
AUTHID  : DB2INST1
FUNCTION: DB2 UDB, oper system services, sqlofica, probe:10
DATA #1 : SQLCA, PD_DB2_TYPE_SQLCA, 136 bytes
 sqlcaid : SQLCA     sqlcabc: 136   sqlcode: -911   sqlerrml: 1
 sqlerrmc: 2
 sqlerrp : SQLRIOAE
 sqlerrd : (1) 0x80100002       (2) 0x00000002       (3) 0x00000000
           (4) 0x00000000       (5) 0xFFFFFE0C       (6) 0x00000000
 sqlwarn : (1)       (2)       (3)       (4)         (5)       (6)
           (7)       (8)       (9)       (10)        (11)
 sqlstate: 40001

2007-07-16-23.49.53.668005+540 I262083A682        LEVEL: Info
PID     : 970826               TID  : 1           PROC : db2bp
INSTANCE: db2inst1                NODE : 000
APPID   : *LOCAL.db2inst1.070716144819
FUNCTION: DB2 UDB, oper system services, sqlofica, probe:10
DATA #1 : SQLCA, PD_DB2_TYPE_SQLCA, 136 bytes
 sqlcaid : SQLCA     sqlcabc: 136   sqlcode: -911   sqlerrml: 1
 sqlerrmc: 2
 sqlerrp : SQLRIOAE
 sqlerrd : (1) 0x80100002       (2) 0x00000002       (3) 0x00000000
           (4) 0x00000000       (5) 0xFFFFFE0C       (6) 0x00000000
 sqlwarn : (1)       (2)       (3)       (4)         (5)       (6)
           (7)       (8)       (9)       (10)        (11)
 sqlstate: 40001
```

SQLCODE=-911

（参考）ロックに関する考慮点

- 同時稼動性を高めるためには
  - 作業単位の区切りに必ず COMMIT を入れ、出来るだけ早くロックをはずす
    - 照会処理でもロック (Read Lock) は取得されるため、RR、RS のアプリケーションでは、作業単位の区切りに必ず COMMIT を入れる
    - CLOSE カーソルで、WITH RELEASE オプションを利用する (ISOLATION が、RR、RS の場合)
    - UR のアプリケーションでも、動的SQL の場合には、カタログへのロックが取得される
    - WITH オプションによる、ステートメント単位の ISOLATION 指定も検討する
  - ISOLATION=UR での照会処理を検討する
    - UR 以外では、COMMIT されていない更新データは、他のアプリケーションからはアクセスすることは出来ない
  - ロック関連の構成パラメーターをチューニングし、エスカレーションを減らす
  - ロック競合を防ぐべく、索引スキャン / 同じ順番でのスキャンになるように索引設計、業務設計を行う
- アプリケーション要件を満たすロックの制御を行うには
  - アプリケーションに適切な 分離レベルを選択する
  - ロックの単位を用途に応じて設定する
    - LOCK TABLE により表全体にロックを取得することが可能
    - ALTER TABLE によりロックの単位を ROW または TABLE に設定することが可能
- 問題判別を行うには
  - イベント・モニター、ロック・スナップショット、障害ログによる原因究明

3．6． ロック・イベントモニター (V9.7-)

V9.7 New

- ロック・イベント・モニター （V9.7新機能）
  - 従来より詳細にロックに関する情報を収集
- 作業単位イベント・モニター
  - トランザクションの完了ごとに情報を収集

- 新イベント・モニターは“未フォーマット・イベント表”に格納される
  - バイナリー形式

- 未フォーマット・イベント表をフォーマットする方法として以下が提供される
  - EVMON_FORMAT_UE_TO_XML
  - EVMON_FORMAT_UE_TO_TABLES
  - db2evmonfmt

V9.7での非推奨機能

- 機能強化に伴い、非推奨となる機能
  - ロック関連モニター機能は、V9.7より「ロック・イベントモニター」として提供される。

非推奨となる機能	V9.7で提供された機能
DB2_CAPTURE_LOCKTIMEOUT レジストリー変数	ロックイベント・モニター
デッドロック・イベント・モニター	ロックイベント・モニター
DB2DETAILDEADLOCK イベント・モニター	ロックイベント・モニター
トランザクション・イベント・モニター	作業単位イベント・モニター
Dynamic SQL snapshot	-

ロック・イベント・モニターとは

- 収集されるロック・イベントの情報
  - 原因となったロック
  - 原因となったロックを保持しているアプリケーション
  - 原因となるロックを待機または要求していたアプリケーション
  - ロック・イベント時のアプリケーションの実行内容
- ロック・データの収集対象にできるアクティビティー
  - SQL ステートメント
    - DML 、DDL 、CALL
  - **LOAD** コマンド
  - **REORG** コマンド
  - **BACKUP DATABASE** コマンド
  - ユーティリティー要求

ロック・イベント・モニターとは

- 従来の方法に比べ、より詳細なロック情報を出力することができるようになりました。
  - ロック待ち、ロックタイムアウト、デッドロックそれぞれについて収集するかどうかを設定可能になりました。
  - ロック保持者と、ロック待ちの両方の情報を出力されます。
  - 設定によりステートメントの履歴を出すことも可能です。

- DB2 V9.7からデッドロック、ロックタイムアウトの情報をdb2diag.logに出力するようになりました。ロックが多発するシステムではログファイルが大きくなる可能性がありますのでご注意ください。場合によってはDIAGSIZEの指定も有効です。
  - 出力される情報は以下の通りです。
    - タイムスタンプと、データベースパーティション番号
    - ロックを待たされた側のアプリケーションID
    - ロックを保持したアプリケーションID
    - 競合したロックの情報

ロック・イベント・モニターとは

- ロック・イベント・モニターを収集するための設定

  以下のいずれかを設定

  - CREATE/ALTER WORKLOADで取得したいロックイベントを設定
    - 詳細は「7-3. モニタリング機能とWLMの連携」を参照
  - ロックに関するDB CFGの設定
    - MON_LOCKWAIT
      - ロック待ちイベントの収集
      - DB CFGで設定するより、ワークロード・レベルで有効にすることが推奨される
    - MON_LOCKTIMEOUT
      - ロック・タイム・アウトのイベントの収集
      - 予期されないイベントであれば、DB CFGで有効にし、予期しているイベントであればワークロード・レベルで有効にすることが推奨される
    - MON_DEADLOCK
      - デッドロック・イベントの収集
      - DB CFGで有効にすることが推奨される
    - MON_LW_THRESH
      - MON_LOCKWAITイベントが収集されるまでのロック待ち時間を制御

  設定の上、イベント・モニターを作成

  - CREATE EVENT MONITOR FOR LOCKING

ロック・イベント・モニターとは

- MON_LOCKWAIT、MON_LOCKTIMEOUT、MON_DEADLOCK
  - NONE
    - イベントを収集しない
  - WITHOUT_HIST
    - ロック・イベントを収集
  - HISTORY
    - ロック・イベントを収集　同トランザクション内の過去のアクティビティ履歴も表示
  - HIST_AND_VALUE
    - ロック・イベントを収集　アクティビティ履歴と入力データ値を表示
- MON_LW_THRESH
  - マイクロ秒　（1秒 = 1,000,000マイクロ秒）
    - MON_LOCKWAITのイベントが発生するまでのロック待機時間

ロック・イベント・モニター作成シンタックス

```
>>-CREATE EVENT MONITOR--event-monitor-name----FOR LOCKING------>

>----WRITE TO UNFORMATTED EVENT TABLE--+--------------------------------------+---->
                                       '-(--| unformatted-event-table-options |--)-'

   .-AUTOSTART---.
>--+-------------+-------------------------------------------><
   '-MANUALSTART-'

unformatted-event-table-options

   .-------------------------------.
   V                               |
|----+-TABLE--table name----------+-+---
     +-IN--tablespace name--------+
     | .-PCTDEACTIVATE--100-----. |
     '-+-PCTDEACTIVATE -integer-+-'
```

どの程度表スペースが満たされた時点でイベント・モニターが非活動化されるかを指定（DMSのみ有効）

詳しいオプションなどはマニュアルをご参考ください。
http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.sql.ref.doc/doc/r0054074.html

ロック・イベント・モニター考慮点

- データベースごとに1つのロック・イベント・モニターを作成
  - 追加のイベント・モニターは、同じデータのコピーを作成するだけ
- DB2DETAILDEADLOCKイベント・モニターは除去する
  - デフォルトではすべてのデータベースは、DB2DETAILDEADLOCK イベント・モニターが有効な状態
  - これは、V9.7からは推奨されておらず、今後のリリースで除去される可能性がある
  - DB2DETAILDEADLOCK イベント・モニターを除去しないと、新しいイベント・モニターのどちらもデータを収集することになる
  - 以下を実行して、DB2DETAILDEADLOCK イベント・モニターを除去する
    - SET EVENT MONITOR DB2DETAILDEADLOCK state 0
    - DROP EVENT MONITOR DB2DETAILDEADLOCK
- DPF環境では未フォーマット・イベント表が存在するパーティションのみのイベントをモニターする

未フォーマット・イベント表へのアクセス方法

- プロシージャーを使用する
  - EVMON_FORMAT_UE_TO_XML
    - 未フォーマット・イベント表から XML 文書にデータを抽出
  - EVMON_FORMAT_UE_TO_TABLES
    - 未フォーマット・イベント表から一連のリレーショナル表にデータを抽出

```
例）UOWEVMONイベントモニター表から、Javaアプリケーションの UOWに関する情報をフォーマット
db2 call "evmon_format_ue_to_tables('UOW',NULL,NULL,NULL,NULL,NULL,NULL,-1,
                                    'select * from uowevmon where appl_name=''java'' ')"
```

- db2evmonfmtを使用する
  - テキスト・レポートまたはフォーマット済み XML 文書のどちらの形式で出力するかを選択可能
  - Java ソース・コードとして提供されるツール
  - 事前にセットアップが必要
    - Java ソース・ファイルに組み込まれている説明に従って、ツールをセットアップおよびコンパイルする
      - sqllib/samples/java/jdbc ディレクトリーにソース・コードがある

```
例）SAMPLEDBのLOCK表から、直近24時間以内に発生したLOCKTIMEOUTのデータをフォーマット
java db2evmonfmt -d sample -ue LOCK -ftext -hours 24 -type locktimeout
```

表関数

- EVMON_FORMAT_UE_TO_XML

```
>>-EVMON_FORMAT_UE_TO_XML--(--options--,----------------------->

>--FOR EACH ROW OF--(--fullselect-statement--)--)-------------><
```

- EVMON_FORMAT_UE_TO_TABLES

```
>>-EVMON_FORMAT_UE_TO_TABLES--(--evmon_type--,--xsrschema--,---->

>--xsrobjectname--,--xmlschemafile--,--tabschema--,------------->

>--tbsp_name--,--options--,--commit_count--,--fullselect--)---><
```

詳しいオプションなんどはマニュアルをご参考ください。
http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.sql.rtn.doc/doc/r0054909.html
http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.sql.rtn.doc/doc/r0054910.html

db2evmonfmt

- Javaベースのツール

```
>>-java--db2evmonfmt-----------------------------------------><
> --+-+-| connect |--+--| filter options |-+------------------><
   | '-| XML file |-'                     |
   '- -h-------------------------------'

connect
|-- -d--db_name-- -ue--table_name-------------------------------->
>--+-----------------------------+-------------------------------|
   '- -u--user_id-- -p--password-'

XML file
|-- -f--xml_filename----------------------------------------------|

filter options
|--+- -fxml-----------------------+--+---------------+--->
   '- -ftext--+-------------------+-'  '- -id--event_id-'
              '- -ss--stylesheet_name-'
>--+--------------------+--+--------------------+-------------->
   '- -type--event_type-'  '- -hours--num_hours-'
>--+--------------------+--+----------------+------------------>
   '- -w--workload_name-'  '- -a--appl_name-'
>--+------------------------+--------------------------------|
   '- -s--srvc_subclass_name-'
```

解説:

- db2evmonfmt.java に記載されている通りのパスを設定する必要があります
- CLASSPATHに db2evmonfmt をコピーしたワークディレクトリを追加する必要があります
  - http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.admin.mon.doc/doc/c0053977.html

```
// Steps to run the sample with command line window:
//
 (一部省略)
//
//   2) Modify the CLASSPATH to include:
//
//         <install_path>/sqllib/java/db2java.zip
//         <install_path>/sqllib/java/db2jcc.jar
//         <install_path>/sqllib/java/db2jcc_license_cu.jar
//         <install_path>/sqllib/java/<jdkDirName>/lib
//         <install_path>/sqllib/lib
//         <install_path>/sqllib/function
//         <install_path>/sqllib/java/sqlj.zip
//       where <jdkDirName> is the name of the
//       jdk directory under <install_path>/sqllib/java.
//
//   3) Modify the PATH to include ...
 (一部省略)
```

CLASSPATH に db2evmonfmt.java をコピーしたディレクトリを追加
例：/home/db2inst/mon

（参考）ロック・イベント・モニター作成例

- 手順
  - イベント・モニターを定義する

```
$ db2 create event monitor evmon1 for locking write to unformatted event table
DB20000I  The SQL command completed successfully.
```

  - イベント・モニターを活動化させる

```
$ db2 set event monitor evmon1 state 1
DB20000I  The SQL command completed successfully.
```

  - （データベースに対する処理が行われる）
    - イベントのタイプに応じて情報が自動的に収集される
  - イベント・モニター情報の強制出力（オプション）

```
$ db2 flush event monitor evmon1
DB20000I  The SQL command completed successfully..
```

  - イベント・モニターを非活動化する

```
$ db2 set event monitor evmon1 state 0
DB20000I  The SQL command completed successfully.
```

  - db2evmonfmtでフォーマットする

```
$ java db2evmonfmt -d sample -ue evmon1 -ftext
```

出力例: ロック・イベントモニター (1/2)

ロックのOwner情報

ロックのRequester情報

```
------------------------------------------------------

Event ID                : 5
Event Type              : LOCKWAIT
Event Timestamp         : 2009-07-02-12.13.55.240137
Partition of detection : 0

------------------------------------------------------


Participant No 1 requesting lock

---------------------------------

Lock Name               : 0x0002000F000000000000000452
Lock Type               : Row
Lock Specifics          : rowID:x0000000000000004
Lock Attributes         : 00000000
Lock mode requested : Update
Lock mode held          : Exclusive
Lock Count              : 1
Lock Hold Count         : 0
Lock rrIID              : 0
Lock Status             : Waiting
Lock release flags  : 40000000
Tablespace TID          : 2
Tablespace Name         : USERSPACE1
Table FID               : 15
Table Schema            : YANAV97
Table Name              : STAFF
```

STAFF表に対する
Update処理をしたいが
ロック待ちになっている

Attributes	Requester	Owner
Participant No	1	2
Application Handle	07269	07268
Application ID	*LOCAL.yanav97.090702030019	*LOCAL.yanav97.090702030016
Application Name	db2bp	db2bp
Authentication ID	YANAV97	YANAV97
Requesting AgentID	6627	6884
Coordinating AgentID	6627	6884
Agent Status	UOW Executing	UOW Waiting
Application Action	No action	No action
Lock timeout value	0	0
Lock wait value	5000	0
Workload ID	1	1
Workload Name	SYSDEFAULTUSERWORKLOAD	SYSDEFAULTUSERWORKLOAD
Service subclass ID	13	13
Service subclass	SYSDEFAULTSUBCLASS	SYSDEFAULTSUBCLASS
Current Request	Execute Immediate	Execute Immediate
TEntry state	1	2
TEntry flags1	00000000	00000000
TEntry flags2	00000200	00000200
Lock escalation	no	no
Client userid		
wrkstnname		
applname		
acctng		

出力例: ロック・イベントモニター （1/2）

ロック保持者はUPDATE処理事態は終わっておりCOMMITしていない状態のため、”Past” acitvitiesに記載されている

```
Current Activities of Participant No 1
--------------------------------------
Activity ID         : 1
Uow ID              : 5
Package Name        : SQLC2H20
Package Schema      : NULLID
Package Version     :
Package Token       : AAAAAZBZ
Package Sectno      : 203
Reopt value         : none
Incremental Bind    : no
Eff isolation       : CS
Eff degree          : 0
Eff locktimeout     : -1
Stmt unicode        : no
Stmt query ID       : 0
Stmt nesting level  : 0
Stmt invocation ID  : 0
Stmt source ID      : 0
Stmt pkgcache ID    : 4361355264
Stmt type           : Dynamic
Stmt operation      : DML, Insert/Update/Delete
Stmt text           : update staff set name='UPDATE2' where id=10
```

このUPDATEが待ちになっている

```
Past Activities of Participant No 2
-----------------------------------

Past Activities wrapped: no

Activity ID         : 1
Uow ID              : 5
Package Name        : SQLC2H20
Package Schema      : NULLID
Package Version     :
Package Token       : AAAAAZBZ
Package Sectno      : 203
Reopt value         : none
Incremental Bind    : no
Eff isolation       : CS
Eff degree          : 0
Eff locktimeout     : -1
Stmt unicode        : no
Stmt query ID       : 0
Stmt nesting level  : 0
Stmt invocation ID  : 0
Stmt source ID      : 0
Stmt pkgcache ID    : 4359651328
Stmt type           : Dynamic
Stmt operation      : DML, Insert/Update/Delete
Stmt text           : update staff set name='UPDATE' where id=10
```

この項目に値を表示するためには、MON_LOCKWAITをHISTORYに設定する

このUPDATEが待ちの原因になっている

4. オプティミスティック・ロッキング拡張サポート（V9.5～）

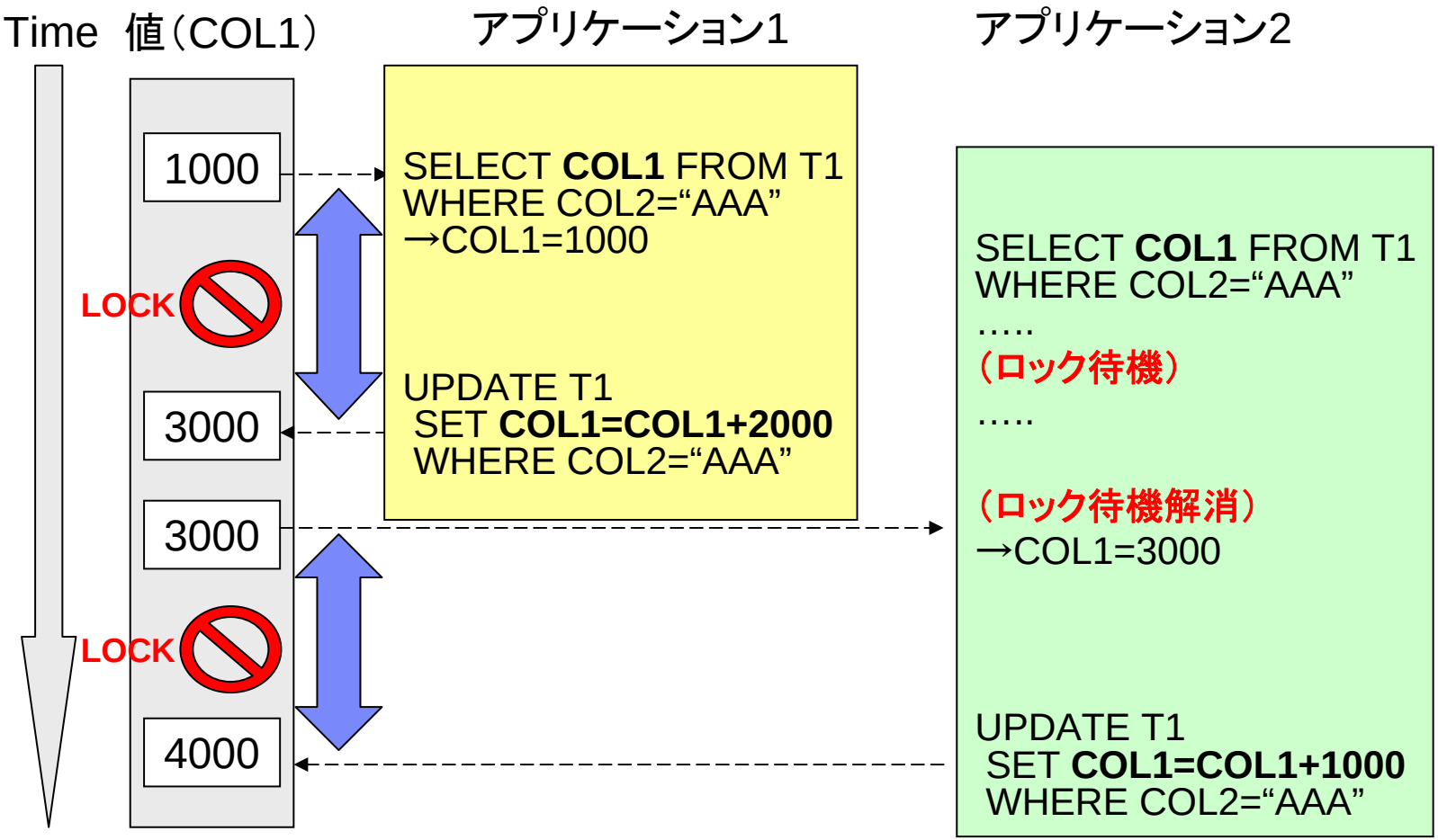
ペシミスティック・ロッキングとオプティミスティック・ロッキングの概念

- あるデータに対して更新を意図した参照を行う場合のロッキング手法
  - 参照したデータの更新を確実に成功させたい場合
    →**ペシミスティック・ロッキング**（悲観的ロッキング）を使用
    - 更新するデータの参照時に更新（U）ロックを取得する（SELECT ... FOR UPDATE）
    - 他のアプリケーションは、参照はできるが更新はできない

  - アプリケーションの同時並行性を向上させたい場合
    →**オプティミスティック・ロッキング**（楽観的ロッキング）を使用
    - 参照から更新の間にロックを保持しない
    - 他のアプリケーションは、参照、更新ともに可能
    - 参照から更新の間に
      - その行が他のアプリケーションによって更新されていなければ、更新は成功する
      - その行が他のアプリケーションによって更新されていれば、更新は失敗する

  - 更新を意図して参照するものの、更新まで実行されないことが多いケースで有効

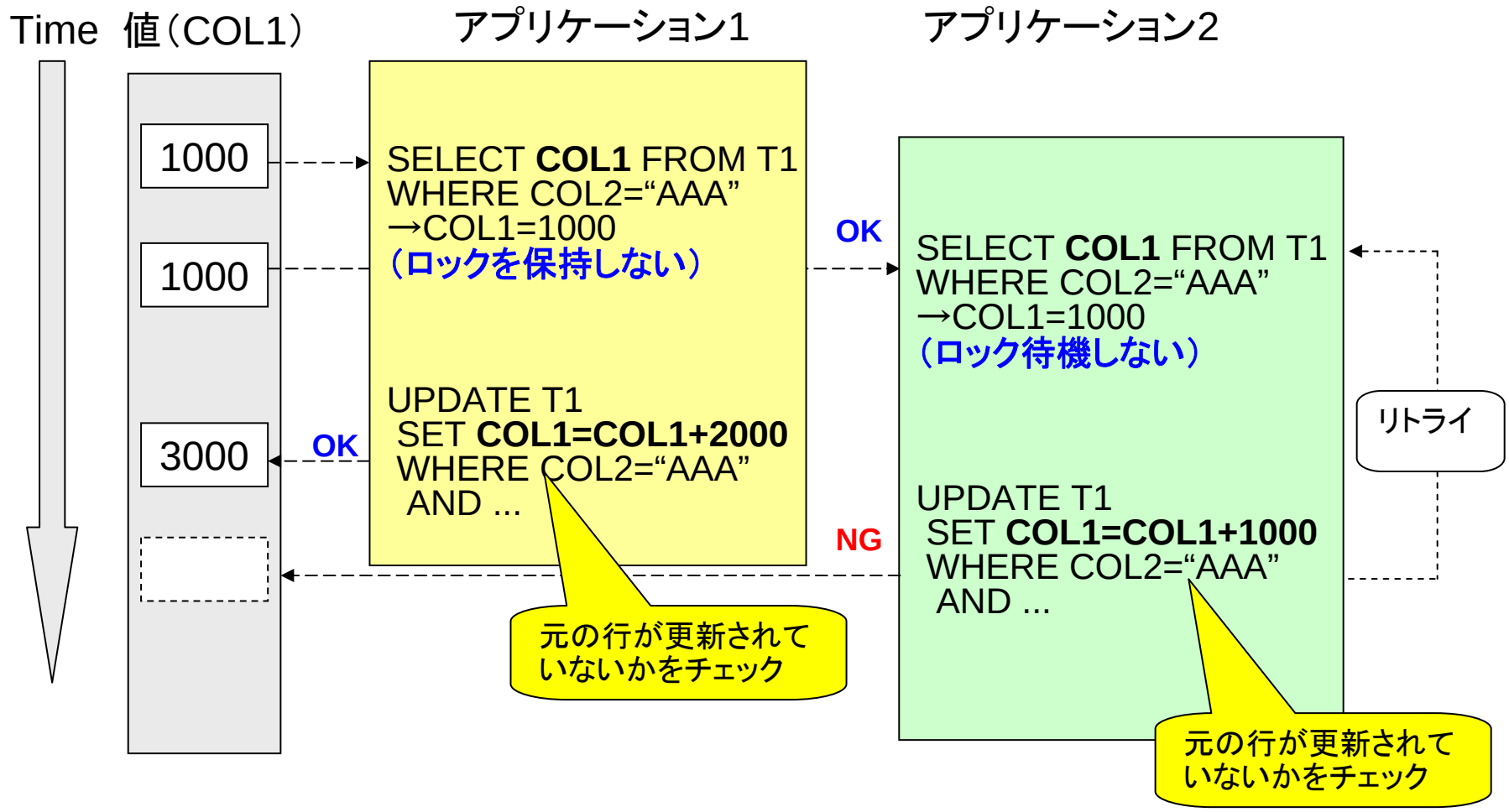
ペシミスティック・ロッキング（悲観的ロッキング）の例



参照～更新の間にロックを保持し、他のアプリケーションからの更新を防ぐ。

→アプリケーションの同時並行性は低下するが、アプリケーション1の更新は確実に成功する。

オプティミスティック・ロッキング（楽観的ロッキング）の例



参照～更新の間にロックを保持せず、他のアプリケーションからも参照、更新が可能。
→アプリケーション1の更新が成功することは必ずしも保証されないが、アプリケーションの同時並行性が向上する。

オプティミスティック・ロッキングの実装方法

- V9.1以前（例）
  - 以下のような方法で、参照から更新の間に他のアプリケーションからの更新が実行されなかったかを判断する
    - 表に更新タイムスタンプを記録するUPDATEトリガーを定義しておき、参照時に取得したタイムスタンプと更新時のタイムスタンプを比較する
    - 参照時に全列をSELECTし、更新時に全列の値を比較する

- V9.5新機能
  - 直近の更新時の情報（RID、行変更トークン）をDB2が各行に保持するようになったため、これらの情報を元に、更新の有無を判断することができる

→これまでより容易に、オプティミスティック・ロッキングを実装することが可能に。

RID_BIT()組み込み関数、ROW CHANGE TOKEN

- **RID_BIT()組み込み関数**（および、RID()組み込み関数）
  - 行のRIDをVARCHAR FOR BIT DATAとして返す
  - SELECTリスト内、もしくは述部ステートメントで使用可能
    - 参考）RID()組み込み関数
      - RIDをBIGINTとして返す
      - z/OSとの互換性のために用意されており、z/OSへ移植されるアプリケーションのみこの関数を使用する
      - DPFでは使用不可
- **ROW CHANGE TOKEN（行変更トークン）**
  - 行が最後に変更された時点を示すトークンをBIGINTとして返す
  - SELECTリスト内、もしくは述部ステートメントで使用可能

```
select col1, RID_BIT(tab1) as row_id, row change token for tab1 as row_change_tok from tab1
```

COL1	ROW_ID	ROW_CHANGE_TOK
1	x'0400000300000000000000000019998AF'	2282930290697437184
2	x'0500000300000000000000000019998AF'	2282930290697437184
3	x'0600000300000000000000000019998AF'	2282930290697437184

RIDと行変更トークンを使用したオプティミスティック・ロッキングの実装

1. SELECTステートメント実行時に、RIDと行変更トークンをSELECTする

```
SELECT SALARY, ROW CHANGE TOKEN FOR EMPLOYEE, RID_BIT (EMPLOYEE)
FROM EMPLOYEE WHERE EMPNO = '000010'
```

2. （ロックを解放する）
3. 1.でSELECTしたRIDと行変更トークンを条件として、ターゲット行に対する更新（UPDATE/DELETE）を実行する（元の行が更新されていないと仮定）

```
UPDATE EMPLOYEE E SET SALARY= SALARY*1.1
 WHERE EMPNO = '000010' AND
          RID_BIT (E) =x'00000000014000040000000000FA9023' AND
          ROW CHANGE TOKEN FOR E =14127277298218153 2"
```

   - 元の行が更新されていなければ、更新は成功
   - 元の行が更新されていれば、更新は失敗（1.からリトライする）

ROW CHANGE TIMESTAMP列の定義

- ROW CHANGE TIMESTAMP 列
  - 行が最後に更新された時刻（更新されていなければ、挿入された時刻）を保持する
  - 生成列として定義する（CREATE TABLE, ALTER TABLE）

GENERATED ALWAYS	**FOR EACH ROW ON UPDATE AS ROW CHANGE TIMESTAMP**
GENERATED BY DEFAULT	**FOR EACH ROW ON UPDATE AS ROW CHANGE TIMESTAMP**

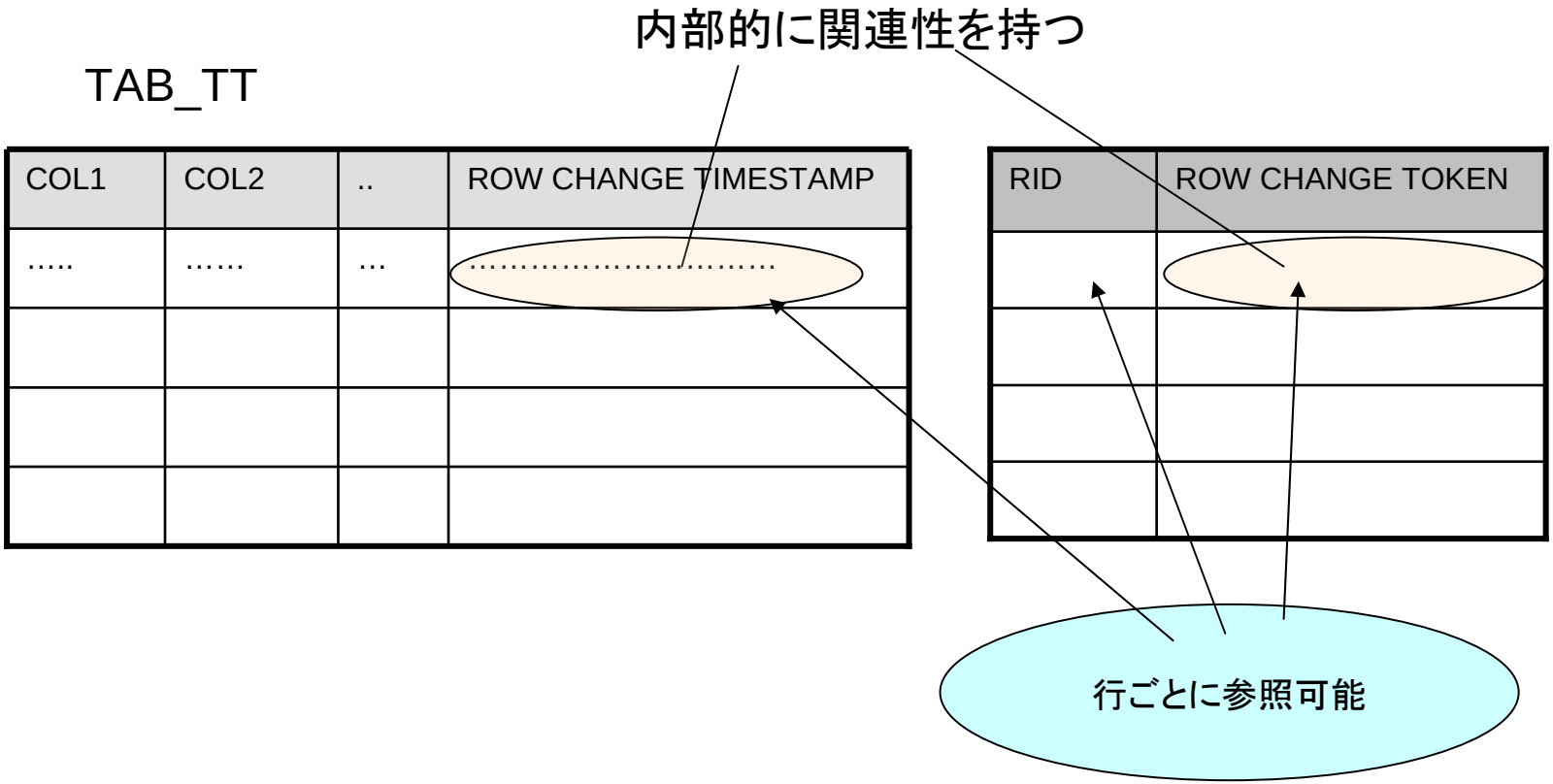
  - 使用方法（SELECTリスト、述部にて指定）
    - 列名を直接指定する
    - ROW CHANGE TIMESTAMP FOR <表名>を使用
  - IMPLICITLY HIDDEN属性を定義し、隠し列にすることが可能
    - 明示的に列を指定したときのみ、列が表示される（”SELECT ＊ FROM . . . “では表示されない）
    - V9.5からの新機能。現時点ではROW CHANGE TIMESTAMP列にのみ使用可能
  - ROW CHANGE TIMESTAMP列は、以下のキー、列、名前ではサポートされない
    - 主キー、外部キー
    - MDCの次元列
    - パーティション表のパーティション・キー
    - データベース・パーティションの分散キー
    - DETERMINED BY（機能従属関係）制約列
    - ニックネーム

参考)使用例

```
$ db2 "create table tab_tt (col1 int, col2 int, rct timestamp not null generated always for each row on update as row
change timestamp)"
DB20000I  SQL コマンドが正常に完了しました。

$ date;db2 "insert into tab_tt(col1,col2) values(1,1)"
Fri Sep 21 16:29:50 JST 2007
DB20000I  SQL コマンドが正常に完了しました。

$ db2 "select col1, rct, row change timestamp for tab_tt as row_change_timestamp from tab_tt"

COL1        RCT                        ROW_CHANGE_TIMESTAMP
----------- -------------------------- --------------------------
          1 2007-09-21-16.29.50.427331 2007-09-21-16.29.50.427331

  1 レコードが選択されました。

$ date;db2 "update tab_tt set col1=11 where col1=1"
Fri Sep 21 16:30:19 JST 2007
DB20000I  SQL コマンドが正常に完了しました。

$ db2 "select col1, rct, row change timestamp for tab_tt as row_change_timestamp from tab_tt"

COL1        RCT                        ROW_CHANGE_TIMESTAMP
----------- -------------------------- --------------------------
         11 2007-09-21-16.30.19.333966 2007-09-21-16.30.19.333966

  1 レコードが選択されました。
```

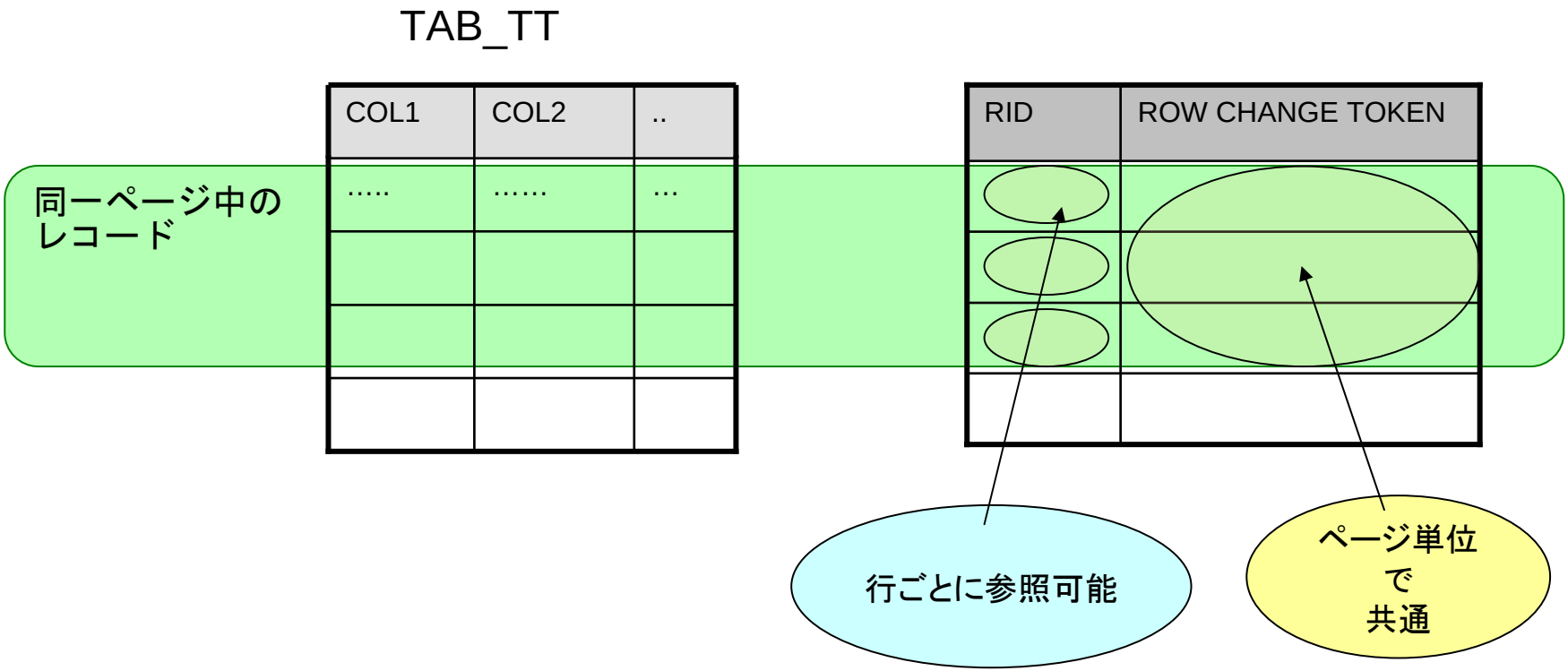
ROW CHANGE TIMESTAMPとROW CHANGE TOKEN

- ROW CHANGE TIMESTAMP列が定義されている場合、ROW CHANGE TOKENの値は、ROW CHANGE TIMESTAMP列から派生する



ROW CHANGE TIMESTAMPとROW CHANGE TOKEN

- ROW CHANGE TIMESTAMP列が定義されて**いない**場合、ROW CHANGE TOKENの値は、データ・ページ中の管理情報から導出される。
  - TIMESTAMP列を定義しない場合、表/ページの構造は変更無し
  - ある行が更新されると、同一ページ内のすべての行のトークンが変わる



ROW CHANGE TIMESTAMPとROW CHANGE TOKEN

- 表にROW CHANGE TIMESTAMP 列が定義されているかどうかで、ROW CHANGE TOKENの細分性が異なる
  - ※同一ページ内の別の行が更新されたことによってROW CHANGE TOKENが変更され、後の更新が失敗しないよう、ROW CHANGE TIMESTAMP列を作成する

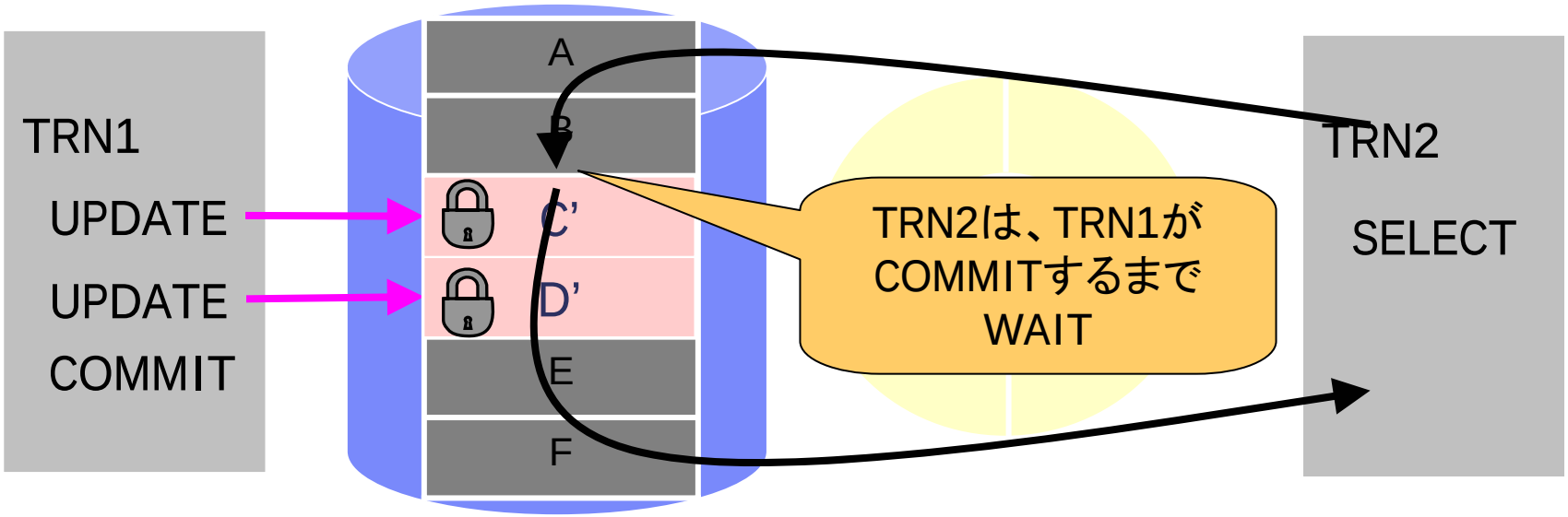
	ROW CHANGE TIMESTAMP列あり	ROW CHANGE TIMESTAMP列なし
ROW CHANGE TOKENの値	ROW CHANGE TIMESTAMPから導出され、行ごとに異なるトークンを持つ （ある行が更新されると、更新された行のみ、トークンが変わる）	ページ・ヘッダ中の情報から導出され、ページ単位でトークンを共有する （ある行が更新されると、同一ページ内のすべての行のトークンが変わる）
false negative	起こらない （例外：REORGによりRIDが変更された場合は起こる）	起こる
表構造の変更	必要あり	必要なし

- false negative：更新されていない行が更新されたと見なされ、後の更新が失敗すること

5. Currently Committedの詳細 (V9.7～)

DB2（V9.5以前）の読み取り一貫性

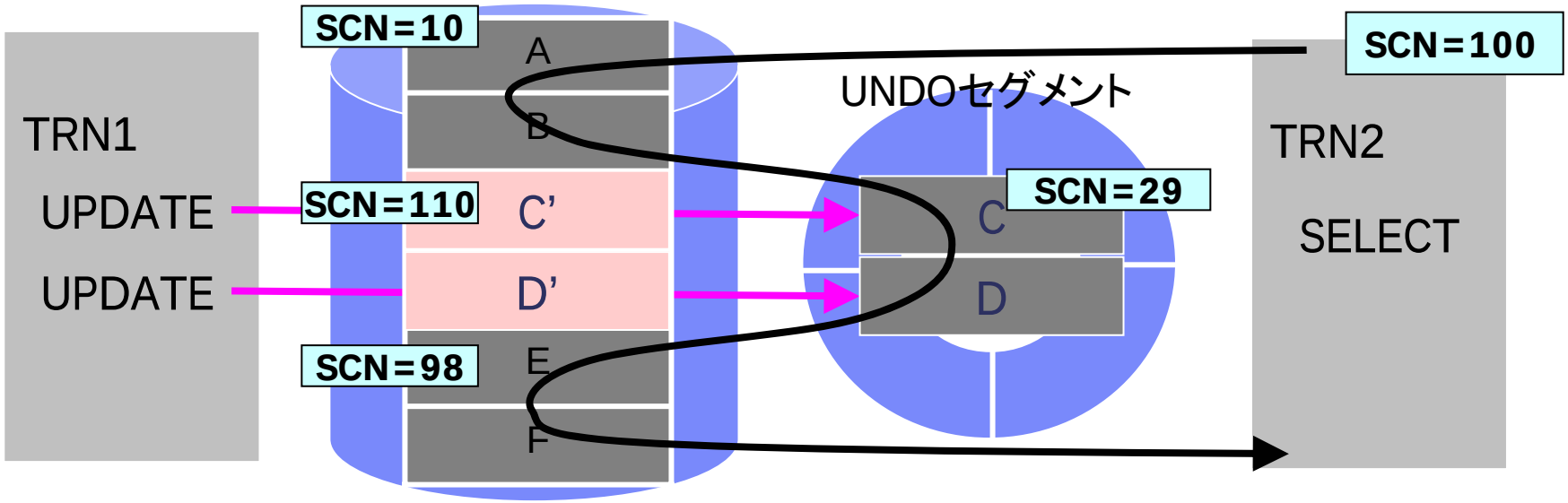
- DB2は最新のデータを読む
  - 参照処理は、更新処理がコミットされるまで待つ



読むデータは常に最新だが、参照処理がロック待機する可能性あり

Oracleの読み取り一貫性（Read Committed）

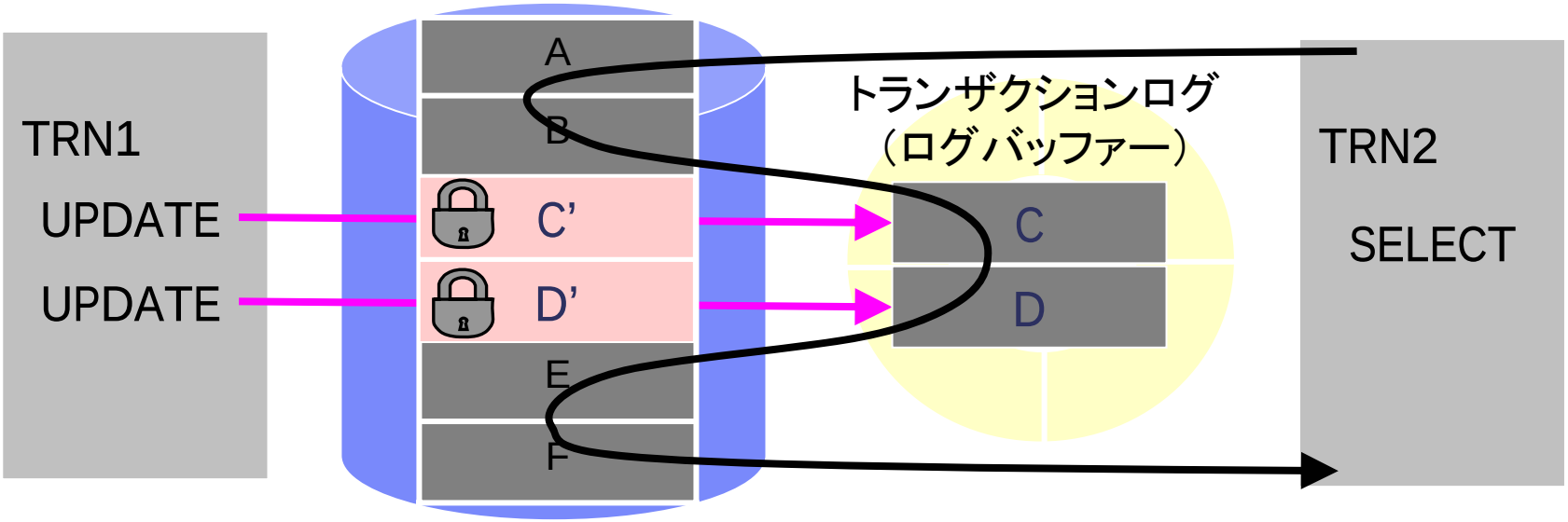
- マルチ・バージョン一貫性制御 (Multi-Version Concurrency Control)
  - 更新処理により、更新前イメージが専用の領域へ書き出される
  - 参照処理はコミットを待たずに更新前イメージを読む
    - Select発行時点でコミット済みのデータを読む
    - 下記の例では、UPDATEがCOMMITされていたとしてもTRN2は更新前データを読む



読んだデータが最新とは限らないが、参照処理はロック待機しない

DB2 9.7の読み取り一貫性(Currently Committed)

- 未コミットの更新があってもロック待機をしない
  - 参照処理は、更新処理に伴うロックの開放を待たず、更新前のデータ(コミット済みの最新データ)をログ(ログバッファー)から読む
  - 常に最新のコミット済みデータを読む



参照処理はロック待機せず、かつコミット済みの最新データを読む

同時実行性の向上: Currently Committed

		別アプリから参照可能か	別アプリから更新可能か
DB2 9.5以前の CS 分離レベル	参照中の行を	Yes	**Maybe**
	更新中の行を	**No**	**No**

		別アプリから参照可能か	別アプリから更新可能か
Oracleの Read Committed分離レベル	参照中の行を	Yes	**Yes**
	更新中の行を	**Yes**	**No**

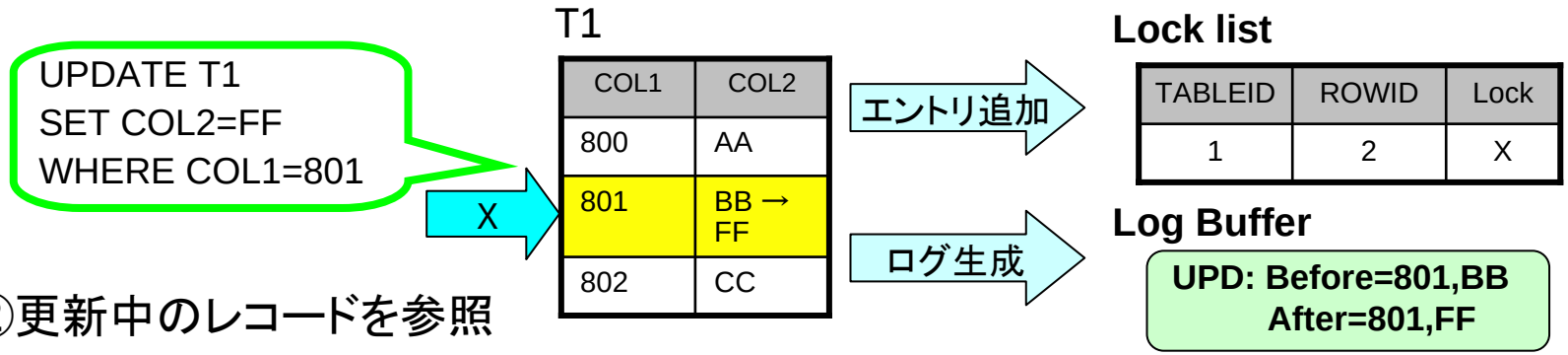
		別アプリから参照可能か	別アプリから更新可能か
DB2 9.7以降の CS分離レベル w/CC	参照中の行を	Yes	**Yes**
	更新中の行を	**Yes**	**No**

Yes : 参照 or 更新可
No : ロック待ち

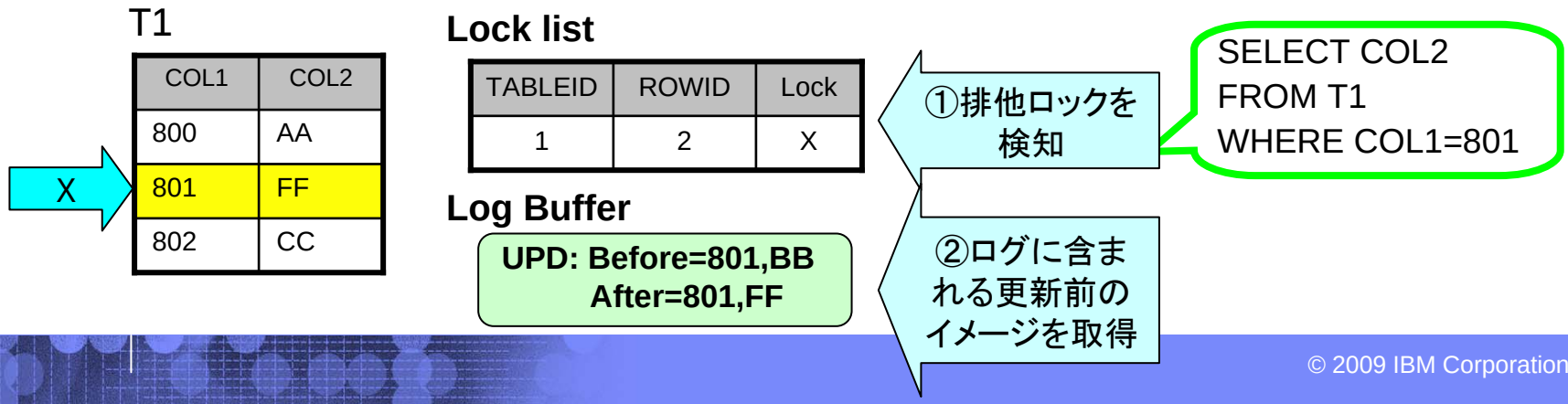
読み取り処理が書き込み処理を妨害しない (ReadがLockを取らない)
書き込み処理が読み取り処理を妨害しない(ReadはLock行をバイパス)

更新前イメージを取得するためのしくみ

- Currently Committedが有効な照会が更新中のレコードをスキャンした場合、DB2はログから更新前のイメージを取り出して返却する
- ①レコードの更新
  - 更新処理は、ロックリストに排他ロックを追加しログバッファーに更新ログを書き込む



- ②更新中のレコードを参照
  - 参照したいレコードの排他ロックを検知すると、データページではなくログを参照する。



ログの検索

- 更新前イメージを取得するためのログ検索の順序
  - ①ログ・バッファー
    - ほとんどのケースではログ・バッファー中に存在するログファイルにヒットする
    - メモリー上に保持されたデータを取得できるため、オーバーヘッドはほとんどなし
  - ②アクティブ・ログ・ファイル
    - ログ・バッファー上に更新ログが存在しない場合、ログ・ファイルから読み込む
      - ログバッファー・フルにより、ログ・バッファーから未コミット更新を含むログ・ページがバッファー中から追い出された場合のみ発生する
  - **アーカイブ・ログへの検索は発生しない**
    - 未コミット更新を含むログ・ファイルは、通常はアクティブ・ログパスに存在し続けるため。
    - 無限ロギングを有効にしているデータベースでは、未コミット更新を含むログ・ファイルがアクティブ・ログパスから無くなるケースがある。しかし、**この場合はアーカイブ・ログのリトリーブは行わずにロック待ちとなる（CCが無効になる）。**

Currently Committedの設定方法

- **Currently Committedを有効にするために**
  - データベース・アプリケーション両方の設定が必要

- 1. データベース構成パラメーターCUR_COMMIT=ONの設定
  - V9.7で新規作成したデータベースではデフォルト
  - 静的SQLを使用している場合、パラメータ変更後に**反映のための再BINDが必須**

- **2. アプリケーションでCCを有効とする**
  - **アプリケーション・ロジックの変更は不要**
  - 動的SQLでは、BLOICKING ALLの設定であればCCが有効となる
    - JDBC、SQLJ、CLI等ではデフォルトでBLOCKING ALL
  - 静的SQLでは、BIND時の指定を推奨
    - カーソルタイプに関わりなく有効にするため、BINDオプションとしてSTATICREADONLY=YESを指定
    - 指定がない場合、あいまいなカーソルでの照会は未コミットINSERTしかスキップしない（UPDATE/DELETEに対してはロック待ちが発生する）

厳密には、カーソルのタイプ等により、CCが有効になるかどうかが異なる。詳細は後続の「カーソルタイプ、パラメータによる動作の違い」を参照

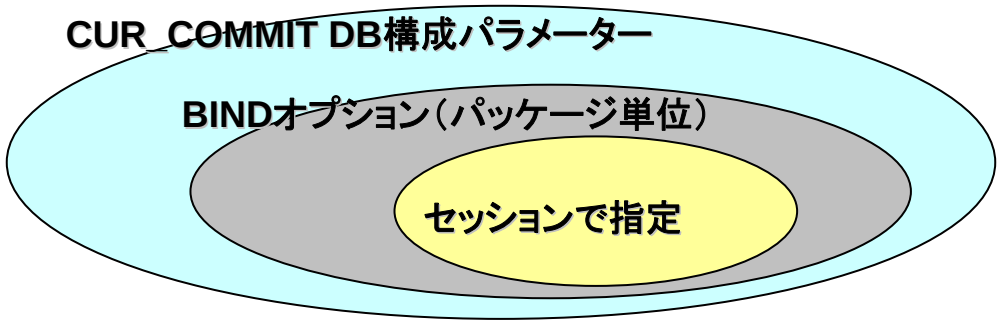
Currently Committedの設定方法

- **CUR_COMMITパラメータの設定による違い**
  - **ONの場合**
    - アプリケーションから明示的な指定がなくても、CCが有効となる
      - CCを無効にしたい場合、パッケージ/セッション単位で指定する
    - V9.7での新規作成データベースでは、ONがデフォルト値
  - **AVAILABLEの場合**
    - アプリケーションから明示的な指定をした場合のみCCが有効となる
    - 下記の様な場合に有効
      - 基本動作をCC無効（V9.5までの動き）とし、特定のアプリケーション（レポーティング処理等）からのSQLだけをCCで使用したい
      - 既存のアプリケーションへの動作はそのまま維持し、新規アプリケーションのみでCCを使用したい
  - **DISABLEの場合**
    - 常にCCが無効となる
      - アプリケーションから指定をしても効果無し
    - V9.5以前のバージョンから移行したデータベースでは、DISABLEがデフォルト値

アプリケーションごとのCurrently Committed設定

- **CC設定の原則**
  - **スコープが狭い設定パラメータがより優先される**
    - DB構成パラメーターで有効になっていても、セッションで無効の指定がある場合は無効になる
  - **アクセスプラン決定時の設定がセクションに書き込まれ、有効になる。**
    - 静的SQLではBIND時の設定が有効になるため注意する

- **セッション単位の指定**
  - JDBC、CLI、.NET等のアプリ・インターフェースからセッション単位で指定可能
    - 例: JDBCの場合、「concurrentAccessResolution」プロパティで設定
      CLI/ODBCの場合、「ConcurrentAccessResolution」 CLIキーワードをdb2cli.iniに記述



Currently Committedの設定確認方法

- CCの有効/無効はセクションに書き込まれるため、EXPLAINを使用して設定内容を確認する。
  - 動的SQLの場合

```
db2expln -d <DB name> -t -f <SQL file>
```

  - 静的SQLの場合

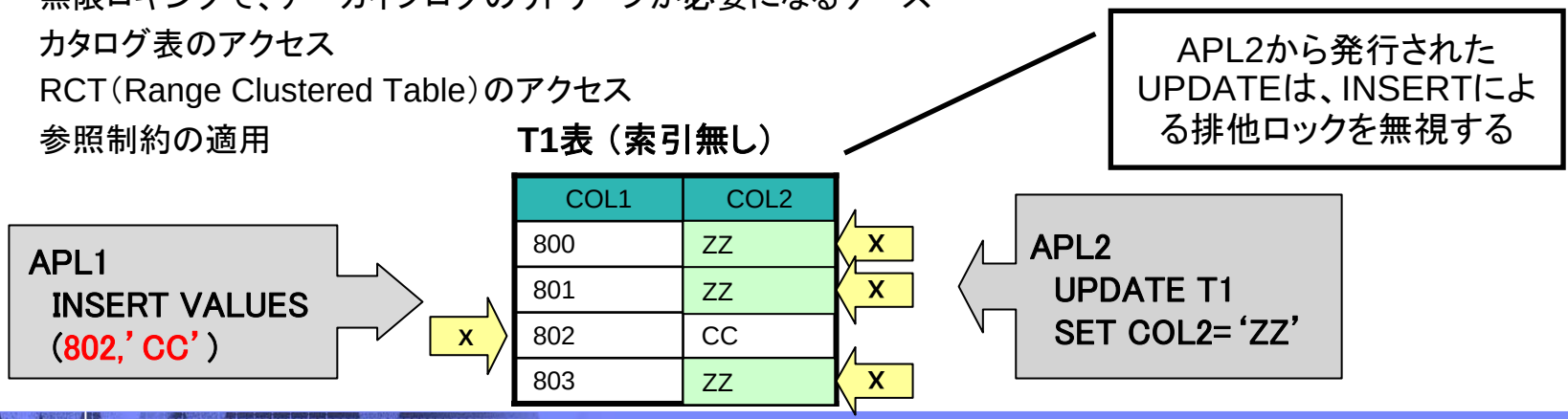
```
db2expln -d <DB name> -t -c <package schema> -p <package name>
```

  - CCが有効な場合の出力例

```
Access Table Name = TUKIV97.T1  ID = 2,19
|  #Columns = 1
|  Skip Inserted Rows
|  Avoid Locking Committed Data
|  Currently Committed for Cursor Stability
...
|  Relation Scan
|  |  Prefetch: Eligible
|  Lock Intents
|  |  Table: Intent Share
|  |  Row  : Next Key Share
...
```

Currently Committed が有効にならないケース

- 分離レベルRS、RRで実行された照会
  - Currently Committedの挙動はRS/RRの要件を満たさないため、CCが無効となる
  - RSによるスキャンの場合、未コミットINSERTのスキップのみ有効となる
- 行レベル以外のロック競合
  - CCによるアクセスであっても、表、パーティション、Blockレベルのロック競合は発生する
  - MDC表からのCell Delete（MDCロールアウト）は、Blockレベルのロックを取得するため、読み取りアクセスとロック競合する
- 更新、削除に伴う表アクセス
  - 更新、削除するデータを特定するための表へのスキャンは、CCの対象外
  - 更新、削除に伴う表アクセスでも、未コミットInsert行はスキップする（下記の図を参照）
- その他
  - Long Varchar/Vargraphic列の処理（SELECT列、条件等）を含むアクセス （LOBはOK）
  - 無限ロギングで、アーカイブログのリトリーブが必要になるケース
  - カタログ表のアクセス
  - RCT（Range Clustered Table）のアクセス
  - 参照制約の適用



Currently Committedを使用する際の考慮点

- **ログ・バッファーを十分に大きくする**
  - CCの導入に伴い、ログ・バッファーサイズ（LOBBUFSZ）のデフォルト値は8から256へ拡張された
    - InfoSphere balanced warehouseでは、LOGBUFSIZ=2048 (8MB)が推奨されている
    - レプリケーションを使用する場合、バッファーサイズが1024を越えるとCPU高騰の懸念があるため注意
  - 更新前イメージがログ・バッファーにヒットしやすくすることで、性能劣化を回避することを推奨
- **ログ量の増加**
  - CCが有効なケースでは更新処理時にレコード全体がロギングの対象となる
    - DB2では更新列のみをロギングしていたためログ量が少なかった
  - 更新量が多いシステムでは、ロギング量の増加による更新処理のパフォーマンス低下に注意
    - 特に、インラインLOBを併用する場合、インライン格納されたLOBを含むレコード全体が更新対象となるため、ログ量が増加しやすい

（参考）カーソルタイプ、パラメータによる動作の違い

- 動的SQLの場合

未コミット更新のスキップ	Cursor type	CUR_COMMITパラメータ	BINDオプション		備考
			CONCURRENTACCESS RESOLUTION	BLOCKING	
有効	読み込み専用	ON	-	全て	読み込み専用カーソルではBLOCKINGを問わずCCが有効
	あいまい	ON	-	B	あいまいなカーソルではBLOCKING=Bが必須
無効	すべて	DISABLED	-	全て	データベース単位でCCが無効となる
	すべて	ON	WAIT FOR OUTCOME	全て	パッケージ単位でCCが無効となる
未コミットINSERTのみスキップ可能	更新を意図	ON	-	全て	
	あいまい	ON	-	N/U	BLOCKING=N/Uの場合、あいまいなカーソルは未コミットUPDATE/DELETEをスキップできない

(参考)カーソルタイプ、パラメータによる動作の違い

- **静的SQLの場合**
  - **静的SQLではBLOCKINGの設定は影響しない**
  - **その代わり、BINDオプションのSTATICREADONLYが重要となる**

未コミット更新のスキップ	Cursor type	CUR_COMMITパラメータ	BINDオプション		備考
			CONCURRENTACCESSRESOLUTION	STATICREADONLY	
有効	読み込み専用	ON	-	すべて	
	あいまい	ON	-	Y	STATICREADONLYD=YESが必要
無効	すべて	ON	WFO	すべて	WFO指定によりパッケージ単位で無効となる
未コミットINSERTのみスキップ	あいまい	ON	-	N	STATICREADONLYDを指定しない場合、あいまいなカーソルではUPDATE/DELETEのスキップができない。
	更新意図	ON	-	すべて	

  - **BIND時とSQL実行時でCUR_COMMITパラメータの設定が一致しないケース。**
    - **CUR_COMMITパラメーターの変更時は再バインドを推奨**

未コミット更新のスキップ	Cursor type	CUR_COMMITパラメータ		備考
		BIND時	SQL実行時	
無効	すべて	DISABLED	ON	
未コミットINSERTのみスキップ可能	すべて	ON	DISABLED	CUR_COMMITパラメータがDISABLEDでも未コミットInsertがスキップされるため注意。

（参考）既存のレジストリ変数とCurrently Committedとの関係

- CSのreadスキャンでは、CCの挙動に包含される
  - そのため、レジストリー変数による挙動の変化無し
- CSのWriteスキャン、RSのRead/Writeスキャンでは、
  - Implicit CCの場合のみ有効となる（未コミットINSERTはCCに包含）
- WAIT FOR OUTCOMEの設定では、
  - レジストリー変数の効果をパッケージ単位で抑止可能

BINDオプション / レジストリー変数の設定	CC暗黙・明示指定 CS Read Only	CC暗黙指定 CS Write RS Read/Write	CC明示指定 CS Write RS Read/Write	Wait For Outcome CS/RS
DB2_SKIPINSERTED	CCに含まれるため、未コミットの挿入行はスキップ	CCに含まれるため、未コミットの挿入行はスキップ	CCに含まれるため、未コミットの挿入行はスキップ	コミットを待機
DB2_SKIPDELETED	スキップせずに未コミットの削除行を返す	未コミットの削除行をスキップ	コミットを待機	コミットを待機
DB2_EVALUNCOMMITTED	最新のコミット済みデータを使って述部評価	未コミットのデータを使って述部評価	コミットを待機	コミットを待機

CC暗黙指定：BINDオプションの指定無しに、CUR_COMMIT構成パラメーターにより有効となったCC
CC明示指定：BINDオプションで明示的にUSE CURRENTLY COMMITTEDを指定した

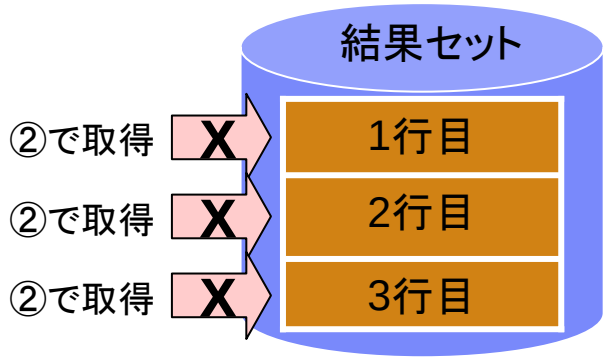
Oracleのロック動作との違い

Oracleのロック動作との違い ― FOR UPDATE句の違い

- 最新のデータを読む場合/最新のデータに基づきデータを更新したい場合、Oracleでは予めSELECT FOR UPDATE を実行して適合行をロックしておくのが一般的
  - OPEN CURSOR時に、結果セット全てに対してXロック

Oracle

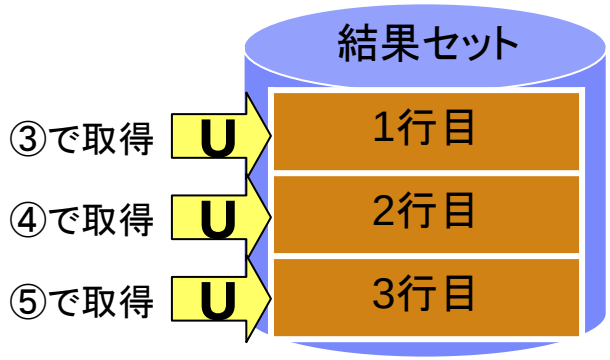
①DCL CSR FOR UPDATE
②OPEN CSR
③1行目FETCH
…



- DB2ではOPEN CURSOR時ではなく実際のFETCH時にロックを獲得する
  - アプリケーション設計上、結果セット全体を排他ロックする必要がある場合は、WITH RS (USE AND KEEP ~ LOCKS)を指定した上で、全行Fetchすればよい
  - CSによるスキャンでは、FOR UPDATEが付いていてもFETCHが終わったレコードからはロックを解放する（Oracleはコミットまで保持する）

DB2

①DCL CSR
WITH RS USE AND KEEP UPDATE LOCKS
②OPEN CSR
③1行目FETCH
④2行目FETCH
⑤3行目FETCH



Oracleのロック動作との違い ― FOR UPDATE句の違い

- OracleではSQL単位にタイムアウト値を設定する
  - SELECT * FROM EMP WHERE EMPNO=100 FOR UPDATE **NOWAIT**

- DB2ではトランザクション単位でタイムアウト値を設定する
  - DB2ではSET LOCK TIMEOUTステートメントを使用して、それ以降にセッションで使用されるロック待ちの振る舞いを指定
  - SET LOCK TIMEOUT
    - SET LOCK TIMEOUT WAIT
      - -1に設定。開放までずっと待つ。もしくはデッドロックが起こるまで。
    - SET LOCK TIMEOUT NOT WAIT ←Oracleの NOWAIT相当
      - ロック待ちせず、SQL0911を発生
    - SET LOCK TIMEOUT integer
      - 指定した秒数の間、ロックの解放を待つ
    - SET LOCKTIMEOUT NULL
      - DB構成パラメーターのLOCKTIMEOUT値を使用する（デフォルトの動作）。

注：1ステートメントだけに適用する場合はSET LOCKTIMEOUT NULLでデフォルト動作に戻す

Oracleのロック動作との違い ― その他

- デッドロックやタイムアウトのメッセージや振る舞いの相違
  - Oracleのデッドロック： ORA-00060
    - デッドロック時は当該のSQL文のみロールバックされる
  - Oracleのタイムアウト： ORA-00054/ORA-30006