

**IBM i**  
バージョン 7.2

**データベース  
コミットメント制御**

**IBM**



**IBM i**  
バージョン 7.2

**データベース  
コミットメント制御**

**IBM**

**お願い**

本書および本書で紹介する製品をご使用になる前に、127ページの『特記事項』に記載されている情報をお読みください。

本製品およびオプションに付属の電源コードは、他の電気機器で使用しないでください。

本書にはライセンス内部コードへの参照が含まれる場合があります。ライセンス内部コードはマシン・コードであり、マシン・コードの IBM のご使用条件に基づいて使用許諾されます。

お客様の環境によっては、資料中の円記号がバックslashと表示されたり、バックslashが円記号と表示されたりする場合があります。

原典： IBM i  
Version 7.2  
Database  
Commitment control

発行： 日本アイ・ビー・エム株式会社

担当： トランスレーション・サービス・センター

第1刷 2014.4

© Copyright IBM Corporation 1998, 2013.

# 目次

コミットメント制御	1	コミットメント制御の開始	53
コミットメント制御の PDF ファイル	1	通知オブジェクトのコミット	55
コミットメント制御の概念	2	ロック・レベルのコミット	56
コミットメント制御の機能	2	コミットメント制御の終了	59
コミット操作およびロールバック操作の機能	3	システム起動によるコミットメント制御の終了	61
コミット操作	3	活性化グループの終了時のコミットメント制御	61
ロールバック操作	4	暗黙コミットおよびロールバック操作	61
コミットメント定義	5	経路指定ステップの正常終了時のコミットメント制御	65
コミットメント定義の有効範囲	6	システムまたはジョブの異常終了時のコミットメント制御	65
コミットメント定義名	9	通知オブジェクトの更新	67
例：ジョブおよびコミットメント定義	10	異常終了後の初期プログラム・ロード中のコミットメント制御リカバリー	69
コミットメント制御によるオブジェクトの処理方法	12	トランザクションおよびコミットメント制御の管理	71
コミット可能リソースのタイプ	13	コミットメント制御情報の表示	71
ローカルおよびリモート・コミット可能リソース	15	トランザクションのロックされたオブジェクトの表示	72
コミット可能リソースのアクセス・インテント	15	トランザクションに関連付けられたジョブの表示	72
コミット可能リソースのコミット・プロトコル	16	トランザクションのリソース状況の表示	73
ジャーナル・ファイルおよびコミットメント制御	17	トランザクションのプロパティの表示	73
コミットメント制御下でのジャーナル項目の順序	17	コミットメント制御のパフォーマンスの最適化	73
コミット・サイクル ID	20	ロックの最小化	76
レコードのロック	21	トランザクション・サイズの管理	78
コミットメント制御および独立ディスク・プールの考慮事項	22	ソフト・コミット	79
コミットメント定義の独立ディスク・プールの考慮事項	23	コミットメント制御のシナリオおよび例	80
XA トランザクションの独立ディスク・プールの考慮事項	25	シナリオ：コミットメント制御	80
コミットメント制御に関する考慮事項と制約事項	25	コミットメント制御の実習問題	83
バッチ・アプリケーションのコミットメント制御	27	実習問題のロジックのフロー	90
2 フェーズ・コミットメント制御	28	実習プログラムのロジックのフローに関連したステップ	91
コミット処理での役割	29	例：アプリケーション開始のためのトランザクション・ログ記録ファイルの使用	93
2 フェーズ・コミットメント制御のトランザクションの状態	32	例：アプリケーション開始のための通知オブジェクトの使用	97
2 フェーズ・コミットメント制御のコミットメント定義	35	例：プログラムごとに固有の通知オブジェクト	99
2 フェーズ・コミットのコミットメント定義：読み取り専用ポートの許可	35	例：全プログラム用の単一通知オブジェクト	104
2 フェーズ・コミットのコミットメント定義：結果を待機しない	37	例：アプリケーション開始のための標準処理プログラムの使用	105
2 フェーズ・コミットのコミットメント定義：「流用可能」の指示	41	例：標準処理プログラムのコード	105
2 フェーズ・コミットのコミットメント定義：最後のエージェントを選択しない	43	処理のフロー	107
コミット処理のフローに影響するポート信頼性	44	例：標準コミット処理プログラムのコード	108
コミットメント制御の XA トランザクション・サポート	46	例：アプリケーションを再始動するかどうかを決定するための標準処理プログラムの使用	110
コミットメント制御のための SQL サーバー・モードおよびスレッド範囲トランザクション	52	トランザクションおよびコミットメント制御のトラブルシューティング	111
		コミットメント制御エラー	112
		エラー状態	112
		非エラー状態	114

コミットメント制御中にモニターすべきエラー・メッセージ . . . . .	114
CALL コマンド後のエラーのモニター . . . . .	117
通常のコミットまたはロールバック処理の障害 . . . . .	118
デッドロックの検出 . . . . .	120
通信障害後のトランザクションのリカバリー . . . . .	121
コミットとロールバック操作の強制時点および再同期の取り消し時点 . . . . .	122

長時間を要するロールバックの終了 . . . . .	123
大規模トランザクションまたは古いトランザクションを検出する . . . . .	124
コミットメント制御の関連情報 . . . . .	125

**特記事項. . . . . 127**

プログラミング・インターフェース情報 . . . . .	129
商標 . . . . .	129
使用条件 . . . . .	129

---

## コミットメント制御

コミットメント制御は、データ安全性を保証するための機能です。コミットメント制御は、データベース・ファイルまたはテーブルなどのリソースへの変更をまとめてトランザクションとして定義し、処理します。

コミットメント制御は、個々の変更をまとめたグループ全体の変更がトランザクションに参加しているすべてのシステムで確実に生じることを保証するか、またはその変更が何も生じないことを保証します。DB2® for IBM® i は、コミットメント制御機能を使用して、\*NONE (コミットなし) 以外の分離レベルで実行中のデータベース・トランザクションをコミットおよびロールバックします。

コミットメント制御を使用して、ジョブ、ジョブ内の活動化グループ、またはシステムが異常終了する場合にシステムが再始動できるようにアプリケーションを設計することができます。コミットメント制御を使用すると、アプリケーションを再び開始させた時に、以前の障害に起因する未完了のトランザクションのためにデータベースに部分的な更新が残ることがないという保証を得られます。

注: コード例を使用すると、126 ページの『コードに関するライセンス情報および特記事項』の条件に同意したことになります。

---

## コミットメント制御の PDF ファイル

この情報の PDF ファイルを表示または印刷できます。

本書の PDF 版を表示またはダウンロードするには、コミットメント制御を選択します。

### PDF ファイルの保存

表示または印刷のために PDF をワークステーションに保存するには、以下のようになります。

1. ご使用のブラウザで PDF リンクを右クリックする。
2. PDF をローカルに保存するオプションをクリックする。
3. PDF を保存したいディレクトリーに進む。
4. 「保存」をクリックする。

### Adobe Reader のダウンロード

これらの PDF を表示または印刷するには、Adobe Reader がご使用のシステムにインストールされている

必要があります。Adobe の Web サイト ([www.adobe.com/products/acrobat/readstep.html](http://www.adobe.com/products/acrobat/readstep.html))  から無償コピーをダウンロードできます。

#### 関連資料:

125 ページの『コミットメント制御の関連情報』

製品マニュアル、IBM Redbooks® 資料、Web サイト、およびその他の Information Center トピック・コレクションには、コミットメント制御トピック集に関連する情報が収められています。以下の PDF ファイルのいずれも表示または印刷できます。

---

## コミットメント制御の概念

以下のコミットメント制御の概念を、コミットメント制御の機能、システムとの相互作用、およびネットワーク内の他のシステムとの相互作用の理解に役立ててください。

## コミットメント制御の機能

コミットメント制御は、個々の変更をまとめたグループ全体の変更が、関係するすべてのシステム上で生じるか、その変更が何も生じないかのいずれかを保証します。

例えば、普通預金口座から当座預金口座に預金を移す場合、複数回の変更がグループとして発生します。ユーザーにとっては、この移動は単一の変更に見えます。ところが、データベースにとっては、普通預金口座および当座預金口座が更新されるため、複数の変更が発生します。両方の口座を正確に保持するには、普通預金口座および当座預金口座に対してすべての変更が行われるか、あるいはまったく変更が行われないかのいずれかでなければなりません。

コミットメント制御により、以下のタスクを実行できます。

- 影響を受けるすべてのリソースに対するトランザクション内のすべての変更の完了を確認する。
- 処理が中断する場合にトランザクション内のすべての変更が除去されることを確認する。
- アプリケーションがトランザクションにエラーがあることを判別するときにトランザクション実行中になされた変更を除去する。

ジョブ、ジョブ内の活動化グループ、またはシステムが異常終了する場合にコミットメント制御がアプリケーションを再始動できるようにアプリケーションを設計することもできます。コミットメント制御を使用すると、アプリケーションを再び開始させた時に、以前の障害に起因する未完了のトランザクションのためにデータベースに部分的な更新が残ることがないという保証を得られます。

## トランザクション

トランザクションは、システム上のオブジェクトへの個々の変更内容をまとめたもので、ユーザーには 1 つの最小の変更に見えます。

**注:** IBM Navigator for i ではトランザクション という用語を使用しますが、文字ベース・インターフェースでは作業論理単位 (LUW) という用語を使用します。2 つの用語は同義です。このトピックでは、文字ベース・インターフェースに明確に関係する場合を除いて、トランザクションという用語を使用します。

トランザクションには、以下の状態のものがあります。

- データベース・ファイルの変更が起こらないような照会。
- 1 つのデータベース・ファイルを変更する単純なトランザクション。
- 1 つ以上のデータベース・ファイルを変更する複雑なトランザクション。
- 1 つ以上のデータベース・ファイルを変更する複雑なトランザクションであるが、トランザクションの論理グループの一部のみに変更がある場合。
- 2 つ以上の位置でのデータベース・ファイルを含む単純または複雑なトランザクション。これらのデータベース・ファイルは、以下のいずれか 1 つの状態にあることが可能です。
  - 単一のリモート・システム上にある。
  - ローカル・システム上および 1 つ以上のリモート・システム上にある。
  - ローカル・システム上の複数のジャーナルに割り当てられる。各ジャーナルはローカル・ロケーションと見なされます。



- データベース・ファイル以外のオブジェクトを含むローカル・システムでの単純または複雑なトランザクション。

## コミット操作およびロールバック操作の機能

コミット操作とロールバック操作は、コミットメント制御下で行われた変更に影響します。

次のプログラム言語およびアプリケーション・プログラミング・インターフェース (API) では、コミット操作およびロールバック操作がサポートされています。

言語または API	コミット	ロールバック
CL	COMMIT コマンド	ROLLBACK コマンド
IBM 統合化言語環境 (Integrated Language Environment®) (ILE) RPG	COMIT 操作コード	ROLBK 操作コード
ILE COBOL	COMMIT verb	ROLLBACK verb
ILE C	_Rcommit 機能	_Rrollbck 機能
PLI	PLICOMMIT サブルーチン	PLIROLLBACK サブルーチン
SQL	COMMIT ステートメント	ROLLBACK ステートメント
SQL 呼び出しレベル・インターフェース (CLI)	SQLTransact() 関数 (トランザクションのコミットおよびロールバックに使用)	
XA API	xa_commit() および db2xa_commit() API	xa_rollback() および db2xa_rollback() API

### 関連概念:

SQL コール・レベル・インターフェース  
データベース・プログラミング

### 関連情報:



WebSphere Development Studio: ILE C/C++ プログラマーの手引き (PDF)

CL プログラミング

アプリケーション・プログラミング・インターフェース

## コミット操作

直前のコミット操作またはロールバック操作以降にコミットメント制御下で行われたすべての変更が確定されます。システムは、トランザクションに関連したすべてのロックも解除します。

システムはコミット要求を受け取ったときに、以下のステップを実行します。

- システムがコミット ID を持つ場合、その ID をリカバリー時に使用するために保管します。
- 次の条件がどちらも当てはまる場合、システムはコミット操作を実行する前に、レコードをファイルに書き込みます。
  - レコードがコミットメント制御下でローカルまたはリモート・データベース・ファイルに追加された。
  - さらに、ブロック化入出力フィードバックがシステムによって使用され、レコードの部分的ブロックが存在するように、ファイルのオープン時に SEQONLY(\*YES) が指定された。

上記の場合以外には、入出力フィードバックと入出力バッファは変更されません。

- コミットメント定義にある各 API コミットメント・リソースごとに、コミットおよびロールバックの出口プログラムを呼び出します。ロケーションに複数の出口プログラムが登録されている場合には、そのロケーションの出口プログラムを、登録されている順序で呼び出します。
- ジャーナルに割り当てられたリソースのレコードが変更されると、コミットメント定義に関連付けられたローカル・ジャーナルすべてに C CM ジャーナル項目を書き込みます。コミットメント定義の下でのジャーナル項目の順序に、コミットメント定義がアクティブの間に書き込まれる典型的な項目が示されています。
- 保留されている永続オブジェクト・レベルの変更を確定します。
- コミットメント制御の目的で獲得および保持が行われていたレコード・ロックおよびオブジェクト・ロックをアンロックします。これらのリソースは、他のユーザーで使用可能になります。
- 現行のトランザクションが終了したことを示すために、コミットメント定義の情報を変更します。

コミット操作を正常に実行するために、上記のすべてのステップが正しく実行されなければなりません。

#### 関連概念:

5 ページの『コミットメント定義』

コミットメント定義には、トランザクション中にコミットメント制御下で変更されるリソースに関する情報が入っています。

17 ページの『コミットメント制御下でのジャーナル項目の順序』

この表には、コミットメント定義がアクティブな場合に書き込まれる典型的な項目の順序が示されています。ジャーナル項目情報ファインダーを使用して、ジャーナル項目の内容の詳細を入手することができます。

## ロールバック操作

直前のコミット操作またはロールバック操作以降に行われたすべての変更が除去されます。システムは、トランザクションに関連したすべてのロックも解除します。

システムは、ロールバック要求を受け取ると、次のステップを実行します。

- 次の両方の条件に該当する場合、システムは入出力バッファからレコードを消去します。
  - コミットメント制御下でローカルまたはリモート・データベース・ファイルにレコードが追加された場合。
  - さらに、ブロック化入出力がシステムによって使用され、まだデータベースに書き込まれていないレコードの部分的ブロックが存在するように、ファイルのオープン時に SEQONLY(\*YES) が指定された場合。

上記以外の場合には、入出力フィールドバック域と入出力バッファは未変更のままです。

- コミットメント定義にある各 API コミットメント・リソースごとに、コミットまたはロールバックの出口プログラムを呼び出します。ロケーションに複数の出口プログラムが登録されている場合、そのロケーションの出口プログラムを、登録されている順序の逆順で呼び出します。
- レコードがファイルから削除された場合は、そのレコードを再びそのファイルに追加します。
- このトランザクション中にレコードに対して行われた変更を除去し、元のレコード (変更前イメージ) をファイルに戻します。
- このトランザクション中にレコードがファイルに追加された場合、そのレコードは削除済みレコードとしてファイル内に残ります。
- トランザクション中にジャーナルに割り当てられたリソースに対して何らかのレコード変更が行われた場合には、ロールバック操作が行われたことを示すためにジャーナル項目 C RB をジャーナルに追加します。ジャーナルにはまた、ロールバックされたレコード変更のイメージも含まれます。ロールバック

操作を要求する前に、変更済みレコードの変更前イメージと変更後イメージはジャーナルに入れられています。コミット可能リソースがデフォルトのジャーナルに割り当てられている場合には、C RB 項目もデフォルトのジャーナルに書き込まれます。

- コミットメント制御下でオープンされているファイルを、次のいずれかに位置付けます。
  - 直前のトランザクションでアクセスされた最後のレコード
  - このコミットメント定義を使ってコミット操作をファイルに行っていない場合は、オープン位置この考慮事項は、順次処理を行う場合には重要です。
- データベース・ファイルに対してコミット可能でない変更については、ロールバックしません。例えば、オープンされたファイルはクローズされず、また消去されたファイルは復元されません。このトランザクション中にクローズされたファイルは、再オープンまたは再位置付けされません。
- コミットメント制御の目的で獲得されたレコード・ロックをアンロックして、他のユーザーがそれらのレコードを使用できるようにします。
- このシステムによって保管されているコミット ID は、同一のコミットメント定義の最後のコミット操作で与えられたコミット ID と同じままになります。
- このトランザクション中に行われたオブジェクト・レベルのコミット可能な変更を、無効にするかまたはロールバックします。
- コミットメント制御の目的で獲得されたオブジェクト・ロックはアンロックされ、他のユーザーはそれらのオブジェクトを使用できるようになります。
- 前のコミットメント境界を現行のコミットメント境界として設定します。
- 現行のトランザクションが終了したことを示すために、コミットメント定義の情報を変更します。

ロールバック操作を正常に実行するために、上記のすべてのステップが正しく実行されなければなりません。

## コミットメント定義

コミットメント定義には、トランザクション中にコミットメント制御下で変更されるリソースに関する情報が入っています。

コミットメント定義を作成するには、コミットメント制御の開始 (STRCMTCTL) コマンドを使用して、システム上でコミットメント制御を開始します。また、\*NONE (コミットなし) 以外の分離レベルの場合は、DB2 for i によってコミットメント定義が自動的に作成されます。

コミットメント定義内のコミットメント制御情報は、そのコミットメント定義が終了するまで、コミットメント・リソース変更としてシステムによって維持されます。システム上のアクティブな各トランザクションは、コミットメント定義によって表されます。後続のトランザクションでは、アクティブなトランザクションのコミットまたはロールバックが終わるたびにコミットメント定義の再利用が可能です。

一般的に、コミットメント定義には次の情報が含まれます。

- STRCMTCTL コマンドのパラメーター
- コミットメント定義の現在の状況
- データベース・ファイルおよび現行トランザクション中になされた変更を含む他のコミット可能リソースについての情報

ジョブ範囲ロックを使用したコミットメント定義の場合は、コミットメント制御を開始するジョブのみがコミットメント定義を認識できます。他のジョブはそのコミットメント定義を認識できません。

複数のプログラムが複数のコミットメント定義を開始および使用することが可能です。ジョブの各コミットメント定義は、それに関連するコミット可能リソースを持つ別々のトランザクションを識別します。これらのトランザクションは、ジョブのために開始された他のコミットメント定義に関連するトランザクションとは別にコミットまたはロールバックすることができます。

#### 関連概念:

3 ページの『コミット操作』

直前のコミット操作またはロールバック操作以降にコミットメント制御下で行われたすべての変更が確定されます。システムは、トランザクションに関連したすべてのロックも解除します。

22 ページの『コミットメント制御および独立ディスク・プール』

独立ディスク・プールおよび独立ディスク・プール・グループは、それぞれ別々の i5/OS™ データベースを持つことが可能です。これらのデータベースに対してコミットメント制御を使用することができます。

23 ページの『コミットメント定義の独立ディスク・プールの考慮事項』

独立ディスク・プールを使用する場合は、コミットメント定義についての考慮事項に注意する必要があります。

## コミットメント定義の有効範囲

コミットメント定義の有効範囲により、当該コミットメント定義を使用するプログラム、およびトランザクションの有効範囲限定中にロックを獲得する方法が決定します。

コミットメント定義を開始するインターフェースにより、コミットメント定義の有効範囲が決定されます。以下のように、コミットメント定義には 4 種類の有効範囲があり、それらは全体に 2 つに分類されます。

### ジョブ範囲ロックを使用したコミットメント定義

- 活動化グループ・レベルのコミットメント定義
- ジョブ・レベルのコミットメント定義
- 明示指定のコミットメント定義

### トランザクション範囲ロックを使用したコミットメント定義

- トランザクション範囲コミットメント定義

ジョブ範囲ロックを使用したコミットメント定義は、コミットメント定義を開始させたジョブで実行するプログラムでのみ使用できます。一方、トランザクション範囲ロックを使用したコミットメント定義は、複数のジョブで使用できます。

アプリケーションは、通常、活動化グループ・レベルまたはジョブ・レベルのいずれかのコミットメント定義を使用します。これらのコミットメント定義は、コミットメント制御の開始 (STRCMTCTL) コマンドを使用して明示的に、または \*NONE 以外の分離レベルで SQL アプリケーションを実行する時にシステムによって暗黙的に作成されます。

## 活動化グループ・レベルのコミットメント定義

最も一般的な有効範囲は、活動化グループに対するものです。活動化グループ・レベルのコミットメント定義は、STRCMTCTL コマンドによってコミットメント定義を明示的に開始させる場合、または、\*NONE (コミットなし) 以外の分離レベルで実行する SQL アプリケーションがコミットメント定義を暗黙的に開始させる場合のデフォルトの有効範囲です。その活動化グループ内で実行されるプログラムだけが、そのコミットメント定義を使用します。ある 1 つのジョブに対して、一度に多数の活動化グループ・レベル・コミットメント定義をアクティブにすることができます。ただし、活動化グループ・レベル・コミットメント

定義はおのおの、1つの活動化グループとしか関連付けることができません。その活動化グループ内で稼働するプログラムは、そのプログラムのコミット可能変更を、その活動化グループ・レベル・コミットメント定義としか関連付けることはできません。

IBM Navigator for i、コミットメント定義処理 (WRKCMDFN) コマンド、ジョブ表示 (DSPJOB) コマンド、またはジョブ処理 (WRKJOB) コマンドが活動化グループ・レベルのコミットメント定義を表示する場合、これらのフィールドに次の情報が表示されます。

- コミットメント定義フィールドに活動化グループの名前が表示されます。特殊値 \*DFACTGRP が表示されて、デフォルトの活動化グループであることが示されます。
- 活動化グループ・フィールドに活動化グループ番号が表示されます。
- ジョブ・フィールドにコミットメント定義を開始したジョブが表示されます。
- スレッド・フィールドに \*NONE が表示されます。

### ジョブ・レベルのコミットメント定義

STRCMTCTL CMTSCOPE(\*JOB) を実行することにより、コミットメント定義をジョブのみの範囲内に限定することができます。活動化グループ・レベル・コミットメント定義が開始していない活動化グループで稼働するすべてのプログラムは、それがジョブの別のプログラムによって既に開始されている場合、ジョブ・レベルのコミットメント定義を使用します。1つのジョブに対して、1つのジョブ・レベル・コミットメント定義だけを開始することができます。

IBM Navigator for i、コミットメント定義処理 (WRKCMDFN) コマンド、ジョブ表示 (DSPJOB) コマンド、またはジョブ処理 (WRKJOB) コマンドがジョブ・レベルのコミットメント定義を表示する場合、これらのフィールドに次の情報が表示されます。

- コミットメント定義フィールドに特殊値 \*JOB が表示されます。
- 活動化グループ・フィールドに空白が表示されます。
- ジョブ・フィールドにコミットメント定義を開始したジョブが表示されます。
- スレッド・フィールドに \*NONE が表示されます。

特定の活動化グループの場合、その活動化グループ内で実行されるプログラムは、1つのコミットメント定義しか使用することができません。したがって、ある活動化グループ内で実行されるプログラムはジョブ・レベルまたは活動化グループ・レベルのコミットメント定義を使用することはできませんが、両方を同時に使用することはできません。SQL サーバー・モードを使用しないマルチスレッド・ジョブでは、プログラムに対するトランザクション作業は、どのスレッドが実行しているかにかかわらず、プログラムの活動化グループについての適切なコミットメント定義に範囲限定されます。複数のスレッドが同じ活動化グループを使用する場合、それらは共同してトランザクション作業を実行し、正しい時刻にコミットまたはロールバックを行うようにしなければなりません。

ジョブのジョブ・レベル・コミットメント定義がアクティブであっても、コミットメント制御要求または操作がジョブ・レベル・コミットメント定義のために活動化グループ内で稼働するプログラムによって実行されていない場合、プログラムはまだその活動化グループ・レベル・コミットメント定義を開始することができます。その他の場合、活動化グループ・レベル・コミットメント定義を開始できるようにするには、まず最初にジョブ・レベル・コミットメント定義を終了する必要があります。ジョブ・レベルのコミットメント定義に対する以下のコミットメント制御要求または操作は、活動化グループ・レベルのコミットメント定義の開始を妨げます。

- コミットメント制御下でのデータベース・ファイルのオープン (全体または共用)
- API コミットメント・リソースに追加するための、コミットメント制御リソースの追加 (QTNADDCR) API



- トランザクションのコミット
- トランザクションのロールバック
- コミットメント制御下でのリモート・リソースの追加
- コミットメント・オプションを変更するための、コミットメント・オプションの変更 (QTNCHGCO) API の使用
- ロールバック必須 (QTNRBRQD) API を使用して、コミットメント定義をロールバックが必要な状態にする
- コミット・サイクル ID 組み込みパラメーターを持つジャーナル項目送信 (QJOSJRNE) API を使用した、現行のコミット・サイクル ID を含むユーザー・ジャーナル項目の送信

同じように、活動化グループ内のプログラムが活動化グループ・レベル・コミットメント定義を現在使用している場合、それをまず終了してからでないとジョブ・レベル・コミットメント定義をその同じ活動化グループ内で稼働するプログラムが使用することはできません。

データベース・ファイルを開くときに、オープンされるファイルのオープン範囲は活動化グループまでか、または 1 つの制約事項の下のジョブまでになります。プログラムがコミットメント制御下でファイルを開いていて、そのファイルがジョブに範囲限定されている場合、オープン要求を出すプログラムはジョブ・レベルのコミットメント定義を使用しなければなりません。

## 明示指定のコミットメント定義

明示指定のコミットメント定義は、アプリケーションによって使用されるトランザクションに影響を与えずに独自のコミットメント制御トランザクションを実行する必要がある場合に、システムによって開始されます。この種のコミットメント定義を開始する機能の例として、問題ログがあります。アプリケーションは明示指定のコミットメント定義を開始することはできません。

IBM Navigator for i、コミットメント定義処理 (WRKCMDFN) コマンド、ジョブ表示 (DSPJOB) コマンド、またはジョブ処理 (WRKJOB) コマンドが明示指定のコミットメント定義を表示する場合、これらのフィールドに次の情報が表示されます。

- コミットメント定義フィールドには、システムから与えられた名前が表示されます。
- 活動化グループ・フィールドにブランクが表示されます。
- ジョブ・フィールドにコミットメント定義を開始したジョブが表示されます。
- スレッド・フィールドに \*NONE が表示されます。

## トランザクション範囲コミットメント定義

トランザクション範囲コミットメント定義は、トランザクション範囲ロック用の XA API を使用して開始されます。

これらの API は、活動化グループ・ベースではなく、スレッド・ベースまたは SQL 接続ベースのコミットメント制御プロトコルを使用します。言い換えれば、これらの API は、トランザクション作業の実行中にコミットメント定義を特定のスレッドまたは SQL 接続に関連付けるのに使用され、またトランザクションをコミットまたはロールバックするのに使用されます。システムは、これらのコミットメント定義を、API プロトコルに関してはトランザクション作業を実行するスレッドに付けます。それらのコミットメント定義は、別々のジョブでスレッドにより使用されます。

IBM Navigator for i、コミットメント定義処理 (WRKCMDFN) コマンド、ジョブ表示 (DSPJOB) コマンド、またはジョブ処理 (WRKJOB) コマンドがトランザクション範囲コミットメント定義を表示する場合、これらのフィールドに次の情報が表示されます。

- コミットメント定義フィールドに特殊値 \*TNSOBJ が表示されます。
- 活動化グループ・フィールドに空白が表示されます。
- ジョブ・フィールドにコミットメント定義を開始したジョブが表示されます。あるいは、コミットメント定義が現在スレッドに接続されている場合は、そのスレッドのジョブが表示されます。
- スレッド・フィールドには、コミットメント定義が接続されるスレッドが表示されます (または、コミットメント定義が現在どのスレッドにも接続されていない場合は、\*NONE)。

#### 関連資料:

XA API

## コミットメント定義名

システムは、ジョブ用に開始されたすべてのコミットメント定義に名前を付けます。

次の表に、ある特定のジョブの各種コミットメント定義とそれに関連した名前を示します。

活動化グループ	コミットの範囲	コミットメント定義名
任意	ジョブ	*JOB
デフォルト活動化グループ	活動化グループ	*DFTACTGRP
ユーザー命名の活動化グループ	活動化グループ	活動化グループ名 (例えば、PAYROLL)
システム命名の活動化グループ	活動化グループ	活動化グループ番号 (例えば、0000000145)
なし	明示指定	QDIR001 (システム専用の、システム定義のコミットメント定義の例)。システム定義のコミットメント定義名は Q で始まります。
なし	トランザクション	*TNSOBJ

IBM 統合化言語環境 (ILE) コンパイル済みプログラムだけが、デフォルト活動化グループ以外の活動化グループのコミットメント制御を開始することができます。したがって、ジョブが複数のコミットメント定義を使用できるのは、そのジョブが 1 つ以上の ILE コンパイル済みプログラムを実行している場合だけです。

オリジナル・プログラム・モデル (OPM) プログラムはデフォルト活動化グループ内で稼働し、デフォルトでは \*DFTACTGRP コミットメント定義を使用します。OPM と ILE の混合環境で、すべてのプログラムによって行われたコミット可能な変更がすべてまとめてコミットまたはロールバックされる場合、ジョブはジョブ・レベルのコミットメント定義を使用しなければなりません。

オープンされたデータベース・ファイルの範囲が活動化グループまでの場合は、そのデータベース・ファイルは活動化グループ・レベルまたはジョブ・レベルのコミットメント定義のどちらにも関連づけることができます。オープンされたデータベース・ファイルの範囲がジョブまでの場合は、そのデータベース・ファイルはジョブ・レベルのコミットメント定義にしか関連付けることはできません。したがって、OPM または ILE のプログラムがコミットメント定義の下で、ジョブまでの範囲のデータベース・ファイルをオープンする場合には、ジョブ・レベルのコミットメント定義を使用する必要があります。

アプリケーション・プログラムがコミットメント制御要求を出すときには、個々のコミットメント定義を識別するのにコミットメント定義名を使用しません。コミットメント定義名は、主として、ジョブの特定のコミットメント定義を識別するため、メッセージ内で使用されます。

活動化グループ・レベルのコミットメント定義の場合、システムは、要求を出したプログラムがどの活動化グループで実行されているかに基づいて、どのコミットメント定義を使用するかを判断します。これが可能な理由は、任意の時点である活動化グループ内で稼働するプログラムが使用できるコミットメント定義は 1 つだけだからです。

トランザクション範囲ロックを使用したトランザクションの場合は、XA API、および CLI に追加された属性に関連したトランザクションによって、呼び出しスレッドを使用するコミットメント定義が決定されます。

#### 関連情報:



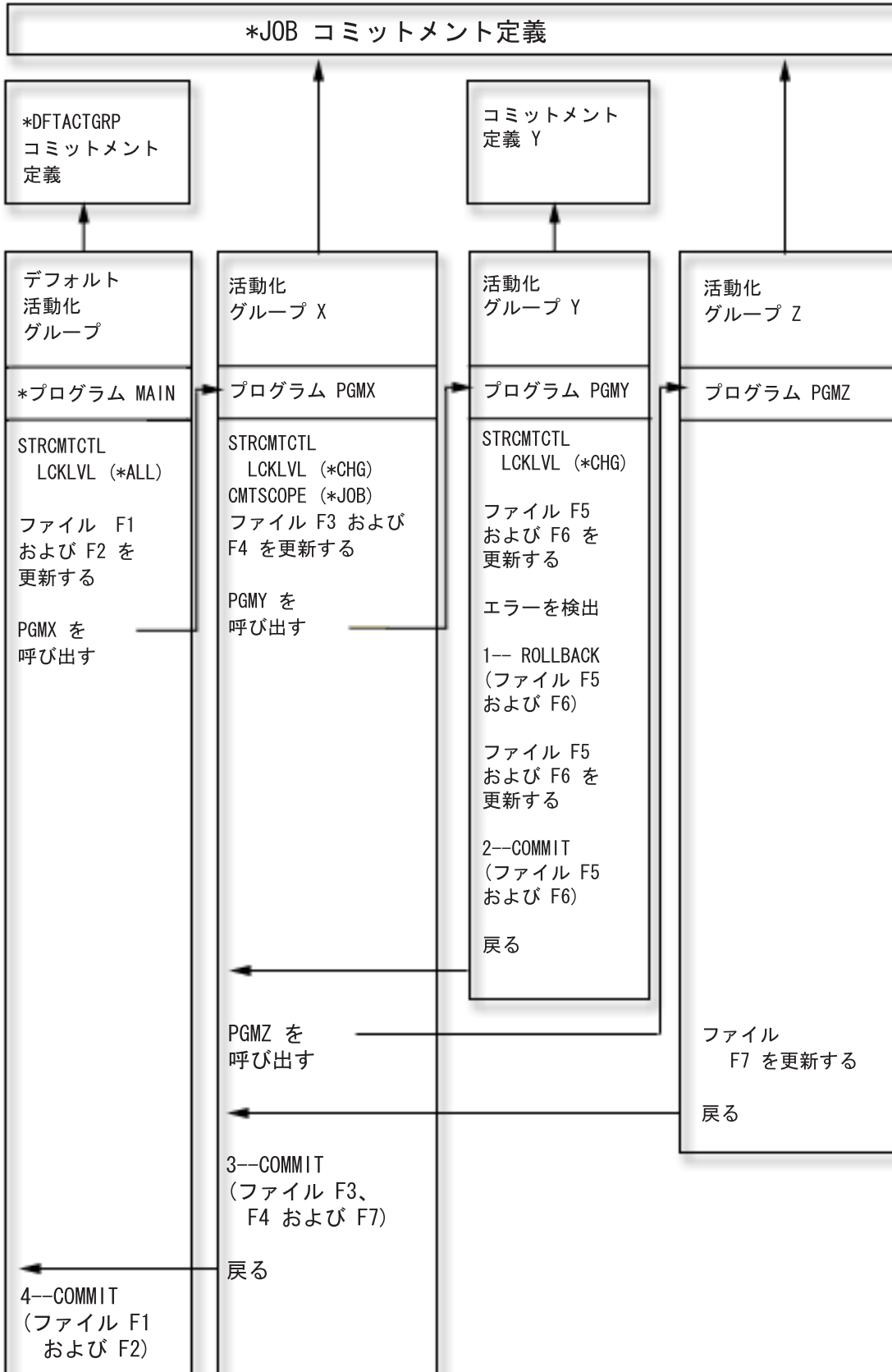
ILE 概念 (PDF)

### 例：ジョブおよびコミットメント定義

この図は、複数のコミットメント定義を使用するジョブの例を示します。

以下の図は、各活動化グループ・レベルでコミットまたはロールバックされるのはどのファイル更新かを示します。例では、すべてのプログラムによってデータベース・ファイルに対して行われる更新はすべて、コミットメント制御下で行われることを想定しています。





次の表では、前の図で示されているシナリオが変更される場合、ファイルがどのようにコミットまたはロールバックされるかを示しています。

### ジョブの複数のコミットメント定義の追加例

シナリオの変更	これらのファイルに対する変更の影響			
	F1 および F2	F3 および F4	F5 および F6	F7
PGMX はコミット操作の代わりにロールバック操作を実行します (3= =COMMIT が ROLLBACK になります)。	依然保留中	ロールバックされた	既にコミット済み	ロールバックされた
PGMZ は PGMX に戻る前にコミット操作を実行します。	依然保留中	PGMZ によってコミットされた	既にコミット済み	コミット済み
PGMZ は、ファイル F7 を更新した後に CMTSCOPE(*ACTGRP) を指定してコミットメント制御を開始しようとします。その試みは、ジョブ・レベル・コミットメント定義を使用して変更が保留中のため失敗します。	依然保留中	依然保留中	既にコミット済み	依然保留中
PGMX は、コミットメント制御を開始せず、 COMMIT(*YES) を持つファイル F3 および F4 をオープンしません。PGMZ は COMMIT(*YES) を持つファイル F7 をオープンしようとします。	依然保留中	コミットメント制御下でない	既にコミット済み	ファイル F7 は、*JOB コミットメント定義が存在しない (PGMX はそれを作成していなかった) ためオープンすることはできません。

### コミットメント制御によるオブジェクトの処理方法

オブジェクトをコミットメント制御下に置くと、コミット可能リソースになります。このコミット可能リソースは、コミットメント定義に登録されます。そのコミットメント定義に対して行われるすべてのコミット操作およびロールバック操作に参加します。

以下の項では、コミット可能リソースの属性について説明します。

- リソース・タイプ
- 位置
- コミット・プロトコル
- アクセス・インテント

## コミット可能リソースのタイプ

この表には、FILE、データ定義言語 (DDL)、分散データ管理 (DDM)、論理装置 (LU) 6.2、Distributed Relational Database Architecture™ (DRDA®)、API、TCP などのさまざまなタイプのコミット可能リソースをリストしています。

表には、以下の項目が示されています。

- コミット可能リソースのタイプ。
- コミットメント制御下に配置する方法。
- コミットメント制御から除去する方法。
- リソース・タイプに適用される制約事項。


リソース・タイプ	コミットメント制御下に置く方法	コミットメント制御から除去する方法	コミット可能な変更の種類	制約事項
FILE - ローカル・データベース・ファイル	コミットメント制御下のオープン <sup>1</sup>	保留中の変更がない場合は、ファイルをクローズする。  ファイルのクローズ時に保留中の変更がある場合は、次のコミット操作またはロールバック操作の実行後。	レコード・レベル変更	500 000 000 以下のレコードを単一ランザクション <sup>2</sup> にロックすることができます。
DDL - オブジェクト・レベルのローカル SQL 表および SQL コレクションへの変更	コミットメント制御下での SQL の稼働	オブジェクト・レベル変更後のコミット操作またはロールバック操作。	以下に示す、オブジェクト・レベル変更 <ul style="list-style-type: none"> <li>• SQL パッケージの作成</li> <li>• SQL 表の作成</li> <li>• SQL 表の除去</li> </ul>	SQL を使って実行したオブジェクト・レベル変更だけがコミットメント制御下になります。
DDM リモート分散データ管理 (DDM) ファイル	コミットメント制御下のオープン。DDM のコミットメント制御サポートには、コミットメント制御および分散データ管理についての詳細情報があります。	保留中の変更がない場合は、ファイルをクローズする。  ファイルのクローズ時に保留中の変更がある場合は、次のコミット操作またはロールバック操作の実行後。	レコード・レベル変更	
LU 6.2- 保護会話	会話の開始 <sup>3</sup>	会話の終了		
DRDA- 分散リレーショナル・データベース	SQL CONNECT ステートメントの使用	接続の終了		

リソース・タイプ	コミットメント制御下に置く方法	コミットメント制御から除去する方法	コミット可能な変更の種類	制約事項
API- ローカル API コミットメント・リソース	コミットメント制御リソースの追加 (QTNADDCR) API	コミットメント・リソースの除去 (QTNRMVCR) API	ユーザー・プログラムがこれを判別します。ジャーナル項目送信 (QJOSJRNE) API を使ってユーザー・プログラムによりジャーナル項目を書き込んで、これらの変更の追跡を援助することができます。	アプリケーションは、コミット、ロールバック、または再同期操作中に出口プログラムを呼び出すことができないようにしなければなりません。
TCP-TCP/IP 接続	TCP/IP 接続を使用するよう定義されている RDB に対して SQL CONNECT ステートメントを使用するか、TCP/IP ロケーションを使って定義されている DDM ファイルをオープンします。	SQL 接続を終了するか、保留中の変更がなければ DDM ファイルをクローズします。保留中の変更がある状態で DDM ファイルをクローズする場合、次のコミットまたはロールバック操作を実行した後に接続がクローズされます。		

**注:**

- <sup>1</sup> データベース・ファイルをコミットメント制御下に置くための詳細は、該当の言語解説書を参照してください。コミットメント制御の関連情報は、利用可能な言語マニュアルにリンクしています。
- <sup>2</sup> QAQQINI ファイルを使用して、制限値 500 000 000 を減らすことができます。手順については、78 ページの『トランザクション・サイズの管理』を参照してください。
- <sup>3</sup> DDM 接続が開始されて DDM ファイルが PTCCNV(\*YES) を指定し、そして DDM ファイルが SNA リモート・ロケーションを使って定義されている場合、DDM リソースに LU 6.2 リソースが追加されます。

DRDA 接続が開始されたときに、以下の条件の両方が当てはまる場合には、DRDA リソースに LU 6.2 リソースが追加されます。

- プログラムが分散作業単位 (DUW) 接続プロトコルを使用している。
- 接続は SNA リモート・ロケーションで定義されたリレーショナル・データベース (RDB) に対するものです。保護会話の開始について詳しくは、「APPC プログラミング」 を参照してください。

**関連概念:**

分散データベース・プログラミング

67 ページの『通知オブジェクトの更新』

システムは、そのコミットメント定義に対して正常に行われた最後のコミット操作のコミット ID を使用して、通知オブジェクトを更新します。

**関連資料:**

コミットメント制御リソースの追加 (QTNADDCR) API

コミットメント・リソースの除去 (QTNRMVCR) API

ジャーナル項目送信 (QJOSJRNE) API

## ローカルおよびリモート・コミット可能リソース

コミット可能リソースは、ローカル・リソースまたはリモート・リソースのいずれかです。

### ローカル・コミット可能リソース

ローカル・コミット可能リソースは、アプリケーションと同じシステム上にあります。コミットメント制御下でリソースに関連した各ジャーナルは、ローカル・ロケーションと見なすことができます。ジャーナルなしで登録されているすべてのリソース (DDL リソースと API リソースの両方についてオプション) は、別のローカル・ロケーションと見なすことができます。

コミット可能リソースが独立ディスク・プールにあり、コミットメント定義が別のディスク・プールにある場合、そのリソースはローカルとは見なされません。

### リモート・コミット可能リソース

リモート・コミット可能リソースは、アプリケーションとは異なるシステム上にあります。リモート・システムに対して固有の各会話にリモート・ロケーションがあります。コミットメント定義には 1 つまたは複数のリモート・システム上に、1 つまたは複数のリモート・ロケーションがあります。

ローカル・リソースをシステム・ディスク・プールまたは任意の独立ディスク・プールのコミットメント制御下に置く場合、他の独立ディスク・プールのコミットメント制御下のリソースにアクセスするには、Distributed Relational Database Architecture (DRDA) を使用する必要があります。

以下の表は、コミット可能リソースのタイプとロケーションを示しています。

リソース・タイプ	位置
API	ローカル
DDL	ローカル
DDM	リモート
DRDA	ローカルまたはリモート
FILE	ローカル
LU62	リモート
TCP	リモート

#### 関連概念:

22 ページの『コミットメント制御および独立ディスク・プール』

独立ディスク・プールおよび独立ディスク・プール・グループは、それぞれ別々の i5/OS データベースを持つことが可能です。これらのデータベースに対してコミットメント制御を使用することができます。

### コミット可能リソースのアクセス・インテント

アクセス・インテントによって、トランザクションへのリソースの関与の程度が決定されます。

リソースがコミットメント制御下に置かれると、リソース・マネージャーはリソースのアクセス方法を指示します。

- 更新
- 読み取り専用
- 未決

次の表は、特定のタイプのリソースに対して利用できるアクセス・インテント、およびリソースが登録された場合にそれに対するアクセス・インテントをシステムが判別する方法を示しています。

リソース・タイプ	可能なアクセス・インテント	アクセス・インテントの決定方法
FILE	更新、読み取り専用	ファイルのオープン方法に基づく
DDL	更新	常に更新
API	更新	常に更新
DDM	更新、読み取り専用	ファイルのオープン方法に基づく
LU62	未決	常に未決
DRDA	更新、読み取り専用、未決	DRDA レベル 1 の場合、別のリモート・リソースが登録されていない場合は、アクセス・インテントは、更新。登録されていれば、アクセス・インテントは読み取り専用。DRDA レベル 2 の場合、アクセス・インテントは常に未決。
TCP	未決	常に未決

登録済みのリソースのアクセス・インテントにより、新しいリソースが登録可能かどうか判別されます。次の規則が適用されます。

- アクセス・インテントが更新である 1 フェーズ・リソースは、以下の状態のいずれかに該当する場合、登録できません。
  - アクセス・インテントが更新であるリソースが、既に別の位置に登録されている。
  - アクセス・インテントが未決であるリソースが、既に別の位置に登録されている。
  - アクセス・インテントが未決であるリソースが、既に同じ位置に登録されており、現行のトランザクション時に変更が加えられている。
- アクセス・インテントが更新である 1 フェーズ・リソースが登録済みの場合、アクセス・インテントが更新である 2 フェーズ・リモート・リソースを登録することはできません。

## コミット可能リソースのコミット・プロトコル

コミット・プロトコルとは、1 フェーズまたは 2 フェーズ・コミット処理に参加するためにリソースがもつ機能のことです。ローカル・リソース (API コミット可能リソースを除く) は、常に 2 フェーズ・リソースです。

コミット可能リソースが独立ディスク・プールにあり、コミットメント定義が別のディスク・プールにあれば、そのリソースはローカル・リソースまたは 2 フェーズ・リソースとは見なされません。

2 フェーズ・リソースは、保護リソースとも呼ばれます。リモート・リソースおよび API コミット可能リソースは、コミットメント制御下に置く場合は 1 フェーズ・リソースまたは 2 フェーズ・リソースとして登録する必要があります。次の表には、コミットメント定義内で 1 フェーズ・リソースと共存することのできるコミット可能リソースのタイプが示されています。

リソース・タイプ	共存可能対象
1 フェーズ API リソース	他のローカル・リソース。リモート・リソースは不可。
1 フェーズ・リモート・リソース	同位置にある他の 1 フェーズ・リソース。ローカル・リソースは不可。

## 関連概念:

22 ページの『コミットメント制御および独立ディスク・プール』  
独立ディスク・プールおよび独立ディスク・プール・グループは、それぞれ別々の i5/OS データベースを持つことが可能です。これらのデータベースに対してコミットメント制御を使用することができます。

## ジャーナル・ファイルおよびコミットメント制御

データベース・ファイル (リソース・タイプが FILE または DDM) については、コミットメント制御下で出力用にオープンする前に、または \*NONE (コミットなし) 以外の分離レベルを使用する SQL アプリケーションによって参照される前に、ジャーナル処理 (ログに記録) する必要があります。コミットメント制御下で入力専用ファイルを開く場合は、そのファイルをジャーナル処理する必要はありません。

以下の状態のいずれかが当てはまる場合、エラーが発生します。

- コミットメント制御下の出力用データベース・ファイルを開こうとしたが、ファイルがジャーナル処理中でなかった。
- コミットメント制御下で開かれようとしているファイルが使用できるコミットメント定義が開始されていなかった。

あるファイルがコミットメント制御下で開かれたときに、データベース・ファイルに対して変更後イメージのみがジャーナル処理されている場合、システムは変更前イメージと変更後イメージの両方に対して、ジャーナル処理を自動的に開始します。変更前イメージは、コミットメント制御下でファイルに行った変更に対してのみ書き込まれます。ファイルに対して行った変更がコミットメント制御下でなかった場合には、変更後イメージのみが書き込まれます。

レコード・レベルのコミット可能な変更およびオブジェクト・レベルのコミット可能な変更は自動的にジャーナルに書き込まれます。レコード・レベルの変更の場合、システムは、必要に応じてジャーナル項目をリカバリーの目的で使用しますが、オブジェクト・レベルのコミット可能な変更からの項目をリカバリーの目的では使用しません。さらに、システムは、API コミットメント・リソースについてはジャーナル項目を自動的に書き込みません。しかし、API リソースの出口プログラムは、ジャーナル項目の送信 (QJOSJRNE) API を使ってジャーナル項目を書き込むことにより、監査証跡を提供したり、リカバリーを援助することができます。ジャーナル項目の内容は、ユーザー出口プログラムによって制御されます。

システムは、ジャーナル以外の手法を使用して、オブジェクト・レベルのコミットメント・リソースのリカバリーを実行します。API コミットメント・リソースのリカバリーは、個々の API コミットメント・リソースと関連したコミットおよびロールバックの出口プログラムを呼び出して行われます。出口プログラムは、その状態に必要な実際のリカバリーを行う責任があります。

## 関連概念:

ジャーナル管理

## コミットメント制御下でのジャーナル項目の順序

この表には、コミットメント定義がアクティブな場合に書き込まれる典型的な項目の順序が示されています。ジャーナル項目情報ファインダーを使用して、ジャーナル項目の内容の詳細を入手することができます。

コミットメント制御項目がローカル・ジャーナルに書き込まれるのは、少なくとも以下の条件の 1 つが該当する場合だけです。

- ジャーナルがコミットメント制御の開始 (STRCMTCTL) コマンドでデフォルトのジャーナルとして指定された。



- ジャーナルに記録されたファイルのうち、少なくとも 1 つのファイルがコミットメント制御下でオープンされた。
- ジャーナルに関連した API コミットメント・リソースのうち、少なくとも 1 つのリソースがコミットメント制御下で登録されている。

項目タイプ	説明	書き込まれる場所	書き込まれる時
C BC	コミットメント制御開始	デフォルトのジャーナル (STRCMTCTL コマンドで指定されている場合)。	STRCMTCTL コマンドが使用された時。
		ジャーナル	ジャーナルに記録された最初のファイルがオープンされた時、または API リソースがジャーナルに登録された時。
C SC	コミット・サイクルの開始	ジャーナル	このジャーナルに記録されたファイルのトランザクションに対して最初のレコード変更が行われた時 <sup>1</sup> 。
		API リソースのジャーナル	QJOSJRNE API を最初にコミット・サイクル ID の組み込み キーとともに使用した時。
ジャーナル・コード D および F	DDL オブジェクト・レベル項目	更新対象のオブジェクトに関連したジャーナル。コミット・サイクル ID を含むジャーナル項目だけが、トランザクションの一部である DDL オブジェクト・レベル変更を表します。	更新が行われた時。
ジャーナル・コード R	レコード・レベル項目	更新中のファイルに関連したジャーナル	更新が行われた時。
ジャーナル・コード U	ユーザー作成項目	API に関連したジャーナル	QJOSJRNE API をコミット・サイクル ID の組み込み キーとともに最初に使用した場合。
C CM	コミット	ジャーナル	コミットが正常に完了した時。
		デフォルトのジャーナル	コミット可能リソースのいずれかがジャーナルに関連している場合。
C RB	ロールバック	ジャーナル	ロールバック操作が完了した後。
		デフォルトのジャーナル	コミット可能リソースのいずれかがジャーナルに関連している場合。



項目タイプ	説明	書き込まれる場所	書き込まれる時
C LW	トランザクションの終了	デフォルトのジャーナル (STRCMTCTL コマンドで指定されている場合)。システムは LW 見出しレコードと 1 つまたは複数の詳細レコードを書き込みます。これらの項目が書き込まれるのは、STRCMTCTL コマンドで OMTJRNE(*NONE) が指定された場合、またはシステム・エラーが発生した場合だけです。	コミット操作またはロールバック操作が完了した時。
C EC	コミットメント制御の終了	ジャーナル	コミットメント制御の終了 (ENDCMTCTL) コマンドが完了したとき。
		デフォルトのジャーナルでないローカル・ジャーナル	ジャーナルに関連したすべてのコミット可能リソースがコミットメント制御から除去された後で、コミット境界が確立された時。
C SB	保管場所またはネストされたコミット・サイクルの開始	ジャーナル	アプリケーションが SQL SAVEPOINT を作成する時、またはシステムが一連のデータベース機能を 1 つの操作として処理するために内部でネストされたコミット・サイクルを作成する時 <sup>2</sup> 。
C SQ	保管場所の解放またはネストされたコミット・サイクルのコミット	ジャーナル	アプリケーションが SQL SAVEPOINT を解放する時、またはシステムが内部でネストされたコミット・サイクルをコミットする時 <sup>2</sup> 。
C SU	保管場所またはネストされたコミット・サイクルのロールバック	ジャーナル	アプリケーションが SQL SAVEPOINT をロールバックする時、またはシステムが内部でネストされたコミット・サイクルをロールバックする時 <sup>2</sup> 。

項目タイプ	説明	書き込まれる場所	書き込まれる時
<b>注:</b>			
<p><sup>1</sup> ジャーナル作成 (CRTJRN) またはジャーナル変更 (CHGJRN) コマンドの固定長データ (FIXLENDTA) パラメーターに作業論理単位 (LUW) 値を指定することによって、ジャーナル項目の固定長部分にトランザクション情報を含めるように指定することができます。FIXLENDTA (*LUW) パラメーターを指定することによって、各 C SC ジャーナル項目の固定長部分に現行トランザクションの作業論理単位 ID (LUWID) が含まれます。同様に、XA トランザクションの場合、FIXLENDTA(*XID) パラメーターを指定すると、各 C SC ジャーナル項目の固定長部分に現行トランザクションの XID が含まれます。トランザクションに複数のジャーナルまたはシステムが関係している場合、LUWID または XID は、特定のトランザクションのコミット・サイクルのすべてを検出するのに役立ちます。</p> <p><sup>2</sup> これらの項目は、QTN_JRNSAVPT_MYLIB_MYJRN 環境変数が *YES に設定された場合にのみ送られます。ここで、MYJRN は使用するジャーナル、MYLIB はジャーナルが保管されるライブラリーです。特殊値 *ALL は MYLIB および MYJRN 値でサポートされます。これらの変数は、システム全体または特定のジョブに設定できます。1 つのジョブのジャーナル MYLIB/MYJRN にのみ項目を送るには、そのジョブで次のコマンドを使用します。</p> <ul style="list-style-type: none"> <li>• ADDENVVAR ENVVAR(QTN_JRNSAVPT_MYLIB_MYJRN) VALUE(*YES)</li> </ul> <p>すべてのジョブのすべてのジャーナルに項目を送信するには、このコマンドを使用します。</p> <ul style="list-style-type: none"> <li>• ADDENVVAR ENVVAR('QTN_JRNSAVPT_*ALL_*ALL') VALUE(*YES) LEVEL(*SYS)</li> </ul> <p>特定のジャーナルに関連するリソースが初めてコミットメント制御下に置かれるとき、この環境変数値はそれぞれのコミットメント定義ごとに内部的にキャッシュされます。その時点の後で環境変数が変更される場合、そのジャーナルに有効になるようにキャッシュされた値をリフレッシュする必要があります。Retrieve Commit Information (QTNRCMTI) API の呼び出しを実行すると、キャッシュされた値が呼び出しジョブでリフレッシュされます。</p>			

#### 関連概念:

3 ページの『コミット操作』

直前のコミット操作またはロールバック操作以降にコミットメント制御下で行われたすべての変更が確定されます。システムは、トランザクションに関連したすべてのロックも解除します。

ジャーナル項目情報ファインダー

#### 関連資料:

コミットメント制御の終了 (ENDCMTCTL) コマンド

### コミット・サイクル ID

コミット・サイクルとは、各コミットメント境界間の時間です。システムはコミット・サイクル ID を割り当てて、特定のコミットメント・サイクルのすべてのジャーナル項目を関連付けます。トランザクションに参加する各ジャーナルには、独自のコミット・サイクルとコミット・サイクル ID があります。

コミット・サイクル ID は、コミット・サイクルに書き込まれた C SC ジャーナル項目のジャーナル順序番号です。コミット・サイクル ID は、コミット・サイクル時に各ジャーナル項目に書き込まれます。コミット・サイクル時に複数のジャーナルが使用された場合には、各ジャーナルのコミット・サイクル ID はそれぞれ異なります。

ジャーナル作成 (CRTJRN) またはジャーナル変更 (CHGJRN) コマンドの固定長データ (FIXLENDTA) パラメーターに作業論理単位 (LUW) 値を指定することによって、ジャーナル項目の固定長部分にトランザクション情報を含めるように指定することができます。FIXLENDTA (\*LUW) パラメーターを指定することによって、各 C SC ジャーナル項目の固定長部分に現行トランザクションの作業論理単位 ID (LUWID) が含まれます。同様に、XA トランザクションの場合、FIXLENDTA(\*XID) パラメーターを指定すると、各 C SC ジャーナル項目の固定長部分に現行トランザクションの XID が含まれます。トランザクションに

複数のジャーナルまたはシステムが関係している場合、LUWID または XID は、特定のトランザクションのコミット・サイクルのすべてを検出するのに役立ちます。

ジャーナル項目送信 (QJOSJRNE) API を使用して、API リソース用のジャーナル項目を書き込むことができます。これらのジャーナル項目にコミット・サイクル ID を含めるためのオプションがあります。

コミット・サイクル ID を使用して、ジャーナル処理済み変更の適用 (APYJRNCHG) コマンドまたはジャーナル処理済み変更の除去 (RMVJRNCHG) コマンドを使って、コミットメント境界にジャーナル処理した変更を適用したり、除去したりすることができます。次の制約が適用されます。

- コミットメント制御下で行われたオブジェクト・レベル変更の多くはジャーナルに書き込まれますが、APYJRNCHG コマンドや RMVJRNCHG コマンドを使って適用または除去されることはありません。
- QJOSJRNE API は、ジャーナル・コードが U のユーザー作成ジャーナル項目を書き込みます。これらの項目は、APYJRNCHG および RMVJRNCHG コマンドを使って適用または除去することができません。ユーザー作成のプログラムを使って、適用または除去しなければなりません。

## レコードのロック

あるジョブがレコードをロックしていて、別のジョブが更新のためにそのレコードを検索しようとする、要求側のジョブは待機させられ、アクティブな処理から除去されます。

要求側のジョブは、以下の事象のいずれかが発生するまでアクティブです。

- レコードのロックが解除される。
- 指定された待機時間が終了する。

複数のジョブが、別のジョブがロックしているレコードを要求することができます。レコードのロックが解除されると、最初にそのレコードを要求したジョブがそのレコードを受け取ります。ロックされたレコードを待つときには、次の作成、変更、または一時変更コマンドの WAITRCD パラメーターに待機時間を指定してください。

- 物理ファイル作成 (CRTPF)
- 論理ファイル作成 (CRTLF)
- ソース物理ファイル作成 (CRTSRCPF)
- 物理ファイル変更 (CHGPF)
- 論理ファイル変更 (CHGLF)
- ソース物理ファイル変更 (CHGSRCPF)
- データベース・ファイルによる指定の変更 (OVRDBF)

待機時間を指定する時には、次の情報を考慮してください。

- 値を指定しないと、プログラムはデフォルトの待機時間で処理します。
- トランザクション範囲ロックのみを使用したコミットメント定義の場合、ジョブのデフォルト待機時間を、以下のものに対して指定できるトランザクション・ロック待ち時間でオーバーライドすることができます。
  - xa\_open API。
  - JDBC インターフェースまたは JTA インターフェース。分散トランザクションにより、これらの API がリストされます。
- 指定時間内にレコードが割り振られないと、通知メッセージが高水準言語プログラムに送られます。
- レコードの待機時間を超えると、ジョブ・ログにメッセージが送られます。そのメッセージには、「ロックされたレコードを保持して要求側ジョブを待機させているジョブの名前」が示されています。レコ

ード・ロック例外が生じた場合には、ジョブ・ログを使ってどのプログラムを変更すればよいかを判別し、そのプログラムがロックを長期間保持しないようにすることができます。

プログラムは、次のいずれかの理由でレコードのロックを長期間継続します。

- ワークステーション・ユーザーが変更を考慮している間、レコードはロックされたままです。
- レコード・ロックは、長いコミットメント・トランザクションの一部です。コミット操作をもっと頻繁に行えるように、トランザクションをより小さくすることを考慮してください。
- 望ましくないロックが起きました。例えば、ファイルが固有のキーをもつ更新ファイルとして定義されていて、プログラムはそのファイルを更新し、さらにレコードを追加するものと想定します。ワークステーション・ユーザーがそのファイルにレコードを追加したい場合には、プログラムはレコードにアクセスし、キーが既に存在するかどうかを判別しようと試みます。キーが既に存在する場合には、プログラムはワークステーション・ユーザーに無効な要求を行ったことを通知します。レコードは同じファイルに対する別の読み取り操作によって暗黙に解放されるか、または明示的に解放されるまでロックされたままになります。

**注:** レコードのロック解除のための各高水準言語インターフェースの使用法については、該当する高水準言語の解説書を参照してください。コミットメント制御の関連情報は、コミットメント制御と共に使用できる高水準言語マニュアルにリンクしています。

LCKLVL(\*ALL) を指定した場合には、ファイルから検索されたレコードは次のコミットまたはロールバック操作までロックされているので、ロックの持続期間はかなり長くなります。このレコードは別の読み取り操作で暗黙に解放されず、また明示的に解放することもできません。

ファイルにロックを設定することができる別の方法として、活動時保管機能があります。

#### 関連概念:

JDBC 分散トランザクション

活動時保管機能

#### 関連資料:

125 ページの『コミットメント制御の関連情報』製品マニュアル、IBM Redbooks 資料、Web サイト、およびその他の Information Center トピック・コレクションには、コミットメント制御トピック集に関連する情報が収められています。以下の PDF ファイルのいずれも表示または印刷できます。

## コミットメント制御および独立ディスク・プール

独立ディスク・プールおよび独立ディスク・プール・グループは、それぞれ別々の i5/OS データベースを持つことが可能です。これらのデータベースに対してコミットメント制御を使用することができます。

ただし、それぞれの独立ディスク・プールまたは独立ディスク・プール・グループに別々の SQL データベースを持つため、いくつかの考慮事項に注意する必要があります。

#### 関連概念:

5 ページの『コミットメント定義』

コミットメント定義には、トランザクション中にコミットメント制御下で変更されるリソースに関する情報が入っています。

15 ページの『ローカルおよびリモート・コミット可能リソース』

コミット可能リソースは、ローカル・リソースまたはリモート・リソースのいずれかです。

16 ページの『コミット可能リソースのコミット・プロトコル』

コミット・プロトコルとは、1 フェーズまたは 2 フェーズ・コミット処理に参加するためにリソースがもつ機能のことです。ローカル・リソース (API コミット可能リソースを除く) は、常に 2 フェーズ・リソースです。

## コミットメント定義の独立ディスク・プールの考慮事項

独立ディスク・プールを使用する場合は、コミットメント定義についての考慮事項に注意する必要があります。

### QRECOVERY ライブラリーに関する考慮事項

コミットメント制御を開始すると、コミットメント定義が QRECOVERY ライブラリー内に作成されます。それぞれの独立ディスク・プールまたは独立ディスク・プール・グループには、QRECOVERY ライブラリーの独自のバージョンがあります。独立ディスク・プールでは、QRECOVERY ライブラリーの名前は QRCYxxxxx です。ここで、xxxxx は独立ディスク・プールの番号です。例えば、独立ディスク・プール 39 の QRECOVERY ライブラリーの名前は QRCY00039 です。さらに、独立ディスク・プールがディスク・プール・グループの一部である場合は、1 次ディスク・プールのみ QRCYxxxxx ライブラリーがあります。

コミットメント制御を開始すると、そのジョブに関連付けられた独立ディスク・プールの QRECOVERY ライブラリー内にコミットメント定義が作成され、独立ディスク・プール上でコミットメント制御がアクティブになります。

### ASP グループ設定に関する考慮事項

コミットメント制御が独立ディスク・プール上でアクティブな時に ASP グループ設定 (SETASPGRP) コマンドを使用すると、以下のような影響があります。

- 独立ディスク・プールから切り替える場合で、リソースがそのディスク・プール上のコミットメント制御に登録されている場合、SETASPGRP コマンドはメッセージ CPDB8EC、理由コード 2、「コミットされていないトランザクションがスレッドにあります」で失敗します。このメッセージの後に、メッセージ CPF88E9 が続きます。
- 独立ディスク・プールから切り替える場合で、リソースがコミットメント制御に登録されていない場合、コミットメント定義は、切り替え先の独立ディスク・プールに移動されます。
- システム・ディスク・プール (ASP グループ \*NONE) から切り替える場合は、コミットメント制御は影響を受けません。コミットメント定義は、システム・ディスク・プールに留まります。その後、独立ディスク・プールのリソースを、システム・ディスク・プールのリソースより前にコミットメント制御下に置く場合、コミットメント定義は独立ディスク・プールに移動します。
- 通知オブジェクトを使用する場合、その通知オブジェクトは、コミットメント定義と同じ独立ディスク・プールまたは独立ディスク・プール・グループ上になければなりません。
- コミットメント定義を別の独立ディスク・プールまたは独立ディスク・プール・グループに移動する場合、通知オブジェクトはその別の独立ディスク・プールまたは独立ディスク・プール・グループ上になければなりません。コミットメント定義が異常終了すると、他の独立ディスク・プールまたは独立ディスク・プール・グループ上にある通知オブジェクトが更新されます。他の独立ディスク・プールまたは独立ディスク・プール・グループ上に通知オブジェクトが見付からないと、更新処理はメッセージ CPF8358 で失敗します。

ジョブの現行ネーム・スペースにより、コミットメント定義が作成される独立 ASP が決定されます。ジョブが独立 ASP に関連付けされない場合、コミットメント定義は \*SYSBAS 内に作成されます。それ以外の場合は独立 ASP 内に作成されます。ジョブが独立 ASP に関連付けされる場合、現行ライブラリーのネ



ーム・スペースに存在するコミットメント制御下でファイルを開くことができます。言い換えれば、ファイルは独立 ASP または \*SYSBAS のどちらにも置くことができます。コミットメント制御下に置かれる最初のリソースが、コミットメント定義と同じ ASP 内に存在しない場合、コミットメント定義はリソースの ASP に移動します。\*SYSBAS および独立 ASP リソースの両方が同じコミットメント定義に登録される場合、システムは暗黙的に 2 フェーズ・コミット・プロトコルを使用して、システム障害が発生した場合にリソースが自動的にコミットされるようにします。従って、\*SYSBAS および独立 ASP の両方にあるデータを含むトランザクションは、単一 ASP グループに対して隔離されているトランザクションと同様に、実行されません。

## デフォルト・ジャーナルに関する考慮事項

以下のデフォルト・ジャーナルに関する考慮事項に注意する必要があります。

- デフォルト・ジャーナルを使用する場合、そのジャーナルは、コミットメント定義と同じ独立ディスク・プールまたは独立ディスク・プール・グループ上になければなりません。
- コミットメント制御を開始したときに他の独立ディスク・プールまたは独立ディスク・プール・グループ上にデフォルト・ジャーナルが見付からないと、コミットメント制御の開始はメッセージ CPF9873 で失敗します。
- コミットメント定義を他の独立ディスク・プールまたは独立ディスク・プール・グループに移動する場合、デフォルト・ジャーナルは移動先の独立ディスク・プールまたは独立ディスク・プール・グループ上にもなければなりません。他の独立ディスク・プールまたは独立ディスク・プール・グループ上にデフォルト・ジャーナルが見付からない場合、コミットメント定義は移動されますが、この時点からデフォルト・ジャーナルは使用されなくなります。

## 初期プログラム・ロード (IPL) およびオフに構成変更に関する考慮事項

以下の IPL およびオフに構成変更に関する考慮事項に注意する必要があります。

- 独立ディスク・プールにあるコミットメント定義のリカバリーは、独立ディスク・プールをオンに構成変更する処理の間に実行され、IPL リカバリーと類似しています。
- 独立ディスク・プール内のコミットメント定義は、システム IPL 中にリカバリーされません。
- \*SYSBAS と独立 ASP の両方に存在するリソースが入っているコミットメント定義にリカバリーが必要な場合、2 つの ASP グループ間にリモート・データベース接続があるかのように、コミットメント定義はリカバリー時に 2 つの (\*SYSBAS と独立 ASP) コミットメント定義に分割されます。両方の ASP グループのデータが自動的にコミットまたはロールバックされるように、リカバリー中にシステムによって再同期が開始されます。
- 独立ディスク・プールをオフに変更すると、コミットメント定義に次のような影響があります。
  - 独立ディスク・プールに関連付けられたジョブが終了します。
  - 独立ディスク・プールに新規のコミットメント定義を作成することはできません。
  - 独立ディスク・プールにあるコミットメント定義は使用不能になります。
  - 独立ディスク・プールにあるがジョブには関連付けられていないコミットメント定義は、トランザクション範囲ロックを解放します。

## リモート・データベースに関する考慮事項

以下のリモート・データベースに関する考慮事項に注意する必要があります。

- LU 6.2 SNA 接続 (保護会話または分散作業単位 (DUW)) を使用して、独立ディスク・プール・データベースからリモート・データベースに接続することはできません。無保護 SNA 会話を使用して独立ディスク・プール・データベースからリモート・データベースに接続することはできます。

- コミットメント制御がジョブまたはスレッドに対してアクティブな場合、コミットメント定義のある独立ディスク・プールまたは独立ディスク・プール・グループとは別の場所にあるデータへのアクセスは、他のシステムにあるデータであるかのように、リモートでのみ可能です。独立ディスク・プール上のリレーショナル・データベース (RDB) に接続するために SQL CONNECT ステートメントを実行した場合は、リモート接続が行われます。
- システム・ディスク・プールおよび基本ディスク・プールは、独立ディスク・プールにあるデータへの読み取り専用アクセスではリモート接続を必要としません。同様に、独立ディスク・プールは、システム・ディスク・プールまたは基本ディスク・プールにあるデータへの読み取り専用アクセスではリモート接続を必要としません。

#### 関連概念:

5 ページの『コミットメント定義』

コミットメント定義には、トランザクション中にコミットメント制御下で変更されるリソースに関する情報が入っています。

55 ページの『通知オブジェクトのコミット』

通知オブジェクトはメッセージ・キュー、データ域、またはデータベース・ファイルで、そのコミットメント定義が正常に終了しなかった場合、特定のコミットメント定義の最後に正常に完了したトランザクションを識別する情報が含まれています。

69 ページの『異常終了後の初期プログラム・ロード中のコミットメント制御リカバリー』

システムの異常終了後、初期プログラム・ロード (IPL) を実行すると、システムは、システムの終了時にアクティブであったすべてのコミットメント定義をリカバリーしようとしています。

## XA トランザクションの独立ディスク・プールの考慮事項

XA 環境では、各データベースが独立したリソース・マネージャーと見なされます。トランザクション・マネージャーが、同じトランザクションの下で 2 つのデータベースをアクセスする場合は、XA プロトコルを使用して 2 つのリソース・マネージャーとの 2 フェーズ・コミットを実行しなければなりません。

それぞれの独立ディスク・プールは独立した SQL データベースなので、XA 環境では、それぞれの独立ディスク・プールは独立したリソース・マネージャーとも見なされます。アプリケーション・サーバーが、2 つの別々の独立ディスク・プールをターゲットとするトランザクションを実行するには、トランザクション・マネージャーは 2 フェーズ・コミット・プロトコルも使用しなければなりません。

#### 関連概念:

46 ページの『コミットメント制御の XA トランザクション・サポート』

DB2 for i は、X/Open グローバル・トランザクションに参加することができます。

独立ディスク・プールの例

## コミットメント制御に関する考慮事項と制約事項

コミットメント制御に関する考慮事項および制約事項を認識しておく必要があります。

### データベース・ファイルに関する考慮事項

- 共用ファイルをコミットメント制御の下でオープンすることを指定した場合には、それ以後そのファイルを使用する際は常にコミットメント制御下でオープンしなければなりません。
- LCKLVL(\*ALL) を使用して読み取り専用でオープンされたファイルに SEQONLY(\*YES) を (暗黙または高水準言語プログラムのいずれかで、あるいはデータベース・ファイルによる指定の変更 (OVRDBF) コマンドで明示的に) 指定した場合には、SEQONLY(\*YES) は無視され、SEQONLY(\*NO) が使用されません。

- コミットメント制御下で行われるレコード・レベルの変更は、ジャーナルに記録されます。このような変更は、ジャーナル処理済み変更の適用 (APYJRNCHG) コマンドまたはジャーナル処理済み変更の除去 (RMVJRNCHG) コマンドを使って、データベースに適用したりデータベースから除去することができます。
- ファイルの変更前イメージと変更後イメージの両方が、コミットメント制御下でジャーナルに記録されます。ファイルの変更後イメージのみをジャーナル処理するよう指定した場合も、システムはコミットメント制御下で行われたファイル変更の変更前イメージも自動的にジャーナル処理します。ただし、そのファイルに対して行われたすべての変更の変更前イメージはキャプチャーされないため、そのファイルに対して RMVJRNCHG コマンドを使用することはできません。

## オブジェクト・レベルおよびレコード・レベルの変更に関する考慮事項

- SQL を使用してコミットメント制御の下で行われるオブジェクト・レベルおよびレコード・レベルの変更は、要求側プログラムが実行されている活動化グループで現在活動中のコミットメント定義を使用します。ジョブ・レベルまたは活動化グループ・レベルのどちらのコミットメント定義も活動化されていない場合には、SQL は活動化グループ・レベルのコミットメント定義を開始します。

### 1 フェーズ・コミットおよび 2 フェーズ・コミットに関する考慮事項

- 1 フェーズのリモート会話または接続が確立されている間は、他のロケーションへのリモート会話または接続は許可されません。コミットメント境界が設定されていて、すべてのリソースが除去される場合には、ロケーションを変更することができます。
- 2 フェーズ・コミットを使用している場合、リモート・ロケーションでコミット制御を開始したり他のコミット制御を実行するのに、リモート・コマンドの投入 (SBMRMTCMD) コマンドを使用する必要はありません。システムがそれらの機能を代わりに実行します。
- 1 フェーズのリモート・ロケーションの場合、SQL が呼び出しスタックにあり、リモート・リレーショナル・データベースがシステム上にない場合には、COMMIT および ROLLBACK の各 CL コマンドは失敗します。SQL が呼び出しスタックにない場合は、COMMIT および ROLLBACK コマンドは異常終了しません。
- 1 フェーズ・リモート・ロケーションの場合、リモート・リソースにコミット可能変更を行う前に、コミットメント制御をソース・システム上でまず開始しなければなりません。SQL プログラムが \*NONE 以外のコミットメント制御オプションを指定して実行される場合には、システムは接続時にソース・システムで分散データベース SQL のコミットメント制御を自動的に開始します。最初のリモート・リソースがコミットメント制御下に置かれると、システムはターゲット・システムでコミットメント制御を開始します。

## 保管に関する考慮事項

保管を実行するジョブに、次のタイプのコミット可能変更のあるアクティブなコミットメント定義が 1 つ以上ある場合には、保管操作はできません。

- 保管しようとしているライブラリー中に存在するファイルへのレコード変更。論理ファイルについては、すべての関連する物理ファイルが検査されます。
- 保管しようとしているライブラリー内のオブジェクト・レベルの変更。
- コミットメント制御リソースの追加 (QTNADDCR) API を使用し、通常の保管処理許可フィールドをデフォルト値 N に設定して追加されたすべての API リソース。

これにより、部分的なトランザクションによる変更を、保管操作が保管メディアに保管することが防止されます。



注: 部分的なトランザクション機能のある新規保管を使用した場合、コミットメント定義を終了せずにオブジェクトを保管できます。

オブジェクト・ロックとレコード・ロックは、他のジョブのコミットメント定義の保留中の変更が保管メディアに保管されるのを防ぎます。変更が API コミットメント・リソースと関連した 1 つまたは複数のオブジェクトに対して行われたときにロックがかけられた場合、これは、API コミットメント・リソースの場合にだけ当てはまります。

## その他の考慮事項および制約事項

- システムを新しいリリースにアップグレードする前に、すべての保留中の再同期を完了するか取り消さなければなりません。
- COMMIT と ROLLBACK の値は、コミット時あるいはロールバック時に WRKACTJOB 機能フィールドに示されます。機能が長期にわたり COMMIT あるいは ROLLBACK になっている場合、次のいずれかの事象が起こり得ます。
  - コミット中あるいはロールバック中にリソース障害が起こり、再同期が必要となる。再同期が完了するか取り消されるまで、制御はアプリケーションへは戻されません。
  - コミット中、このシステムは読み取り専用ポートした。コミットを開始したシステムがこのシステムにデータを送るまで、アプリケーションへ制御は戻されません。
  - コミット中、このシステムは流用可能にポートした。コミットを開始したシステムがこのシステムにデータを送るまで、アプリケーションへ制御は戻されません。

### 関連概念:

#### 2 フェーズ・コミットの整合性の確認

56 ページの『ロック・レベルのコミット』

コミットメント制御の開始 (STRCMTCTL) コマンドで LCKLVL パラメーターに指定する値は、コミットメント定義のコミットメント制御でオープンされて配置されるデータベース・ファイルのレコード・ロックのデフォルトのレベルになります。

### 関連資料:

データベース・ファイル による指定の変更 (OVRDBF) コマンド

ジャーナル処理済み変更の適用 (APYJRNCHG) コマンド

ジャーナル処理済み変更の除去 (RMVJRNCHG) コマンド

SQL プログラミング

リモート・コマンドの投入 (SBMRMTCMD) コマンド

コミット (COMMIT) コマンド

ロールバック (ROLLBACK) コマンド

コミットメント制御リソースの追加 (QTNADDCR) API

## バッチ・アプリケーションのコミットメント制御

バッチ・アプリケーションは、コミットメント制御を必要とする場合とそうでない場合があります。場合によっては、バッチ・アプリケーションは、入力ファイルの読み取りとマスター・ファイルの更新の単一機能しか実行できないことがあります。ただし、異常終了の後に再開することが重要な場合には、この種のアプリケーションに対してコミットメント制御を使用することができます。

入力ファイルは、レコードが処理されたことを示すコードをレコード内にもつ更新ファイルです。このファイルと更新された任意のファイルはコミットメント制御下に入ります。このコードが入力ファイルに存在し

ているときには、このコードは完了済みのトランザクションを表します。プログラムは入力ファイルを読み取り、完了済みコードのあるレコードを迂回します。これにより、通常の状態と再開状態で同じプログラム・ロジックを使用することができます。

バッチ・アプリケーションに、相互に依存する入力レコードが入っており、スイッチまたは合計が入っている場合には、再開についての情報を提供するために通知オブジェクトを使用することができます。入力ファイル内の最後にコミットされたトランザクションから処理を再び開始するためには、通知オブジェクトに保持されている値が使用されます。

入力レコードが相互に依存している場合には、それらのレコードを 1 つのトランザクションとして処理することができます。バッチ・ジョブでは、最大 500 000 000 のレコードをロックすることができます。この限度は QUERY オプション・ファイル (QAQQINI) を使用することにより削減することができます。QUERY 属性の変更 (CHGQRYA) コマンドの QRYOPTLIB パラメーターを使用して、使用するジョブのための QUERY オプション・ファイルを指定してください。QUERY オプション・ファイルの中の COMMITMENT\_CONTROL\_LOCK\_LEVEL 値を、ジョブのロック限度として使用してください。ジャーナル処理されたリソースが初めてコミットメント制御下に置かれるとき、このロックしきい値はそれぞれのコミットメント定義ごとに内部的にキャッシュされます。その時点の後でロック限度が変更される場合、そのコミットメント定義に有効になるようにキャッシュされた値をリフレッシュする必要があります。Retrieve Commit Information (QTNRCMTI) API の呼び出しを実行すると、キャッシュされた値が呼び出しジョブでリフレッシュされます。新しい値は、キャッシュがリフレッシュされる前に開始されたトランザクションには適用されません。

2000 のロックを超えるコミット・サイクルはおそらく、システム・パフォーマンスをかなり低下させるものとなります。そうならない場合には、対話式アプリケーションと同じロック上の考慮事項が存在することになりますが、レコードがバッチ・アプリケーション内でロックされる時間の長さは、対話式アプリケーションの場合ほど重要ではないと考えられます。

#### 関連概念:

55 ページの『通知オブジェクトのコミット』

通知オブジェクトはメッセージ・キュー、データ域、またはデータベース・ファイルで、そのコミットメント定義が正常に終了しなかった場合、特定のコミットメント定義の最後に正常に完了したトランザクションを識別する情報が含まれています。

78 ページの『トランザクション・サイズの管理』

レコード・ロックを最小限に抑える方法として、トランザクションのサイズを管理する方法もあります。

#### 関連資料:

QUERY 属性の変更 (CHGQRYA) コマンド

## 2 フェーズ・コミットメント制御

2 フェーズ・コミットは、コミット可能リソースが複数のシステム上で確実に同期化するようにします。

i5/OS オペレーティング・システムは、SNA LU 6.2 アーキテクチャーに従った 2 フェーズ・コミットをサポートします。2 フェーズ・コミットのためにシステムが使用する内部プロトコルに関する詳細は、*SNA Transaction Programmer's Reference for LU Type 6.2, GC30-3084-05* を参照してください。i5/OS オペレーティング・システムのすべてのサポート対象リリースでは、SNA LU 6.2 の Presumed Nothing プロトコルおよび SNA LU 6.2 の Presumed Abort プロトコルがサポートされます。

分散作業単位 (DUW) 分散リレーショナル・データベース体系 (DRDA) プロトコルとして TCP/IP を使用することにより、2 フェーズ・コミットもサポートされます。TCP/IP DUW 接続を使用するには、すべてのシステム (アプリケーション・リクエスターとアプリケーション・サーバーの両方) が V5R1M0 かそれ

以降でなければなりません。DRDA について詳しくは、Open Group Technical Standard の「*DRDA V2 Vol. 1: Distributed Relational Database Architecture*」を参照してください。これは、The Open Group の Web サイト (英語) にあります。

2 フェーズ・コミット下では、システムはコミット操作を次の 2 つのウェーブで実行します。

- 作成ウェーブ時に、リソース・マネージャーはそのトランザクション・マネージャーにコミット要求を出します。トランザクション・マネージャーは、それが管理する他のすべてのリソースおよび他のトランザクション・マネージャーに対してトランザクションがコミット可能であることを通知します。すべてのリソース・マネージャーが、コミット可能であると応答する必要があります。この応答をポート といいます。
- コミット・ウェーブ時には、コミット要求を開始したトランザクション・マネージャーが、作成ウェーブの結果に基づいて、処置を判別します。作成ウェーブが正常に完了し、すべての参加プログラム・ポートが操作可能になったならば、トランザクション・マネージャーはそれが管理するすべてのリソースおよび他のトランザクション・マネージャーにトランザクションをコミットするよう指示します。作成ウェーブが正常に完了しなかった場合、すべてのトランザクション・マネージャーおよびリソース・マネージャーは、トランザクションをロールバックするように指示されます。

## リモート・リソースを使用したコミット操作およびロールバック操作

リモート・リソースがコミットメント制御下にある場合、イニシエーターはすべてのリモート・エージェントにコミット要求を送ります。ロールバック要求は、トランザクション・プログラム・ネットワーク全体に送信します。各エージェントは、コミット操作の結果を送り返します。

作成ウェーブ時にエラーが発生すると、イニシエーターはすべてのエージェントにロールバック要求を送ります。コミット・ウェーブ時にエラーが発生すると、システムはできるだけ多くのロケーションをコミット状態にしようとします。これにより、ヒューリスティック混合状態になることがあります。起こり得る状態について詳しくは、2 フェーズ・コミットメント制御のトランザクションの状態を参照してください。

エラーはユーザーに通知された所でイニシエーターに送り返されます。コミットメント制御の開始 (STRCMTCTL) コマンドでデフォルトのジャーナルが指定されていた場合は、C LW 項目が書き込まれます。エラーが発生すると、これらの項目は OMTJRNE(\*LUWID) が指定されていた場合でも書き込まれます。エラー・メッセージおよびコミットメント定義の状況情報と共に、これらの項目を使用して、コミット可能リソースを手動で同期化することができます。

リモート・リソースがコミットメント制御下にある場合、イニシエーターはすべてのリモート・エージェントにロールバック要求を送ります。ロールバック要求は、トランザクション・プログラム・ネットワーク全体に送信します。各エージェントは、ロールバック操作の結果を送り返します。

関連概念:

 The Open Group の Web サイト

関連資料:

コミットメント制御の開始 (STRCMTCTL) コマンド

## コミット処理での役割

トランザクションのコミットに複数のリソース・マネージャーが含まれている場合、各リソース・マネージャーはトランザクションでそれぞれの役割を果たします。リソース・マネージャーの役割は、トランザクション時に加えられた変更のコミットまたはロールバックです。

リソース・タイプ別のリソース・マネージャーは以下のとおりです。

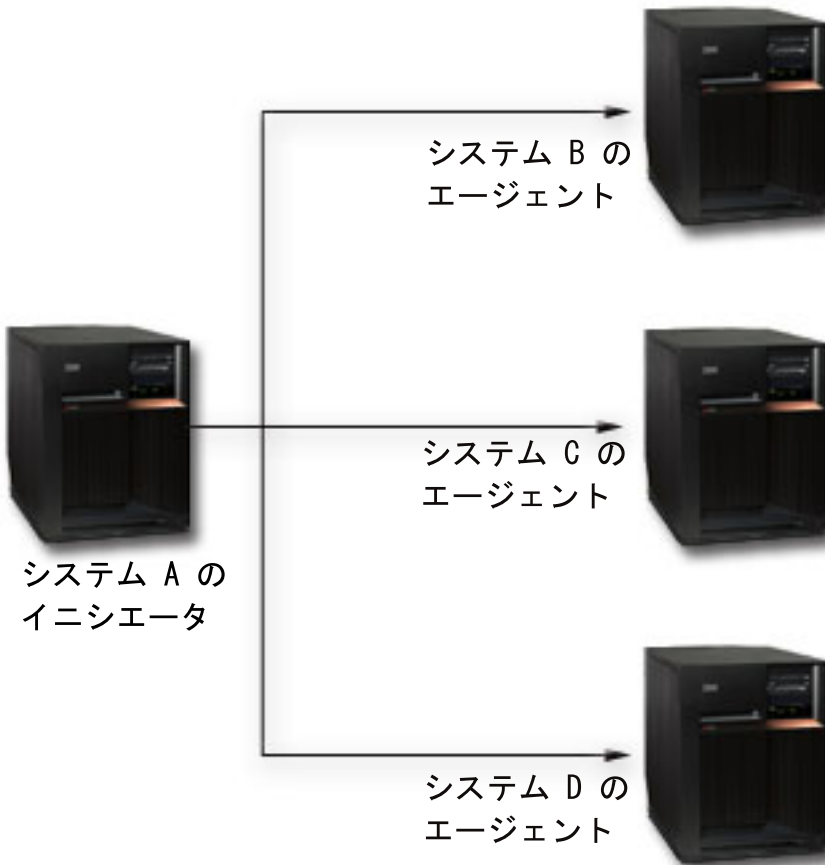
- FILE** データベース・マネージャー
- DDM** データベース・マネージャー
- DDL** データベース・マネージャー
- DRDA** 通信トランザクション・プログラム
- LU62** 通信トランザクション・プログラム
- API** API 出口プログラム

次の図は、トランザクションの基本的な役割を示しています。図で示された構造は、トランザクション・プログラム・ネットワークと呼ばれます。構造は、単一レベル・ツリーでも複数レベル・ツリーでも可能です。

## 2 フェーズ・コミット処理での役割：単一レベル・ツリー

システム A 上のアプリケーションがコミット要求を出すと、システム A のリソース・マネージャーはイニシエーターになります。TCP/IP 上の Distributed Relational Database Architecture (DRDA) 分散作業単位の場合、イニシエーターはコーディネーターと呼ばれます。

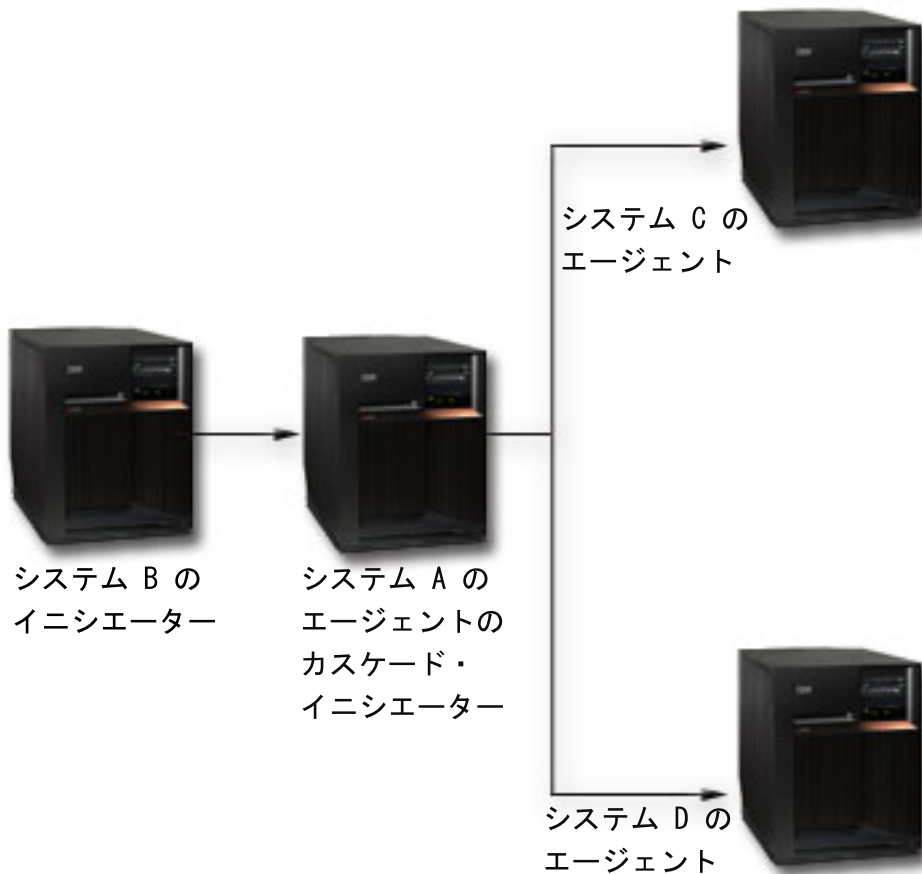
他の 3 つのシステム (B、C、および D) のリソース・マネージャーは、このトランザクションのエージェントになります。TCP/IP 上の DRDA 分散作業単位の場合、エージェントは参加者と呼ばれることもあります。



## 2 フェーズ・コミット処理での役割：複数レベル・ツリー

アプリケーションが APPC 通信を使って 2 フェーズ・コミットを実行している場合、システム間の関係は 1 つのトランザクションから次のものに変更することができます。次の図は、システム B 上のアプリケーションがコミット要求を出した場合の同じシステムを示しています。この構成は複数レベル・ツリーです。

この図にある役割は TCP/IP 上の DRDA 分散作業単位には当てはまりません。複数レベルのトランザクション・ツリーはサポートされていないからです。



システム B はシステム C およびシステム D と直接通信していないので、トランザクション・プログラム・ネットワークには別のレベルがあります。システム A のリソース・マネージャーには、エージェントとしての役割と、カスケード・イニシエーターとしての役割があります。

LU 6.2 の 2 フェーズ・トランザクションのパフォーマンスを向上させるために、イニシエーターは最後のエージェントの役割を他のエージェントに割り当てる場合があります。最後のエージェントは、作成ウェーブには加わりません。コミット・ウェーブでは、最後のエージェントが最初にコミットします。最後のエージェントが正常にコミットしない場合には、イニシエーターは他のエージェントにロールバックするように指示します。

TCP/IP 上の DRDA 分散作業単位の場合、コーディネーターは、参加者に再同期サーバーの役割を割り当てる場合があります。再同期サーバーは、コーディネーターとの通信障害が発生した場合や、コーディネーターにシステム障害が生じた場合に、他の参加者の再同期をとることを担当します。

関連概念:



35 ページの『2 フェーズ・コミットのコミットメント定義：読み取り専用ポートの許可』

通常、トランザクション・マネージャーはコミット処理の両フェーズに参加します。コミット処理のパフォーマンスを向上させるために、トランザクション・マネージャーが読み取り専用をポートできるように、一部またはすべてのロケーションを設定することができます。

## 2 フェーズ・コミットメント制御のトランザクションの状態

コミットメント定義が、トランザクション・プログラム・ネットワークの一部である各ロケーションに確立されます。各コミットメント定義に対し、システムは、現行のトランザクションと前のトランザクションの状態を常に監視しています。

システムはその状態を使用して、トランザクションが通信障害またはシステム障害によって割り込まれた場合にコミットするかまたはロールバックするかを決定します。複数のロケーションがトランザクションに参加している場合、各ロケーションのトランザクションの状態を比較して、正しい処置 (コミットまたはロールバック) が判別されます。正しい処置を判別するために、ロケーション間で行われるこの通信処理を、再同期化といいます。

次の表に、これらの項目を示します。

- トランザクションの間に起こり得る基本状態。
- 起こり得る、上記以外の状態。
- トランザクションが通信障害またはシステム障害によって割り込まれた場合に、状態が再同期化を必要とするかどうか。示される値は、次のとおりです。

### 必要なし

各ロケーションは、それぞれで正しい決定を下せます。

### 必要の可能性あり

各ロケーションは正しい決定を下せるが、イニシエーターに決定を知らせる必要が生じることがあります。

**必須** 正しい決定を下す前に各ロケーションの状態を判別する必要があります。

- 通信障害またはシステム障害の際に取られる処置。

状態名	説明	トランザクションが割り込まれた場合の再同期	通信障害またはシステム障害の際に取られる処置
<b>2 フェーズ・コミット処理時の基本状態</b>			
リセット (RST)	コミットメント境界からプログラムがコミットまたはロールバックの要求を出すまで。	必要なし	保留中の変更はロールバックされます。
作成進行中 (PIP)	イニシエーターは作成ウェーブを開始しました。すべてのロケーションはまだポートしていません。	必要の可能性あり	保留中の変更はロールバックされます。

状態名	説明	トランザクションが割り込まれた場合の再同期	通信障害またはシステム障害の際に取られる処置
作成済み (PRP)	このロケーションおよびトランザクション・プログラム・ネットワークのこれ以降のすべてのロケーションは、コミットのポートをしました。このロケーションは、イニシエーターのコミットを知らせる通知をまだ受け取っていません。	必須	不確定。再同期化処理の結果によって異なります。
コミット進行中 (CIP)	すべてのロケーションがコミットをポートしました。イニシエーターはコミット・ウェーブを開始しました。	必須	保留中の変更はコミットされます。すべてのロケーションがコミットするように再同期化が実行されます。別のロケーションでヒューリスティック・ロールバックが報告されると、エラーが報告されます。
コミット済み (CMT)	すべてのエージェントがコミットして、このノードに応答を戻しました。	必要の可能性あり	なし
<b>2 フェーズ・コミット処理時の上記以外の状態</b>			
最終エージェントの保留 (LAP)	最終エージェントが選択された場合、この状態はイニシエーターで、PIP 状態と CIP 状態の間で起きます。イニシエーターは最後のエージェントにコミットするように指示しましたが、応答をまだ受け取っていません。	必須	不確定。再同期化処理の結果によって異なります。
読み取り専用のポート (VRO)	このエージェントは、作成ウェーブに保留中の変更がないことを応答しました。読み取り専用のポート状態が許可されると、このエージェントはコミット・ウェーブに含まれません。	必要の可能性あり	なし

状態名	説明	トランザクションが割り込まれた場合の再同期	通信障害またはシステム障害の際に取られる処置
ロールバック必須 (RBR)	<p>以下の事象のいずれかが生じました。</p> <ul style="list-style-type: none"> <li>エージェントがコミット操作の前にロールバック要求を出した。</li> <li>トランザクションに障害が発生した。</li> <li>トランザクションをロールバック必須状態にするのに、QTNRBRQD API が使用された。</li> </ul> <p>トランザクション・プログラムは、コミットメント制御下では他の変更を実行することはできません。</p>	必要の可能性あり	保留中の変更はロールバックされます。
<b>オペレーターの処置またはエラーが原因で起きる条件</b>			
強制ロールバック	このロケーションおよびトランザクション・プログラム・ネットワークのこれ以降のすべてのロケーション(ただし、最後のエージェントを除く) がオペレーターの介入によりロールバックされました。	必要の可能性あり	保留中の変更は既にロールバックされました。
強制コミット	このロケーションおよびトランザクション・プログラム・ネットワークのこれ以降のすべてのロケーション(ただし、最後のエージェントを除く) は、オペレーターの介入によりコミットしました。	必要の可能性あり	保留中の変更は既にコミットされました。
ヒューリスティック混合 (HRM)	リソース・マネージャーには、コミットしたものもありますが、ロールバックしたものもあります。オペレーターの介入が使用されたか、システム・エラーが起きました。ヒューリスティック混合は、コミットメント定義画面には状況として表示されません。通知メッセージがオペレーターに送られます。	必要の可能性あり	オペレーターは、すべての参加ロケーションで復元操作を実行して、データベースを整合した状態にする必要があります。

関連概念:



『2 フェーズ・コミットのコミットメント定義：読み取り専用ポートの許可』

通常、トランザクション・マネージャーはコミット処理の両フェーズに参加します。コミット処理のパフォーマンスを向上させるために、トランザクション・マネージャーが読み取り専用をポートできるように、一部またはすべてのロケーションを設定することができます。

37 ページの『2 フェーズ・コミットのコミットメント定義：結果を待機しない』

コミット操作中に通信またはシステム障害が生じて再同期が修復される場合、デフォルトの方式では、再同期が終了してコミット操作が完了するまで待機しています。

53 ページの『コミットメント制御の開始』

コミットメント制御を開始するには、コミットメント制御の開始 (STRCMTCTL) コマンドを使用します。

69 ページの『異常終了後の初期プログラム・ロード中のコミットメント制御リカバリー』

システムの異常終了後、初期プログラム・ロード (IPL) を実行すると、システムは、システムの終了時にアクティブであったすべてのコミットメント定義をリカバリーしようとします。

112 ページの『コミットメント制御エラー』

コミットメント制御を使用するときには、どの状態がエラーを起こし、またどの状態がエラーを起こさないかを理解することが必要です。

## 2 フェーズ・コミットメント制御のコミットメント定義

コミットメント制御の開始後、トランザクションのコミットメント・オプションを変更する場合は、コミットメント・オプションの変更 (QTNCHGCO) API を使用します。

ユーザーの環境およびアプリケーションによっては、コミットメント・オプションの変更によりシステムのパフォーマンスが改善されることがあります。

注: TCP/IP 接続上の DRDA 分散作業単位 (DUW) を使用している場合、適用されるオプションは「読み取り専用ポートの許可」のみです。

### 関連資料:

コミットメント・オプションの変更 (QTNCHGCO) API の使用

## 2 フェーズ・コミットのコミットメント定義：読み取り専用ポートの許可:

通常、トランザクション・マネージャーはコミット処理の両フェーズに参加します。コミット処理のパフォーマンスを向上させるために、トランザクション・マネージャーが読み取り専用をポートできるように、一部またはすべてのロケーションを設定することができます。

トランザクション時にコミット可能な変更がロケーションにない場合、トランザクション・マネージャーは作成ウェーブ時に読み取り専用をポートします。ロケーションはコミット・ウェーブには参加しません。このことにより全体的なパフォーマンスを改善できます。なぜなら、コミット済みウェーブの最中に通常生じる通信フローは、1 つまたは複数のリモート・ロケーションで何の更新も行われないトランザクション中で除去されるからです。

コミットメント制御を開始した後、コミットメント・オプションの変更 (QTNCHGCO) API を使用して、「許可された読み取り専用ポート」オプションを Y に変更することができます。以下を満たせば、これを行うことができます。

- 1 つまたは複数のリモート・システムにトランザクションのコミット可能な変更がない場合が多い。
- トランザクションが、前のトランザクションによって設定されたファイル・カーソル (次のレコード) の位置に依存していない。ロケーションが読み取り専用をポートした場合、トランザクションがロールバ

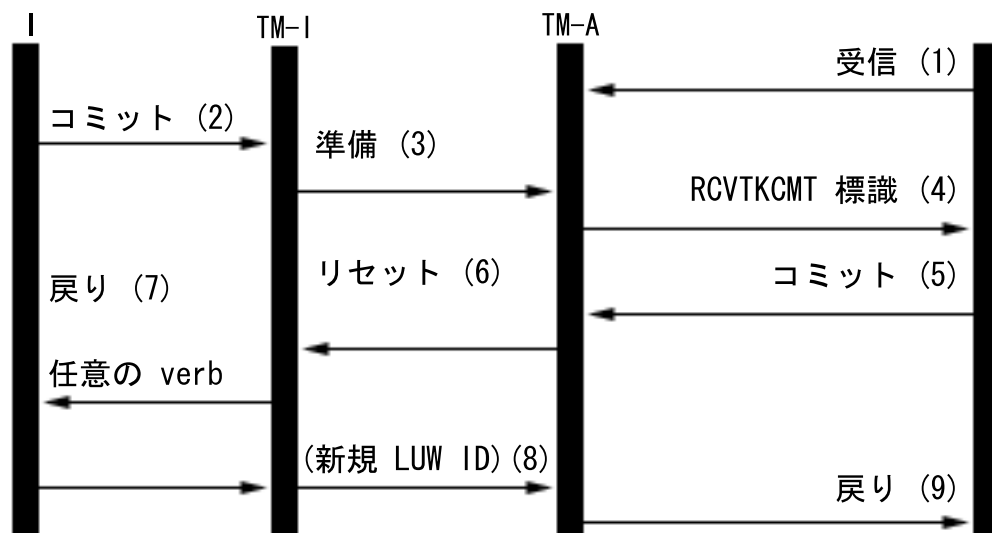
ックされてもアプリケーションには通知されません。ロケーションは、データベース・ファイルに対する読み取り操作をコミットしているのに、カーソル位置を移動しました。ファイル・カーソルの位置が重要なのは、順次処理を行っている場合だけです。

コミットメント定義が読み取り専用をポートできるように設定した場合、アプリケーションは別のロケーションからの次のメッセージ・フローを待機します。

「許可された読み取り専用ポート」オプションは、クライアント/サーバーの性格を持つアプリケーションを対象にしています。プログラム A の目的がプログラム I の要求を満たすことだけで、独立した作業をするのであれば、プログラム A に読み取り専用ポートオプションを許可するのは適切です。

**エージェントが読み取り専用をポートするとき、最終エージェント最適化をしないコミット処理のフロー**

次の図には、エージェントが読み取り専用をポートするとき、最終エージェント最適化をせずにアプリケーション・プログラムがコミット命令を出したときの、アプリケーション・プログラムとトランザクション・マネージャーとの間のメッセージのフローを示しています。イニシエーター・アプリケーション・プログラムとエージェント・アプリケーション・プログラムの両方とも、2 フェーズ・コミット処理を認識しません。図の括弧 ( ) の中の数字は、それに続く説明の番号に対応しています。



凡例

- I = イニシエーター (コミット要求を開始するアプリケーション)
- TM-I = イニシエーターのトランザクション・マネージャー
- A = エージェント (コミット要求を受信するアプリケーション)
- TM-A = エージェントのトランザクション・マネージャー

次に、エージェントが読み取り専用をポートするときの最終エージェント最適化をしない通常処理のイベントを説明します。ここでは、基本的なフローを説明します。トランザクション・プログラム・ネットワークが複数レベルになったり、エラーが起これば、イベントの手順はもっと複雑になります。

1. アプリケーション・プログラム A は、プログラム I からの要求を受け取る準備ができたことを示す受信要求を行います。
2. イニシエーター・アプリケーション (I) がコミット命令を出します。

3. イニシエーターのトランザクション・マネージャー (TM-I) がこのトランザクションのイニシエーターの役割を果たします。それは、トランザクションに参加している他のすべてのロケーションに作成メッセージを送信することによって、作成ウェーブを開始します。
4. 他のすべてのロケーションのトランザクション・マネージャーが、エージェントの役割を果たします (TM-A)。アプリケーション・プログラム A は、コミット要求を受け取ったことを TM-A により通知されます。ICF ファイルの場合、その通知はテーク・コミット受信 (RCVTKCMT) ICF 標識がオンに設定される形で行われます。
5. アプリケーション・プログラム A は、応答としてコミット命令 (またはロールバック命令) を出します。これがアプリケーション・プログラムのポートです。
6. アプリケーション・プログラム A が、コミットメント・オプションの変更 API (QTNCHGCO) を使用して、許可された読み取り専用ポート・コミットメント・オプションを Y に設定した場合と、トランザクションの最中にエージェントで何の変更もなされなかった場合に、エージェント (TM-A) はリセット・メッセージでイニシエーター (TM-I) に応答します。エージェントのコミット・ウェーブはありません。
7. アプリケーション・プログラム (A) に戻りが送られ、トランザクションがエージェント TM-A で完了していることを通知します。
8. イニシエーター (TM-I) が次にエージェント (TM-A) にメッセージ (データ・フローまたはコミットメント命令のどちらか) を出すとき、TM-I は現行トランザクション ID をメッセージと共に送るようにします。その理由は、コミット操作中に TM-I と別のシステム間で通信障害が生じると、TM-I で新しいトランザクション ID が生成されることがあるからです。
9. アプリケーション・プログラム (A) に戻りが送られ、トランザクションがエージェント TM-A で完了していることを通知します。次のメッセージを受け取るまで、戻りは遅れます。なぜなら、TM-I から新しいトランザクション ID を受け取らなければ、アプリケーション A によって次のトランザクションを開始できないからです。

#### 関連概念:

29 ページの『コミット処理での役割』

トランザクションのコミットに複数のリソース・マネージャーが含まれている場合、各リソース・マネージャーはトランザクションでそれぞれの役割を果たします。リソース・マネージャーの役割は、トランザクション時に加えられた変更のコミットまたはロールバックです。

32 ページの『2 フェーズ・コミットメント制御のトランザクションの状態』

コミットメント定義が、トランザクション・プログラム・ネットワークの一部である各ロケーションに確立されます。各コミットメント定義に対し、システムは、現行のトランザクションと前のトランザクションの状態を常に監視しています。

73 ページの『コミットメント制御のパフォーマンスの最適化』

コミットメント制御を使用するには、システム・パフォーマンスに影響する可能性のあるリソースが必要です。次のようないくつかの要因がシステム・パフォーマンスに影響します。

#### 関連資料:

コミットメント・オプションの変更 (QTNCHGCO) API の使用

#### 2 フェーズ・コミットのコミットメント定義：結果を待機しない:

コミット操作中に通信またはシステム障害が生じて再同期が修復される場合、デフォルトの方式では、再同期が終了してコミット操作が完了するまで待機しています。

注: 「結果の待機なし」オプションは、TCP/IP 接続で Distributed Relational Database Architecture (DRDA) 分散作業単位を使用している場合には適用されません。TCP/IP 接続で DRDA 分散作業単位 (DUW) は、出力待ちをまったく行いません。

次の条件が当てはまる場合には、この性質を変更することを考慮してください。

- 参加するアプリケーションが互いに独立している。
- データベース・ファイルの同期を確実に維持するのに、プログラム・ロジックに前のトランザクションの結果が必要でない。

コミットメント制御の開始後、コミットメント・オプションの変更 (QTNCHGCO) API を使用して、コミットメント定義が再同期化の結果を待機しないように指定することができます。結果の待機オプションに N (No) を指定すると、システムはデータベース・サーバー・ジョブ (QDBSRVnn) を使用して再同期を非同期で処理します。

注: これらのデータベース・サーバー・ジョブは、初期プログラム・ロード (IPL) 処理時に開始されます。コミットメント制御のオプションを変更した場合、システムが開始するジョブの数には影響ありません。

このトピックでは、解決される結果の待機オプションの 2 つの値、Y (Yes) と N (No) のみ記述していません。指定できる値は実際にはあと 2 つあり、それは L (Yes またはイニシエーターからの継承)、および U (No またはイニシエーターからの継承) です。これらの値を使用するときは、個々のコミット操作中に使用される実際の値は、システムによって Yes か No に解決されます。コミットメント・オプションの変更 (QTNCHGCO) API のトピックに、これらの値についての詳細説明があります。

注: イニシエーターの値は、イニシエーターとエージェントが presumed abort をサポートしているなら、エージェントで継承することができます。

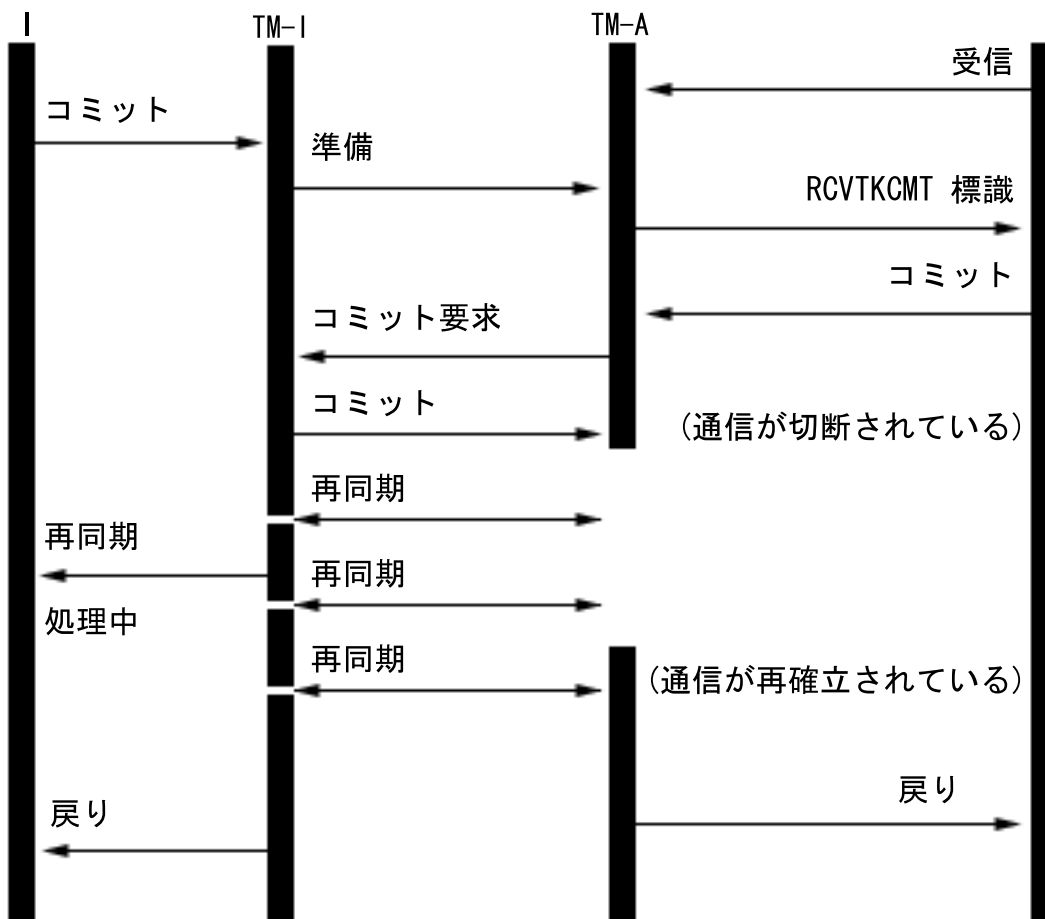
結果の待機 (WFO) オプションは通常の、エラーのないコミット処理には影響ありません。エラーが起きると、以下の条件で、WFO オプションによりアプリケーションが再同期を待機するかどうかを決定します。

- 解決した WFO オプションが Y (Yes) の場合、アプリケーションは再同期の結果を待ちます。
- 解決した WFO オプションが N (No) であり、presumed abort プロトコルをサポートしているロケーションのウェーブまたはロールバックを準備している間に通信障害が発生する場合、再同期は実行されず、コミットメント定義はロールバックされます。
- コミットメント定義が疑わしい (トランザクション状況が準備済みかまたは最終エージェント保留の) 場合、アプリケーションは、WFO 値の解決結果にかかわらず、再同期の結果を待ちます。
- 解決した WFO オプションが N で、2 つ目と 3 つ目の条件のどちらも当てはまらない場合は、システムはもう一度再同期化を試みます。再同期化が正常に実行されなかった場合、システムは状況メッセージ CPF83E6 をアプリケーションに送り、再同期化中であることを通知します。

CPF83E6 は状況メッセージであるため、アプリケーションがそれをモニターしているときにのみ効果があります。通常は、アプリケーションはこのメッセージを通知メッセージとして処理します。トランザクションに参加しているシステムは、障害が修復されるまでトランザクションを再同期化しようとします。これに続く再同期化の試みは、データベース・サーバー・ジョブで実行されます。ジョブが障害を起こしたデータベース・サーバーでこの再同期化の試みが実行される場合、メッセージ CPI83D0 が QSYSOPR へ送られます。

## 結果の待機の値: Yes

次の図では、イニシエーター (I) のコミットメント定義は、結果の待機オプションのデフォルト値である Y (Yes) を使用します。TM-I と TM-A との通信が失われると、アプリケーション A とアプリケーション I の両方が、トランザクションが再同期化されるまで待機します。

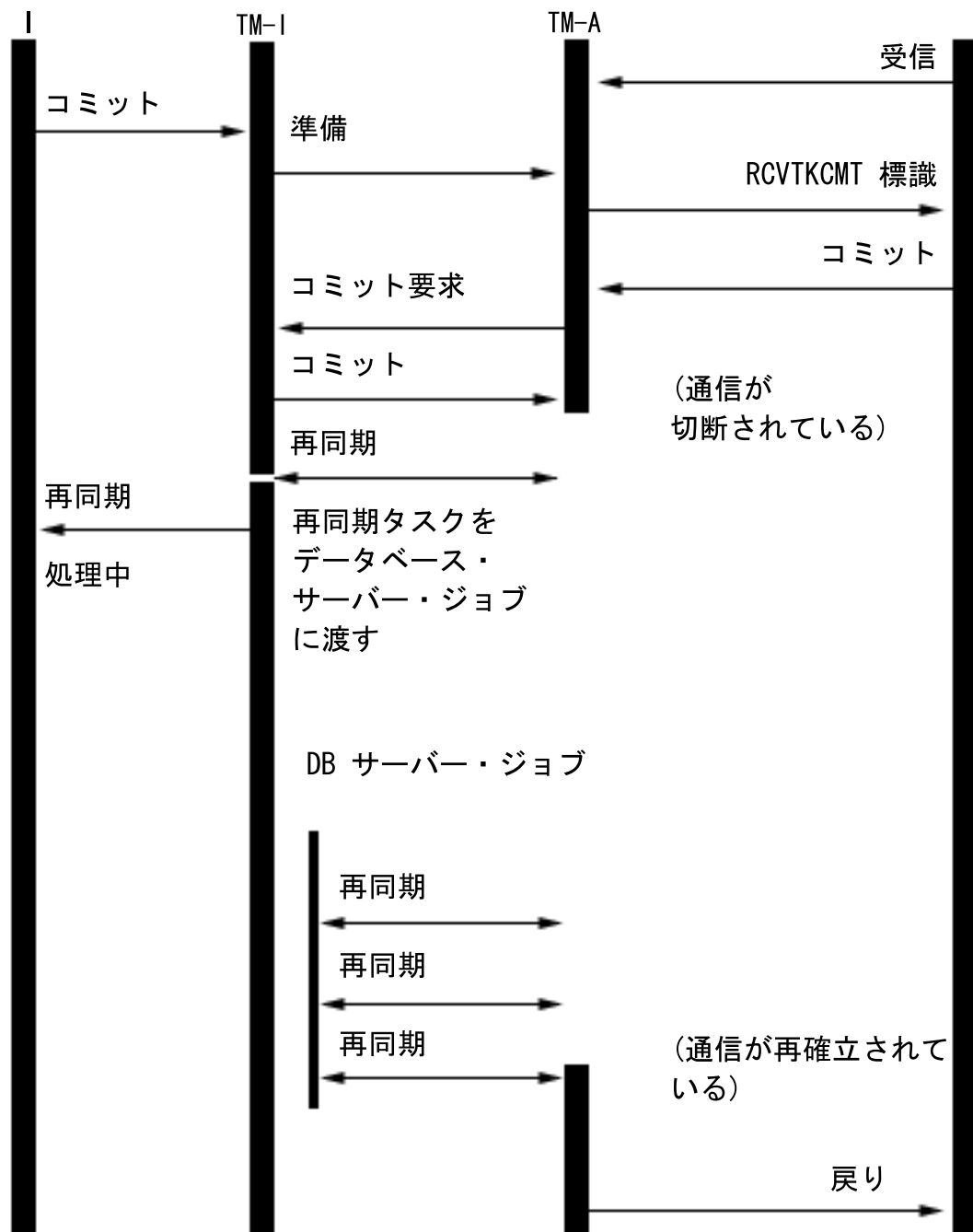


## 結果の待機の値: No

次の図では、イニシエーターのコミットメント定義により、解決される WFO が N (No) に設定されます。TM-A が前述のリストの条件 3 に合致する一方、TM-I は条件 4 に合います。アプリケーション I が TM-A により再同期化を試行した後、制御がアプリケーション I に戻されます。データベース・サーバー・ジョブが再同期化を試行します。アプリケーション I は、コミット要求が正常に完了しても戻り標識を受け取りません。通信が再確立された後まで、制御はエージェント・アプリケーション (A) に戻されません。これは、障害が生じたタイミングによって決まります。この場合、イニシエーターからコミット・メッセージを受け取る前に通信障害が生じ、TM-A がコミットするかロールバックするかについて決めていません。トランザクション・マネージャーが疑わしい場合、そのシステムの解決される WFO 値にかかわらず、再同期が完了するまでそれが制御を保存します。

再同期が完了する前にすべてのシステムのアプリケーションを続けたい場合、解決される WFO オプションをすべてのシステムで N (No) に変更するか、またはイニシエーターを N に、残りのシステムを U (No またはイニシエーターより継承) に設定する必要があります。しかし、解決される WFO オプション

は、トランザクション・マネージャーがコミットするかロールバックするか決まっていなかった場合には無視され、制御を戻す前に再同期化が完了するまで常に待機するというのを覚えておいてください。



リモート・リレーショナル・データベースへの接続が確立されるときに、保護会話がまだ開始されていないと、システムは結果の待機の値を暗黙的に N に変更します。この理由は、結果の待機の値が N で、リモート・システムが presumed abort をサポートする場合に、コミット操作のパフォーマンスが向上するためです。この結果の待機の値の暗黙的な変更は、DRDA および DDM アプリケーションでのみ実行されます。APPC アプリケーションでは、QTNCHGCO API を呼び出して変更しない限り、デフォルトの結果の待機の値 Y が使用されます。

関連概念:



32 ページの『2 フェーズ・コミットメント制御のトランザクションの状態』

コミットメント定義が、トランザクション・プログラム・ネットワークの一部である各ロケーションに確立されます。各コミットメント定義に対し、システムは、現行のトランザクションと前のトランザクションの状態を常に監視しています。

112 ページの『コミットメント制御エラー』

コミットメント制御を使用するときには、どの状態がエラーを起こし、またどの状態がエラーを起こさないかを理解することが必要です。

#### 関連資料:

コミットメント・オプションの変更 (QTNCHGCO) API の使用

### 2 フェーズ・コミットのコミットメント定義：「流用可能」の指示:

通常、トランザクション・マネージャーは、トランザクション・プログラム・ネットワーク内のすべてのロケーションで、すべてのコミットあるいはロールバック操作に参加します。パフォーマンスを向上させるために、トランザクション内の特定またはすべてのロケーションを、トランザクション・マネージャーが「流用可能」を示せるように設定することができます。

注：「流用可能の指示」オプションは、TCP/IP 接続で DRDA 分散作業単位 (DUW) を使用している場合には当てはまりません。

トランザクション中に通信フローがロケーションに送られないと、コミットまたはロールバック操作が実行されるときにそのロケーションは無視されます。このことにより全体的なパフォーマンスを改善できます。なぜなら、通常はコミットまたはロールバックの最中に生じる通信フローは、トランザクション中にデータが 1 つ以上のリモート・ロケーションに送られないと除去されるからです。

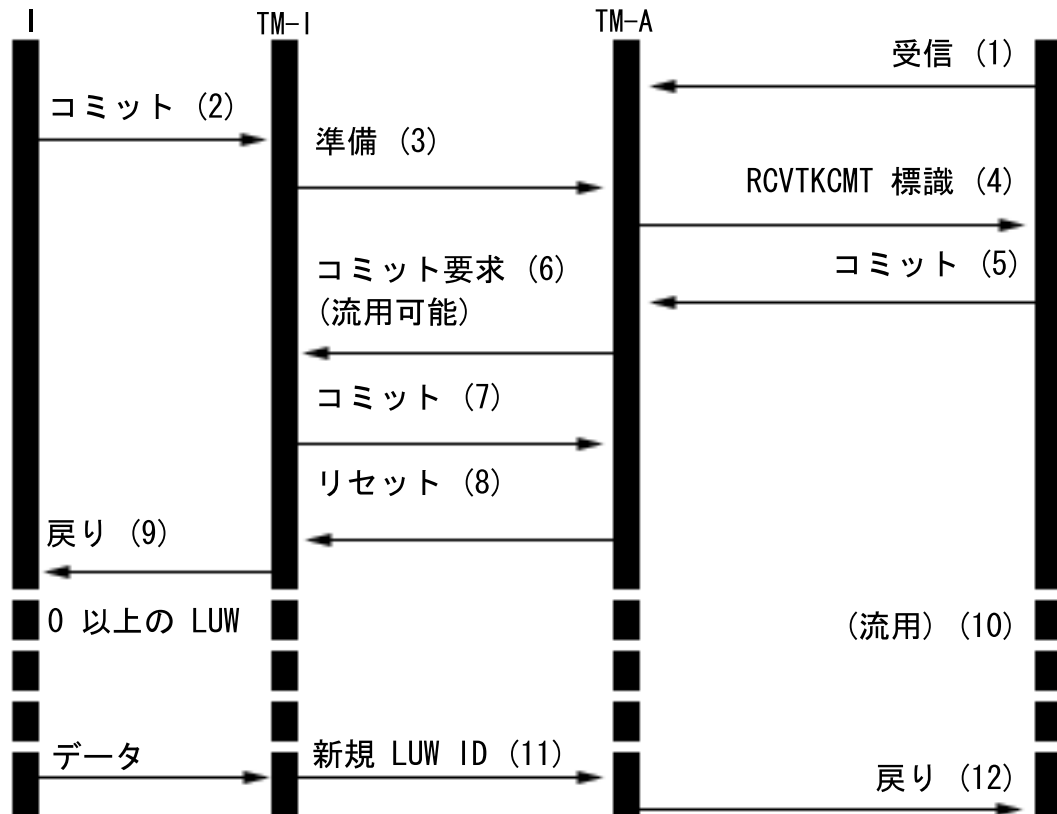
コミットメント制御を開始した後、コミットメント・オプション変更 (QTNCHGCO) API を使用して「流用可能 (OK to leave out)」オプションを Y (Yes) に変更できます。この変更は、1 つ以上のリモート・システムが頻繁にトランザクションに関係しない場合に行えます。

コミットメント定義を「流用可能」を示すように設定した場合、アプリケーションは、別のロケーションからの次のメッセージ・フローを待機します。

「流用可能」オプションは、クライアント/サーバーの性質を持つアプリケーションを想定しています。プログラム A の目的が、プログラム I の要求を満たすことだけで、独立した作業をするのであれば、プログラム A に「流用可能」オプションを使用するのは適切です。

### エージェントが「流用可能」をボートするときの、最終エージェント最適化をしないコミット処理のフロー

次の図は、エージェントが「流用可能」を示しているとき、最終エージェント最適化をせずにアプリケーション・プログラムがコミット命令を発行する場合の、アプリケーション・プログラムとトランザクション・マネージャー間のメッセージのフローを示しています。イニシエーター・アプリケーション・プログラムとエージェント・アプリケーション・プログラムの両方とも、2 フェーズ・コミット処理を認識しません。図の括弧 ( ) の中の数字は、それに続く説明の番号に対応しています。



凡例

- I = イニシエーター (コミット要求を開始するアプリケーション)
- TM-I = イニシエーターのトランザクション・マネージャー
- A = エージェント (コミット要求を受信するアプリケーション)
- TM-A = エージェントのトランザクション・マネージャー

次に、エージェントが「流用可能」をポートするときの、最終エージェント最適化をしない通常の処理のイベントについて説明します。ここでは、基本的なフローを説明します。トランザクション・プログラム・ネットワークが複数レベルになったり、エラーが起こると、イベントの手順はもっと複雑になります。

1. アプリケーション・プログラム A は、プログラム I からの要求を受け取る準備ができたことを示す受信要求を行います。
2. イニシエーター・アプリケーション (I) がコミット命令を出します。
3. イニシエーターのトランザクション・マネージャー (TM-I) がこのトランザクションのイニシエーターの役割を果たします。それは、トランザクションに参加している他のすべてのロケーションに作成メッセージを送信することによって、作成ウェーブを開始します。
4. 他のすべてのロケーションのトランザクション・マネージャーが、エージェントの役割を果たします (TM-A)。アプリケーション・プログラム A は、コミット要求を受け取ったことを TM-A により通知されます。ICF ファイルの場合、その通知はテーク・コミット受信 (RCVTKCMT) ICF 標識がオンに設定される形で行われます。
5. アプリケーション・プログラム A は、応答としてコミット命令 (またはロールバック命令) を出します。これがアプリケーション・プログラムのポートです。

6. アプリケーション・プログラム A が、コミットメント・オプション変更 API (QTNCHGCO) を使用して、「流用可能」コミットメント・オプションを Y に設定した場合、コミット要求メッセージでエージェント (TM-I) がイニシエーター (TM-A) に応答したとき、標識が送られます。

注: 「流用可能」コミットメント・オプションの変更内容は、次の正常なコミット・オプションまで効力はありません。

7. イニシエーター (TM-I) がすべてのポートを受け取ると、TM-I はコミット・メッセージを送ります。これにより、コミット・ウェーブが開始されます。
8. 各エージェント (TM-A) はコミットし、リセット・メッセージで応答します。
9. アプリケーション・プログラム (I) に戻りが送られ、トランザクションがイニシエーターで完了していることを通知します。
10. TM-I でいくつかのトランザクションが生じます。この場合、TM-A または TM-A からのデータを変更する必要はありません。TM-A は、これらのトランザクションには関係しません。
11. イニシエーター (TM-I) が次に、エージェント (A) にメッセージを発行すると、新しいトランザクション ID がメッセージと共に送られます。エージェントにメッセージを送る前にイニシエーターがコミットまたはロールバック操作を実行すると、この操作中にそのエージェントには何のメッセージも送られません。(そのエージェントは、それらのコミットやロールバックを「除外」します。) エージェントが除外されている間、イニシエーターで 1 つ以上のトランザクションがコミットまたはロールバックされるので、次のメッセージがエージェントに送られるときにイニシエーターはその現行トランザクション ID と通信する必要があります。
12. 戻りがアプリケーション・プログラム (A) へ送られ、元のコミットが完了して現行のトランザクションに参加していることを示します。

#### 関連概念:

73 ページの『コミットメント制御のパフォーマンスの最適化』

コミットメント制御を使用するには、システム・パフォーマンスに影響する可能性のあるリソースが必要です。次のようないくつかの要因がシステム・パフォーマンスに影響します。

#### 2 フェーズ・コミットのコミットメント定義：最後のエージェントを選択しない:

デフォルトでは、イニシエーターのトランザクション・マネージャーは、コミット操作中にエージェントを自由に最終エージェントとして選択することができます。

注: 「最後のエージェントを選択しない」オプションは、TCP/IP 接続で DRDA 分散作業単位 (DUW) を使用している場合には当てはまりません。

複数レベルのツリーの場合、そのイニシエーターによって最終エージェントとして選択されたエージェントは、それ自身の最終エージェントを自由に選択することもできます。コミット操作中に最終エージェントが選択されると、パフォーマンスが改善されます。イニシエーターとその最終エージェントとの間で 2 つの通信フローが削除されるからです (それらのシステムの作成段階が削除されます)。

しかし、イニシエーターが要求コミットをその最終エージェントに送ると、イニシエーターが続けられるには、その前に最終エージェントのポートを受け取るまで待機する必要があります。これは、コミットメント定義の結果の待機の値とは関係ありません。通常の、エラーがないコミット処理中では、これは問題ではありません。しかし、このウィンドウ中にエラーが生じる場合、イニシエーターは再同期が完了するまで続けることはできません。イニシエーター・アプリケーションが端末装置のユーザーからの要求を処理している場合、これは使用可能度の問題である可能性があります。

そのようなエラーが生じた場合、通常のコミット操作中のパフォーマンスの改善が、使いやすさに対する影響よりも重要であるかどうかを考慮する必要があります。要求コミットが最終エージェントに送られる前にエラーが生じる場合、LUW はすぐにロールバックし、イニシエーターは待機しないということに注意してください。そのため、エラーが生じたことでイニシエーターが待機するときのウィンドウはとて小さいので、そのようなエラーが生じることはほとんどありません。

使いやすさに対する影響がパフォーマンスの改善よりも重要でないと判断する場合、コミットメント定義を最終エージェントを選択しないように変更することができます。コミットメント制御の開始後、コミットメント・オプションの変更 (QTNCHGCO) API を使用して、「許可された最終エージェント」オプションを N に変更できます。

#### 関連資料:

コミットメント・オプションの変更 (QTNCHGCO) API の使用

#### コミット処理のフローに影響するポート信頼性:

ポート信頼性は、コミット操作後にイニシエーター・アプリケーションに早めに戻ることに、コミット操作中に 1 つのメッセージを削除することにより、最適化のパフォーマンスを向上させる最適化です。

TCP/IP 上の Distributed Relational Database Architecture (DRDA) 分散作業単位では、明示的なポート信頼性の最適化は行われません。ただし、i5/OS オペレーティング・システムは TCP/IP 接続に対してリセット (forget) 確認を要求することはありません。したがって、リセット (forget) は TCP/IP 接続に対して常に暗黙指定されます。

コミットメント制御の開始後、コミットメント・オプションの変更 (QTNCHGCO) API を使用して、「ポート信頼性の受け入れ」オプションを Y に変更することができます。

ポート信頼性は、エージェントが疑わしいときに通信障害が発生した場合にヒューリスティック判定は行われないことを、エージェントがイニシエーターに約束したものと考えることができます。ポート信頼性の最適化を使用しているエージェントは、コミットの作成ウェーブの間に、イニシエーターに標識を送ります。イニシエーターもポート信頼性の最適化を使用している場合、イニシエーターはエージェントに標識を送り、コミット・メッセージへの応答でリセットが必要ないことを示します。この方法でリセット・メッセージを削減し、またトランザクション・マネージャーはコミット・メッセージを送った後すぐにアプリケーションに戻れます。

次の条件に該当する場合には、ポート信頼性の最適化の使用を考慮してください。

- 障害を修復できない場合を除いて、システムまたは通信障害が発生した場合に疑わしいエージェントでヒューリスティック判定が行われそうもない。
- データベース・ファイルの同期を確実に維持するのに、プログラム・ロジックに前のトランザクションの結果が必要でない。

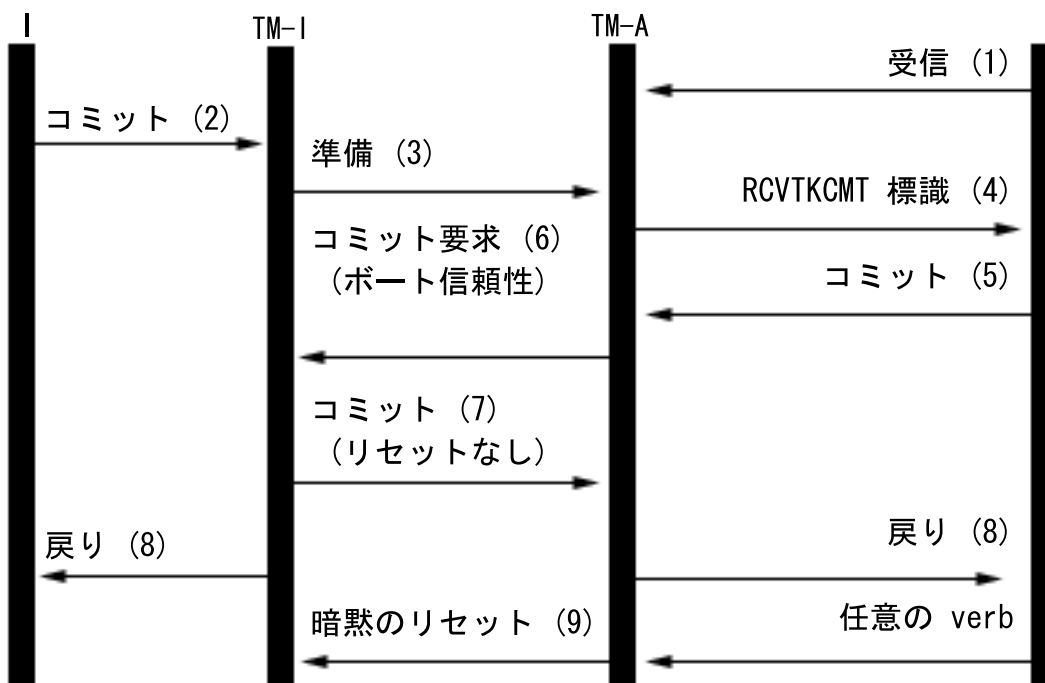
ポート信頼性の最適化は、以下の条件のすべてが該当する場合に限り、i5/OS オペレーティング・システムで使用されます。

- イニシエーターとエージェントのロケーションが、presumed abort level のコミットメント制御をサポートしている。
- イニシエーターのロケーションが、エージェントからのポート信頼性の指示を受け付ける。i5/OS イニシエーターでは、この条件は 2 つのコミットメント・オプションの値に関係しています。
  - 結果の待機のコミットメント・オプションの値が No でなければならない (デフォルトは Yes です)。

- ポート信頼性受け入れのコミットメント・オプションの値が Yes でなければならない (デフォルトは Yes です)。
- 作成ウェーブ中に、エージェントのロケーションがポート信頼性を行う。i5/OS エージェントは、常にポート信頼性を行います。その理由は、ヒューリスティック判定は、拒否の影響の可能性を警告する手動の手順を使用しなければ行えないからです。

### ポート信頼性最適化があるコミット処理のフロー

次の図は、ポート信頼性の最適化が使用されるとき、アプリケーション・プログラムとトランザクション・メッセージ間のメッセージのフローを示しています。イニシエーター・アプリケーション・プログラムとエージェント・アプリケーション・プログラムの両方とも、2 フェーズ・コミット処理を認識しません。図の括弧 ( ) の中の数字は、それに続く説明の番号に対応しています。



#### 凡例

- I = イニシエーター (コミット要求を開始するアプリケーション)
- TM-I = イニシエーターのトランザクション・マネージャー
- A = エージェント (コミット要求を受信するアプリケーション)
- TM-A = エージェントのトランザクション・マネージャー

以下に、エージェントがポート信頼性を行うとき最終エージェント最適化をしない通常処理のイベントについて説明します。ここでは、基本的なフローを説明します。トランザクション・プログラム・ネットワークが複数レベルになったり、エラーが起これば、イベントの手順はもっと複雑になります。

1. アプリケーション・プログラム A は、プログラム I からの要求を受け取る準備ができたことを示す受信要求を行います。
2. イニシエーター・アプリケーション (I) がコミット命令を出します。
3. イニシエーターのトランザクション・マネージャー (TM-I) がこのトランザクションのイニシエーターの役割を果たします。それは、トランザクションに参加している他のすべてのロケーションに作成メッセージを送信することによって、作成ウェーブを開始します。



4. 他のすべてのロケーションのトランザクション・マネージャーが、エージェントの役割を果たします (TM-A)。アプリケーション・プログラム A は、コミット要求を受け取ったことを TM-A により通知されます。ICF ファイルの場合、その通知はテーク・コミット受信 (RCVTKCMT) ICF 標識がオンに設定される形で行われます。
5. アプリケーション・プログラム A は、応答としてコミット命令 (またはロールバック命令) を出します。これがアプリケーション・プログラムのポートです。
6. エージェント (TM-A) は、イニシエーター (TM-I) にコミット要求メッセージを送ります。i5/OS システムは、ポート信頼性標識を要求コミットと共に送ります。
7. イニシエーター (TM-I) がすべてのポートを受け取ると、TM-I はコミット・メッセージを送ります。結果の待機コミットメント・オプションが N (No) で、ポート信頼性受け入れコミット・オプションが Y (Yes) の場合、コミット・メッセージと共にリセット標識は送られません。これがエージェントに、コミットへの応答にリセット・メッセージが必要でないことを知らせます。
8. トランザクションは完了します。アプリケーション・プログラム (I および A) に戻りが送られます。この戻りは、コミット操作が正常に実行されたことを示します。コミット済みのメッセージを受け取る前にヒューリスティック決定が行われたためにシステム A でヒューリスティック障害が発生すると、アプリケーション I には何も知らされません。代わりに、QSYSOPR メッセージ・キューにメッセージが送られます。しかし、アプリケーション A は、ヒューリスティック障害の標識を受け取ります。
9. 次にエージェント (TM-A) がイニシエーター (TM-I) に、データ・フローかコミットメント命令のどちらかのメッセージを送るさいに、TM-A が正常にコミットを終了したことを TM-I に知らせるメッセージと共に、暗黙のリセット標識が送られます。これは、TM-A がステップ 7 のコミット・メッセージを正常に受け取ったことを確認するまで、TM-I は完全なトランザクション情報を保持する必要があるという理由によります。

#### 関連資料:

コミットメント・オプションの変更 (QTNCHGCO) API の使用

## コミットメント制御の XA トランザクション・サポート

DB2 for i は、X/Open グローバル・トランザクションに参加することができます。

Open Group では、トランザクション作業用に業界標準モデルを定義しており、この関連を持たないリソースに対して行われた変更が 1 つのグローバル・トランザクションの一部になることを可能にします。この例として、2 つの別個のベンダーが提供するデータベースへの変更があります。このモデルは、X/Open 分散トランザクション処理 モデルと呼ばれます。

以下の資料では、X/Open 分散トランザクション処理モデルが説明されています。

- *X/Open Guide*, February 1996, Distributed Transaction Processing: Reference Model, Version 3 (ISBN:1-85912-170-5, G504), The Open Group。
- *X/Open CAE Specification*, December 1991, Distributed Transaction Processing: The XA Specification (ISBN:1-872630-24-3, C193 または XO/CAE/91/300), The Open Group。
- *X/Open CAE Specification*, April 1995, Distributed Transaction Processing: The TX (Transaction Demarcation) Specification (ISBN:1-85912-094-6, C504), The Open Group。

DB2 for i によって提供される XA トランザクション・サポートの使用を試みる前に、これらの資料の情報 (特に、XA 仕様) に精通していなければなりません。これらの資料は、The Open Group の Web サイトにあります。

分散トランザクション処理 (DTP) モデルには、5 つの構成要素があります。



### Application Program (AP)

データベースなどのリソースに関連する一連の操作を指定することによって、ユーザーの必要な機能を実装します。グローバル・トランザクションの始まりと終わりを定義し、トランザクション境界内のリソースにアクセスし、通常、それぞれのトランザクションをコミットするかロールバックするかの判断を行います。

### Transaction Manager (TM)

グローバル・トランザクションを管理します。さらに、それらを開始およびコミットまたはロールバックする判断を調整して、アトミック・トランザクションの完了を保証します。また、TM は、ある構成要素に障害が起こった場合に、RM とともにリカバリー活動を調整します。

### Resource Manager (RM)

データベース管理システムなど、コンピューターの共用リソースの定義された部分を管理します。AP は、それぞれの RM によって定義されるインターフェースを使用してトランザクションの作業を実行します。TM は、RM によって提供されるインターフェースを使用してトランザクションの完了を実行します。

### Communications Resource Manager (CRM)

モデルのインスタンスが、現行の TM ドメインの内側または外側にある別のインスタンスにアクセスすることを可能にします。CRM は DB2 for i の範囲外であり、ここでは説明されていません。

### 通信プロトコル

プロトコルは、CRM が相互に通信するために使用します。このプロトコルは DB2 for i の範囲外であり、ここでは説明されていません。

XA 仕様は DTP モデルの一部であり、DTP モデルの TM および RM 構成要素によって使用されるインターフェースのセットを記述します。DB2 for i は、これらのインターフェースを UNIX プラットフォーム・スタイルの API と出口プログラムのセットとして実装しています。これらの API の詳しい解説と、RM としての DB2 for i の使用方法について詳しくは、『XA API』を参照してください。

## IBM Navigator for i および XA トランザクション

IBM Navigator for i は、グローバル・トランザクションとしての XA トランザクションの管理をサポートします。

グローバル・トランザクションには、DB2 for i の外部と内部の両方の変更が含まれることがあります。グローバル・トランザクションは、Open Group XA アーキテクチャーまたは他の同様のアーキテクチャーを使用して、外部トランザクション・マネージャーによって調整されます。アプリケーションは、トランザクション・マネージャーによって提供されるインターフェースを使用してグローバル・トランザクションをコミットまたはロールバックします。トランザクション・マネージャーは、XA アーキテクチャーまたは他のアーキテクチャーによって定義されたコミット・プロトコルを使用して、トランザクションを完了させます。DB2 for i は、グローバル・トランザクションに参加する時に XA リソース・マネージャーの役割を果たします。グローバル・トランザクションには 2 つのタイプがあります。

- **トランザクション範囲ロック:** トランザクションのために獲得されたロックは、そのトランザクションに範囲限定されます。トランザクションは、あるジョブまたはスレッドから別のジョブまたはスレッドへ移動することが可能です。
- **ジョブ範囲ロック:** トランザクションのために獲得されたロックは、そのジョブに範囲限定されます。トランザクションは、開始されたジョブから移動することはできません。

## XA トランザクションの独立ディスク・プールの考慮事項

XA トランザクション・サポートの新規ユーザーには、トランザクション範囲ロックの XA API をお勧めします。ジョブ範囲ロックの XA API は、引き続きサポートされますが、トランザクション範囲ロックの XA API に比べて利点はなくなりました。トランザクション範囲ロックの XA API には制約事項がほとんどなく、以下の状態では、より良いパフォーマンスが得られます。

- 単一の XA トランザクション・ブランチを処理するために、常に複数の SQL 接続が使用される場合。
- 複数の並行した XA トランザクション・ブランチを処理するために、単一の SQL 接続が使用される場合。

これらの状態では、ジョブ範囲ロックに XA API を使用する場合、XA トランザクション・ブランチを実行するために別々のジョブを開始する必要があります。

DB2 for i を RM として使用する前に、以下の考慮事項と制約事項を理解しておいてください。「スレッド」という用語は、スレッド化可能ではないジョブ、またはスレッド化可能ジョブ内の単一スレッドのどちらかを意味します。

次の考慮事項は、特に記載されていない限り、トランザクション範囲ロックのトランザクションおよびジョブ範囲ロックのトランザクションの両方に適用されます。

### DB2 for i に関する考慮事項

- ローカル・データベースに対する XA トランザクションは、SQL サーバー・モードで実行しているジョブで実行する必要があります。そのようなトランザクションの場合、SQL サーバー・モードで実行されていないジョブで `xa_open()` または `db2xa_open()` API が使用されると、SQL サーバー・モードが暗黙的に開始されます。サポート対象データベース・インターフェースに関する制約事項については、『XA API』を参照してください。
- ジョブ範囲ロックに XA API を使用する場合、SQL サーバー・モードを使用するには、リモート・データベースに対する XA トランザクションが必要です。ただし、トランザクション範囲ロックに XA API を使用する場合は、リモート・データベースに対する XA トランザクションではサーバー・モードはオプションです。なお、従来の i5/OS データベース・アクセス方式を使用した DDM ファイルへの変更は、SQL サーバー・モードが使用されていない場合に、リモート・データベースに対する XA トランザクション内で許可されます。
- XA API 呼び出しの際に、XA 仕様は、DB2 for i によって検出されたすべてのエラーを戻りコードを介して報告します。エラーの意味が戻りコードだけでは判別できない可能性がある場合は、ジョブ・ログに診断メッセージが残されます。

### 組み込み SQL に関する考慮事項

- XA トランザクションに構造化照会言語 (SQL) 接続を使用するには、SQL 接続が行われる前に `xa_open()` または `db2xa_open()` アプリケーション・プログラミング・インターフェース (API) を使用しなければなりません。接続先のリレーショナル・データベースは、`xainfo` パラメーターによって `xa_open()` または `db2xa_open()` API に渡されなければなりません。接続が送られる先のジョブで 사용되는ユーザー・プロファイルとパスワードを `xa_open()` または `db2xa_open()` API に渡すことができます。それが渡されない場合、プロファイルは、接続の試行時に指定されたもの、またはデフォルトとして使用されたものを使用します。

注: 次の考慮事項は、ジョブ範囲ロックのトランザクションにのみ適用されます。

- XA トランザクションの実行に組み込み SQL が使用されると、それぞれの接続に対して実行される作業は別々のジョブに送られます (それらの接続が同じスレッド内で確立された場合でも)。これは、1 つのスレッド内のすべての接続に対して実行される作業が同じジョブに送られる、XA を使用しない SQL サ

ーバー・モードとは異なります。これは、XA 仕様が、それぞれのリソース・マネージャー・インスタンスについて別々の準備、コミット、またはロールバック呼び出しを必要とするためです。

注: 次の考慮事項は、ジョブ範囲ロックのトランザクションにのみ適用されます。

- XA トランザクションを実行するのに組み込み SQL が使用される場合、それぞれのスレッドで、各リレーショナル・データベースに 1 つの接続のみが作成できます。スレッドがアクティブにトランザクション・ブランチと関連付けられていない場合は、スレッドの接続の 1 つを使用して要求された処理は、RM で TM の `ax_reg()` 出口プログラムを使用して、その処理がトランザクション・ブランチを開始、再開、または結合するかどうかを判別します。

トランザクション・ブランチが開始される場合、対応するリレーショナル・データベースへのスレッド接続を使用して実行されます。

トランザクション・ブランチが結合される場合、トランザクション・ブランチを開始したスレッドで作成された、対応するリレーショナル・データベースへの接続を使用して経路指定されます。システムは、その接続のユーザー・プロファイルを、結合スレッドの接続のユーザー・プロファイルと同じものとするように強制はしないことに注意してください。TM は、これがセキュリティー上影響がないことを保証する責任があります。一般に TM は、すべての接続用に同じユーザー・プロファイルを使用します。このユーザー・プロファイルは、TM によって管理されているすべてのデータの権限が与えられています。このデータへのアクセスに対するこれ以上のセキュリティーは、標準的な IBM i セキュリティー手法を使用する代わりに、TM または AP によって管理されることとなります。

注: 次の考慮事項は、ジョブ範囲ロックのトランザクションにのみ適用されます。

- トランザクション・ブランチが再開される場合、使用される接続は、中断されているトランザクション・ブランチの関連が、トランザクション・ブランチの開始または結合で確立されたかどうかによって異なります。

続く処理は、`db2xa_end()` API が、スレッドのトランザクション・ブランチとの関連を中断するか終了するまで、同一の接続上で実行されます。

## CLI に関する考慮事項

- XA トランザクションを実行するために CLI が使用されている場合、`db2xa_open()` API を使用した後に複数の接続が同一のスレッド内で作成されます。他のスレッドが最初に `db2xa_open()` API に同じ `xainfo` パラメーター値を指定して使用していれば、それらの接続は、XA トランザクションを実行するために他のスレッドで使用することができます。

注: 次の考慮事項は、ジョブ範囲ロックのトランザクションにのみ適用されます。

- XA トランザクションを実行するために CLI が使用されている場合、トランザクション・ブランチを開始するために使用される接続は、そのトランザクション・ブランチで行われるすべての作業に使用されなければなりません。別のスレッドがトランザクション・ブランチに加わる場合には、トランザクション・ブランチを開始するために使用される接続用の接続ハンドルは、参加するスレッドに渡される必要があります。そうすれば、それは同じ接続上で作業を実行できます。同様に、スレッドがトランザクション・ブランチを再開しようとする場合も、同じ接続を使用する必要があります。

CLI 接続ハンドルは別のジョブでは使用できないため、結合機能の対象は、CLI の使用時にトランザクション・ブランチを開始したその同じジョブで実行しているスレッドに限られます。

## リモート・リレーショナル・データベースに関する考慮事項

注: リモート・リレーショナル・データベースについての以下の考慮事項は、ジョブ範囲ロックのトランザクションにのみ適用されます。

- リモート・リレーショナル・データベースへの XA 接続がサポートされるのは、そのリレーショナル・データベースが、分散作業単位 (DUW) DRDA 接続をサポートするシステムに存在する場合だけです。これには、SNA LU 6.2 会話で Distributed Relational Database Architecture (DRDA) を実行するか、または TCP/IP 接続を使用して DRDA を実行するときに V5R1 以降を使用する System i<sup>®</sup> 製品が含まれます。また、SNA LU 6.2 で DRDA をサポートするか、または TCP/IP で DRDA を使用して XA プロトコルをサポートする、その他のプラットフォームも含まれます。
- XA 結合機能を使用する前に、結合を行うスレッドで `db2xa_open()` API を使用しなければなりません。トランザクション・ブランチを開始したスレッドと、結合を行うスレッドの両方の `db2xa_open()` API で、同じリレーショナル・データベース名と `RMID` を指定しなければなりません。結合の試行時にトランザクション・ブランチがアクティブである場合、結合スレッドはブロックされます。この結合スレッドは、アクティブなスレッドが、関連付けられたトランザクション・ブランチを中断するか終了するまで、ブロックされたままになります。

## リカバリーでの考慮事項

- すべてのコミットメント定義に提供される手動のヒューリスティック・コミットおよびロールバック・サポートは、準備済み状態にあるトランザクション・ブランチを強制的にコミットまたはロールバックすることが必要となったときに使用できます。
- 手動のヒューリスティック・ロールバック・サポートは、アクティブまたはアイドル状態のトランザクション・ブランチにも使用できます。`xa_end` API がトランザクション・ブランチをアイドル状態に移動するために使用された後 (ただし、`xa_commit` または `xa_rollback` がそのトランザクションを完了するために使用される前) で、クライアント接続が失敗したときに、このサポートは特に重要になります。接続が失敗した後で、クライアントのトランザクション・マネージャーがアイドルのトランザクション・ブランチを完了するために復帰しない場合、アイドルのトランザクション・ブランチは孤立化し、システムを再始動するか、手動でヒューリスティック・ロールバックが実行されない限り、保留のままとなります。
- また、ヒューリスティックに完了した状態にあるトランザクション・ブランチを無視するための、手動オプションもあります。トランザクション・マネージャーが、ヒューリスティック判定戻りコードを受け取った後で、`xa_forget` API を発行するために XA プロトコルに従わない場合、システムを再始動しても、トランザクション・ブランチは孤立化しヒューリスティックに完了した状態のままになります。トランザクション・ブランチは、この状態でいかなる保留中の変更またはロックも保持していませんが、`forget` オプションが実行されるときに解放されるシステム記憶域を消費します。

## トランザクション・ブランチに関する考慮事項

- XA トランザクション・ブランチに関する情報は、IBM Navigator for i、およびジョブ処理 (WRKJOB)、ジョブ表示 (DSPJOB)、およびコミットメント定義の処理 (WRKCMTDFN) の各コマンドによって表示されるコミットメント制御情報の一部として示されます。TM 名、トランザクション・ブランチの状態、トランザクション ID、およびブランチ修飾子がすべて示されます。コマンド `WRKCMTDFN JOB(*ALL) STATUS(*XOPEN)` を使用するか、または IBM Navigator for i で「グローバル・トランザクション」フォルダーを表示することにより、現在アクティブなすべての XA トランザクションに関連するコミットメント定義を表示することができます。

注: 次の項目は、ジョブ範囲ロックのトランザクションにのみ適用されます。

- スレッドと既存のトランザクション・ブランチとの関係が `db2xa_end()` API を使用して中断または終了されている場合、スレッドは新しいトランザクション・ブランチを開始することができます。新しいト



ランザクション・ブランチを開始するために使用される接続が、別のランザクション・ブランチの開始よりも前に使用されており、スレッドとそのランザクション・ブランチとの関連が、`db2xa_end()` API により終了または中断していた場合には、新しい SQL サーバー・ジョブが開始されることとなります。新しい SQL サーバー・ジョブが必要になるのは、`db2xa_commit()` または `db2xa_rollback()` API により最初のランザクション・ブランチがまだ完了していない場合だけです。この場合、別の完了メッセージ SQL7908 がジョブ・ログに送られます。これは、接続が確立されたときに、新しい SQL サーバー・ジョブが、接続のオリジナル SQL サーバー・ジョブと同様に識別されたことを示しています。新しいランザクション・ブランチに対するすべての SQL 要求は、新しい SQL サーバー・ジョブに経路指定されます。`db2xa_commit()` または `db2xa_rollback()` API によりランザクション・ブランチが完了すると、新しい SQL サーバー・ジョブはリサイクルされ、事前開始ジョブ・プールに戻されます。

- トランザクション・ブランチは、ジョブ範囲ロックの XA トランザクションの場合にのみ、以下の状況でロールバックのみとマークされます。
  - スレッドがランザクション・ブランチと関連したまま、そのスレッドが終了する。これには、プロセス終了の結果として終了するスレッドが含まれます。
  - システムに障害が起こる。
- トランザクション範囲ロックの XA トランザクションでは、次のいずれかの状況が起こったときにランザクション・ブランチと関連したままのスレッドがある場合、そのランザクション・ブランチはシステムによってロールバックされます。
  - トランザクション・ブランチに関連する接続が終了される。
  - トランザクション・ブランチを開始したジョブが終了される。
  - システムに障害が起こる。

注: 次の考慮事項は、ジョブ範囲ロックのランザクションにのみ適用されます。

- 関連しているスレッドが存在するかどうかにかかわらず、ランザクション・ブランチがシステムによってロールバックされる次のような 1 つの状況があります。これは、接続の処理の経路指定先の SQL サーバー・ジョブが終了するときです。これは、そのジョブに対してジョブ終了 (ENDJOB) CL コマンドが使用されるときにのみ起こります。
- 次のいずれかの状況が起こったとしても、ランザクション・ブランチとアクティブに関連しているスレッドがない場合、そのランザクション・ブランチは影響を受けません。TM は、ランザクション・ブランチを開始したスレッドで指定されたのと同じ `xainfo` パラメーター値が指定されている `xa_open()` または `db2xa_open()` API を使用した任意のスレッドから、ランザクション・ブランチをコミットまたはロールバックすることができます。
  - トランザクション・ブランチに関連する接続が終了される。
  - トランザクション・ブランチの処理を実行したスレッドまたはジョブが `xa_close()` または `db2xa_close()` API を使用する。
  - システムに障害が起こる。この場合、ランザクション・ブランチは、準備済み状態にある場合にのみ影響を受けません。アイドル状態にある場合は、システムはそのランザクション・ブランチをロールバックします。
- 2 つの XA トランザクション・ブランチのランザクション ID (XID) が同じグローバル・ランザクション ID (GTRID) をもっているが、ブランチ修飾子 (BQUAL) が異なる場合、これらのランザクション・ブランチは疎結合であるといえます。デフォルトで、疎結合ランザクション・ブランチはロックを共用しません。ただし、ランザクション範囲ロックに XA API を使用した場合には、疎結合ランザクションがロックを共用することを許可するオプションがあります。

関連概念:

25 ページの『XA トランザクションの独立ディスク・プールの考慮事項』

XA 環境では、各データベースが独立したリソース・マネージャーと見なされます。トランザクション・マネージャーが、同じトランザクションの下で 2 つのデータベースをアクセスする場合は、XA プロトコルを使用して 2 つのリソース・マネージャーとの 2 フェーズ・コミットを実行しなければなりません。

 The Open Group の Web サイト

『コミットメント制御のための SQL サーバー・モードおよびスレッド範囲トランザクション』  
ジョブ範囲ロックのコミットメント定義は通常、活動化グループに範囲が限定されます。

**関連タスク:**

122 ページの『コミットとロールバック操作の強制時点および再同期の取り消し時点』

コミットまたはロールバック操作の強制の決定は、ヒューリスティック判定 と呼ばれます。この処置により、オペレーターは、準備済み状態にあるトランザクションのリソースを手動でコミットまたはロールバックできるようになります。

## コミットメント制御のための SQL サーバー・モードおよびスレッド範囲トランザクション

ジョブ範囲ロックのコミットメント定義は通常、活動化グループに範囲が限定されます。

あるジョブがマルチスレッドの場合、そのジョブ内のすべてのスレッドがコミットメント定義にアクセスし、特定のトランザクションについて行われた変更は複数のスレッドに広がります。つまり、プログラムが同じ活動グループで実行されるすべてのスレッドは、1 つのトランザクションに参加することになります。

トランザクションの作業を活動グループではなく、スレッドに範囲限定することが望ましい場合があります。その場合は、それぞれのスレッドが独自のコミットメント定義を持ち、それぞれのコミットメント定義に対応するトランザクションの作業は、他のスレッドで実行される作業とは独立して行われます。

DB2 for iは、ジョブの変更 (QWTCGJJB) API を使用して、ジョブが SQL サーバー・モードで実行されるように変更することによって、このサポートを提供します。SQL サーバー・モードで SQL 接続が要求されると、それは独自のジョブに送られます。その接続に対して実行される後続のすべての SQL 命令も、そのジョブに送られます。接続が確立されると、完了メッセージ SQL7908 が SQL サーバー・モード・ジョブのジョブ・ログに送られて、SQL 要求が送られるジョブを示します。コミットメント定義は、このメッセージで示されたジョブによって所有されます。エラーが発生した場合は、問題の発生源を理解するために両方のジョブのジョブ・ログを調べなければならないことがあります (SQL ステートメントを実行するジョブでは、実際の作業が行われていないため)。

SQL サーバー・モードでの実行時には、コミットメント制御のもとで作業を実行するために SQL インターフェースのみ使用できます。組み込み SQL またはコール・レベル・インターフェース (CLI) も使用できます。1 つのスレッド内で組み込み SQL を介して確立されるすべての接続は、同じバックエンド・ジョブに送られます。これにより、SQL サーバー・モードで実行されていないジョブの場合と同様に、1 つのコミット要求ですべての接続に対する作業をコミットすることができるようになります。CLI を介して確立されるそれぞれの接続は、別々のジョブに送られます。CLI では、それぞれの接続に対して実行される作業が独立してコミットまたはロールバックされることが必要です。

SQL サーバー・モードでの実行時には、コミットメント制御のもとで次の操作を実行することはできません。

- SQL インターフェース以外のインターフェースで行われたレコード変更



- DDM ファイルへの変更
- API コミットメント・リソースへの変更

コミットメント制御は、SQL サーバー・モードで実行されているジョブで直接開始することはできません。

#### 関連概念:

46 ページの『コミットメント制御の XA トランザクション・サポート』  
DB2 for i は、X/Open グローバル・トランザクションに参加することができます。  
サーバー・モードでの DB2 CLI の実行  
SQL サーバー・モードでの DB2 CLI の開始  
サーバー・モードで DB2 CLI を実行する場合の制約事項

#### 関連資料:

ジョブの変更 (QWTCHGJB) API

---

## コミットメント制御の開始

コミットメント制御を開始するには、コミットメント制御の開始 (STRCMTCTL) コマンドを使用します。

注: コミットメント制御を SQL アプリケーションから開始する必要はありません。SQL 分離レベルが \*NONE でない場合、SQL は接続時にコミットメント制御を暗黙的に開始します。

STRCMTCTL コマンドを使用するとき、次のパラメーターを指定できます。

#### コミット・ロック・レベル

STRCMTCTL コマンドの LCKLVL パラメーターでロック・レベルを指定します。指定したレベルは、コミットメント定義のコミットメント制御下でオープンされて配置されるデータベース・ファイルのレコード・ロックのデフォルトのレベルになります。

#### 通知オブジェクトのコミット

通知オブジェクトを指定するには、NTFY パラメーターを使用します。通知オブジェクトはメッセージ・キュー、データ域、またはデータベース・ファイルで、そのコミットメント定義が正常に終了しなかった場合、特定のコミットメント定義の最後に正常に完了したトランザクションを識別する情報が含まれています。

#### コミット範囲パラメーター

コミット範囲を指定するには、CMTSCOPE パラメーターを使用します。コミットメント制御を開始すると、システムはコミットメント定義を作成します。コミット範囲パラメーターは、コミットメント定義の有効範囲を識別します。デフォルトでは、コミットメント制御の開始の要求を行ったプログラムの活動化グループまでをコミットメント定義の有効範囲とします。もう 1 つの範囲はジョブまでです。

#### デフォルトのジャーナル・パラメーター

コミットメント制御を開始するとき、デフォルトのジャーナルを指定することができます。次の理由でデフォルトのジャーナルを使用することができます。

- トランザクション・ジャーナル項目を取り込みたい。これらの項目はユーザーがトランザクションに関連するリソースのヒストリーを分析するのに役立つことができます。それらの項目はジャーナル変更を適用および除去するためには使用されません。除外するジャーナル項目 (OMTJRNE) パラメーターは、システムがトランザクション項目を書き込むかどうかを判別します。

- 経路指定ステップ内でファイルをクローズして再びオープンするジョブのパフォーマンスを改善したい。デフォルトのジャーナルではないジャーナルに割り当てられたすべてのファイルをクローズするとき、ジャーナルについてのすべてのシステム情報は経路指定ステップから除去されます。そのジャーナルに割り当てられたファイルが後でオープンされる場合、ジャーナルについてのすべての情報を再び作成しなければなりません。ジャーナルに割り当てられたリソースがアクティブであるかどうかに関係なく、システムはコミットメント定義でのデフォルトのジャーナルについての情報を保持します。

### コミット・テキスト・パラメーター

TEXT パラメーターを使用して、ジョブ用に開始されたコミットメント定義についての情報を表示するときの、そのコミットメント定義に関連付けられる特定のテキストを識別します。テキストを指定しないと、システムがデフォルトのテキスト記述を提供します。

### ジャーナル項目パラメーターの省略

パフォーマンスを改善するためにデフォルトのジャーナルを指定する場合、OMTJRNE パラメーターを使用して、システムがトランザクション・ジャーナル項目を書き込むのを防ぐことができます。システムにトランザクション項目の書き込みを許可すると、ジャーナル・レシーバーのサイズが大幅に増し、コミットおよびロールバック操作中のパフォーマンスが大幅に低下します。

トランザクション項目は、コミットメント制御環境か新しいアプリケーションのどちらかをセットアップまたはテストしているときに役立ちます。

トランザクション項目は、次の条件において OMTJRNE パラメーターの値にかかわらず、デフォルトのジャーナルに書き込まれます。

- コミットまたはロールバック操作中にシステム・エラーが生じる。
- トランザクションに関係するリソースに変更が手作業で加えられ、その変更によりヒューリスティック混合条件が生じる。ヒューリスティック混合条件についての説明は、2 フェーズ・コミットメント制御のトランザクションの状態を参照してください。この種の手作業変更は、ヒューリスティック判定と呼ばれます。

これらの状態をとるべき処置を判別するために、トランザクションに関係するリソースについての情報を使用することができます。

ジャーナル項目情報ファインダーを使用して、トランザクション (コミットメント制御) ジャーナル項目について項目固有のデータのレイアウトを表示することができます。

### 関連概念:

32 ページの『2 フェーズ・コミットメント制御のトランザクションの状態』

コミットメント定義が、トランザクション・プログラム・ネットワークの一部である各ロケーションに確立されます。各コミットメント定義に対し、システムは、現行のトランザクションと前のトランザクションの状態を常に監視しています。

ジャーナル項目情報ファインダー

### 関連タスク:

122 ページの『コミットとロールバック操作の強制時点および再同期の取り消し時点』

コミットまたはロールバック操作の強制の決定は、ヒューリスティック判定 と呼ばれます。この処置により、オペレーターは、準備済み状態にあるトランザクションのリソースを手動でコミットまたはロールバックできるようになります。

### 関連資料:

コミットメント制御の開始 (STRCMTCTL) コマンド

## 通知オブジェクトのコミット

通知オブジェクトはメッセージ・キュー、データ域、またはデータベース・ファイルで、そのコミットメント定義が正常に終了しなかった場合、特定のコミットメント定義の最後に正常に完了したトランザクションを識別する情報が含まれています。

コミットメント定義の最後に正常に終了したトランザクションのコミット ID は、特定のコミット可能リソース、変更コミット操作に関連付けられます。

コミットメント定義の最後に正常に終了したトランザクションのコミット ID は、コミットメント定義が正常に終了しない場合だけ通知オブジェクトに入れられます。この情報は、アプリケーションを再び開始するにはどこでアプリケーションの処理を終了するかを判断するために使用することができます。

独立ディスク・プールの場合、その通知オブジェクトは、コミットメント定義と同じ独立ディスク・プールまたは独立ディスク・プール・グループ上になければなりません。コミットメント定義を別の独立ディスク・プールまたは独立ディスク・プール・グループに移動する場合、通知オブジェクトはその別の独立ディスク・プールまたは独立ディスク・プール・グループ上になければなりません。コミットメント定義が異常終了すると、他の独立ディスク・プールまたは独立ディスク・プール・グループ上にある通知オブジェクトが更新されます。他の独立ディスク・プールまたは独立ディスク・プール・グループ上に通知オブジェクトが見付からないと、更新処理はメッセージ CPF8358 で失敗します。

ジャーナルされたリソースが現行トランザクションに関係していて、コミット操作がコミット ID を指定して実行される場合、コミット ID はその特定のトランザクションがコミット中であることを識別するコミット・ジャーナル項目 (ジャーナル・コードおよび項目タイプ C CM) に入れられます。コミット ID を含むコミット・ジャーナル項目は、トランザクションに関係するリソースと関連がある各ジャーナルに送られます。

次の表では、コミット ID およびその最大サイズを指定する方法を示します。コミット ID がその最大サイズを超える場合、通知オブジェクトに書き込まれるとき切り捨てられます。

言語	操作	コミット ID 中の最大文字数
CL	COMMIT コマンド	3000 <sup>1</sup>
統合化言語環境 (Integrated Language Environment) (ILE) RPG	COMIT 操作コード	4000 <sup>1</sup>
PLI	PLICOMMIT サブルーチン	4000 <sup>1</sup>
ILE C	_Rcommit 機能	4000 <sup>1</sup>
ILE COBOL	COMMIT verb	サポートしない
SQL	COMMIT ステートメント	サポートしない

注: <sup>1</sup>通知オブジェクトがデータ域である場合、最大サイズは 2000 文字です。

通知オブジェクトがコミット ID で更新されるときは、次のように更新が行われます。

### データベース・ファイル

データベース・ファイルを通知オブジェクトとして使用すると、コミット ID がファイルの終わりに追加されます。既存のレコードはすべてのファイル内に残ります。複数のユーザーやジョブが同時にレコードを変更する可能性があるため、ファイルの各コミット ID には、失敗したジョブおよびコミットメント定義にデータを関連付けるために固有の情報が含まれています。通知オブジェクトとして働くファイルをジャーナルすることができます。

## データ域

データ域を通知オブジェクトとして使用した場合には、コミット ID がデータ域に入れられたときにデータ域の内容全体が置き換えられます。複数のユーザーまたはジョブが同じプログラムを使用している場合には、データ域には正常に終了しなかった最後のコミットメント定義からのコミット ID だけが含まれます。したがって、1 つのデータ域通知オブジェクトだけでは、再びアプリケーション・プログラムを開始するための正しい情報は得られません。この問題を解決するには、各ワークステーション・ユーザーまたはジョブの各コミットメント定義ごとに別々のデータ域を使用してください。

## メッセージ・キュー

メッセージ・キューを通知オブジェクトとして使用した場合には、メッセージ CPI8399 がメッセージ・キューに送られます。コミット ID はメッセージ CPI8399 の 2 次レベルのテキストに入れます。通知オブジェクトにデータベース・ファイルを使用する場合と同様に、アプリケーション・プログラムを再開するために、各コミット ID の内容が、ジョブの各コミットメント定義を固有に識別します。

## 関連概念:

27 ページの『バッチ・アプリケーションのコミットメント制御』

バッチ・アプリケーションは、コミットメント制御を必要とする場合とそうでない場合があります。場合によっては、バッチ・アプリケーションは、入力ファイルの読み取りとマスター・ファイルの更新の単一機能しか実行できないことがあります。ただし、異常終了の後に再開することが重要な場合には、この種のアプリケーションに対してコミットメント制御を使用することができます。

97 ページの『例: アプリケーション開始のための通知オブジェクトの使用』

異常終了後にプログラムが開始されると、そのプログラムは通知オブジェクトの中の項目を検索します。項目が存在すると、プログラムは再度トランザクションを開始することができます。トランザクションが再開されると、通知オブジェクトはプログラムにより消去されます。これにより、同じトランザクションが別の時に開始されないようにします。

## ロック・レベルのコミット

コミットメント制御の開始 (STRCMTCTL) コマンドで LCKLVL パラメーターに指定する値は、コミットメント定義のコミットメント制御でオープンされて配置されるデータベース・ファイルのレコード・ロックのデフォルトのレベルになります。

ローカル・データベース・ファイルをオープンするときにデフォルトのレコード・ロック・レベルを指定変更することはできません。ただし、SQL によってアクセスされるデータベース・ファイルは、最初に SQL ステートメントが出された時に有効な現行 SQL 分離レベルを使用します。

ロック・レベルはユーザーの要件、許容待機時間、および最も頻繁に使用される解放手順について指定する必要があります。

以下の説明は、コミットメント制御下でオープンされるファイルだけに当てはまります。

### \*CHG ロック・レベル

同時に稼働する他のジョブによる変更から変更レコードを保護したい場合にはこの値を使用してください。コミットメント制御下でオープンされるファイルでは、トランザクションの期間中そのロックが保持されます。コミットメント制御下でオープンされないファイルでは、レコード上のロックはレコードが読み取られるときから更新操作が完了するまで保持されます。

### \*CS ロック・レベル

同時に稼働する他のジョブによる変更から変更および検索レコードの両方を保護するためにこの値

を使用してください。変更されないが検索されたレコードが保護されるのは、それらのレコードが解放されるまで、または別のレコードが検索されるまでです。

\*CS ロック・レベルは、このジョブによって読み込まれた更新レコードを他のジョブは読み取ることができないことを保証します。さらに、別のジョブで \*UPDATE のレコード・ロック・タイプを使用してロックされたレコードをプログラムが読み取れるのは、そのジョブが別のレコードにアクセスしてからです。

**\*ALL ロック・レベル**

コミットメント制御下で同時に稼働する他のジョブによる変更から、コミットメント制御下にある変更レコードおよび検索レコードを保護するためにこの値を使用してください。検索または変更されたレコードが保護されるのは、次のコミットまたはロールバック操作までです。

\*ALL ロック・レベルは、このジョブによって読み込まれた更新レコードに他のジョブがアクセスできないことを保証します。これはロックの標準プロトコルとは異なります。ロック・レベルを \*ALL に指定すると、レコードが別のジョブで \*UPDATE のレコード・ロック・タイプでロックされている場合は、更新のために読み取るのではなくてもそのレコードにアクセスすることはできません。

次の表は、ファイルがコミットメント制御下にある場合とそうでない場合のレコード・ロックの持続期間を示しています。

要求	LCKLVL パラメーター	ロックの持続期間	ロックのタイプ
読み取り専用	コミットメント制御なし	ロックなし	なし
	*CHG	ロックなし	なし
	*CS	読み取りから次の読み取り、コミット、またはロールバック	*READ
	*ALL	読み取りからコミットまたはロールバック	*READ
更新用に読み取ってから更新または削除する <sup>1</sup>	コミットメント制御なし	読み取りから更新または削除	*UPDATE
	*CHG	読み取りから更新または削除	*UPDATE
		更新または削除から次のコミットまたはロールバック <sup>2</sup>	*UPDATE
	*CS	読み取りから更新または削除	*UPDATE
		更新または削除から次のコミットまたはロールバック <sup>2</sup>	*UPDATE
	*ALL	読み取りから更新または削除	*UPDATE
		更新または削除から次のコミットまたはロールバック <sup>2</sup>	



要求	LCKLVL パラメーター	ロックの持続期間	ロックのタイプ
更新用に読み取ってから解放する <sup>1</sup>	コミットメント制御なし	読み取りから解放	*UPDATE
	*CHG	読み取りから解放	*UPDATE
	*CS	読み取りから解放、コミットまたはロールバック	*UPDATE
		解放から次の読み取り、コミット、またはロールバック	*UPDATE
	*ALL	読み取りから解放、コミットまたはロールバック	*UPDATE
		解放から次のコミットまたはロールバック	
追加	コミットメント制御なし	ロックなし	なし
	*CHG	追加からコミットまたはロールバック	*UPDATE
	*CS	追加からコミットまたはロールバック	*UPDATE
	*ALL	追加からコミットまたはロールバック	*UPDATE
直接書き込み	コミットメント制御なし	直接書き込み期間	*UPDATE
	*CHG	直接書き込みからコミットまたはロールバック	*UPDATE
	*CS	直接書き込みからコミットまたはロールバック	*UPDATE
	*ALL	直接書き込みからコミットまたはロールバック	*UPDATE

**注:**

<sup>1</sup>コミットまたはロールバック操作が更新用読み取り操作後でレコードが更新、削除、または解放される前に実行される場合、レコードはコミットまたはロールバック操作中にアンロックされます。レコード上の保護は、コミットまたはロールバックが完了するとすぐに失われます。

<sup>2</sup>レコードが削除されてもコミットまたはロールバックがまだトランザクションに出されていないとき、削除されたレコードはロックされたままではありません。同じまたは別のジョブが削除されたレコードをキーによって読み取ろうとすると、ジョブはレコードが見つからないという指示を受け取ります。しかし、固有キー・アクセス・パスがファイルを超えて存在する場合、トランザクションがコミットされるまで、削除されたレコードと同じ固有キー値で別のジョブがレコードを挿入または更新することはできません。

ロック・レベルが \*CS または \*ALL のときに更新のために読み取られることのないレコードには、\*READ のタイプのレコード・ロックがレコード上で取得されます。このタイプのロックは、別のジョブが更新のためにレコード読み取りをできなくしますが、レコードが読み取り専用操作からアクセスされるのは防げません。

\*UPDATE レコード・ロック・タイプは、更新、削除、追加、または更新のための読み取りが行われるレコード上で入手されます。このタイプのロックは、他のジョブが更新のためのレコードを読み取るのを防ぎ、コミットメント制御下で稼働してレコード・ロック・レベル \*CS または \*ALL を持つジョブが読み取り専用操作のためのレコードにもアクセスできないようにします。



コミットメント制御を使用していないプログラムは、別のジョブがロックしたレコードを読み取ることはできませんが、LCKLVL パラメーターに指定される値にかかわらず、レコードを更新のために読み取ることはできません。

活動化グループまたはジョブのためのコミットメント制御が開始するときにコミットメント定義に指定されるロック・レベルは、特定のコミットメント定義に関連するオープンだけに適用されます。

注: \*CS および \*ALL ロック・レベル値は、現在未処理の変更のあるレコードをユーザーが別のジョブから取り出さないよう保護します。しかし、\*CS および \*ALL ロック・レベル値は、同じジョブ内の異なる活動化グループで稼働するプログラムからの変更を現在保留している活動化グループで稼働するプログラムを使用して、レコードを検索しないよう保護することはありません。

同じジョブ内では、同じコミットメント定義を使用して再びレコードがアクセスされる限り、プログラムは現行のトランザクション内で既に変更されたレコードを変更することができます。ジョブ・レベルのコミットメント定義を使用しているときには、変更済みのレコードへのアクセスは、ジョブ・レベルのコミットメント定義を使用する任意の活動化グループで稼働しているプログラムから行うことができます。

#### 関連概念:

25 ページの『コミットメント制御に関する考慮事項と制約事項』

コミットメント制御に関する考慮事項および制約事項を認識しておく必要があります。

#### 関連資料:

コミットメント制御の開始 (STRCMTCTL) コマンド

---

## コミットメント制御の終了

ジョブ・レベルまたは活動化グループ・レベルのコミットメント定義のいずれのコミットメント制御も、コミットメント制御の終了 (ENDCMTCTL) コマンドを使用して終了させることができます。

ENDCMTCTL コマンドを出すと、要求を出しているプログラムが使用中のコミットメント定義を終了することをシステムに指示します。ENDCMTCTL コマンドは、ジョブの 1 つのコミットメント定義のみを終了し、そのジョブの他のすべてのコミットメント定義は変わりません。

活動化グループ・レベルのコミットメント定義を終了した場合は、ジョブに対してそのジョブ・レベルのコミットメント定義が既に開始されていない限り、その活動化グループ内で実行されているプログラムは、コミットメント制御下で変更を行えなくなります。ジョブ・レベルのコミットメント定義が活動化されている場合、コミットメント制御を終了したばかりの活動化グループ内で稼働するプログラムは、その定義をただちに使用できるようになります。

ジョブ・レベルのコミットメント定義を終了した場合には、そのジョブ・レベル・コミットメント定義を使用していたジョブで実行されていたプログラムは、まず STRCMTCTL コマンドを使って再度コミットメント制御を開始しないと、コミットメント制御の下で変更を行えなくなります。

ENDCMTCTL コマンドを出す前に、コミットメント定義を終了するために以下の条件を満足する必要があります。

- 終了したいコミットメント定義のコミットメント制御下でオープンされたすべてのファイルをまずクローズしなければなりません。ジョブ・レベルのコミットメント定義を終了する場合は、これにはそのジョブ・レベル・コミットメント定義を使っている活動化グループ内で稼働するプログラムによってコミットメント制御下でオープンされたすべてのファイルが含まれます。
- まず QTNRMVCR API を使って、終了したいコミットメント定義の API コミットメント・リソースをすべて除去しなければなりません。ジョブ・レベルのコミットメント定義を終了する場合、これにはそ

のジョブ・レベル・コミットメント定義を使っている活動化グループ内で稼働するプログラムによって追加されたすべての API コミットメント・リソースが含まれます。

- 終了したいコミットメント定義と関連したリモート・データベースを切り離さなければなりません。
- コミットメント定義に関連したすべての保護会話は、正しい同期レベルを使用して正常終了しなければなりません。

コミットメント制御が対話式ジョブにおいて終了される場合で、しかもそのコミットメント定義と関連した 1 つまたは複数のコミット可能リソースに保留中の変更がある場合は、照会メッセージ CPA8350 がユーザーに送られ、保留中の変更のコミット、保留中の変更のロールバック、または ENDCMTCTL 要求の取り消しのいずれを行うかが問われます。

コミットメント制御がバッチ・ジョブ内で終了し、コミットメント定義に関連した 1 つまたは複数のクローズされたファイルに保留中の変更がある場合には、その変更はロールバックされ、以下のメッセージが送られます。

- ローカル・リソースだけが登録されている場合には、CPF8356。
- リモート・リソースだけが登録されている場合には、CPF835C。
- ローカル・リソースおよびリモート・リソースの両方が登録されている場合には、CPF83E4。

終了されるコミットメント定義に対して通知オブジェクトが定義されていると、そのオブジェクトは更新されることがあります。

最後のエージェントとして登録された API を持つ活動化グループが終了すると、API の出口プログラムが呼び出され、コミットまたはロールバックのいずれかが決定されます。この場合、活動化グループは正常に終了したとしても、ロールバック要求が API 出口プログラムから戻されることがあります。このとき、暗黙コミット操作は実行されません。

コミットメント定義が正常に終了すると、すべての必要なリカバリー (もしあれば) が行われます。それ以外のリカバリーは、終了したばかりのコミットメント定義に関連したコミットメント・リソースに対しては行われません。

コミットメント定義を終了した後でも、活動化グループ内で実行されるプログラム用にジョブ・レベルまたは活動化グループ・レベルのコミットメント定義を再び開始することが可能です。ジョブ・レベルのコミットメント定義を開始できるのは、そのジョブ用にそれがまだ開始されていない場合のみです。

コミットメント定義の開始と終了は、活動化グループ内で稼働するプログラムによって何回でも行うことができますが、開始および終了操作を繰り返す行うために必要なシステム・リソース量は、ジョブ・パフォーマンスと全体的なシステム・パフォーマンスを低下させることがあります。このため、後で呼び出されるプログラムがコミットメント定義を使用する予定の場合は、そのコミットメント定義をアクティブのままにしておくことをお勧めします。

#### 関連概念:

67 ページの『通知オブジェクトの更新』

システムは、そのコミットメント定義に対して正常に行われた最後のコミット操作のコミット ID を使用して、通知オブジェクトを更新します。

#### 関連資料:

コミットメント制御の終了 (ENDCMTCTL) コマンド

---

## システム起動によるコミットメント制御の終了

システムでは、コミットメント制御を終了させるか、あるいは暗黙的なコミット操作またはロールバック操作を実行することができます。コミットメント制御をシステム起動により終了させることが正常な場合もあります。他の場合、コミットメント制御はシステム異常終了またはジョブ異常終了で終了します。

## 活動化グループの終了時のコミットメント制御

活動化グループが終了すると、システムは活動化グループ・レベル・コミットメント定義を自動的に終了します。

活動化グループ・レベルのコミットメント定義に保留中の変更が存在し、そしてその活動化グループが正常終了中の場合、システムはその終了前に、コミットメント定義に対して暗黙のコミット操作を行います。活動化グループが異常終了している場合か、またはコミットメント制御下でオープンされた、その活動化グループまでの範囲のファイルをクローズするときにシステムがエラーを検出した場合には、終了前にその活動化グループ・レベルのコミットメント定義に対して暗黙のロールバック操作が行われます。

**注:** \*JOB または \*DFACTGRP コミットメント定義に対しては、活動化グループの終了処理の時に暗黙のコミットまたはロールバック操作が行われることは決してありません。それは、\*JOB および \*DFACTGRP コミットメント定義は、活動化グループが終了したために終了することはないからです。これらのコミットメント定義は、ENDCMCTCL コマンドを使って明示的に終了されるか、またはジョブの終了時にシステムによって終了されます。

活動化グループが終了すると、システムは、その活動化グループまでの範囲のすべてのファイルを自動的にクローズします。これには、コミットメント制御の下でオープンされた、その活動化グループまでの範囲のすべてのデータベース・ファイルも含まれます。このようなファイルがクローズされるのは、その活動化グループ・レベルのコミットメント定義に対して暗黙でコミット操作が行われる前です。したがって、入出力バッファに常駐するすべてのレコードがまずデータベースに強制的に書き込まれてから、暗黙のコミット操作が実行されます。

実行される暗黙のコミットまたはロールバック操作の一環として、活動化グループ・レベルのコミットメント定義に関連した各 API コミットメント・リソースごとに API コミットおよびロールバック出口プログラムが呼び出されます。出口プログラムは 5 分以内にその処理を完了する必要があります。API コミットおよびロールバック出口プログラムが呼び出されると、システムは API コミットメント・リソースを自動的に除去します。

活動化グループが終了したために終了するコミットメント定義に対して暗黙ロールバック操作が実行された場合、通知オブジェクトがコミットメント定義に定義されていた場合には、その通知オブジェクトが更新されることがあります。

### 関連概念:

67 ページの『通知オブジェクトの更新』

システムは、そのコミットメント定義に対して正常に行われた最後のコミット操作のコミット ID を使用して、通知オブジェクトを更新します。

## 暗黙コミットおよびロールバック操作

場合によっては、システムがコミットメント定義用にコミット操作またはロールバック操作を開始することがあります。このようなタイプのコミット操作およびロールバック操作は、暗黙的なコミットおよびロールバック要求 と呼ばれます。

通常、コミット操作またはロールバック操作は、コミットメント制御をサポートする使用可能なプログラム言語の 1 つを使ってアプリケーション・プログラムから開始されます。このようなタイプのコミットおよびロールバック操作は、明示的なコミットおよびロールバック要求 と呼ばれます。

次の 2 つの表には、保留中の変更を持つコミットメント定義に関連した特定の事象が生じたときに、システムが行う処置が示されています。以下の状態のいずれかが当てはまる場合、コミットメント定義には保留中の変更があります。

- コミット可能なリソースが更新された。
- ファイルの読み取りを行ったためにファイル位置が変更されたので、コミットメント制御下でオープンされたデータベース・ファイルが読み取られた。
- コミットメント定義には API リソースがあり、API リソースに対してユーザー・プログラムによって変更が行われたので、システムはすべての API リソースに保留中の変更があると想定した。

C CM (コミット操作) ジャーナル項目および C RB (ロールバック操作) ジャーナル項目は、操作が明示的であるか、暗黙であるかを示します。

次の表は、ジョブの終了 (正常または異常終了) 時に、以下の状態に基づいてシステムが取る処置を示しています。

- トランザクションの状態。
- コミットメント定義のジョブ終了時の処置の値
- API リソースが最後のエージェントであるか。

状態	最後のエージェント API	ジョブ終了の場合の処置 <sup>1</sup> オプション	コミットまたはロールバック操作
RST	適用外	適用外	<p>コミットメント定義が X/Open グローバル・トランザクションに関連付けられていない場合は、暗黙のロールバックが実行されます。</p> <p>コミットメント定義が X/Open グローバル・トランザクションに関連付けられている場合は、次の事象が起こります。</p> <ul style="list-style-type: none"> <li>• トランザクション・ブランチの状態がアクティブ (S1) でない場合は、処置は実行されず、トランザクション・ブランチは同じ状態のままになります。</li> <li>• トランザクション・ブランチの状態がアクティブ (S1) である場合は、暗黙のロールバックが実行されます。</li> </ul>
PIP	適用外	適用外	<p>コミットメント定義が X/Open グローバル・トランザクションに関連付けられていない場合は、暗黙のロールバックが実行されます。</p> <p>コミットメント定義が X/Open グローバル・トランザクションに関連付けられている場合は、トランザクション・ブランチはアイドル (S2) 状態になり、アイドル (S2) 状態のまま残されます。</p>

状態	最後のエージェント API	ジョブ終了の場合の処理 <sup>1</sup> オプション	コミットまたはロールバック操作
PRP	適用外	WAIT	<p>コミットメント定義が X/Open<sup>2</sup> グローバル・トランザクションに関連付けられていない場合は、次のことが起こります。</p> <ul style="list-style-type: none"> <li>再同期化が開始され、コミット操作のイニシエーターによる決定が受け取られます。</li> <li>コミットするか、ロールバックするかの戻された判断が実行されます。これは明示操作と見なされます。</li> </ul>
PRP	適用外	C	<p>コミットメント定義が X/Open<sup>2</sup> グローバル・トランザクションに関連付けられていない場合は、暗黙のコミット操作が実行されません。</p>
		R	<p>コミットメント定義が X/Open グローバル・トランザクションに関連付けられていない場合は、暗黙のロールバック操作が実行されます。</p> <p>コミットメント定義が X/Open グローバル・トランザクションに関連付けられている場合は、次のことが起こります。</p> <ul style="list-style-type: none"> <li>トランザクションを開始したジョブが終了する場合、トランザクションは、XA TM によってコミットまたはロールバックされるまで準備済み状態のままになります。このケースでは、XA トランザクション・ブランチの状態は準備済み (S3) のままになります。</li> <li>トランザクションの作業が送られている SQL サーバー・ジョブが終了される場合は、強制ロールバックが暗黙的に実行されます。このケースでは、XA トランザクション・ブランチの状態はヒューリスティック完了 (S5) に変わります。</li> </ul>
CIP	適用外	適用外	明示コミット操作が実行されます。
LAP	いいえ	WAIT	<p>1. 最後のエージェントに対する再同期を使用して、コミットするかロールバックするかの判断が取り出されます。</p>
			<p>2. コミットするかロールバックするかの戻された判断が実行されます。これは明示操作と見なされます。</p>
LAP	はい	WAIT	<p>1. コミットするかロールバックするかの判断が取り出すために、最後のエージェント API が呼び出されます。</p>
			<p>2. コミットまたはロールバック操作が実行されます。これは明示操作と見なされます。</p>



状態	最後のエージェント API	ジョブ終了の場合の処置 <sup>1</sup> オプション	コミットまたはロールバック操作
LAP	適用外	C	暗黙コミット操作が実行されます。
		R	暗黙ロールバック操作が実行されます。
CMT	適用外	適用外	このコミットメント定義および下位 (ダウンストリーム) ロケーションに対するコミット操作は既に完了しています。コミット操作は完了しました。
VRO	適用外	適用外	ローカルおよびリモート・エージェントは、読み取り専用ポートしました。すべての下位 (ダウンストリーム) エージェントも読み取り専用ポートしていなければなりません。処置は必要ありません。
RBR	適用外	適用外	ロールバック操作が必要です。明示ロールバック操作が実行されます。

**注:**

<sup>1</sup> コミットメント・オプションの変更 (QTNCHGCO) API を使用して、ジョブ終了の場合の処置オプションを変更することができます。

<sup>2</sup> コミットメント定義が X/Open グローバル・トランザクションに関連付けられている場合は、次の事象が起こりません。

- トランザクションを開始したジョブが終了する場合、トランザクションは、XA TM によってコミットまたはロールバックされるまで準備済み状態のままになります。このケースでは、XA トランザクション・ブランチの状態は準備済み (S3) のままになります。
- トランザクション範囲ロックの場合のみ、トランザクションの作業が送られている SQL サーバー・ジョブが終了される場合は、強制ロールバックが暗黙的に実行されます。このケースでは、XA トランザクション・ブランチの状態はヒューリスティック完了 (S5) に変わります。

次の表は、活動化グループの終了時にシステムが取る処置を示し、ジョブ範囲ロックのトランザクションにのみ適用されます。システム処置は、以下の事項に基づいています。

- トランザクションの状態。(活動化グループの終了時には、常にリセット (RST) になります。)
- 活動化グループの終了方法 - 正常終了または異常終了か。
- API リソースが最後のエージェントであるか。

**注:** API リソースが最後のエージェントとして登録された場合、コミットまたはロールバックの決定は、最後のエージェントによって制御されます。その決定は、明示的操作と見なされます。

状態	最後のエージェント API	終了のタイプ	コミットまたはロールバック操作
RST	いいえ	通常	暗黙コミット操作が実行されます。保護会話が存在する場合、コミットメント定義がコミット操作のルート・イニシエーターになります。
RST	いいえ	異常	暗黙ロールバックが実行された。
RST	はい	通常	API 出口プログラムが呼び出された。コミット操作かロールバック操作のいずれかが API により決定されます。



状態	最後のエージェント API	終了のタイプ	コミットまたはロールバック操作
RST	はい	異常	API 出口プログラムが呼び出された。コミット操作かロールバック操作のいずれかが API により決定されます。

#### 関連概念:

『システムまたはジョブの異常終了時のコミットメント制御』

ジョブが異常終了すると、システムはジョブのすべてのコミットメント定義を終了します。この場合のコミットメント定義は、ジョブ終了処理時に終了されます。このトピックは、ジョブ範囲ロックのコミットメント定義にのみ適用されます。

#### 関連資料:

コミットメント・オプションの変更 (QTNCHGCO) API の使用

## 経路指定ステップの正常終了時のコミットメント制御

経路指定ステップが正常に終了すると、システムはジョブのすべてのコミットメント定義を終了します。

注: 以下の情報は、ジョブ範囲ロックのコミットメント定義にのみ適用されます。

経路指定ステップは、以下の状態の 1 つにより、正常に終了されます。

- バッチ・ジョブの正常終了
- 対話式ジョブの正常サインオフ
- ジョブ経路再指定 (RRTJOB)、ジョブ転送、(TFRJOB)、またはバッチ・ジョブ転送、(TFRBCHJOB) コマンドは、現行経路指定ステップを終了して新しい経路指定ステップを開始します。

上記以外の経路指定ステップの終了は異常と見なされ、ジョブ・ログ内のジョブ完了メッセージ CPF1164 の非ゼロ完了コードで認知されます。

コミットメント定義に保留中の変更がある場合は、経路指定ステップの終了時にコミットメント定義を終了するより前にシステムは暗黙でロールバック操作を行います。これには、そのコミットメント定義と関連した各 API コミットメント・リソースごとに、API コミットおよびロールバック出口プログラムを呼び出すことが含まれます。出口プログラムは 5 分以内にその処理を完了する必要があります。API コミットおよびロールバック出口プログラムが呼び出されると、システムは API コミットメント・リソースを自動的に除去します。

そのコミットメント定義に通知オブジェクトが定義されている場合は、その通知オブジェクトは更新されることがあります。

#### 関連概念:

67 ページの『通知オブジェクトの更新』

システムは、そのコミットメント定義に対して正常に行われた最後のコミット操作のコミット ID を使用して、通知オブジェクトを更新します。

## システムまたはジョブの異常終了時のコミットメント制御

ジョブが異常終了すると、システムはジョブのすべてのコミットメント定義を終了します。この場合のコミットメント定義は、ジョブ終了処理時に終了されます。このトピックは、ジョブ範囲ロックのコミットメント定義にのみ適用されます。

システムが異常終了した場合、システムは、システムの異常終了時にアクティブであったすべてのジョブによって開始され使用されていたすべてのコミットメント定義を終了します。この場合のコミットメント定義は、システムの異常終了後の次の IPL 時に実行されるデータベース・リカバリー処理の一環として終了されます。

**重要:** コミットメント定義のリカバリーは、電源障害、ハードウェア障害、またはオペレーティング・システムやライセンス内部コードでの障害に起因するシステムまたはジョブの異常終了に関連します。ジョブの異常終了を強制するためにジョブ異常終了 (ENDJOBABN) コマンドを使用しないでください。異常終了によって、終了するジョブのアクティブなトランザクションについて部分的にコミットまたはロールバックされる保留中の変更が発生することがあります。ENDJOBABN コマンドを使って終了したジョブの部分的なトランザクションがある場合、次の IPL で、リカバリーを試みることがあります。

ENDJOBABN コマンドを使用して終了させたジョブについてシステムが IPL 時に実行するコミットメント制御リカバリーの結果は不確実です。ジョブが異常終了するとコミットメント・リソースのすべてのロックが解除されるためです。部分的トランザクションに起因する保留中の変更は、他のジョブで使用できるようになります。このような保留中の変更があると、他のアプリケーション・プログラムがデータベースに対して誤った変更をさらに行うことがあります。同様に、その後で引き続き IPL リカバリーを行うと、ジョブが異常終了した後にアプリケーションによって行われた変更が悪影響が出る可能性があります。例えば、未処理作成テーブルのロールバック処置としての IPL リカバリーの時に SQL テーブルがドロップすることがあります。ただし、ジョブが異常終了した後で、他のアプリケーションがそのテーブルに既に数行挿入していたということもあります。

システムは、ジョブの異常終了時またはシステムの異常終了後の次の IPL 時に終了されるコミットメント定義に対して以下のように実行します。

- コミットメント定義の処理がコミット操作の途中で中断されていない限り、コミットメント定義に保留中の変更がある場合には、システムはコミットメント定義を終了する前に暗黙のロールバック操作を行います。コミット操作の途中で終了した場合、トランザクションはその状態に応じて、ロールバック、再同期化、またはコミットされます。暗黙でロールバック操作を行ったり、コミット操作を完了するための処理には、そのコミットメント定義と関連した各 API コミットメント・リソースごとに、API コミットおよびロールバック出口プログラムを呼び出すことが含まれます。API コミットおよびロールバック出口プログラムが呼び出されると、システムは API コミットメント・リソースを自動的に除去します。

**重要:** トランザクションが疑わしいとき (トランザクションの状態が LAP または PRP) にジョブを終了すると、データベースに矛盾をきたします (変更は 1 つ以上のシステムでコミットされ、他のシステムでロールバックされます)。

- ジョブ終了の場合の処置 コミットメント・オプションが COMMIT の場合、ジョブが終了するとこのシステムの変更はコミットされます。これは、トランザクションに参加している他のシステム上での変更がコミットされるかロールバックされるかどうかにかかわらず行われます。
- ジョブ終了の場合の処置 コミットメント・オプションが ROLLBACK の場合、ジョブが終了するとこのシステムの変更はロールバックされます。これは、トランザクションに参加している他のシステム上での変更がコミットされるかロールバックされるかどうかにかかわらず行われます。
- ジョブ終了の場合の処置 コミットメント・オプションが WAIT の場合、コミットまたはロールバック決定権を所有しているシステムで再同期化が完了するまで、ジョブは終了しません。再同期化が完了する前にジョブを終了させるには、ヒューリスティック決定され、再同期化は取り消されなければなりません。

長時間を要するロールバックの間は、ジョブやシステムを異常終了させないことをお勧めします。異常終了させると、初めのジョブが終了したときに (またはシステムが終了した場合は、次の IPL 中に)、別のロールバックが生じる結果になります。この別のロールバックは元のロールバックで実行される作業を繰り返すため、実行にははるかに長い時間がかかります。

- そのコミットメント定義に通知オブジェクトが定義されている場合は、その通知オブジェクトは更新されることがあります。

コミットメント制御が終了する前に処理が終了し、さらに保護会話が依然としてアクティブである場合、コミットまたはロールバックするためにコミットメント定義が必要となることがあります。行われる処置は、そのコミットメント定義の「状態」オプションおよびジョブ終了の場合の「処置」オプションによって決まります。

#### 関連概念:

61 ページの『暗黙コミットおよびロールバック操作』

場合によっては、システムがコミットメント定義用にコミット操作またはロールバック操作を開始することがあります。このようなタイプのコミット操作およびロールバック操作は、暗黙的なコミットおよびロールバック要求 と呼ばれます。

『通知オブジェクトの更新』

システムは、そのコミットメント定義に対して正常に行われた最後のコミット操作のコミット ID を使用して、通知オブジェクトを更新します。

## 通知オブジェクトの更新

システムは、そのコミットメント定義に対して正常に行われた最後のコミット操作のコミット ID を使用して、通知オブジェクトを更新します。

通知オブジェクトでは、以下は非コミット変更と見なされます。

- コミットメント制御でレコードになされる更新。
- コミットメント制御下で削除されるレコード。
- コミットメント制御下でローカル DDL オブジェクトになされるオブジェクト・レベル変更。
- コミットメント制御下でオープンされたデータベース・ファイルのために実行された読み取り操作。それは、ロールバック操作が行われるときに、ファイル位置は直前のコミットメント境界まで戻されるからです。コミットメント制御下で読み取り操作を実行する場合、ファイル位置は変更され、そのためコミットメント定義の未コミット変更が存在するようになります。

- 次のリソースのいずれかが加えられるコミットメント定義は、常に未コミット変更を持つと考えられる。
  - API コミットメント・リソース
  - リモート 分散リレーショナル・データベース体系 (DRDA \*) リソース
  - 分散データベース管理体系 (DDM) リソース
  - LU 6.2 リソース

これは、オブジェクトまたはこれらのタイプのリソースに関連するオブジェクトに変更が実際に加えられるのはいつかをシステムは知らないためです。コミット可能リソースのタイプには、これらのタイプのリソースを追加して処理する方法についての詳細な説明があります。

通知オブジェクトの変更は、コミットメント定義が終了できる次の方法に基づいてシステムが行います。

- ジョブが正常に終了して未コミットが存在しない場合、システムは正常に行われた最後のコミット操作のコミット ID を通知オブジェクトに入れることはありません。
- 活動化グループが終了するときに活動化グループ・レベル・コミットメント定義の暗黙のコミット操作が実行される場合、システムが正常に行われた最後のコミット操作を通知オブジェクトに入れることはありません。

注: \*DFACTGRP または \*JOB コミットメント定義に対しては、暗黙コミット操作は決して実行されません。

- システム、ジョブ、または活動化グループが、コミットメント定義の最初の正常なコミット操作の前に異常終了した場合、最後のコミット ID はないため、システムが通知オブジェクトを更新することはありません。この条件と通常のプログラム完了とを区別するためには、ユーザーのプログラムはコミットメント定義の最初の正常なコミット操作を完了する前に特定の項目を持つ通知オブジェクトを更新しなければなりません。
- 少なくとも 1 つの正常なコミット操作の後に、ジョブの異常終了またはシステムの異常終了が生じた場合、システムはそのコミット操作のコミット ID を通知オブジェクトに入れます。正常に行われた最後のコミット操作でコミット ID を指定しなかった場合には、通知オブジェクトは更新されません。ジョブの異常終了の場合、この通知オブジェクト処理は、そのジョブに対して活動化されていた各コミットメント定義ごとに実行されます。システムの異常終了の場合、この通知オブジェクト処理は、システムのすべてのジョブに対して活動化されていた各コミットメント定義ごとに実行されます。
- 以下のすべての事象が発生した場合、システムは、そのコミットメント定義に対して、正常に行われた最後のコミット操作のコミット ID を使って通知オブジェクトを更新します。
  - 非デフォルトの活動化グループが終了した。
  - 活動化グループ・レベルのコミットメント定義に対して暗黙でロールバック操作が実行された。
  - そのコミットメント定義に対して実行されたコミット操作が少なくとも 1 回正常に行われた。

正常に行われた最後のコミット操作でコミット ID を指定しなかった場合には、通知オブジェクトは更新されません。活動化グループが異常終了するか、活動化グループに範囲を限定されたコミットメント制御下でオープンされたファイルをクローズするときにエラーが生じる場合、活動化グループ・レベルのコミットメント定義の暗黙のロールバック操作が実行されます。データベース・ファイルの有効範囲を活動化グループに限定すること、および活動化グループを終了できる方法についての詳細は、ご使用になっている ILE 言語の解説書を参照してください。

- ジョブが正常に終了したときに未コミット変更が存在して少なくとも 1 つの正常なコミット操作が実行された場合、正常に行われた最後のコミット操作のコミット ID は通知オブジェクトに入れられて未コミット変更はロールバックされます。正常に行われた最後のコミット操作でコミット ID を指定しな



った場合には、通知オブジェクトは更新されません。この通知オブジェクトの処理は、ジョブが終了するときにそのジョブに対して活動化されていた各コミットメント定義ごとに実行されます。

- ENDCMTCTL コマンドの実行時に未コミットの変更が存在すると、正常に行われた最後のコミット操作がコミット ID を指定していた場合にのみ通知オブジェクトが更新されます。
  - バッチ・ジョブの場合、未コミット変更はロールバックされ、正常に行われた最後のコミット操作のコミット ID が通知オブジェクトに入れられます。
  - 対話式ジョブでは、照会メッセージ CPA8350 に対する応答が変更をロールバックを指定していれば、未コミット変更はロールバックされて正常に行われた最後のコミット操作のコミット ID は通知オブジェクトに入れられます。
  - 対話式ジョブの場合、照会メッセージ CPA8350 に対する応答が変更のコミットを指定していれば、使用するコミット ID を入力するようシステムがプロンプトを出し、変更内容はコミットされます。プロンプト画面で入力するコミット ID が通知オブジェクトに入れられます。
  - 対話式ジョブでは、照会メッセージ CPA8350 に対する応答が ENDCMTCTL 要求を取り消す場合、変更保留はそのまま通知オブジェクトは更新されません。

#### 関連概念:

59 ページの『コミットメント制御の終了』

ジョブ・レベルまたは活動化グループ・レベルのコミットメント定義のいずれのコミットメント制御も、コミットメント制御の終了 (ENDCMTCTL) コマンドを使用して終了させることができます。

61 ページの『活動化グループの終了時のコミットメント制御』

活動化グループが終了すると、システムは活動化グループ・レベル・コミットメント定義を自動的に終了します。

65 ページの『経路指定ステップの正常終了時のコミットメント制御』

経路指定ステップが正常に終了すると、システムはジョブのすべてのコミットメント定義を終了します。

65 ページの『システムまたはジョブの異常終了時のコミットメント制御』

ジョブが異常終了すると、システムはジョブのすべてのコミットメント定義を終了します。この場合のコミットメント定義は、ジョブ終了処理時に終了されます。このトピックは、ジョブ範囲ロックのコミットメント定義にのみ適用されます。

13 ページの『コミット可能リソースのタイプ』

この表には、FILE、データ定義言語 (DDL)、分散データ管理 (DDM)、論理装置 (LU) 6.2、Distributed Relational Database Architecture (DRDA)、API、TCP などのさまざまなタイプのコミット可能リソースをリストしています。

55 ページの『通知オブジェクトのコミット』

通知オブジェクトはメッセージ・キュー、データ域、またはデータベース・ファイルで、そのコミットメント定義が正常に終了しなかった場合、特定のコミットメント定義の最後に正常に完了したトランザクションを識別する情報が含まれています。

## 異常終了後の初期プログラム・ロード中のコミットメント制御リカバリー

システムの異常終了後、初期プログラム・ロード (IPL) を実行すると、システムは、システムの終了時にアクティブであったすべてのコミットメント定義をリカバリーしようとしています。

同様に、独立ディスク・プールをオンに変更すると、システムは、オフに変更されたかまたは異常終了した時にアクティブであった独立ディスク・プールに関係付けられたすべてのコミットメント定義をリカバリーしようとしています。

リカバリーは、IPL 時にシステムが開始したデータベース・サーバー・ジョブにより実行されます。データベース・サーバー・ジョブは、他のジョブでは実行することのできない作業を処理するために、システムにより開始されます。

データベース・サーバー・ジョブの名前は QDBSRVnn です。ここで、nn は 2 桁の数字です。データベース・サーバー・ジョブの番号は、システムのサイズによって異なります。同様に、独立ディスク・プールまたは独立ディスク・プール・グループのデータベース・サーバー・ジョブの名前は QDBSxxxVnn です。ここで、xxx は独立ディスク・プール番号で、nn は 2 桁の数字です。例えば、独立ディスク・プール 35 用のデータベース・サーバー・ジョブの名前として、QDBS035V02 が可能です。

2 フェーズ・コミットメント制御のトランザクションの状態では、障害発生時のトランザクションの状態に応じてシステムが取る処置が示されています。2 つの状態、PRP および LAP の場合は、システム処置は確定していません。

**注:**

- 以下は、ジョブ範囲ロックのコミットメント定義にのみ適用されます。
- トランザクション・マネージャーは、XA トランザクション (ロックがジョブ範囲かトランザクション範囲かにかかわらず) に関連付けられたコミットメント定義を、このトピックで記述されている再同期処理ではなく XA API を使用してリカバリーします。

システムは、トランザクションに参加している他のロケーションとの再同期化を実行するまで、処置を判別することができません。この再同期化は、IPL またはオンへの変更操作が完了後に実行されます。

システムは、データベース・サーバー・ジョブを使ってこの再同期化を実行します。リカバリーの必要なコミットメント定義は、データベース・サーバー・ジョブと関連しています。IPL 時にシステムは、システムの終了前にコミットメント定義が保持していたすべてのレコード・ロックおよび他のオブジェクト・ロックを獲得します。これらのロックは、再同期化が完了し、リソースがコミットまたはロールバックされるまで、ローカル・コミットメント・リソースを保護するのに必要です。

リモート・ロケーションとの再同期化の状況を示すメッセージが、データベース・サーバー・ジョブのジョブ・ログに送られます。トランザクションが予測できない場合、ローカル・リソースがコミットまたはロールバックされる前にトランザクションに対する決定を行うロケーションとの再同期化を完了する必要があります。

トランザクションに対する決定が行われると、次のメッセージがデータベース・サーバー・ジョブのジョブ・ログに送られます。

**CPI8351**

保留中の &1 個の変更がロールバック中である。

**CPC8355**

ジョブ &19/&18/&17 についてコミットメント定義 &8 の IPL 後リカバリーが完了した。

**CPD835F**

ジョブ &19/&18/&17 のコミットメント定義 &8 の IPL リカバリーが正常に実行されなかった。

リカバリーに関連した他のメッセージも送られます。これらのメッセージは、ヒストリー (QHST) ログに送られます。エラーが起きると、メッセージが QSYSOPR メッセージ・キューにも送られます。

IBM Navigator for i を使用するか、データベース・サーバー・ジョブのジョブ・ログを表示するか、またはコミットメント定義の処理 (WRKCMDFN) コマンドを使用すると、リカバリーの進行状況を判別することができます。IBM Navigator for i および「コミットメント定義の処理」画面を使用すると、システム



にコミットまたはロールバックを強制実行させることができますが、これは最後の手段として使用してください。トランザクションに参加しているすべてのロケーションが操作可能状態になることが予測される場合、システムの再同期化を行う必要があります。これにより、データベースの健全性が保持されます。

#### 関連概念:

23 ページの『コミットメント定義の独立ディスク・プールの考慮事項』

独立ディスク・プールを使用する場合は、コミットメント定義についての考慮事項に注意する必要があります。

32 ページの『2 フェーズ・コミットメント制御のトランザクションの状態』

コミットメント定義が、トランザクション・プログラム・ネットワークの一部である各ロケーションに確立されます。各コミットメント定義に対し、システムは、現行のトランザクションと前のトランザクションの状態を常に監視しています。

---

## トランザクションおよびコミットメント制御の管理

以下の指示に従って、コミットメント制御情報を表示し、コミットメント制御のパフォーマンスを最適化することができます。

### コミットメント制御情報の表示

IBM Navigator for i は、システム上のすべてのトランザクション (作業論理単位) に関する情報を表示することができます。IBM Navigator for i は、トランザクションに関連付けられたジョブの情報がある場合は、それも表示することができます。

注: 以下の表示操作では、SQL アプリケーションの分離レベルを表示しません。

情報を表示するには、次のように操作を進めます。

1. IBM Navigator for i で、使用するシステムを展開します。
2. 「データベース」を展開します。
3. 処理対象のシステムを展開します。
4. 「トランザクション」を展開します。

注: X/Open グローバル・トランザクションに関連付けられているトランザクションを表示するには、「グローバル・トランザクション」を展開します。DB2 管理のトランザクションを表示するには、「データベース・トランザクション」を展開します。

5. 「グローバル・トランザクション」または「データベース・トランザクション」を展開します。

これにより、以下の情報が表示されます。

- 作業単位 ID
- 作業単位状態
- ジョブ
- ユーザー
- 番号
- 進行中の再同期
- コミットメント定義

すべての状況画面および各画面のフィールドに関する情報が、オンライン・ヘルプにあります。

#### 関連タスク:

### 120 ページの『デッドロックの検出』

あるジョブがあるオブジェクト (オブジェクト A) のロックを保持しながら他のオブジェクト (オブジェクト B) のロックの獲得を待っており、同時に他のジョブまたはトランザクションがオブジェクト B のロックを保持しながらオブジェクト A のロックの獲得を待っている場合、デッドロック状態が発生することがあります。

### 121 ページの『通信障害後のトランザクションのリカバリー』

この手順を使用すると、リモート・システムとの通信の障害発生後に、そのシステム上で処理を実行するトランザクションを取り扱う際に役立ちます。

### 122 ページの『コミットとロールバック操作の強制時点および再同期の取り消し時点』

コミットまたはロールバック操作の強制の決定は、ヒューリスティック判定 と呼ばれます。この処置により、オペレーターは、準備済み状態にあるトランザクションのリソースを手動でコミットまたはロールバックできるようになります。

### 124 ページの『大規模トランザクションまたは古いトランザクションを検出する』

大規模トランザクションまたは古いトランザクションを検出するには、**コミットメント定義の処理 (WRKCMTDFN)** コマンドを使用します。

## トランザクションのロックされたオブジェクトの表示

トランザクション範囲ロックのみを持つグローバル・トランザクションについて、ロックされたオブジェクトを表示することができます。

トランザクションのロックされたオブジェクトを表示するには、次の手順に従います。

1. IBM Navigator for i で、使用するシステムを展開します。
2. 「データベース」を展開します。
3. 処理対象のシステムを展開します。
4. 「トランザクション」を展開します。
5. 「グローバル・トランザクション」を展開します。
6. 処理対象のトランザクションを右クリックして、「**ロックされたオブジェクト**」を選択します。

#### 関連タスク:

### 120 ページの『デッドロックの検出』

あるジョブがあるオブジェクト (オブジェクト A) のロックを保持しながら他のオブジェクト (オブジェクト B) のロックの獲得を待っており、同時に他のジョブまたはトランザクションがオブジェクト B のロックを保持しながらオブジェクト A のロックの獲得を待っている場合、デッドロック状態が発生することがあります。

## トランザクションに関連付けられたジョブの表示

トランザクションに関連付けられたジョブを表示するには、次の手順に従います。

1. 「IBM Navigator for i」ウィンドウで、使用するシステムを展開します。
2. 「データベース」を展開します。
3. 処理対象のシステムを展開します。
4. 「トランザクション」を展開します。
5. 「グローバル・トランザクション」または「データベース・トランザクション」を展開します。
6. 処理対象のトランザクションを右クリックして、「**ジョブ**」を選択します。

ジョブ範囲ロックのデータベース・トランザクションおよびグローバル・トランザクションの場合は、トランザクションに関連付けられたジョブのリストが表示されます。

トランザクション範囲ロックのグローバル・トランザクションの場合は、このトランザクション・オブジェクトに接続されたジョブ、またはこのトランザクション・オブジェクトに接続されるのを待っているジョブのリストが表示されます。

#### 関連タスク:

120 ページの『デッドロックの検出』

あるジョブがあるオブジェクト (オブジェクト A) のロックを保持しながら他のオブジェクト (オブジェクト B) のロックの獲得を待っており、同時に他のジョブまたはトランザクションがオブジェクト B のロックを保持しながらオブジェクト A のロックの獲得を待っている場合、デッドロック状態が発生することがあります。

### トランザクションのリソース状況の表示

トランザクションのリソース状況を表示するには、次の手順に従います。

1. 「IBM Navigator for i」ウィンドウで、使用するシステムを展開します。
2. 「データベース」を展開します。
3. 処理対象のシステムを展開します。
4. 「トランザクション」を展開します。
5. 「グローバル・トランザクション」または「データベース・トランザクション」を展開します。
6. 処理対象のトランザクションを右クリックして、「リソース状況」を選択します。

### トランザクションのプロパティの表示

トランザクションのプロパティを表示するには、次の手順に従います。

1. 「IBM Navigator for i」ウィンドウで、使用するシステムを展開します。
2. 「データベース」を展開します。
3. 処理対象のシステムを展開します。
4. 「トランザクション」を展開します。
5. 「グローバル・トランザクション」または「データベース・トランザクション」を展開します。
6. 処理対象のトランザクションを右クリックして、「プロパティ」を選択します。

## コミットメント制御のパフォーマンスの最適化

コミットメント制御を使用するには、システム・パフォーマンスに影響する可能性のあるリソースが必要です。次のようないくつかの要因がシステム・パフォーマンスに影響します。

### パフォーマンスに影響しない要因

#### ファイルのオープン

コミット・オープン・オプションを指定せずにファイルをオープンすると、コミットメント定義が開始されていても、さらにシステム・リソースが使用されることはありません。コミット・オープン・オプションの指定に関する詳細は、該当する高水準言語の解説書を参照してください。

### パフォーマンスを低下させる要因

#### ジャーナル

ファイルをジャーナル処理するにはシステム・リソースが必要です。ただし、ほとんどの場合、ジャーナル処理はコミットメント制御がない時よりもコミットメント制御がある方がより良く実行されます。変更後イメージだけを指定しても、コミットメント制御が有効な間はコミットメント制御はこれを変更前イメージと変更後イメージの両方に変更します。通常これはパフォーマンスではなくスペースの考慮事項です。

## コミット操作

トランザクション中にジャーナル・リソースに何らかの変更を加えた場合は、トランザクションをコミットするたびに、それらのリソースに関連した各ジャーナルに 2 つの項目が追加されます。追加される項目の数は、小さなトランザクションが大量にある場合には、著しく増大することがあります。ジャーナル・レシーバーをジャーナルとは別のディスク・プールに入れたほうが良い場合もあります。

## ロールバック操作

コミットメント制御はデータベースに記録された保留中の変更を元に戻さなければならないため、ロールバックが行われる時にはさらにシステム・リソースが必要になります。また、レコード変更が保留中の場合には、ロールバック操作が行われるとさらに別の項目がジャーナルに追加されます。

## コミットメント制御の開始 (STRCMTCTL) およびコミットメント制御の終了 (ENDCMTCTL) コマンド

コミットメント定義が、STRCMTCTL コマンドを使用して開始され、ENDCMTCTL コマンドを使用して終了されるたびに、システムにはさらにオーバーヘッドが発生します。各トランザクションに STRCMTCTL および ENDCMTCTL のコマンドを使用しないようにします。これらは必要なときにだけ使用してください。対話式ジョブの始めにコミットメント定義を確立し、そのジョブの間中これを使用することができます。

## コミットメント制御トランザクションに複数のジャーナルを使用

2 フェーズ・コミットを使うと、コミットメント制御下でオープンされたファイルを複数のジャーナルにジャーナル処理することができます。ところが、数のジャーナルを使用すると、コミットメント定義を管理するためのシステム・リソースがさらに必要になります。複数のジャーナルを使用すると、リカバリーがもっと複雑になる可能性もあります。

## レコードのロック

レコードをロックすると、他のアプリケーションに影響があります。特定のジョブ内のロック化されたレコードの数によって、そのジョブで使用するシステム・リソース全体が増加します。同じレコードにアクセスする必要のあるアプリケーションは、トランザクションの終了を待たなければなりません。

## SEQONLY(\*YES) の要求

OVRDBF コマンドを使用して SEQONLY(\*YES) オプションを要求するか、またはアプリケーション・プログラムが暗黙に SEQONLY(\*YES) の使用を試み、ファイルが LCKLVL(\*ALL) のコミットメント制御下で入力のためだけにオープンされた場合には、オプションは、SEQONLY(\*NO) に変わります。レコードはブロック化されないため、このオプションが入力ファイルのパフォーマンスに影響する場合があります。

## 活動時保管処理または独立 ASP 静止処理がアクティブのときのデータベース・ファイルに対するレコード・レベルの変更の要求

コミットメント定義がコミットメント境界にあり、さらに活動時保管操作または独立 ASP 静止操作が別のジョブで実行されている場合は、コミットメント制御下でデータベース・ファイルに対するレコード・レベルの変更を要求すると遅らされることがあります。活動時保管操作の場合、保管要求の特定のオブジェクトと同じジャーナルにファイルがジャーナル処理される場合に、この遅延が発生することがあります。

注: ジョブが活動時保管チェックポイント処理のために保留されているときには、「活動ジョブの処理 (WRKACTJOB)」コマンド出力の状況欄には、CMTW (コミット待機) が表示されます。

## 活動時保管処理または独立 ASP 静止処理がアクティブのときの変更のコミットまたはロールバック

活動時保管操作または独立 ASP 静止操作が別のジョブで実行されている場合は、コミットまたはロールバック操作がコミットメント境界で遅らされることがあります。このことは、API コミット

メント・リソースがコミットメント定義に以前追加されていた場合に起きることがあります。ただし、以下の条件のいずれかが当てはまる場合は除きます。

- 活動時保管操作の場合に、コミットメント・リソースの追加 (QTNADDCR) API を使用して API リソースが追加され、さらに通常の保管処理許可フィールドに値 Y が指定されている。
- 独立 ASP 静止操作の場合に、QTNADDCR API を使用して API リソースが追加され、さらに独立 ASP 静止許可フィールドに、値 Y が指定されている。

コミットまたはロールバック要求時にはジョブは保持され、かつコミットまたはロールバック要求は一度に 1 つのコミットメント定義に対してだけ実行できるため、API コミットメント・リソースを備えた複数のコミットメント定義を持つジョブの場合、活動時保管操作または独立 ASP 静止操作が完了しないことがあります。

注: 部分的なトランザクション機能のある新規保管を使用した場合、コミットメント定義を終了せずにオブジェクトを保管できます。

#### 活動時保管処理または独立 ASP 静止処理がアクティブのときのオブジェクト・レベル変更の要求

コミットメント定義がコミットメント境界にあり、さらに活動時保管操作または独立 ASP 静止操作が別のジョブで実行されている場合は、コミットメント制御下でオブジェクト・レベルの変更を行う要求は遅らされることがあります。このことは、オブジェクトが入っているライブラリーに対する活動時保管操作または独立 ASP 静止操作の実行中に、オブジェクト・レベルの変更が行われたときに起こる場合があります。例えば、ライブラリー MYSQLLIB に対して活動時保管操作または独立 ASP 静止操作が実行されている場合は、コミットメント制御下のライブラリー MYSQLLIB 内のテーブル MYTBL に対して行われる SQL テーブル作成操作は遅らされることがあります。

注: 待機時間が 60 秒を超えた場合には、操作を待機し続けるか取り消すかをユーザーに尋ねる照会メッセージ CPA8351 が送られます。

#### QTNADDCR API を使用した API リソースの追加

ジョブのすべてのコミットメント定義がコミットメント境界にあり、さらに活動時保管操作または独立 ASP 静止操作が別のジョブで実行されている場合は、QTNADDCR API を使用して API コミットメント・リソースを追加する要求は遅らされることがあります。

注:

1. 待機時間が 60 秒を超えた場合には、操作を待機し続けるか取り消すかをユーザーに尋ねる照会メッセージ CPA8351 が送られます。
2. 活動時保管操作の場合、このことは、通常の保管処理許可フィールドに値 Y が指定されている場合は、QTNADDCR API を使用して追加された API リソースには適用されません。独立 ASP 静止操作の場合、このことは、独立 ASP 静止許可フィールドに値 Y が指定されている場合は、QTNADDCR API を使用して追加された API リソースには適用されません。

## パフォーマンスを向上させる要因

### デフォルトのジャーナルを使用

コミットメント定義がアクティブのときに、コミットメント制御下ですべてのファイルを一度クローズしてから再びオープンした場合、デフォルトのジャーナルを使用するとパフォーマンスが向上します。しかし、OMTJRNE(\*NONE) を指定したデフォルトのジャーナルを使用すると、コミット操作およびロールバック操作のパフォーマンスが低下します。

### 最後のエージェントを選択

最終エージェントが選択されるとパフォーマンスが改善されます。コミット操作中は、システムと最終エージェントとの間にはほとんど対話が必要ないからです。しかし、コミット操作中に通信障



害が生じる場合、結果の待機オプションの値にかかわらず、再同期が完了するまでコミット操作は完了しません。そのような障害はまれですが、このオプションを使うと、障害が生じたときに再同期が完了するのをユーザーがかなりの時間待機することのマイナスの影響を、アプリケーション書き込み機能が考慮することができます。これを、正常なコミット操作中の最終エージェント最適化によってなされるパフォーマンスの改善と比較してください。この考慮事項は通常、バッチ・ジョブの場合よりも対話式ジョブの場合の方が重要です。

デフォルトでは最終エージェントはシステムによる選択を許可されていますが、ユーザーは QTNCHGCO API を使ってこの値を修正することができます。

### 結果の待機オプションを使用しない

リモート・リソースがコミットメント制御下にあり、結果の待機オプションが N (No) に設定されているときに、すべてのリモート・システムが presumed abort をサポートする場合には、パフォーマンスが向上します。DRDA および DDM アプリケーションの場合、結果の待機オプションは、リモート・システムへの最初の接続が確立されるときに、システムによって N に設定されます。APPC アプリケーションでは、結果の待機オプションを明示的に設定しなければ、デフォルトの Y が使用されます。

### 「流用可能」オプションを選択する。

「流用可能」オプションを選択するとパフォーマンスは改善されます。

### 読み取り専用ポート・オプションを選択する。

読み取り専用ポート・オプションを選択するとパフォーマンスは改善されます。

### 関連概念:

#### ジャーナル管理

41 ページの『2 フェーズ・コミットのコミットメント定義：「流用可能」の指示』

通常、トランザクション・マネージャーは、トランザクション・プログラム・ネットワーク内のすべてのロケーションで、すべてのコミットあるいはロールバック操作に参加します。パフォーマンスを向上させるために、トランザクション内の特定またはすべてのロケーションを、トランザクション・マネージャーが「流用可能」を示せるように設定することができます。

35 ページの『2 フェーズ・コミットのコミットメント定義：読み取り専用ポートの許可』

通常、トランザクション・マネージャーはコミット処理の両フェーズに参加します。コミット処理のパフォーマンスを向上させるために、トランザクション・マネージャーが読み取り専用をポートできるように、一部またはすべてのロケーションを設定することができます。

### 関連情報:

コミットメント制御リソースの追加 (QTNADDCR) API

## ロックの最小化

レコード・ロックを最小限に抑える一般的な方法は、レコード・ロックを解除することです。(LCKLVL(\*ALL) を指定している場合には、この手法はうまくいきません。)

以下に、レコード・ロックを解除することによりレコード・ロックを最小限に抑える例を示します。単一ファイルの保守アプリケーションは、一般に、以下の手順に従います。

1. 変更するレコードを識別するためのプロンプトを表示する。
2. 要求されたレコードを検索する。
3. レコードを表示する。
4. ワークステーション・ユーザーが変更を行うのを認める。
5. レコードを更新する。



ほとんどの場合、レコードは、要求されたレコードのアクセスから更新の終わりまでロックされています。レコードを待っている別のジョブで、レコード待機時間を超える場合があります。ワークステーション・ユーザーが変更を考慮している間にレコードをロックしておくのを避けるには、そのレコードをデータベースから取り出した後に（レコード表示が現れる前に）解放します。この場合、更新の前にレコードへ再アクセスする必要があります。レコードが解放されてから再度アクセスされるまでにそのレコードが変更された場合は、ワークステーション・ユーザーに通知しなければなりません。プログラムは、元のレコードの 1 つまたは複数のフィールドを保管し、これを検索後の同じレコードのフィールドと比較して、レコードが変更されたかどうかを次のように判別することができます。

- レコード中の更新カウント・フィールドを使用して、更新直前にそのフィールドに 1 を加えます。プログラムは元の値を保管し、レコードが再度検索されたときにフィールドの値とそれを比較します。変更があった場合には、ワークステーション・ユーザーに通知され、レコードが再度表示されます。更新カウント・フィールドが変更されるのは、更新があった場合だけです。ワークステーション・ユーザーが変更を考慮している間、レコードは解放されます。この手法を使用する場合には、ファイルを更新するすべてのプログラムでこれを使用しなければなりません。
- データ・レコード全体の内容を保管し、次にそのデータ・レコードを検索したときのレコードと比較します。

上記のいずれの場合でも、この一連の操作は、マスター・レコードと表示装置ファイルで同じフィールド名を使用する RPG では、外部記述データを単純に使用する妨げとなります。レコードを再度検索するとワークステーション・ユーザーの変更はオーバーレイされるので、(RPG で) 同じフィールド名を使用することはできません。この問題は、レコード・データをデータ構造に移動することにより解決できます。あるいは、DDS キーワード RTNDTA を使用する場合は、引き続き、外部記述データを使用できます。RTNDTA キーワードを使用することにより、オペレーティング・システムがデータを画面からプログラムへ移動しなくても、プログラムは画面上のデータを再度読み取ることができます。これにより、プログラムは以下のタスクを実行することができます。

1. レコードを識別するためのプロンプトを出す。
2. 要求されたレコードをデータベースから検索する。
3. レコードを解放する。
4. レコードが変更されたかどうかを判別するために使用する 1 つまたは複数のフィールドを保管する。
5. レコードを表示し、ワークステーション・ユーザーの応答を待つ。

ワークステーション・ユーザーが画面上のレコードを変更した場合には、プログラムは次の順序に従います。

1. 再びレコードをデータベースから検索する。
2. 保管されたフィールドを比較して、データベース・レコードが変更されているかどうかを判別する。変更されている場合には、プログラムはレコードを解放し、レコードが表示されるときにメッセージを出します。
3. RTNDTA キーワードを指定した読み取り操作を実行することによって画面からレコードを検索し、データベース・レコードでそのレコードを更新します。
4. ワークステーション・ユーザーが要求を取り消した場合には、解放するレコードはほかにないので、次の論理プロンプトに進みます。

LCKLVL(\*CHG) および LCKLVL(\*CS) をこの状況で適切に使用できます。LCKLVL(\*ALL) を使用した場合には、コミットまたはロールバック操作を使用して、レコード・ロックを解除しなければなりません。

**関連タスク:**

120 ページの『デッドロックの検出』

あるジョブがあるオブジェクト (オブジェクト A) のロックを保持しながら他のオブジェクト (オブジェクト B) のロックの獲得を待っており、同時に他のジョブまたはトランザクションがオブジェクト B のロックを保持しながらオブジェクト A のロックの獲得を待っている場合、デッドロック状態が発生することがあります。

## トランザクション・サイズの管理

レコード・ロックを最小限に抑える方法として、トランザクションのサイズを管理する方法もあります。

この項では、対話式のトランザクションを想定しています。(コミットメント制御はバッチ・アプリケーションにも使用することができます。多くの場合、バッチ・アプリケーションは連続したトランザクションと見なすことができます。)バッチ・アプリケーションには多くの同じ考慮事項が適用されます。これについては、バッチ・アプリケーションのコミットメント制御の項で説明しています。

トランザクションに関連したそれぞれのジャーナルごとに、トランザクション中に最大 500 000 000 のレコードをロックすることができます。この限度は QUERY オプション・ファイル (QAQQINI) を使用することにより削減することができます。QUERY 属性の変更 (CHGQRYA) コマンドの QRYOPTLIB パラメータを使用して、使用するジョブのための QUERY オプション・ファイルを指定してください。QUERY オプション・ファイルの中の COMMITMENT\_CONTROL\_LOCK\_LEVEL 値を、ジョブのロック限度として使用してください。ジャーナル処理されたリソースが初めてコミットメント制御下に置かれるとき、このロックしきい値はそれぞれのコミットメント定義ごとに内部的にキャッシュされます。その時点の後でロック限度が変更される場合、そのコミットメント定義に有効になるようにキャッシュされた値をリフレッシュする必要があります。Retrieve Commit Information (QTNRCMTI) API の呼び出しを実行すると、キャッシュされた値が呼び出しジョブでリフレッシュされます。新しい値は、キャッシュがリフレッシュされる前に開始されたトランザクションには適用されません。

レコードのロック・レベルを選択するときには、トランザクションのサイズを考慮してください。サイズを使用して、トランザクションが終了する前にレコードがロックされる時間の長さを決定します。コミットメント制御のコミットまたはロールバック操作で実行キーの使用を 1 回に制限するか、あるいは 1 つのトランザクションで実行キーを複数回使用しなければならないかを決定しなければなりません。

**注:** トランザクションが短ければ短いほど、活動時保管チェックポイント処理の開始を待つジョブがより早く処理を続行し、完了することができます。

例えば、受注アプリケーションの場合には、顧客の 1 回の受注で複数品目が受注され、それぞれの品目ごとに受注明細レコードと在庫マスター・レコードを更新しなければなりません。トランザクションが 1 つの受注全体として定義され、実行キーを押すたびに 1 品目が入力される場合には、その受注全体の処理の過程で関連するすべてのレコードがロックされます。したがって、使用頻度の高いレコード (例えば在庫マスター・レコード) が長時間ロックされ、他の作業が妨げられることがあります。サブファイルを使用して 1 回実行キーを押すだけで全品目が入力される場合には、1 つの受注全体のロック時間が最小になります。

一般にロックの数と継続時間を最小限にして、他の多くのワークステーション・ユーザーが長時間待つことなく同じデータにアクセスできるようにしなければなりません。これは、ユーザーが画面にデータを入力している間は、ロックを保持しないようにすることによって行うことができます。アプリケーションによっては、複数のワークステーション・ユーザーが同じデータをアクセスする必要がないものもあります。例えば、得意先ごとに多くの未入金項目がある入金帳アプリケーションでの代表的な手法は、ワークステーション・ユーザーが 1 つの入金記録の入金帳を完了するまで、すべてのレコードをロックし、保持することです。

ワークステーション・ユーザーが 1 つのトランザクションで複数回実行キーを押す場合には、トランザクションをいくつかのセグメントに分けて実行することができます。以下にその例を示します。

- 最初のセグメントは、ワークステーション・ユーザーが情報を要求する照会です。
- 2 番目のセグメントは、ワークステーション・ユーザーが 1 つのトランザクション全体を完了するための確認です。
- 3 番目のセグメントは、関連するレコードの検索と更新です。

この方法では、レコード・ロックを 1 回の実行キーの使用に限定することができます。

この照会を主にした手法は、通常、表示される情報から決定が下されるアプリケーションで使用されます。例えば、航空機の座席予約アプリケーションでは、顧客は飛行時間、利用できる接続便、座席の配列などを知ってから、利用する便を決定するのが普通です。顧客が決定を下すと、トランザクションが入力されます。トランザクションが失敗した (希望する便が現在満席である) 場合には、ロールバック機能を使用して別の要求を入力することができます。最初の照会から決定が下されるまでレコードがロックされる場合には、別の予約担当者は、他のトランザクションが完了するまで待つことになります。

#### 関連概念:

27 ページの『バッチ・アプリケーションのコミットメント制御』

バッチ・アプリケーションは、コミットメント制御を必要とする場合とそうでない場合があります。場合によっては、バッチ・アプリケーションは、入力ファイルの読み取りとマスター・ファイルの更新の単一機能しか実行できないことがあります。ただし、異常終了の後に再開することが重要な場合には、この種のアプリケーションに対してコミットメント制御を使用することができます。

## ソフト・コミット

ソフト・コミット は、システムが、トランザクションに関連したジャーナル項目をディスクに書き込む回数を制限するコミットメント制御の形式です。

ソフト・コミットにより、トランザクションのパフォーマンスは向上しますが、システム障害の場合に 1 つ以上のトランザクションが失われる可能性があります。DB2 for i 上の従来のコミットメント制御により、トランザクションの持続性が確保されます。つまり、トランザクションがコミットされている間、トランザクションはシステム上に存続します。ソフト・コミットにより、トランザクションの原子性はまだ確保されますが、この持続性は提供されません。換言すれば、システムはコミット境界を保証しますが、システム障害が発生した場合、1 つ以上のトランザクションが丸ごと失われる可能性があります。

特定のジョブに対して、あるいはシステム全体に対して、ソフト・コミットを使用するには、**QIBM\_TN\_COMMIT\_DURABLE** 環境変数で **\*NO** を指定してください。この変数は、**Add Environment Variable (ADDENVVAR)** コマンドで変更できます。

例えば、特定のジョブからソフト・コミットを要求するには、そのジョブから、次のコマンドを実行します。

```
ADDENVVAR ENVVAR (QIBM_TN_COMMIT_DURABLE) VALUE (*NO)
```

システム全体に対してソフト・コミットを要求するには、次のコマンドを実行します。

```
ADDENVVAR ENVVAR (QIBM_TN_COMMIT_DURABLE) VALUE (*NO) LEVEL (*SYS)
```

注: この環境変数をシステム規模で設定するためには、**\*JOBCTL** 権限をもっている必要があります。

`QIBM_TN_COMMIT_DURABLE` 環境変数が追加されていない場合、または環境変数が `*NO` 以外の値に設定されている場合、システムはソフト・コミットを使用しません。トランザクションの持続性が確保されるように、代わりに、従来のコミットメント制御を使用します。

**Work with Environment Variables (WRKENVVAR)** コマンドを使用すると、この新規環境変数が存在するかどうか、また、存在する場合はその値とレベルを検査することができます。

この環境変数値は、コミットメント制御が開始されるたびに、内部的にキャッシュされます。その時点の後に環境変数が変更される場合、有効になるようにキャッシュされた値をリフレッシュする必要があります。Retrieve Commit Information (QTNRCMTI) API の呼び出しを実行すると、キャッシュされた値が呼び出しジョブでリフレッシュされます。

一部のトランザクションについては、オペレーティング・システムはソフト・コミットについての要求を無視するよう選択し、その代わりに、従来のコミットメントを実行します。これは、複数のデータベース接続が必要であったり、DDL 操作が進行中であるような、一部の複合環境で発生します。オペレーティング・システムは、要求を実行することが適している場合と、従来のコミットメント操作を実行する方がより妥当である場合とを判別できます。したがって、そのような環境でソフト・コミットを要求しても、悪影響はありません。

---

## コミットメント制御のシナリオおよび例

これらのシナリオおよび例では、ある会社がコミットメント制御をセットアップする方法を示しています。このトピック・コレクションには、コミットメント制御を使用するプログラムのコード例も記載されています。

以下のシナリオでは、JKL Toy Company がローカル・データベース上のトランザクションを追跡するためにコミットメント制御をインプリメントする方法が示されています。

次の例では、コミットメント制御のサンプル・コードが提供されています。実習問題は、コミットメント制御をインプリメントする RPG プログラムです。これには、その方法の各ステップで行われることを示すロジック・フローが含まれています。

### シナリオ：コミットメント制御

JKL Toy Company では、コミットメント制御を使用して製造および在庫のデータベース・レコードを保護しています。このシナリオは、JKL Toy Company がコミットメント制御を使用して、その在庫部門から製造部門に部品を移動する方法を示しています。

『シナリオ: ジャーナル管理』のトピックには、JKL Toy Company のネットワーク環境についての記述を記載します。次のシナリオでは、実動システム JKLPROD 上でコミットメント制御がどのように機能するかを示しています。

このシナリオでは、コミットメント制御を使用する利点を 2 つの例で示します。最初の例は、会社の在庫プログラム (プログラム A) をコミットメント制御なしで作動する方法、および発生する可能性のある問題点を示しています。2 番目の例は、コミットメント制御を使用してプログラムを作動する方法を示しています。

JKL Toy Company では、システム JKLPROD 上で在庫アプリケーション・プログラム (プログラム A) が使用されます。プログラム A は 2 つのレコードを使用します。1 つのレコードで、在庫に保管される品目が追跡されます。もう 1 つのレコードで、倉庫から移されて生産に使用される品目が把握されます。



## コミットメント制御を使用しないプログラム A

次のアプリケーション・プログラムでは、コミットメント制御を使用しないと仮定します。システムは、レコードを更新用の読み取りでロックします。次のステップでは、ダイオードが倉庫から製造部門に移されるのを、アプリケーション・プログラムが追跡する方法について説明します。

- プログラム A は在庫レコードをロックし、検索します (レコードを別のプログラムがロックしている場合には、この処置は待機する場合があります)。
- プログラム A は生産レコードをロックし、検索します (この処置にも時間がかかる場合があります)。プログラム A は現在、両方のレコードをロックしているので、他のプログラムはこれらを変更することができません。
- プログラム A は在庫レコードを更新します。これによってレコードは解放され、他のプログラムが更新のためにこれを読み取ることができます。
- プログラム A は生産レコードを更新します。これによってレコードは解放され、他のプログラムが更新のためにこれを読み取ることができます。

コミットメント制御を使用しないと、すべての状況でこのプログラムが正しく作動するように問題を解決する必要があります。例えば、ジョブ障害またはシステム障害のためにプログラム A がレコードを両方とも更新しない場合には、問題が起きます。この場合には、2 つのファイルには整合性がありません。つまり、ダイオードが在庫レコードから除去されてもそれは生産レコードには追加されません。コミットメント制御を使用することにより、トランザクションの処理が割り込まれたときにトランザクションに含まれるすべての変更が完了するか、またはファイルの状態が元の状態に戻ります。

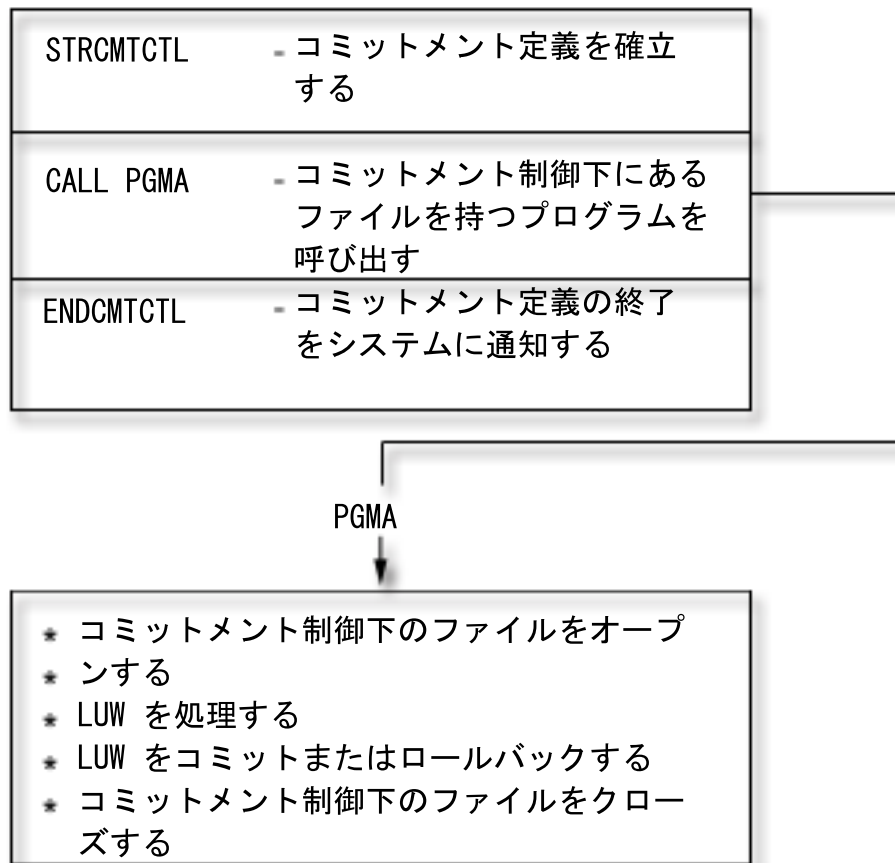
## コミットメント制御を使用するプログラム A

コミットメント制御を使用すると、前の例は次のように変更されます。

1. コミットメント制御が開始されます。
2. プログラム A は在庫レコードをロックし、検索します (レコードを別のプログラムがロックしている場合には、この処置に時間がかかる場合があります)。
3. プログラム A は生産レコードをロックし、検索します (この処置にも時間がかかる場合があります)。プログラム A は現在、両方のレコードをロックしているので、他のプログラムはこれらを変更することができません。
4. プログラム A は在庫レコードを更新し、コミットメント制御はレコードのロックを継続します。
5. プログラム A は生産レコードを更新し、コミットメント制御はレコードのロックを継続します。
6. プログラム A は、トランザクションをコミットします。在庫レコードと生産レコードに対する変更がファイルに永続的に適用されます。変更はジャーナルに記録され、そのことによって変更がディスクに入ったものと想定されます。コミットメント制御は両方のレコードのロックを解除します。これで、他のプログラムは更新の目的でこれを読み取ることができます。

両方のレコードに対するロックが、トランザクションがコミットされるまでコミットメント制御によって保持されるので、あるレコードが更新されるが他のレコードは更新されない、といった状態は起きません。トランザクションがコミットされる前に経路指定ステップまたはシステムに障害が起きた場合には、システムは、行われた変更を除去 (ロールバック) して、最後のトランザクションがコミットされた時点までファイルを更新します。

ファイルがコミットメント制御下に置かれる各経路指定ステップでは、次の図で示すステップが行われま



コミットメント制御下で実行される操作はジャーナルに記録されます。コミットメント制御開始ジャーナル項目は、コミットメント制御下の最初のファイル・オープン項目の後に現れます。これは最初のファイル・オープン項目がコミットメント制御にどのジャーナルを使用するかを決定するためです。さらに最初のオープン操作からのジャーナル項目を使用して 2 回目以降のオープン操作を検査し、すべてのファイルが同じジャーナルを使用することを確認します。

ジョブ障害またはシステム障害が起きたときには、コミットメント制御下にあるリソースはコミットメント境界に更新されます。トランザクションが開始されたが、完了する前に経路指定ステップが終了した場合は、そのトランザクションはシステムによりロールバックされ、経路指定ステップの終了後はファイルに表されません。トランザクションが完了する前にシステムが異常終了した場合、そのトランザクションはシステムによりロールバックされ、ライセンス内部コードの初期プログラム・ロード (IPL) が正常に実行された後はファイルに表されません。ロールバックが行われると、いつでも取り消し項目がジャーナルに入ります。

例えば、JKL Toy Company にダイオードの在庫が 100 個あるとします。製造で在庫から 20 個持ち出されて、残りが 80 個になります。データベースの更新によって、変更前イメージ (100) と変更後イメージ (80) の両方のジャーナル項目が作られます。

システムが項目をジャーナル処理した後で、コミットメント点またはロールバック点に達する前に異常終了したと仮定します。IPL の後、システムはジャーナル項目を読み取り、対応するデータベース・レコードを更新します。この更新によって更新を逆転させる 2 つのジャーナル項目が生じます。すなわち、最初の項目が変更前イメージ (80) で、2 番目の項目が変更後イメージ (100) になります。



異常終了後に IPL が正常に完了すると、システムはコミットされていないデータベース変更を除去 (ロールバック) します。前の例では、システムは在庫レコードの変更を除去します。その理由は、コミット操作がそのトランザクションのジャーナル内にはないからです。この場合には、在庫レコードの変更前イメージがファイルに入ります。ジャーナルには、ロールバックされた変更とロールバック操作が行われたことを示す標識が入ります。

#### 関連概念:

シナリオ: ジャーナル管理

## コミットメント制御の実習問題

この実習問題は、コミットメント制御およびその要件を理解する上で役立ちます。ここに記載の手順は、i5/OS ライセンス・プログラム、データ・ファイル・ユーティリティ (DFU)、およびこのトピック・コレクションについて読者がよく理解していることが前提となっています。

注: コード例を使用すると、126 ページの『コードに関するライセンス情報および特記事項』の条件に同意したことになります。

この問題を開始する前に、以下の前提条件の手順に従ってください。

1. この実習問題専用のライブラリーの作成。この指示では、ライブラリーは **CMTLIB** となっています。**CMTLIB** の部分をユーザーのライブラリー名に置き換えてください。
2. ソース・ファイルおよびジョブ記述の作成。

コミットメント制御を使用するには、以下の手順に従ってください。

1. **ITMP** という名前の物理ファイル (品目マスター・ファイル) を作成します。このファイルのデータ記述仕様 (DDS) は次のとおりです。

```
10  A  R  ITMR
20  A  ITEM  2
30  A  ONHAND  5  0
40  A  K  ITEM
```

2. **TRNP** という名前の物理ファイル (トランザクション・ファイル) を作成します。このファイルは、トランザクション・ログ・ファイルとして使用されます。このファイルの DDS は次のとおりです。

```
10  A  R  TRNR
20  A  QTY  5  0
30  A  ITEM  2
40  A  USER  10
```

3. **TRNL** という名前の論理ファイル (トランザクション論理ファイル) を作成します。このファイルは、アプリケーションの再開の援助として使用されます。**USER** フィールドは、タイプ **LIFO** の順序です。このファイルの DDS は次のとおりです。

```
10  LIFO
20  A  R  TRNR  PFILE (TRNP)
30  A  K  USER
```

4. **STRDFU** コマンドをタイプし、**ITMP** ファイルに対して **ITMU** という名前の **DFU** アプリケーションを作成します。アプリケーションの定義時には、**DFU** によって提供されるデフォルトを受け入れてください。
5. **CHGDTA ITMU** コマンドをタイプし、**ITMP** ファイルの次のレコードを入力します。

品目	在庫数量
AA	450
BB	375
CC	4000

- F3 キーを使用してプログラムを終了します。これによって、プログラムが操作対象とするある種のデータが与えられます。
- 次のように品目処理の CL プログラム (ITMPCSC) を作成します。

```
PGM
DCL &USER *CHAR LEN(10)
RTVJOBA USER(&USER)
CALL ITMPCS PARM(&USER)
ENDPGM
```

これは ITMPCS プログラムを呼び出す制御プログラムで、ユーザー名を取り出して処理プログラムに渡します。このアプリケーションでは、固有のユーザー名が使用されるものとしています。

- 次のように DDS から ITMPCSD という名前の表示装置ファイルを作成します。

2 つの様式があります。1 つは基本プロンプト画面に関する様式で、2 番目はオペレーターが直前に入力したものを検討できるようにする様式です。この表示装置ファイルは、ITMPCS プログラムによって使用されます。

```
SEQNBR *... .. 1 ... .. 2 ... .. 3 ... .. 4 ... .. 5 ... .. 6 ... .. 7 ..
```

```
1.00      A          R PROMPT
2.00      A
3.00      A
4.00      A
5.00      A
6.00      A
7.00      A          QTY          5  0I  +1
8.00      A  61
9.00      A
10.00     A
11.00     A          ITEM          2  I  +1
12.00     A  62
13.00     A
14.00     A  63
15.00     A
16.00     A  64
17.00     A
18.00     A
19.00     A
20.00     A
21.00     A
22.00     A
23.00     A          R REWV
24.00     A
25.00     A
26.00     A
27.00     A          QTY          5  0  +1EDTCDE(Z)
28.00     A
29.00     A          ITEM          2      +1
```

- コミットメント制御に関する実習問題のロジックのフローに示してあるロジックのフローを検討します。
- STRSEU コマンドを入力し、次のようにソースをタイプします。

```
SEQNBR *... .. 1 ... .. 2 ... .. 3 ... .. 4 ... .. 5 ... .. 6 ... .. 7 ..
```

```
1.00      FITMP  UF  E          K          DISK
2.00      F*
3.00      FTRNP  0  E          DISK
4.00      F*
5.00      FTRNL  IF  E          K          DISK
6.00      F          TRNR          KRENAMETRR1
7.00      FITMPCSD CF  E          WORKSTN
8.00      C* Enter parameter with User name for -TRNP- file
```

```

 9.00 C *ENTRY PLIST
10.00 C PARM USER 10
11.00 C LOOP TAG
12.00 C EXFMTPROMPT
13.00 C* Check for CF3 for end of program
14.00 C 93 DO End of Pgm
15.00 C SETON LR
16.00 C RETRN
17.00 C END
18.00 C* Check for CF4 for review last transaction
19.00 C 94 DO Review last
20.00 C* Check for existence of a record for this user in -TRNL- file
21.00 C USER CHAINTRNR1 64 Not found
22.00 C 64 GOTO LOOP
23.00 C EXFMTREVV
24.00 C GOTO LOOP
25.00 C END
26.00 C* Access Item record
27.00 C ITEM CHAINITMR 62 Not found
28.00 C* Handle -not found- Condition
29.00 C 62 GOTO LOOP
30.00 C* Does sufficient quantity exist
31.00 C ONHAND SUB QTY TEST 50 61 Minus
32.00 C* Handle insufficient quantity
33.00 C 61 DO
34.00 C* Release Item record which was locked by the CHAIN for update
35.00 C EXCPTRLSITM
36.00 C GOTO LOOP
37.00 C END
38.00 C* Change ONHAND and update the Item record
39.00 C Z-ADDTEST ONHAND
40.00 C UPDATITMR
41.00 C* Test for Special Simulation Conditions
42.00 C ITEM IFEQ 'CC'
43.00 C* Simulate program need for rollback
44.00 C QTY IFEQ 100
45.00 C SETON 63 Simult Rlbck
46.00 C* ROLBK
47.00 C GOTO LOOP
48.00 C END
49.00 C* Simulate an abnormal program cancellation by Div by zero
50.00 C* Operator Should respond -C- to inquiry message
51.00 C QTY IFEQ 101
52.00 C Z-ADD0 ZERO 30
53.00 C TESTZ DIV ZERO TESTZ 30 Msg occurs
54.00 C END
55.00 C* Simulate an abnormal job cancellation by DSPLY.
56.00 C* Operator Should System Request to another job
57.00 C* and cancel this one with OPTION(*IMMED)
58.00 C QTY IFEQ 102
59.00 C 'CC=102' DSPLY Msg occurs
60.00 C END
61.00 C END ITEM=CC
62.00 C* Write the -TRNP- file
63.00 C WRITETRNR
64.00 C* Commit the update to -ITMP- and write to -TRNP-
65.00 C* COMIT
66.00 C GOTO LOOP
67.00 OITMR E RLSITM

```

11. CRTRPGM コマンドを入力し、前のステップで入力したソースからプログラム ITMPCS を作成します。
12. コマンド CALL ITMPCSC をタイプして実行キーを押してから、F4 キーを押します。このオペレーターの項目がないことがメッセージに示されます。
13. 次のデータを入力して、プログラムが正しく作動するかどうかを調べます。

数量	品目
3	AA
4	BB

14. F4 キーを押します。最後に入力された BB 品目を示す検討画面が表示されます。次のデータを入力してください。

数量	品目
5	FF (無効品目番号のメッセージが表示されます。)
9000	BB (数量不足のエラー・メッセージが表示されます。)
100	CC (ロールバック・メッセージが表示されます。)
102	CC (RPG DSPLY 操作が行われます。実行キーを押してください。)
101	CC (プログラムは、0 除算状態が起こったことを示す照会メッセージを表示するか、あるいは終了します。これはジョブ属性 INQMSGRPY の設定によって異なります。照会メッセージが表示された場合には、C を入力して RPG プログラムを取り消してから、C を入力して後続の照会での CL プログラムを取り消してください。これは、予期しないエラー状態をシミュレートするものです。)

15. データ表示コマンド DSPDTA ITMP をタイプします。

レコード AA および BB が正しく更新されたかどうかを調べてください。値は、AA = 447、BB = 371、CC = 3697 となっているはずですが、CC からの数量減算が行われましたが、トランザクション・レコードが書き出されていないということに注意してください。

16. コミットメント制御のジャーナル・レシーバーを作成します。ジャーナル・レシーバー作成 (CRTJRNRCV) コマンドを使用して、RCVR1 というジャーナル・レシーバーを CMLIB ライブラリー内に作成します。少なくとも 5000 KB のしきい値を指定してください。システムに十分なスペースがあるなら、もっと大きな値が推奨されます。それは、新しいジャーナル・レシーバーの生成間の時間を最大にして、ジャーナルを頻繁に変更することによるパフォーマンス上の影響を最小化するためです。
17. コミットメント制御のジャーナルを作成します。ジャーナル作成 (CRTJRN) コマンドを使用して、JRNTST というジャーナルを CMTLIB ライブラリー内に作成します。このジャーナルはコミットメント制御専用で使用されるので、MNGRCV(\*SYSTEM) DLTRCV(\*YES) を指定してください。JRNRVCV パラメーターには、ステップ 16 で作成したジャーナル・レシーバーを指定してください。
18. パラメーター FILE (CMTLIB/ITMP CMTLIB/TRNP) JRN(CMTLIB/JRNTST) を指定した物理ファイル・ジャーナル開始 (STRJRNP) コマンドを使用して、コミットメント制御で使用するファイルをジャーナルします。

IMAGES パラメーターには、デフォルトの \*AFTER を使用してください。これは、レコードの変更後イメージ変更項目だけがジャーナルに示されることを意味します。ファイル ITMP および TRNP のジャーナル処理がここで開始されました。

通常では、ジャーナル処理の後にファイルを保管します。ジャーナル項目の JID とは異なる JID を持つ復元されたファイルに対して、ジャーナル処理済みの変更を適用することはできません。この実習問題ではジャーナル処理された変更を適用する必要はないので、ジャーナル処理されたファイルの保管は省略することができます。

19. コマンド CALL ITMPCSC をタイプして、次のトランザクションを入力します。

数量	品目
5	AA
6	BB

F3 キーを使用してプログラムを終了します。

20. 次のジャーナル表示コマンドをタイプします。DSPJRN CMTLIB/JRNTEST

ジャーナルにある項目に注意してください。ジャーナル内の項目は、プログラムによって実行された場合と同じ項目の順序 (R UP = ITMP の更新、次に R PT = TRNP に追加されたレコード) になります。これは、物理ファイル TRNP に論理ファイルが定義されており、システムが RPG のデフォルトを一時変更したためです。論理ファイルが存在していない場合には、RPG の前提事項の SEQONLY(\*YES) が使用されることになり、1 ブロックの PT 項目が現れます。これは、ブロックがいっぱいになるまでレコードが RPG バッファに保持されているためです。

21. CL プログラム ITMPCSC を次のように変更します (新しいステートメントは、アスタリスクで示してあります)。

```
PGM
DCL &USER *CHAR LEN(10)
RTVJOBA USER(&USER)
* STRCMTCTL LCKLVL(*CHG)
CALL ITMPCS PARM(&USER)
* MONMSG MSGID(RPG9001) EXEC(ROLLBACK)
* ENDCMTCTL
ENDPGM
```

STRCMTCTL コマンドは、コミットメント制御環境を設定します。LCKLVL という語は、レコードが更新のために読み取られるが、更新後のレコードがトランザクションの続行中解放することができないことを指定します。MONMSG コマンドは、RPG エスケープ・メッセージを処理し、RPG プログラムが異常終了した場合には ROLLBACK を実行します。ENDCMTCTL コマンドは、コミットメント制御環境を終了します。

22. 既存の ITMPCSC プログラムを削除してから、作成し直します。
23. ステートメント 2.00、4.00、46.00、および 65.00 の注記記号を除去するように、RPG プログラムを変更します。これで、このソース・プログラムはコミットメント制御で使用できるようになります。
24. 既存の ITMPCS プログラムを削除してから、作成し直します。これで、プログラムはコミットメント制御の下で作動できるようになりました。
25. コマンド CALL ITMPCSC および次のトランザクションを入力します。

数量	品目
7	AA
8	BB

26. システム要求を使用して、現行ジョブを表示するオプションを要求します。「ジョブの表示」画面が表示されたら、オプション 16 を選択して、コミットメント制御の状況の表示を要求してください。

画面の値に注意してください。プログラムでは 2 つのコミットメント・ステートメントが実行されたので、2 つのコミットがあるはずですが。

27. F9 キーを押して、コミットメント制御下にあるファイルおよび各ファイルの活動量を示すリストを表示します。
28. プログラムに戻って、F3 キーを押すことによってプログラムを終了します。



29. DSPJRN CMTLIB/JRNTEST をタイプして、ファイルの項目とコミットメント制御の特殊ジャーナル項目に注意します。

項目	意味
C BC	STRCMTCTL コマンドが実行された。
C SC	コミット・サイクルの開始。これは、トランザクションの最初のデータベース操作によってレコードが、コミットメント制御の一部として挿入、更新、または削除されたときには常に起きます。
C CM	コミット操作が行われた。
C EC	ENDCMTCTL コマンドが実行された。

ジャーナルについて初めに IMAGES (\*AFTER) を要求しても、変更前イメージと変更後イメージ (R UB および R UP タイプ) のコミットメント制御が自動的に行われます。

30. コマンド CALL ITMPCSC および次のトランザクションを入力します。

数量	品目
12	AA
100	CC (これは、ロールバックのアプリケーションでの使用の必要性をシミュレートする条件です。RPG ステートメント 40.00 によって更新される ITMP ファイル中の CC レコードがロールバックされます。)

31. F4 キーを押して、最後に入力されたトランザクションを確認します。

最後にコミットされたトランザクションは、品目 AA に対する入力です。

32. システム要求を使用して、現行ジョブの表示オプションを要求します。「ジョブの表示」画面が表示されたら、コミットメント制御の状況の表示を要求してください。

画面の値およびその値がロールバックによってどのように変更されているかに注意してください。

33. プログラムに戻ります。

34. 基本プロンプト画面に戻り、F3 キーを押すことによってプログラムを終了します。

35. コマンド DSPJRN CMTLIB/JRNTEST をタイプします。

ロールバック項目で使用するジャーナルの他の項目 (C RB 項目) に注意してください。ITMP レコードがロールバックされると、3 つの項目がジャーナルに入れられます。その理由は、コミットメント制御下でデータベース・ファイルに行われた変更は、変更前 (R BR) および変更後 (R UR) 項目を作成するからです。

36. ジャーナル・コード R および次の項目タイプを使って、項目を表示します。UB、UP、BR、および UR。項目全体を表示するには、オプション 5 を使用します。数量 フィールドは、パック 10 進数なので、16 進数の表示を希望する場合は F11 を使用してください。次のことに注意してください。

- UB レコードの ITMP レコードの在庫数量値
- UP レコードによって在庫数量値がどのように減少されたか
- BR レコードが UP レコードと同じか
- UR レコードが UB レコードに最初に表示されていたものとしてどのように値を戻したか

ロールバックの終わりでは最後の項目は RB 項目です。

37. コマンド CALL ITMPCSC をタイプして実行キーを押してから、F4 キーを押します。最後に入力されたトランザクションに注意してください。
38. 次のトランザクションを入力します。

数量	品目
13	AA
101	CC (これは、プログラムの終了を引き起こす予期しないエラー状態をシミュレートする条件です。このシミュレーションは、フィールドを 0 で除算することにより発生します。プログラムは、ジョブ属性 INQMSGRPY の設定に応じて照会メッセージを表示するか、あるいは終了します。照会メッセージが表示された場合には、C を入力してプログラムを終了してください。CL プログラムが RPG プログラム・エラーを監視するように変更されているので、行われた 2 番目の照会は行われません。)

39. コマンド DSPJRN CMTLIB/JRNTEST をタイプします。

同じタイプのロールバック処理が行われますが、今回は RPG プログラムではなく、CL プログラムの MONMSG コマンドの EXEC パラメーターによって行われています。どちらのプログラムが原因であるかを調べるためには、2 つの RB 項目を表示してください。

40. コマンド WRKJOB をタイプして、後で使用できるように完全な修飾ジョブ名を書き留めておきます。
41. コマンド CALL ITMPCSC をタイプして、次のトランザクションを入力します。

数量	品目
14	AA
102	CC (外部メッセージ・キューに対する RPG DSPLY 命令が行われるはずですが、システム要求キーを使用し、システム要求メニューでオプション 1 を選択して 2 次ジョブに移ってください。)

42. 2 次ジョブにサインオンして、環境を確立し直します。
43. コマンド ENDJOB をタイプし、前に示された完全な修飾ジョブ名と OPTION(\*IMMED) を指定します。これは、システムまたはジョブの異常終了をシミュレートします。
44. 30 秒待ってから、コマンド CALL ITMPCSC をタイプして F4 キーを押します。

最後にコミットされたトランザクションに注意してください。これは、前に入力された AA 項目であるはずですが。

45. 基本プロンプト画面に戻り、F3 キーを押すことによってプログラムを終了します。
46. コマンド DSPJRN CMTLIB/JRNTEST をタイプします。

同じタイプのロールバック処理が行われますが、今回のロールバックはプログラムではなく、システムによって行われます。RB 項目は、実行管理機能異常終了プログラムの QWTPITPP によって書き出されます。

ここではコミットメント制御の 2 つの基本的な機能を使用しました。アプリケーションのコミットメント制御を続けるか、あるいは次のような他のいくつかの機能を試みることができます。

- 通知オブジェクトの使用

- LCKLVL(\*ALL) による読み取り専用レコードのロック
- LCKLVL(\*ALL) による同じファイル中の複数レコードのロック

### **実習問題のロジックのフロー**

ロジックのフローは、コミットメント制御に関するこの実習問題をより理解するのに役立ちます。次の図は、コミットメント制御の実習問題のフローを示します。

図に示されている各ステップについて詳しくは、91 ページの『実習プログラムのロジックのフローに関連したステップ』を参照してください。



### 実習プログラムのロジックのフローに関連したステップ

次のステップは実習問題のロジックのフローに関連しています。

1. パラメーターとして渡されたユーザー名を取り出します。これは、TRNP ファイルに書き出しを行うために使用されるとともに、各オペレーターによって最後に入力されたトランザクションを取り出すためにも使用されます。このアプリケーションでは、各オペレーターに固有のユーザー名が使用されるものとしています。
2. 様式名 **PROMPT** を使用して、基本プロンプト画面を表示します。
3. F3 を押した場合には、プログラム終了機能が開始されます。
4. F4 を押した場合には、オペレーターによって最後に入力されたトランザクションをアクセスするルーチンが開始されます。
5. フィールド **ITEM** を使用して品目レコードを読み取ります。ファイルは更新ファイルであるので、この要求によってレコードがロックされます。
6. ファイル **ITMP** 中でレコードが見つからないという状態を検査します。
7. **ITMP** レコードが存在しない場合には、標識 62 をオンに設定し、エラー・メッセージを表示し、ステップ 2 に戻ります。
8. 在庫数量 (**ONHAND**) から要求数量 (**QTY**) を減算し、それを作業域に入れます。
9. 要求を満たすのに十分な数量があるかどうかを調べます。
10. 数量が不足している場合には、**ITMP** ファイル中のレコードのロックを解除します。このステップは、数量が不足することもあるので必要です。
11. 標識 61 をオンに設定して、数量不足の表示エラー・メッセージを示し、ステップ 2 に戻ります。
12. **ONHAND** フィールドを新しい在庫数量に変更して、**ITMR** レコードを更新します。
13. ロールバックが要求されるような状態をシミュレートするために使用できる **ITEM** フィールドの特殊な項目を検査します。
14. **QTY=100** を検査して、**ROLLBACK** 命令を出します。これは、プログラムがロールバックの必要性を感知するような状態をシミュレートします。
15. **QTY=101** を検査します。照会メッセージを生成する例外をプログラム中で起こします。この機能のために 0 除算を使用します。オペレーターは、ジョブ記述 **INQMSGRPH** オプションで自動応答が提供されないかぎり、**C** を入力してプログラムを取り消さなければなりません。これは、予期しないエラーが検出され、オペレーターがプログラムを取り消す状態をシミュレートします。
16. **QTY=102** を検査して、照会操作の画面を出します。これにより、プログラムはこのステップで停止し、システム要求キーを使用して別のジョブに移ることができます。更新ジョブを取り消してください。これは、コミットメント境界の途中でジョブまたはシステムの異常終了が起こった状態をシミュレートします。
17. トランザクション・レコードを **TRNP** に書き出します。
18. トランザクションのレコードをコミットして、ステップ 2 に戻ります。
19. **USER** をキーとして使用し、ファイル **TRNL** のアクセス・パスの最初のレコードを読み取ります。このファイルは **LIFO** 順序であるので、このレコードは、このユーザーが最後に入力したトランザクション・レコードです。
20. **TRNL** ファイルにレコードがないという状態を検査します。この状態は、ファイルにこのユーザーの項目がない場合に起こります。
21. このユーザーのレコードがない場合には、標識 64 をオンに設定して、エラー・メッセージを表示し、ステップ 2 に戻ります。
22. このユーザーが最後に入力したトランザクションを表示します。この情報は、オペレーターが前に入力したことを忘れた場合、あるいはトランザクションを再開するときに使用することができます。オペレーターが応答すると、ステップ 2 に戻ります。



23. プログラム機能のいずれかを実行します。

## 例: アプリケーション開始のためのトランザクション・ログ記録ファイルの使用

この例では、異常終了後にアプリケーションを開始させるためのトランザクション・ログ記録ファイルの使用法について、サンプル・コードおよび説明を提供しています。

注: コード例を使用すると、126 ページの『コードに関するライセンス情報および特記事項』の条件に同意したことになります。

トランザクション・ログ記録ファイルは、通知オブジェクトが使用されないときに、システムまたはジョブの障害後にアプリケーションを再開するために使用されます。トランザクション・ログ記録ファイルは、対話式のアプリケーションにおいて、トランザクションの影響を要約するために多用されます。

例えば、受注アプリケーションにおいてはレコードは通常、受注された各品目ごとにトランザクション・ログ記録ファイルに書き出されます。このレコードには、注文された品目、数量、および価格が入っています。買掛管理アプリケーションでは、料金を受け取る各口座番号ごとにトランザクション・ログ記録ファイルにレコードが書き出されます。このレコードには通常、口座番号、未支払い金額、販売担当者などの情報が入っています。

トランザクション・ログ記録ファイルが既に存在している多くのアプリケーションでは、ワークステーション・ユーザーは、直前に入力されたトランザクションについての情報を要求することができます。トランザクション・ログ記録ファイルが存在しているアプリケーションにコミットメント制御を追加することによって、次のことを行うことができます。

- コミットメント境界に対してデータベース・ファイルが更新されたかを確認する。
- トランザクションの再開を簡単にする。

コミットメント制御の下でアプリケーションを再開するためにトランザクション・ログ記録ファイルを使用する場合には、ワークステーション・ユーザーを固有に識別できなければなりません。システムで固有のユーザー・プロファイル名が使用されている場合には、そのプロファイル名をトランザクション・ログ記録レコードの 1 つのフィールドに入れることがあります。このフィールドは、ファイルへのキーとして使用することができます。

次の例では、受注在庫ファイルを使用してトランザクションを実行すること、およびトランザクション・ログ記録ファイルが既に存在していることを想定しています。プログラムは、以下のタスクを行います。

1. 数量および品目番号についてのプロンプトをワークステーション・ユーザーに出します。
2. 品目マスター・ファイル (PRDMSTP) の数量を更新します。
3. レコードをトランザクション・ログ記録ファイル (ISSLOGL) に書き出します。

在庫数量が不足している場合には、プログラムはトランザクションを拒否します。品目番号、品名、数量、ユーザー名、および日付がトランザクション・ログ記録ファイルに書き出されるため、ワークステーション・ユーザーは、データの入力がどこで中断されたかをプログラムに問い合わせることができます。

### 物理ファイル PRDMSTP の DDS

```
SEQNBR *... .. 1 ... .. 2 ... .. 3 ... .. 4 ... .. 5 ... .. 6 ... .. 7  
  
1.00      A          R PRDMSTR          TEXT('マスター・レコード')  
2.00      A          PRODCT          3          COLHDG('品名' '番号')
```

3.00	A	DESCRP	20	COLHDG('品名')
4.00	A	ONHAND	5 0	COLHDG('在庫' '数量')
5.00	A			EDTCDE(Z)
6.00	A	K PRODC		

### ISSLOGP で使用される物理ファイル ISSLOGP の DDS

SEQNBR	*... .. 1 ... .. 2 ... .. 3 ... .. 4 ... .. 5 ... .. 6 ... .. 7
1.00	A R ISSLOGR TEXT('品目ログ・レコード')
2.00	A PRODC 3 COLHDG('品名' '番号')
3.00	A DESCRP 20 COLHDG('品名')
4.00	A QTY 3 0 COLHDG('数量')
5.00	A EDTCDE(Z)
6.00	A USER 10 COLHDG('ユーザー' '名')
7.00	A DATE 6 0 EDTCDE(Y)
8.00	A COLHDG('日付')

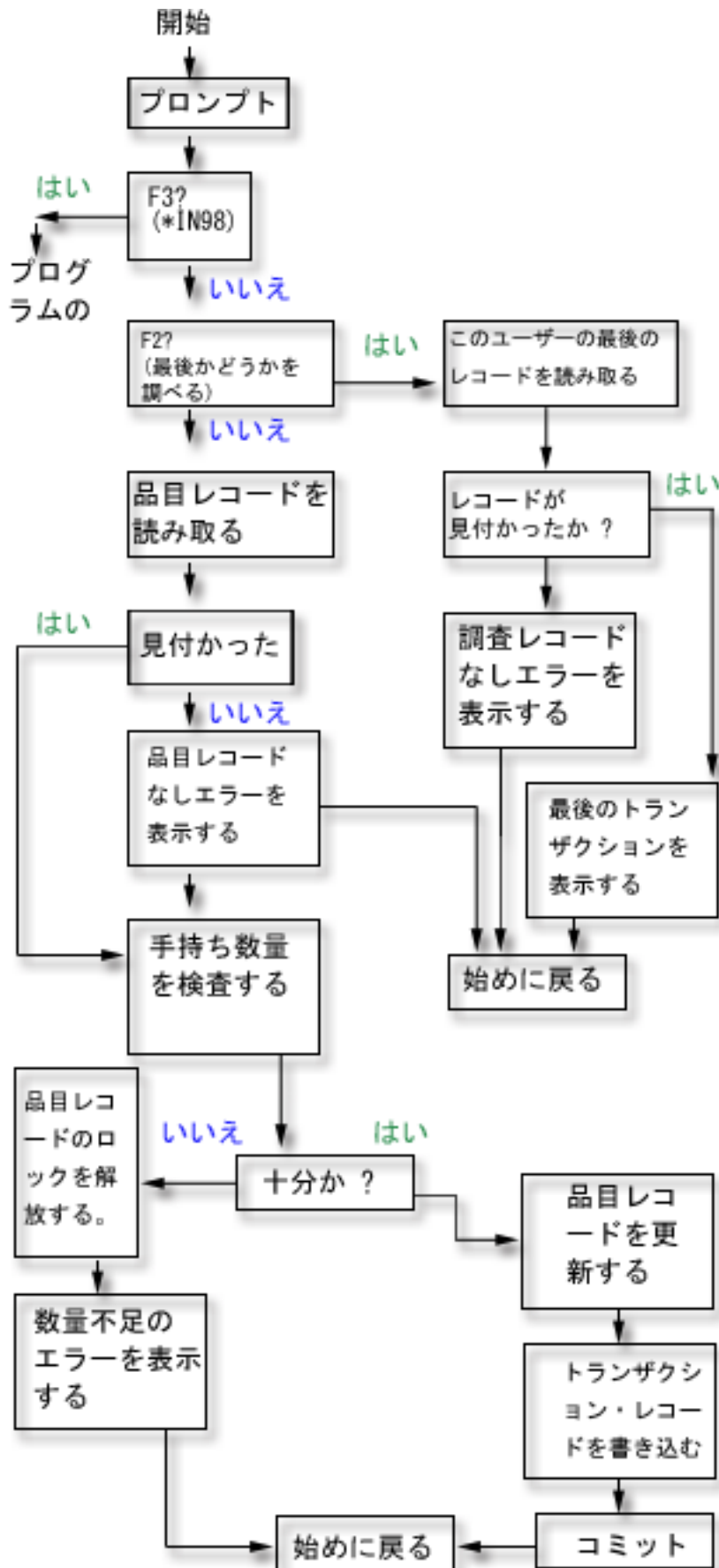
### 論理ファイル ISSLOGL の DDS

SEQNBR	*... .. 1 ... .. 2 ... .. 3 ... .. 4 ... .. 5 ... .. 6 ... .. 7
1.00	A LIFO
2.00	A R ISSLOGR PFILE(ISSLOGP)
3.00	A K USER

### プログラムで使用される表示装置ファイル PRDISSD の DDS

SEQNBR	*... .. 1 ... .. 2 ... .. 3 ... .. 4 ... .. 5 ... .. 6 ... .. 7 ..
1.00	A REF(ISSLOGP)
2.00	A R PROMPT
3.00	A CA03(98 'End of program')
4.00	A CA02(97 'Where am I')
5.00	A 1 20'ISSUES PROCESSING'
6.00	A 3 2'Quantity'
7.00	A QTY R I +1
8.00	A 62 ERRMSG('在庫数量が不足 +
9.00	A しています' 62)
10.00	A +6'品目'
11.00	A PRODC R I +1
12.00	A 61 ERRMSG('品目があり +
13.00	A ません' 62)
14.00	A 55 15 2'No Previous record exists'
15.00	A 24 2'CF2 Last transaction'
16.00	A R RESTART
17.00	A 1 20'LAST TRANSACTION +
18.00	A INFORMATION'
19.00	A 5 2'品目'
20.00	A PRODC R +1
21.00	A 7 2'品名'
22.00	A DESCRP R +1
23.00	A 9 2'数量'
24.00	A QTY R +1REFFLD(QTY)

この処理の概要は、プログラムのフローに示してあります。



PRDMSTP ファイルが更新され、レコードがトランザクション・ログ記録ファイルに書き出された後に RPG COMMIT 命令が指定されています。オペレーターに対して出される各プロンプトは新しいトランザクションの境界を表すので、トランザクションは単一入力トランザクションと見なされます。

プログラムが呼び出されると、プログラムにユーザー名が渡されます。トランザクション・ログ記録ファイルのアクセス・パスは、プログラムが最後に入力されたレコードを容易にアクセスできるように後入れ先出し (LIFO) 順序で定義されます。

ワークステーション・ユーザーは、システムまたはジョブの障害後に、データ入力停止と識別されたのと同じ機能を使用してプログラムを再開できます。プログラムに他のコードを追加する必要はありません。現在トランザクション・ログ記録ファイルを使用しているが、処理個所の判別にそれを使用していない場合には、ユーザー名をトランザクション・ログ記録ファイルに追加して (ユーザー名が固有であると見なします) この方法をプログラムで使用してください。

以下の例には、使用される RPG プログラムを示します。コミットメント制御に必要なステートメントは、矢印 (==>) でマークしてあります。

## RPG プログラム

```

SEQNBR *... 1 ... 2 ... 3 ... 4 ... 5 ... 6 ... 7 ..
=>1.00  FPRDMSTP UP E          K          DISK          KCOMIT
=>2.00  FISSLOGL IF E          K          DISK          KCOMIT
3.00   PRDISSD CP E          WORKSTN
4.00   *ENTRY          PLIST
5.00   PARM          USER 10
6.00   C*
7.00   C* Initialize fields used in Trans Log Rcd
8.00   C*
9.00   C          MOVE UPDATE DATE
10.00  C*
11.00  C* Basic processing loop
12.00  C*
13.00  C          LOOP TAG
14.00  C          EXFMTPROMPT
15.00  C 98          GOTO END          End of pgm
16.00  C 97          DO          Where am I
17.00  C          EXSR WHERE
18.00  C          GOTO LOOP
19.00  C          END
20.00  C          PRODC T          CHAINPRDMSTR          61          Not found
21.00  C 61          GOTO LOOP
22.00  C          ONHAND SUB QTY TEST 50 62 Less than
23.00  C 62          DO          Not enough
24.00  C          EXCPTRLSMST          Release lock
25.00  C          GOTO LOOP
26.00  C          END
27.00  C*
28.00  C* Update master record and output the Transaction Log Record
29.00  C*
30.00  C          Z-ADDTEST ONHAND
31.00  C          UPDATPRDMSTR
32.00  C          WRITEISSLOGR
=>33.00 C          COMMIT
34.00  C          GOTO LOOP
35.00  C*
36.00  C* End of program processing
37.00  C*
38.00  C          END TAG
39.00  C          SETON          LR
40.00  C*

```

```

41.00      C* WHERE subroutine for "Where am I" requests
42.00      C*
43.00      C          WHERE      BEGSR
44.00      C          USER      CHAINISSLOGL          55      Not found
45.00      C N55          EXFMTRESTART
46.00      C          ENDSR
47.00      OPRDMSTR E          RLSMST

```

## RPG プログラム PRDISS を呼び出すために使用される CL プログラム

```
SEQNBR *... .. 1 ... .. 2 ... .. 3 ... .. 4 ... .. 5 ... .. 6 ... .. 7 ..
```

```

1.00      PGM
2.00      DCL          &USER *CHAR LEN(10)
3.00      STRCMTCTL   LCKLVL(*CHG)
4.00      RTVJOBA    USER(&USER)
5.00      CALL       PRDISS PARM(&USER)
6.00      MONMSG     MSGID(RPG9001) EXEC(ROLLBACK)
7.00      ENDCMTCTL
8.00      ENDPGM

```

このプログラムでコミットメント制御を使用するためには、通常、\*CHG のロック・レベルが使用されます。レコードは、変更のコミットメント操作が実行されるまでロックされます。在庫数量が不足している場合には、そのレコードは明示的に解放されることに注意してください。(レコードがプログラムで明示的に解放されなかった場合には、ファイルから更新用に次のレコードが読み取られるとレコードが解放されません。)

この例では、ロック・レベル \*ALL を使用する利点はありません。\*ALL が使用される場合には、数量が不足したときに、レコードを解放するためにロールバック操作またはコミット操作を使用しなければなりません。

前のコードは、RPG プログラム PRDISS を呼び出す CL プログラムです。STRCMTCTL/ENDCMTCTL コマンドが使用されていることに注意してください。固有のユーザー名が検索され (RTVJOBA コマンド)、プログラムに渡されます。ロールバックを起こすような MONMSG コマンドの使用については、例：アプリケーション開始のための標準処理プログラムの使用の項で説明しています。

### 関連概念:

105 ページの『例：アプリケーション開始のための標準処理プログラムの使用』

標準処理プログラムは、すべてのアプリケーションに対する通知オブジェクトとして 1 つのデータベース・ファイルを使用することによってアプリケーションを再開するための 1 つの方法です。この方法では、標準処理プログラムを使用するすべてのアプリケーションでユーザーごとにユーザー・プロファイル名が固有であることを前提としています。

99 ページの『例：プログラムごとに固有の通知オブジェクト』

各ジョブに 1 つの固有の通知オブジェクトを使用すると、同じプログラムにユーザーが複数いても、外部記述コミットメント ID を使用することができます。

## 例：アプリケーション開始のための通知オブジェクトの使用

異常終了後にプログラムが開始されると、そのプログラムは通知オブジェクトの中の項目を検索します。項目が存在すると、プログラムは再度トランザクションを開始することができます。トランザクションが再開されると、通知オブジェクトはプログラムにより消去されます。これにより、同じトランザクションが別の時に開始されないようにします。

通知オブジェクトの使用方法は、次のとおりです。



- データベース・ファイルにコミットメント ID が入っている場合には、このファイルに QUERY を使用して各アプリケーションまたはワークステーション・ジョブをどこから再開するかを決定する。
- 特定のワークステーションのメッセージ・キューにコミットメント ID が入っている場合には、サインオン時にワークステーション・ユーザーにコミットされた最後のトランザクションを知らせるメッセージを送ることができる。
- キー名またはユーザー名をもつデータベース・ファイルにコミットメント ID が入っている場合には、プログラムは開始時にこのファイルを読み取ることができる。ファイルにレコードが存在する場合には、プログラムを再開してください。プログラムは、最後にコミットされたトランザクションを識別するメッセージをワークステーション・ユーザーに送ることができます。リカバリーはすべて、プログラムによって行われます。レコードがデータベース・ファイルに存在していた場合には、プログラムは終了時にそのレコードを削除します。
- バッチ・アプリケーションでは、コミットメント ID を合計、スイッチ設定、およびアプリケーションの再開に必要なその他の状況情報が入っているデータ域に入れることができる。アプリケーションはその開始時にデータ域をアクセスして、そこに保管されている値を調べます。アプリケーションが正常に終了した場合には、そのデータ域は次回の実行に備えてセットアップされます。
- バッチ・アプリケーションの場合、コミットメント ID をメッセージ・キューに送ることができる。アプリケーションの開始時に実行されるプログラムは、メッセージをこの待ち行列から検索し、その他の各プログラムを再開することができます。

アプリケーションの要求に従ってアプリケーションを再開するには、いくつかの方法があります。方法を選択するときには、次の情報を考慮してください。

- 同時にプログラムに複数のユーザーがいる場合には、1 つのデータ域を通知オブジェクトとして使用することはできません。それは、システムの異常終了後にあるデータ域を使用すると、各ユーザーのコミットメント ID がそのデータ域の中で互いに重なることになるためです。
- 通知オブジェクトの情報の削除に関しては、障害が起きた時の状況を即座に以下の情報に従って処理するように設計する必要があります。
  - 情報が即座に削除された場合、割り込まれたトランザクションの処理が行われる前に別の障害が起きた場合は、その情報は存在しない。
  - 通知オブジェクトの情報は、割り込まれたトランザクションが正常に処理されるまで削除してはならない。この場合、通知オブジェクトがデータベース・ファイルまたはメッセージ・キューであると、その中に複数の項目が存在することになる。
  - 複数の項目が存在する場合には、プログラムは最後のレコードをアクセスする必要がある。
- 通知オブジェクトを使用して、最後にコミットされたトランザクションをワークステーション・ユーザーに示すことはできません。それは、システムまたはジョブの障害が起こった場合、あるいはジョブの正常終了時にコミットされていない変更が存在している場合にのみ、通知オブジェクトが更新されるからです。
- 情報がワークステーション・ユーザーに表示される場合には、その情報は意味のあるものでなければなりません。情報を意味のあるものにするためには、プログラムが通知オブジェクト中のコードをユーザーが再開に利用できる情報に変換する必要があります。
- 再開に必要な情報は、ワークステーション・ユーザーが必要とした場合に表示されるようにしなければなりません。意味のなくなった情報が再表示されないようにするためには、プログラムに追加のロジックが必要です。
- 通知オブジェクトがデータベース・ファイルである場合には、1 つの通知オブジェクトおよび標準処理プログラムによって、再開機能が可能になります。この標準処理プログラムは、再開機能を必要とするプログラムによって呼び出され、個別の各プログラムに対する変更を最小限にします。

## 関連概念:

55 ページの『通知オブジェクトのコミット』

通知オブジェクトはメッセージ・キュー、データ域、またはデータベース・ファイルで、そのコミットメント定義が正常に終了しなかった場合、特定のコミットメント定義の最後に正常に完了したトランザクションを識別する情報が含まれています。

## 例：プログラムごとに固有の通知オブジェクト

各ジョブに 1 つの固有の通知オブジェクトを使用すると、同じプログラムにユーザーが複数いても、外部記述コミットメント ID を使用することができます。

次の例では、通知オブジェクトとしてデータベース・ファイルが使用され、しかもこのプログラムによってのみ使用されます。

プログラムには 2 つのデータベース・ファイル (PRDMSTP と PRDLOCP) があり、これらは在庫に対する受け入れで更新されなければなりません。プログラムによって使用される表示装置ファイルは、PRDRCTD という名前になっています。データベース・ファイル PRDRCTP は、通知オブジェクトとして使用されます。この通知オブジェクトは、プログラムに対してファイルとして定義され、通知機能のためのデータ構造の定義としても使用されます。

注: コード例を使用すると、126 ページの『コードに関するライセンス情報および特記事項』の条件に同意したことになります。

## 物理ファイル PRDLOCP の DDS

SEQNBR \*... .. 1 ... .. 2 ... .. 3 ... .. 4 ... .. 5 ... .. 6 ... .. 7

1.00	A	R	PRDLOCR			TEXT('位置レコード')
2.00	A		PRODC	3		COLHDG('品名' '番号')
3.00	A		LOCATN	6		COLHDG('位置')
4.00	A		LOCAMT	5	0	COLHDG('位置' '数量')
5.00	A					EDTCDE(Z)
6.00	A	K	PRODC			
7.00	A	K	LOCATN			

## 表示装置ファイル PRDRCTD の DDS

SEQNBR \*... .. 1 ... .. 2 ... .. 3 ... .. 4 ... .. 5 ... .. 6 ... .. 7 ..

1.00	A					REF(PRDMSTP)
2.00	A	R	PROMPT			
3.00	A					CA03(98 'End of program')
4.00	A					SETOFF(71 'RESTART')
5.00	A			1	20	'PRODUCT RECEIPTS'
6.00	A			3	2	'Quantity'
7.00	A		QTY	3	OI	+1
8.00	A					+6'品目'
9.00	A		PRODC	R	I	+1
10.00	A	61				ERRMSG('マスター・ファイルに +
11.00	A					レコードが +
12.00	A					見つからない' 62)
13.00	A					+6'位置'
14.00	A		LOCATN	R	I	+1REFFLD(LOCATN PRDLOCP)
15.00	A	62				ERRMSG('位置ファイルに +
16.00	A					レコードが +
17.00	A					見つからない' 62)
18.00	A			9	2	'Last Transaction'
19.00	A	71				+6'This is restart +
20.00	A					information'
21.00	A					DSPATR(HI BL)
22.00	A			12	2	'数量'

23.00	A			12	12'品目'
24.00	A			12	23'位置'
25.00	A			12	35'品名'
26.00	A	LSTPRD	R	14	15REFFLD(PRODUCT)
27.00	A	LSTLOC	R	14	26REFFLD(LOCATN *SRC)
28.00	A	LSTQTY	R	14	5REFFLD(QTY *SRC)
29.00	A				EDTCDE(Z)
30.00	A	LSTDSC	R	14	35REFFLD(DESCRP)

## 通知オブジェクトおよび外部記述データ構造 (PRDRCTP) の DDS

SEQNBR \*... .. 1 ... .. 2 ... .. 3 ... .. 4 ... .. 5 ... .. 6 ... .. 7 ..

1.00	A				LIFO
2.00	A				REF (PRDMSTP)
3.00	A	R	PRDRCTR		
4.00	A		USER	10	
5.00	A		PRODUCT	R	
6.00	A		DESCRP	R	
7.00	A		QTY	3 0	
8.00	A		LOCATN	R	REFFLD(LOCATN PRDLOCP)
9.00	A	K	USER		

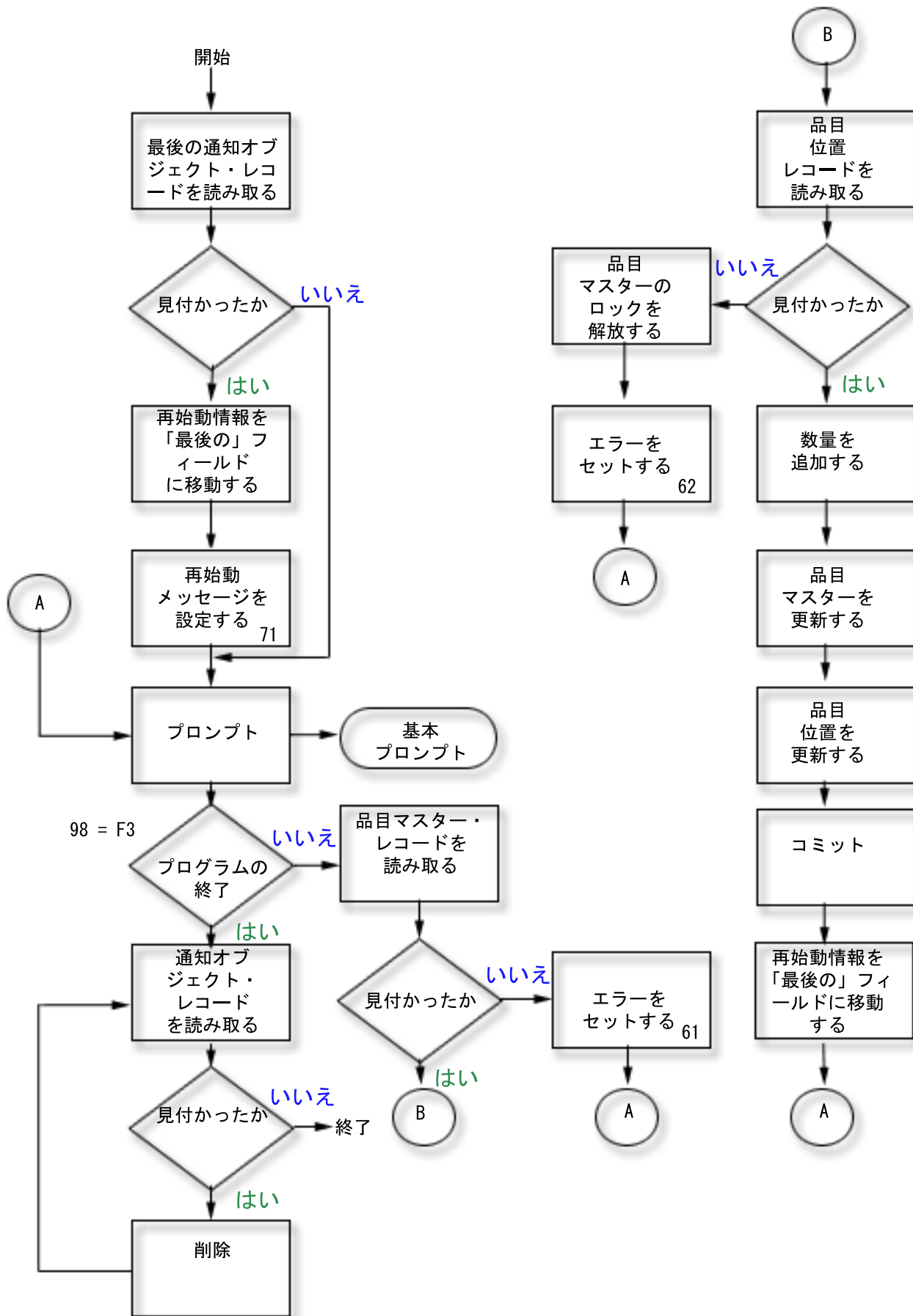
プログラムは、通知オブジェクトを次のように処理します。

- 開始時に、プログラムは通知オブジェクトをランダムに処理し、特定のキーのレコードが存在する場合にはそのレコードを表示します。
  - 複数のレコードが存在する場合には、PRDRCTP ファイルが LIFO の順序になっているため、このキーの最後のレコードが使用される。
  - レコードが存在しない場合は、トランザクションが割り込まれなかったので再開する必要はない。
  - 最初のコミット操作が成功する前にプログラムが正常に実行されなかった場合には、プログラムは再開が必要であるとは見なされない。
- 通知オブジェクトを消去するためのルーチンは、プログラムの終わりにあります。
  - 複数の障害があった場合には、このルーチンは通知オブジェクト中の複数のレコードの削除を処理することができる。
  - システムはコミットメント ID をデータベース・ファイルに入れるが、その ID は RPG プログラム中に変数として定義されていなければならない。
  - RPG ではデータ構造を外部記述できるので、データ構造はコミットメント ID を指定するのに便利な方法である。この例では、データ構造はデータベース・ファイルが通知オブジェクトとして使用されたのと同じ外部記述を使用します。

このプログラムの処理では、品目番号、位置、および数量を入力するためのプロンプトがユーザーに表示されます。

- 次の 2 つのファイルを更新しなければなりません。
  - 品目マスター・ファイル (PRDMSTP)
  - 品目位置ファイル (PRDLOCP)
- いずれかを更新する前に、各ファイルにレコードがなければなりません。
- 毎回、各トランザクションが正常に入力された後で、プログラムは入力フィールドを、最後の対応するフィールドに移動します。この最後のフィールドは、直前に何が入力されたかを示すフィードバックが画面にプロンプトでオペレーターに表示されます。
- 再開のための情報が存在する場合には、その情報が最後のフィールドに移動され、特殊なメッセージが画面が表示されます。

この処理の概要は、次の図に示してあります。通知オブジェクト中のレコードを固有とするためにユーザー名がプログラムに渡されます。





RPG ソース・コードの例を以下に示します。通知オブジェクト (ファイル PRDRCTP) はプログラムの開始時と終了時に通常のファイルとして使用され、プログラムの呼び出し前に CL (STRCMTCTL コマンド) 中に通知オブジェクトとしても指定されています。

## RPG ソース

```

SEQNBR *... .. 1 ... .. 2 ... .. 3 ... .. 4 ... .. 5 ... .. 6 ... .. 7 ..

1.00      FPRDMSTP UF  E          K          DISK          KCOMIT
2.00      FPRDLOCP UF  E          K          DISK          KCOMIT
3.00      FPRDRCTD CF  E          WORKSTN
4.00      F*
5.00      F* The following file is the specific notify object for this pgm.
6.00      F*       It is accessed only in a restart situation and at the
7.00      F*       end of the program to delete any records. The records
8.00      F*       are written to the notify object by Commitment Control.
9.00      F*
10.00     FPRDRCTP UF  E          K          DISK
11.00     ICMTID      E DSPRDRCTP
12.00     C          *ENTRY      PLIST
13.00     C          PARM          USER10 10
14.00     C          MOVE USER10  USER
15.00     C*
16.00     C* Check for restart information - get last rcd per user
17.00     C*       PRDRCTP file access path is in LIFO sequence
18.00     C*
19.00     C          USER      CHAINPRDRCTR          20      Not found
20.00     C N20          DO          Restart
21.00     C          EXSR MOVLST          Move to last
22.00     C          SETON          71      Restart
23.00     C          END
24.00     C*
25.00     C* Basic processing loop
26.00     C*
27.00     C          LOOP      TAG
28.00     C          EXFMTPROMPT
29.00     C 98          GOTO END          End of pgm
30.00     C          PRODCR      CHAINPRDMSTR          61      Not found
31.00     C 61          GOTO LOOP
32.00     C          KEY          KLIST
33.00     C          KFLD          PRODCR
34.00     C          KFLD          LOCATN
35.00     C          KEY          CHAINPRDLOCR          62      Not found
36.00     C 62          DO          Release lck
37.00     C          EXCPTRLSMST
38.00     C          GOTO LOOP
39.00     C          END
40.00     C          ADD QTY      ONHAND          Add
41.00     C          ADD QTY      LOCAMT
42.00     C          UPDATPRDMSTR          Update
43.00     C          UPDATPRDLOCR          Update
44.00     C*
45.00     C* Commit and move to previous fields
46.00     C*
47.00     C          CMTID      COMIT
48.00     C          EXSR MOVLST          Move to last
49.00     C          GOTO LOOP
50.00     C*
51.00     C* End of program processing
52.00     C*
53.00     C          END      TAG
54.00     C          SETON          LR
55.00     C*56.00     C* Delete any records in the notify object
57.00     C*

```

58.00	C	DLTLP	TAG		
59.00	C	USER	CHAINPRDRCTR	20	Not found
60.00	C	N20	DO		
61.00	C		DELETPRDRCTR		Delete
62.00	C		GOTO DLTLP		
63.00	C		END		
64.00	C*				
65.00	C*	Move to	-Last Used-	fields for operator feedback	
66.00	C*				
67.00	C	MOVLST	BEGSR		
68.00	C		MOVE PRODC	LSTPRD	
69.00	C		MOVE LOCATN	LSTLOC	
70.00	C		MOVE QTY	LSTQTY	
71.00	C		MOVE DESCRP	LSTDSC	
72.00	C		ENDSR		
73.00	OPRDMSTR	E	RLSMST		

### 関連概念:

93 ページの『例: アプリケーション開始のためのトランザクション・ログ記録ファイルの使用』

この例では、異常終了後にアプリケーションを開始させるためのトランザクション・ログ記録ファイルの使用法について、サンプル・コードおよび説明を提供しています。

55 ページの『通知オブジェクトのコミット』

通知オブジェクトはメッセージ・キュー、データ域、またはデータベース・ファイルで、そのコミットメント定義が正常に終了しなかった場合、特定のコミットメント定義の最後に正常に完了したトランザクションを識別する情報が含まれています。

『例: 全プログラム用の単一通知オブジェクト』

すべてのプログラムに対して 1 つの通知オブジェクトを使用することには、利点があります。その理由は、再開に必要なすべての情報が同じオブジェクトに入っており、すべてのプログラムで通知オブジェクトに対する標準的な方法を用いることができるためです。

### 例: 全プログラム用の単一通知オブジェクト

すべてのプログラムに対して 1 つの通知オブジェクトを使用することには、利点があります。その理由は、再開に必要なすべての情報が同じオブジェクトに入っており、すべてのプログラムで通知オブジェクトに対する標準的な方法を用いることができるためです。

この場合は、ユーザーとプログラムの識別の固有な組み合わせを使用して、プログラムが再開時に正しい情報をアクセスできるようにしてください。

再開に必要な情報はプログラムによって異なることがあるため、コミットメント ID 用の外部記述データ構造は使用しないでください。1 つの通知オブジェクトを使用する場合には、先行するプログラムは、外部ではなくプログラム内でデータ構造を記述することができます。以下にその例を示します。

1	10	USER
11	20	PGMNAM
21	23	PRODC
24	29	LOCATN
30	49	DESC
50	51 0	QTY
52	220	DUMMY

この通知オブジェクトを使用する各プログラムは、コミットメント ID に指定された情報がプログラムに対して固有になります (ユーザー名とプログラム名は固有ではありません)。この通知オブジェクトは、いずれかのプログラムがコミットメント ID に入れることになる最大の情報を収めるのに十分な大きさでなければなりません。

### 関連概念:

99 ページの『例：プログラムごとに固有の通知オブジェクト』

各ジョブに 1 つの固有の通知オブジェクトを使用すると、同じプログラムにユーザーが複数いても、外部記述コミットメント ID を使用することができます。

55 ページの『通知オブジェクトのコミット』

通知オブジェクトはメッセージ・キュー、データ域、またはデータベース・ファイルで、そのコミットメント定義が正常に終了しなかった場合、特定のコミットメント定義の最後に正常に完了したトランザクションを識別する情報が含まれています。

## 例：アプリケーション開始のための標準処理プログラムの使用

標準処理プログラムは、すべてのアプリケーションに対する通知オブジェクトとして 1 つのデータベース・ファイルを使用することによってアプリケーションを再開するための 1 つの方法です。この方法では、標準処理プログラムを使用するすべてのアプリケーションでユーザーごとにユーザー・プロファイル名が固有であることを前提としています。

注：コード例を使用すると、126 ページの『コードに関するライセンス情報および特記事項』の条件に同意したことになります。

この方法では、物理ファイル NFYOBJP が通知オブジェクトとして使用されますが、これは次のように定義されます。

固有のユーザー・プロファイル名  
プログラム識別コード  
再開のための情報

10 文字  
10 文字  
文字フィールド  
(このフィールドには、再開のための情報を必要とする、プログラムを再び開始できるだけの、最大量の情報を入れる十分の大きさが必要です。このフィールドはアプリケーション・プログラムが必要とするものです。例では、その長さを 200 文字と想定しています。)

このファイルは、SHARE(\*YES) として作成されます。ファイルの最初の 2 つのフィールドはファイルのキーです。(このファイルは、RPG プログラムでデータ構造としても定義することができます。)

### 関連概念:

93 ページの『例：アプリケーション開始のためのトランザクション・ログ記録ファイルの使用』

この例では、異常終了後にアプリケーションを開始させるためのトランザクション・ログ記録ファイルの使用法について、サンプル・コードおよび説明を提供しています。

## 例：標準処理プログラムのコード

この例では、すべてのアプリケーションに対する通知オブジェクトとして 1 つのデータベース・ファイルを使用して、アプリケーションを再開するために使用できる、標準処理プログラム用のコードを示します。

以下のコード例に示したアプリケーションは次のことを行います。

1. このアプリケーション・プログラムはパラメーターのユーザー名を受け取り、通知オブジェクトの固有の ID としてこれをプログラム名とともに使用します。
2. アプリケーション・プログラムは、通知オブジェクトにレコードが存在しているかどうかを判断する標準コミット処理プログラムに要求コードの R を渡します。
3. 標準コミット処理プログラムがコード 1 を戻した場合には、レコードが見つかっており、アプリケーション・プログラムは再開に必要な情報をユーザーに示します。
4. アプリケーション・プログラムは、通常の処理に進みます。

5. トランザクションが完了すると、ワークステーション・ユーザーが以前のトランザクションで行われた処置を参照することができるように、値が保管されます。

保管された情報は、通知オブジェクトによっては提供されません。それは、通知オブジェクトが更新されるのは、ジョブまたはシステムの障害が起こった場合だけであるからです。

注: コード例を使用すると、126 ページの『コードに関するライセンス情報および特記事項』の条件に同意したことになります。

### Application program example

SEQNBR \*... .. 1 ... .. 2 ... .. 3 ... .. 4 ... .. 5 ... .. 6 ... .. 7 ..

```

1.00      FPRDMSTP UF  E          K          DISK          KCOMIT
2.00      FPRDLOCP UF  E          K          DISK          KCOMIT
3.00      FPRDRCTD CF  E          WORKSTN
4.00      F*
5.00      F* The following is a compile time array which contains the
6.00      F*   restart information used in the next example
7.00      F*
8.00      E              RTXT   50  50  1              Restart text
9.00      I*
10.00     I* Data structure used for info passed to notify object
11.00     I*
12.00     ICMTID      DS
13.00     I              1  10  USER
14.00     I              11 20  PGMNAM
15.00     I              21 23  PRODCY
16.00     I              24 29  LOCATN
17.00     I              30 49  DESCRP
18.00     I              P  50 510QTY
19.00     I              52 170  DUMMY
20.00     I              171 220  RSTART
21.00     C              *ENTRY  PLIST
22.00     C              PARM      USER10 10
23.00     C*
24.00     C* Initialize fields used to communicate with std program
25.00     C*
26.00     C              MOVE USER10  USER
27.00     C              MOVE 'PRDRCT2' PGMNAM
28.00     C              MOVE 'R'      RQSCOD          Read Rqs
29.00     C              CALL 'STDCMT'
30.00     C              PARM          RQSCOD  1
31.00     C              PARM          RTNCOD  1
32.00     C              PARM          CMTID 220          Data struct
33.00     C              RTNCOD  IFEQ '1'          Restart
34.00     C              EXSR MOVLST          Move to last
35.00     C SETON              71          Restart
36.00     C              END
37.00     C*
38.00     C* Initialize fields used in notify object
39.00     C*
40.00     C              MOVEARTXT,1  RSTART          Move text
41.00     C*
42.00     C* Basic processing loop
43.00     C*
44.00     C              LOOP  TAG
45.00     C              EXFMTPROMPT
46.00     C  98              GOTO END
47.00     C              PRODCY  CHAINPRDMSTR          61  Not found
48.00     C  61              GOTO LOOP
49.00     C              KEY    KLIST
50.00     C              KFLD          PRODCY
51.00     C              KFLD          LOCATN

```

```

SEQNBR *... .. 1 ... .. 2 ... .. 3 ... .. 4 ... .. 5 ... .. 6 ... .. 7 ..
52.00      C          KEY          CHAINPRDLOCR          62      Not found
53.00      C      62          DO
54.00      C          EXCPTRLSMST          Release lck
55.00      C          GOTO LOOP
56.00      C          END
57.00      C          ADD QTY          ONHAND          Add
58.00      C          ADD QTY          LOCAMT
59.00      C          UPDATPRDMSTR          Update
60.00      C          UPDATPRDLOCR          Update
61.00      C*
62.00      C* Commit and move to previous fields
63.00      C*
64.00      C          CMTID          COMIT
65.00      C          EXSR MOVLST          Move to last
66.00      C          GOTO LOOP
67.00      C* End of program processing
68.00      C*
69.00      C          END          TAG
70.00      C          MOVE 'D'          RQSCOD          Dlt Rqs
71.00      C          CALL 'STDCMT'
72.00      C          PARM          RQSCOD
73.00      C          PARM          RTNCOD
74.00      C          PARM          CMTID
75.00      C          SETON          LR
76.00      C*
77.00      C* Move to -Last Used- fields for operator feedback
78.00      C*
79.00      C          MOVLST          BEGSR
80.00      C          MOVE PRODC T          LSTPRD
81.00      C          MOVE LOCATN          LSTLOC
82.00      C          MOVE DESCRP          LSTDSC
83.00      C          MOVE QTY          LSTQTY
84.00      C          ENDSR
85.00      OPRDMSTR E          RLSMST
86.00 ** RTXT          Restart Text
87.00 Inventory Menu - Receipts Option

```

### 処理のフロー:

標準処理プログラムは、再開しなければならぬアプリケーションから呼び出されます。

アプリケーション・プログラムは、次のパラメーター・リストを標準処理プログラムに渡します。

- 要求コード
- 戻りコード
- データ構造名 (通知オブジェクトの内容)

要求コードは、次の操作を実行します。

- R (読み取り)

同じキーで最後に通知オブジェクトに追加されたレコードを検索します。戻りコードは次のように設定されます。

- 0 使用可能なレコードがありません (再開は必要ありません)。
  - 1 再開のためにレコードが情報フィールドに戻されました (再開が必要です)。
- WA (書き込み)



ファイルにレコードを書き出します。このコードは、通知オブジェクトを独自の目的で使用する場合に使用することができます。例えば、プログラムがトランザクションの再開が必要と決定した場合、プログラムはレコードを通知オブジェクトに書き込み、ジョブまたはシステムに障害が起きたときにシステムが取る処置をシミュレートすることができます。

- DE (削除)

通知オブジェクト中の同じキーをもつすべてのレコードを削除します。戻りコードは次のように設定されます。

**0** 削除するレコードが存在していません。

**1** 1 つまたは複数のレコードが削除されました。

- OE (オープン)

**O** 要求コードはオプションで、処理プログラムが呼び出されるたびにそれを開始する必要性をなくすために使用されます。

- CA (クローズ)

オープン要求コードを使用した後で、クローズ要求コードを使用することによってファイルが確実にクローズされます。

- SA (検索)

このユーザーの最後のレコードを戻します。プログラム名は使用されません。このコードは、再開が必要であるかどうかを判断するために初期プログラムで使用することができます。

## 例：標準コミット処理プログラムのコード

標準コミット処理プログラム (STDCMT) は、すべてのアプリケーションで使用される 1 つの通知オブジェクトと連絡するために必要な機能を実行します。

コミットメント制御機能は自動的に項目を通知オブジェクトに書き出しますが、通知オブジェクトを処理するのは、ユーザー作成の標準プログラムでなければなりません。この標準プログラムは、この方法を単純にして標準化します。

このプログラムは、渡されたパラメーターを検査するために作成され、次のように適切な処置を行います。

### **O=オープン**

呼び出しプログラムは、戻り時に通知オブジェクトがオープン状態のままであるように要求します。通知オブジェクトが **RPG** プログラムによって暗黙的にオープンされているので、このプログラムがそれをクローズしないようにしてください。標識 **98** が設定され、プログラムの作業域を保持するために **LR** をオフにしてプログラムが戻り、通知オブジェクトをオープンのままにします。この結果、余分なオーバーヘッドなしにプログラムを再度呼び出すことができます。

### **C=クローズ**

呼び出しプログラムは、通知オブジェクトをそれ以上必要としないと判断し、クローズを要求します。通知オブジェクトの完全なクローズが可能になるように標識 **98** がオフに設定されます。

### **R=読み取り**

呼び出しプログラムは、キー・フィールドに一致するレコードが読み取られ、送り戻されるように要求します。この処理プログラムは渡されたキー・フィールドを使用して、**NFYOBJP** からレコードを検索しようとします。同じキーで重複したレコードが存在する場合には、最後のレコードが戻されます。戻りコードが結果に従って設定され、レコードが存在する場合にはデータ構造 **CMTID** にレコードが送り返されます。

## W=書き込み

呼び出しプログラムは、次回呼び出されるときに再開できるように通知オブジェクトにレコードが書き出されるように要求します。この処理プログラムは、渡されるデータの内容をレコードとして NFYOBJP に書き出します。

## D=削除

呼び出しプログラムは、このキーに一致するレコードが削除されるように要求します。この機能は通常、呼び出しプログラムが正常に完了した場合、再開についての情報を除去するために実行されます。この処理プログラムは、渡されるキー・フィールドに一致するすべてのレコードを削除しようとします。レコードが存在しない場合には、別の戻りコードが戻されます。

## S=検索

呼び出しプログラムは、どのプログラムが書き出したかに関係なく、特定のユーザーのレコードの検索を要求します。この機能は、サインオンで再開が必要であることを示すためにプログラムで使用されます。この処理プログラムはユーザー名だけをキーとして使用し、レコードが存在するかどうかを判断します。戻りコードが結果に従って設定され、またレコードが存在する場合には、このキーの最後のレコードの内容が読み取られ、送り戻されます。

以下の例に、標準コミット処理プログラム STDCMT を示してあります。

## 標準コミット処理プログラム

注: コード例を使用すると、126 ページの『コードに関するライセンス情報および特記事項』の条件に同意したことになります。

SEQNBR \*... .. 1 ... .. 2 ... .. 3 ... .. 4 ... .. 5 ... .. 6 ... .. 7 ..

```
1.00      FNFYOBJP UF  E          K          DISK          A
2.00      ICMTID      DS
3.00      I
4.00      I
5.00      I
6.00      C          *ENTRY  PLIST
7.00      C          PARM      RQSCOD  1
8.00      C          PARM      RTNCOD  1
9.00      C          PARM      CMTID  220
10.00     C          UNQUSR  CABEQ*BLANKS  BADEND      H1 Invalid
11.00     C          UNQPGM  CABEQ*BLANKS  BADEND      H2 Invalid
12.00     C*
13.00     C* '0' for Open
14.00     C*
15.00     C          RQSCOD  IFEQ '0'          Open
16.00     C          SETON          98      End LR
17.00     C          GOTO END
18.00     C          END
19.00     C*
20.00     C* 'C' for Close
21.00     C*
22.00     C          RQSCOD  IFEQ 'C'          Close
23.00     C          SETOF          98
24.00     C          GOTO END
25.00     C          END
26.00     C*
27.00     C* 'R' for Read - Get last record for the key
28.00     C*
29.00     C          RQSCOD  IFEQ 'R'          Read
30.00     C          KEY      KLIST
31.00     C          KFLD          UNQUSR
32.00     C          KFLD          UNQPGM
33.00     C          KEY      CHAINNFYOBJR  51      Not found
34.00     C  51          MOVE '0'      RTNCOD
```

```

35.00 C 51          GOTO END
36.00 C          MOVE '1'      RTNCOB      Found
37.00 C          LOOP1      TAG
38.00 C          KEY       READENFYOBBR      20 EOF
39.00 C 20          GOTO END
40.00 C          GOTO LOOP1
41.00 C          END
42.00 C*
43.00 C* 'W' FOR Write
44.00 C*
45.00 C          RQSCOD      IFEQ 'W'      Write
46.00 C          WRITENFYOBBR
47.00 C          GOTO END
48.00 C          END
49.00 C*
50.00 C* 'D' for Delete - Delete all records for the key
51.00 C*
52.00 C          RQSCOD      IFEQ 'D'      Delete
53.00 C          KEY       CHAINNFYOBBR      51 Not found
54.00 C 51          MOVE '0'      RTNCOB
55.00 C 51          GOTO END
56.00 C          MOVE '1'      RTNCOB      Found
57.00 C          LOOP2      TAG
58.00 C          DELETNFYOBBR
59.00 C          KEY       READENFYOBBR      20 EOF
60.00 C N20         GOTO LOOP2
61.00 C          GOTO END
62.00 C          END
63.00 C*
64.00 C* 'S' for Search for the last record for this user
65.00 C*          (Ignore the -Program- portion of the key)
66.00 C*
67.00 C          RQSCOD      IFEQ 'S'      Search
68.00 C          UNQUSR      SETLLNFYOBBR      20 If equal
69.00 C N20         MOVE '0'      RTNCOB
70.00 C N20         GOTO END
71.00 C          MOVE '1'      RTNCOB      Found
72.00 C          LOOP3      TAG
73.00 C          UNQUSR      READENFYOBBR      20 EOF
74.00 C N20         GOTO LOOP3
75.00 C          GOTO END
76.00 C          END
77.00 C*
78.00 C* Invalid request code processing
79.00 C*
80.00 C          SETON          H2      Bad RQS code
81.00 C          GOTO BADEND
82.00 C*
83.00 C* End of program processing
84.00 C*
85.00 C          END          TAG
86.00 C N98          SETON          LR
87.00 C          RETRN
88.00 C* BADEND tag is used then fall thru to RPG cycle error return
89.00 C          BADEND      TAG

```

#### 関連概念:

『例: アプリケーションを再始動するかどうかを決定するための標準処理プログラムの使用』  
 初期プログラムは、標準コミット処理プログラムを呼び出して、再開が必要かどうかを判断することができます。その結果、ワークステーション・ユーザーは再開を行うかどうかを決定することができます。

#### 例: アプリケーションを再始動するかどうかを決定するための標準処理プログラムの使用

初期プログラムは、標準コミット処理プログラムを呼び出して、再開が必要かどうかを判断することができます。その結果、ワークステーション・ユーザーは再開を行うかどうかを決定することができます。

初期プログラムは要求コードの S (検索) を標準プログラムに渡します。レコードが存在する場合には、再開に関する情報が初期プログラムに渡され、その情報がワークステーション・ユーザーに表示されます。

通知オブジェクト中のコミットメント ID には、初期プログラムが表示でき、再開にプログラムが必要とする処置を識別する情報が入っています。例えば、コミットメント ID の終わりの 50 文字を、この情報を収めるために確保することができます。アプリケーション・プログラムでは、この情報をコンパイル時配列に入れ、初期設定ステップでデータ構造に移動することができます。例：標準コミット処理プログラムのコードには、これをアプリケーション・プログラムに組み込む方法が示されています。

以下の例は、レコードが通知オブジェクトに存在しているかどうかを判断する初期プログラムを示しています。

### 例: 初期プログラム

注: コード例を使用すると、126 ページの『コードに関するライセンス情報および特記事項』の条件に同意したことになります。

```
SEQNBR *... .. 1 ... .. 2 ... .. 3 ... .. 4 ... .. 5 ... .. 6 ... .. 7

1.00          PGM
2.00          DCLF          CMTINLD
3.00          DCL          &RQSCOD *CHAR LEN(1) VALUE(S) /* Search */
4.00          DCL          &RTNCOD *CHAR LEN(1)
5.00          DCL          &CMTID *CHAR LEN(220)
6.00          DCL          &USER *CHAR LEN(10)
7.00          DCL          &INFO *CHAR LEN(50)
8.00          RTVJOB      USER(&USER)
9.00          CHGVAR      &CMTID (&USER *CAT XX)
10.00         /* The XX is required to prevent a blank Pgm nam */
11.00          CALL      STDCMT PARM(&RQSCOD &RTNCOD &CMTID)
12.00          IF        (&RTNCOD *EQ '1') DO /* RESTART REQD */
13.00          CHGVAR      &INFO %SST(&CMTID 171 50)
14.00          SNDRCVF      RCDfmt(RESTART)
15.00          ENDDO
16.00         /*
17.00         /* Enter normal initial program statements */
18.00         /* or -TFRCTL- to first menu program */
19.00         /*
20.00          ENDPGM
```

#### 関連概念:

108 ページの『例：標準コミット処理プログラムのコード』

標準コミット処理プログラム (STDCMT) は、すべてのアプリケーションで使用される 1 つの通知オブジェクトと連絡するために必要な機能を実行します。

55 ページの『通知オブジェクトのコミット』

通知オブジェクトはメッセージ・キュー、データ域、またはデータベース・ファイルで、そのコミットメント定義が正常に終了しなかった場合、特定のコミットメント定義の最後に正常に完了したトランザクションを識別する情報が含まれています。

---

## トランザクションおよびコミットメント制御のトラブルシューティング

この情報は、コミットメント制御エラーのトラブルシューティング、デッドロックの検出、通信障害後のトランザクションのリカバリー、コミット操作とロールバック操作の強制時点、および再同期の取り消し時点と長時間を要するロールバックの終了時点の理解に役立つことがあります。

## コミットメント制御エラー

コミットメント制御を使用するときには、どの状態がエラーを起こし、またどの状態がエラーを起こさないかを理解することが必要です。

通常、エラーが起きるのは、例えばコミットメント定義を使用するファイルがまだオープンされているときにコミットメント制御の終了 (ENDCMTCTL) コマンドを実行するなど、コミットメント制御機能を矛盾して使用したときです。

### コミット処理中のエラー

コミット操作中に通信またはシステム障害が生じる場合、再同期を実行して、トランザクションに参加しているすべてのシステムで一貫性のあるデータをトランザクション・マネージャーが保持していることを確認する必要があります。再同期の作業およびそれがコミット操作に与える影響は、これらの要因によって決まります。

- 結果の待機コミットメント・オプション。
- トランザクションの状態。

障害が、完全に修復不能か、または直ちに修復できないような壊滅的なものであるとき、トランザクションに参加している他のシステムのシステム・オペレーターはヒューリスティック判定をしなければなりません。ヒューリスティック判定は、トランザクション中にそのシステム上に加えられた変更をコミットまたはロールバックします。そのような決定の後に障害が修復され、その決定を行ったためにデータ保全性に問題が生じたことが再同期によりわかった場合、CPD83D9 メッセージまたは CPD83E9 メッセージが QSYSOPR メッセージ待ち行列に送られます。

#### 関連概念:

37 ページの『2 フェーズ・コミットのコミットメント定義：結果を待機しない』

コミット操作中に通信またはシステム障害が生じて再同期が修復される場合、デフォルトの方式では、再同期が終了してコミット操作が完了するまで待機しています。

32 ページの『2 フェーズ・コミットメント制御のトランザクションの状態』

コミットメント定義が、トランザクション・プログラム・ネットワークの一部である各ロケーションに確立されます。各コミットメント定義に対し、システムは、現行のトランザクションと前のトランザクションの状態を常に監視しています。

### エラー状態

エラーが生じた場合、エスケープ・メッセージが出されますが、これはプログラムでモニターすることができます。

コミットメント制御に関連したいくつかの典型的なエラーを次に示します。

- ENDCMTCTL コマンドを実行する前に、STRCMTCTL コマンドが連続して実行される。
- コミットメント制御下でファイルがオープンされる時に、STRCMTCTL コマンドが実行されていない。

ジョブ・レベル・コミットメント定義を使用する活動化グループで稼働するプログラムの場合、これはエラー状態ではありません。ジョブ・レベルのコミットメント定義は単一のプログラムでしか開始できませんが、あるプログラムがそのジョブ・レベルのコミットメント定義を開始すると、活動化グループ・レベルのコミットメント定義を使用しない任意の活動化グループで実行される任意のプログラムによって使用されます。活動化グループ内で稼働し、活動化グループ・レベルのコミットメント定義を使用するプログラムは、まず STRCMTCTL コマンドを使用して活動化グループ・レベルのコミットメント定義を開始しなければなりません。

- コミットメント制御下で出力用にオープンされたファイルがジャーナル処理されない。



- 共用ファイルの最初のオープン操作ではファイルをコミットメント制御下に置くが、同じ共用ファイルのそれ以降のオープンの操作ではそうしない。
- 共用ファイルの最初のオープン操作ではファイルをコミットメント制御下に置かないが、同じ共用ファイルのそれ以降のオープン操作ではそうする。
- 単一トランザクションにおけるジョブのレコード・ロック限度に達した。
- プログラムが同じレコードに対して読み取り操作、コミット操作、および変更を出す。コミット操作がレコードのロックを解放しているため、コミット操作の後で再び読み取り操作を出さなければなりません。
- フェーズ・ロケーションの場合、コミットメント制御下に置かれたリソースが常駐するロケーションが、既にコミットメント制御下に置かれているコミットメント定義のリソースのロケーションと異なっている。
- ENDCMTCTL コマンドが出されたときに、コミットされていないレコードが存在している。

これがエラー状態でないのは、すべてのファイルがクローズされ、リモート・データベースが切り離され、終了されるコミットメント定義とまだ関連付けられている API コミットメント・リソースがない場合です。

- コミット、ロールバック、または ENDCMTCTL コマンドが実行される前に、STRCMTCTL コマンドが実行されなかった。

活動化グループ内で稼働するプログラムの場合にはこれはエラー状態ではなく、ジョブ・レベルのコミットメント定義がアクティブです。ジョブ・レベルのコミットメント定義は単一のプログラムでしか開始できませんが、あるプログラムがそのジョブ・レベルのコミットメント定義を開始すると、活動化グループ・レベルのコミットメント定義を使用しない任意の活動化グループで実行される任意のプログラムによって使用されます。活動化グループ内で稼働し、活動化グループ・レベルのコミットメント定義を使用するプログラムは、まず STRCMTCTL コマンドを使用して活動化グループ・レベルのコミットメント定義を開始しなければなりません。

- コミットメント定義のコミットメント制御下で引き続きオープンされたままのファイルが存在している時に ENDCMTCTL コマンドが実行される。
- 保管操作を行っているジョブが、コミットメント境界にない 1 つまたは複数のコミットメント定義をもっている。
- コミット可能リソースを持った他のジョブが SAVACTWAIT パラメーターで指定された時間内にコミットメント境界に達しなかったため、活動時保管操作が終了した。
- API コミット可能リソースが単一のジョブの 1 つまたは複数のコミットメント定義に追加されたため、活動時保管処理を続行することができなかった。
- 単一のジョブに 1023 を超えるコミットメント定義がある。
- リモート・ロケーションとの会話が、リソースの障害のため消失した。これにより、トランザクションがロールバックされます。
- 更新のためオープンされた 1 フェーズ・リソースが、コミット操作を開始したのではないノードにある。リソースまたはコミット要求を開始したノードを除去する必要があります。
- トランザクションがロールバック必須 (RBR) 状態のときにコミット操作が要求された。ロールバック操作を実行しなければなりません。
- API 出口プログラムがコミット要求またはロールバック要求を出した。
- トリガー・プログラムが呼び出されたときのコミットメント定義に対して、トリガー・プログラムがコミット要求またはロールバック要求を出した。

トリガー・プログラムは個別のコミットメント定義を開始し、そのコミットメント定義に対してコミット要求またはロールバック要求を出すことができます。

## 非エラー状態

コミットメント制御でエラーが起こらない状況のいくつかを次に示します。

- コミットまたはロールバック操作が実行され、リソースがコミットメント制御下でない。このことにより、リソースがコミットメント制御下にあるかどうかを考慮せずに、コミットまたはロールバック操作をプログラムに入れることができます。また、コミット可能な変更を行う前にコミット ID を指定することもできます。
- コミットまたはロールバック操作が実行されたが、コミットされていないリソース変更が存在していない。この場合、未コミット・リソース変更の有無を考えないでプログラムにコミットおよびロールバック操作を含めることができます。
- コミットメント制御下にあるファイルがクローズされ、コミットされていないレコードが存在している。この状況では、別のプログラムを呼び出してコミットまたはロールバック操作を実行することができます。これはファイルが共用されているかどうかにかかわらず起こります。この機能により、サブプログラムは複数のプログラムを含むトランザクションの一部となるデータベースの変更を行うことができます。
- 1 つまたは複数のコミットメント定義の未コミット変更をもったまま、ジョブが正常終了または異常終了した。すべてのコミットメント定義の変更は、ロールバックされます。
- 活動化グループ・レベルのコミットメント定義の保留中の変更を持ったまま活動化グループが終了した。活動化グループが正常に終了し、かつコミットメント制御の下でオープンされ、終了中のその活動化グループまでの範囲をもつファイルをクローズしたときにエラーが起きなかった場合には、システムによって暗黙のコミットが実行されます。それ以外の場合は、暗黙でロールバックが実行されます。
- プログラムがコミットされていない変更済みレコードに再アクセスする。このことにより、プログラムは次のことを行うことができます。
  - コミット操作を指定する前にレコードの追加および更新を行う。
  - コミット操作を指定する前に同じレコードを 2 回更新する。
  - コミット操作を指定する前にレコードの追加および削除を行う。
  - コミットされていないレコードに、コミットメント制御にある別の論理ファイルで再アクセスする。
- STRCMTCTL コマンドに LCKLVL (\*CHG または \*CS) を指定し、コミット操作を使用して読み取り専用ファイルを開く。この場合には、要求時にロックは起こりません。このファイルはコミットメント制御が有効でないかのように扱われますが、コミットメント制御下にあるファイルの WRKJOB メニュー・オプションには表示されます。
- STRCMTCTL コマンドを出した後で、コミットメント制御の下でいずれのファイルもオープンしない。この場合、ファイルに対するいずれのレコード・レベルの変更もコミットメント制御下では行われません。

## コミットメント制御中にモニターすべきエラー・メッセージ

メッセージのタイプおよびエラーが発生した時点によっては、いくつかの異なるエラー・メッセージがコミットまたはロールバック操作によって戻されるか、またはジョブ・ログに送られます。

エラー・メッセージは、以下の処理時に出される可能性があります。

- 通常のコミットまたはロールバック処理
- ジョブ処理の終了時のコミットまたはロールバック処理
- 活動化グループの終了時のコミットまたはロールバック処理

活動化グループの終了またはジョブ処理の終了時には、以下のどのメッセージもモニターすることはできません。また、CPFxxxx メッセージのみがモニター可能です。CPDxxxx メッセージは常に診断メッセージとして送信されますが、モニターすることはできません。活動化グループの終了時の活動化グループ・レベルのコミットメント定義を終了するときや、ジョブの終了時のコミットメント定義を終了するとき起きたエラーは、診断メッセージとしてジョブ・ログに残ります。

コミットメント制御に関連した、探すべきメッセージは以下のとおりです。

**CPD8351**

変更がコミットされていない可能性がある。

**CPD8352**

リモート・ロケーション &3 で変更がコミットされていない。

**CPD8353**

リレーショナル・データベース &1 の変更がコミットされていない。

**CPD8354**

DDM ファイル &1 に対する変更がコミットされていない可能性がある。

**CPD8355**

DDL オブジェクト &1 に対する変更がコミットされていない可能性がある。

**CPD8356**

ロールバックされた変更はコミットされている可能性がある。

**CPD8358**

リレーショナル・データベース &1 の変更がロールバックされていない可能性がある。

**CPD8359**

DDM ファイル &1 に対する変更がロールバックされていない可能性がある。

**CPD835A**

DDL オブジェクト &3 に対する変更がロールバックされていない可能性がある。

**CPD835C**

&2 の通知オブジェクト &1 は更新されない。

**CPD835D**

DRDA リソースは、SQL カーソル保留を使用できない。

**CPF835F**

コミットまたはロールバック操作が正常に実行されなかった。

**CPD8360**

メンバー・ファイル、またはその両方が既に割り振り解除されている。

**CPD8361**

コミット時に API 出口プログラム &1 が正常に実行されなかった。

**CPD8362**

ロールバックの処理時に API 出口プログラム &1 が正常に実行されなかった。

**CPD8363**

コミット処理時に API 出口プログラム &1 が &4 分後に終了した。

**CPD8364**

ロールバック処理時に API 出口プログラム &1 が &4 分後に終了した。

**CPD836F**

コミットメント制御操作中にプロトコル・エラーが起こった。

**CPD83D1**

API リソース &4 は最終エージェントでない。

**CPD83D2**

リソースがコミットメント制御に対応していない。

**CPD83D7**

コミット操作がロールバックに変更された。

**CPD83D9**

ヒューリスティック混合条件が発生した。

**CPF83DB**

コミット操作の結果ロールバックとなった。

**CPD83DC**

コミット操作またはロールバック操作を決定するために「問題発生時の処置」が使用された。理由コードは &2 です。

**CPD83DD**

会話が打ち切られた。理由コードは &1 です。

**CPD83DE**

戻り情報が正しくない。

**CPD83EC**

API 出口プログラム &1 がロールバックをポートした。

**CPD83EF**

次の作業論理単位に対してロールバック操作が開始された。

**CPF8350**

コミットメント定義が見つからない。

**CPF8355**

ENDCMTCTL を使用することはできない。保留中の変更がアクティブです。

**CPF8356**

&1 ローカル変更がコミットされないでコミットメント制御が終了した。

**CPF8358**

&2 の通知オブジェクト &1 は更新されない。

**CPF8359**

ロールバック操作が正常に実行されなかった。

**CPF835A**

コミットメント定義 &1 の終了が取り消された。

**CPF835B**

コミットメント制御の終了時にエラーが起こった。

**CPF835C**

リモートの変更をコミットしないでコミットメント制御が終了した。

**CPF8363**

コミット操作が正常に実行されなかった。

**CPF8364**

コミットメント制御パラメーター値が正しくない。理由コードは &3 です。

**CPF8367**

コミットメント制御操作を実行できない。

**CPF8369**

API コミットメント・リソースをコミットメント制御下に入れることはできない。理由コードは &1 です。

**CPF83D0**

コミットメント操作は使用できない。

**CPF83D2**

コミット完了 = 進行中の再同期が戻された。

**CPF83D3**

コミット完了 = ヒューリスティック混合が戻された。

**CPF83D4**

作業論理単位ジャーナル項目が送られていない。

**CPF83E1**

制約違反のためにコミット操作が正常に実行されなかった。

**CPF83E2**

ロールバック操作が必要である。

**CPF83E3**

要求されたネスト・レベルがアクティブになっていない。

**CPF83E4**

コミットメント制御が終了したが、リソースがコミットされていない。

**CPF83E6**

コミットメント制御操作が進行中の再同期化とともに完了した。

**CPF83E7**

X/Open グローバル・トランザクションのコミットまたはロールバックは許されない。

**CALL コマンド後のエラーのモニター**

コミットメント制御を使用するプログラムを呼び出すときには、予期しないエラーをモニターして、エラーが起こった場合にはロールバック操作を実行します。

例えば、RPG で 0 で割るなどの予期しないエラーをプログラムが検出したときに、コミットされていないレコードが存在している場合があります。

ジョブの照会メッセージ応答 (INQMSGRPY) パラメーターの状態に応じて、プログラムは照会メッセージを送るか、またはデフォルトの処置を行います。オペレーターの応答またはデフォルトの処置でプログラムが終了しても、コミットされていないレコードはまだ存在し、コミットまたはロールバック操作を待ちます。

別のプログラムが呼び出されてコミット操作が行われた場合は、以前のプログラムで部分的に完了したトランザクションがコミットされます。



部分的に完了したトランザクションがコミットされないようにするには、CALL コマンドの後に出されるエスケープ・メッセージをモニターしてください。例えば、RPG プログラムの場合には、次のコーディングを使用してください。

```
CALL RPGA
MONMSG MSGID(RPG9001)
EXEC(ROLLBACK) /*Rollback if pgm is canceled*/
```

COBOL プログラムの場合には、次のコーディングを使用してください。

```
CALL COBOLA
MONMSG MSGID(CBE9001)
EXEC(ROLLBACK) /*Rollback if pgm is canceled*/
```

## 通常のコミットまたはロールバック処理の障害

コミットまたはロールバック処理時にはいつでもエラーが発生する可能性があります。

次の表では、この処理を 4 つの状況に分けています。中央の欄には、それぞれの状況でエラーになった時のシステム処置を示します。3 番目の欄では、メッセージに対する応答としてユーザーまたはアプリケーションが行うべきことを示しています。これらのものは、システムがコミットメント制御を処理する方法と一致しています。

状況	コミットまたはロールバック処理	推奨する処置
レコード・レベル入出力コミットに障害が起きた。	<ul style="list-style-type: none"> <li>作成ウェーブ時にエラーが起きた場合、トランザクションはロールバックされ、メッセージ CPF83DB が送られます。</li> <li>コミットされたウェーブ時にエラーが起きた場合、コミット処理は続行され、できるだけ多くの残りのリソースをコミットしようとしています。メッセージ CPF8363 がコミット処理の終了時に送られます。</li> </ul>	メッセージをモニターし、必要に応じて処理する。
コミット時に API コミットメント・リソースに対するオブジェクト・レベルまたはコミットおよびロールバック出口プログラムに障害が起きた。	<ul style="list-style-type: none"> <li>作成ウェーブ時にエラーが起きた場合、トランザクションはロールバックされ、メッセージ CPF83DB が送られます。</li> <li>コミットされたウェーブ時にエラーが起きた場合、処理は続行され、できるだけ多くの残りのリソースをコミットまたはロールバックしようとしています。コミットメント・リソース・タイプに応じて、次のメッセージのいずれかが戻されます。 <ul style="list-style-type: none"> <li>- CPD8353</li> <li>- CPD8354</li> <li>- CPD8355</li> <li>- CPD8361</li> </ul> </li> </ul> <p>メッセージ CPF8363 がコミット処理の終了時に送られます。</p>	メッセージをモニターし、必要に応じて処理する。

状況	コミットまたはロールバック処理	推奨する処置
レコード・レベル入出力ロールバックに障害が起きた。	<ol style="list-style-type: none"> <li>1. CPD8356 を戻す。</li> <li>2. オブジェクト・レベルまたは API コミットメント・リソースのロールバック処理を引き続き試みる。</li> <li>3. 処理の終わりに CPF8359 を戻す。</li> </ol>	メッセージをモニターし、必要に応じて処理する。
ロールバック時に API コミットメント・リソースに対するオブジェクト・レベルまたはコミットおよびロールバック出口プログラムに障害が起きた。	<ol style="list-style-type: none"> <li>1. コミットメント・リソース・タイプにより、以下のメッセージのいずれかを戻す。 <ul style="list-style-type: none"> <li>• CPD8358</li> <li>• CPD8359</li> <li>• CPD835A</li> <li>• CPD8362</li> </ul> </li> <li>2. 処理を続行する。</li> <li>3. 処理の終わりに CPF8359 を戻す。</li> </ol>	メッセージをモニターし、必要に応じて処理する。

### ジョブの終了時のコミットまたはロールバック処理

前の表に説明されている状態はすべて、次のメッセージが出された場合を除くジョブの終了時にも適用されます。

- ローカル・リソースだけが登録されている場合には、CPF8356。
- リモート・リソースだけが登録されている場合には、CPF835C。
- ローカル・リソースおよびリモート・リソースの両方が登録されている場合には、CPF83E4。

さらに、API コミット可能リソース用にコミットまたはロールバック出口プログラムが呼び出された場合には、ジョブ完了固有の 2 つのメッセージのうちいずれかが表示されます。コミットおよびロールバック出口プログラムが 5 分以内に完了しない場合には、そのプログラムは取り消され、診断メッセージ CPD8363 (コミットの場合) または CPD8364 (ロールバックの場合) が送られて、そのコミットまたはロールバック処理の残りが続行されます。

### 初期プログラム・ロード (IPL) 時のコミットまたはロールバック処理

前の表に説明のあるすべての状況は、メッセージ CPF8359 または CPF8363 ではなくメッセージ CPF835F が送られることを除き、コミットメント定義の IPL リカバリーのときにも当てはまります。特定のコミットメント定義に対して送られるメッセージは、QDBSRVxx ジョブのうちの 1 つのジョブ・ログまたは QHST ログに現れることもあります。QHST ログのメッセージ CPI8356 は、特定のコミットメント定義の IPL リカバリーの開始点を示します。メッセージ CPC8351 は、特定のコミットメント定義の IPL リカバリーの終わりを示し、そのコミットメント定義のリカバリーに関するその他のメッセージは、この 2 つのメッセージの間に現れます。

API コミット可能リソースのコミットまたはロールバック出口プログラムが呼び出された場合には、コミットメント定義固有のこの 2 つのメッセージのうちいずれかが表示されます。コミットおよびロールバック出口プログラムが 5 分以内に完了しない場合には、そのプログラムは取り消され、診断メッセージ CPD8363 (コミットの場合) または CPD8364 (ロールバックの場合) が送られて、そのコミットまたはロールバック処理の残りが続行されます。

## デッドロックの検出

あるジョブがあるオブジェクト (オブジェクト A) のロックを保持しながら他のオブジェクト (オブジェクト B) のロックの獲得を待っており、同時に他のジョブまたはトランザクションがオブジェクト B のロックを保持しながらオブジェクト A のロックの獲得を待っている場合、デッドロック状態が発生することがあります。

デッドロック状態が発生しているかどうかを見付けるために次のステップを実施し、もしあればそれを修正します。

1. アクティブ・ジョブのリストの中で中断されているジョブを見付けます。
2. ジョブがロックを待っているオブジェクトを調べます。
3. ジョブがロックを待機しているオブジェクトのすべてについて、ロック・ホルダー (トランザクションまたはジョブ) のリストを調べて、中断されているジョブによって要求されたレベルに対応した競合ロックを見付けようとしています。
4. トランザクションが競合ロックを保持している場合、トランザクションに関連付けられたジョブの表示のステップに従って、それらのジョブのいずれかがロックを待機しているかどうかを調べます。
5. この待機しているジョブが、最初に中断されたジョブによりロックされているオブジェクトのいずれかをロックしようとしているかどうかを判別します。最初に中断されたジョブによりロックされているオブジェクトのいずれかをロックしようとしているジョブが見つければ、問題のオブジェクトを障害スポットとして識別することができます。
6. 適切な方針を決定するために、トランザクションを調査します。
  - a. トランザクションのプロパティを調べて、それを開始したアプリケーションを見付け、次にそのアプリケーション・コードを調べます。
  - b. あるいは、トランザクションのプロパティにあるコミット・サイクル ID を見付けてから、ジャーナル内でこの ID を含んでいる項目を探すことにより、この時点までのトランザクションの処置をトレースします。このためには、ジャーナル項目の検索 (RTVJRNE) コマンドを使用し、CMTCYCID パラメーターを指定します。
  - c. 関連情報を入力後、ロールバックまたはコミット操作の強制を選択できます。

### 関連タスク:

76 ページの『ロックの最小化』

レコード・ロックを最小限に抑える一般的な方法は、レコード・ロックを解除することです。(LCKLVL(\*ALL) を指定している場合には、この手法はうまくいきません。)

### ジョブの状況の判別

72 ページの『トランザクションのロックされたオブジェクトの表示』

トランザクション範囲ロックのみを持つグローバル・トランザクションについて、ロックされたオブジェクトを表示することができます。

72 ページの『トランザクションに関連付けられたジョブの表示』

トランザクションに関連付けられたジョブを表示するには、次の手順に従います。

71 ページの『コミットメント制御情報の表示』

IBM Navigator for i は、システム上のすべてのトランザクション (作業論理単位) に関する情報を表示することができます。IBM Navigator for i は、トランザクションに関連付けられたジョブの情報がある場合は、それも表示することができます。

122 ページの『コミットとロールバック操作の強制時点および再同期の取り消し時点』

コミットまたはロールバック操作の強制の決定は、ヒューリスティック判定 と呼ばれます。この処置により、オペレーターは、準備済み状態にあるトランザクションのリソースを手動でコミットまたはロールバック

クできるようになります。

#### 関連資料:

ジャーナル項目の検索 (RTVJRNE) コマンド

#### 関連情報:

124 ページの『大規模トランザクションまたは古いトランザクションを検出する』  
大規模トランザクションまたは古いトランザクションを検出するには、**コミットメント定義の処理 (WRKCMTDFN)** コマンドを使用します。

## 通信障害後のトランザクションのリカバリー

この手順を使用すると、リモート・システムとの通信の障害発生後に、そのシステム上で処理を実行するトランザクションを取り扱う際に役立ちます。

通信障害の場合、システムは通常、すべてのリモート・システムとの再同期を自動的に完了させます。ただし、障害が破滅的でリモート・システムへの通信が再確立されないような場合 (例えば、通信回線が切断された場合)、ユーザー自身で再同期を取り消して、トランザクションを復元する必要があります。トランザクションが、ロックされている場合もありますが、その時はロックを解除します。

1. IBM Navigator for i で、処理中のトランザクションのコミットメント制御情報を表示します。
2. リモート・システムと再同期しようとしている問題のトランザクションを見付けます。該当のトランザクションの「**進行中の再同期**」フィールドは、「はい」に設定されています。
3. 個々のトランザクションのリソース状況を検査することにより、リモート・システムに接続されていたトランザクションを探します。
4. トランザクションを識別後、トランザクションの状態に応じて、コミットの強制またはロールバックの強制を行う必要があります。
5. トランザクションのプロパティを調査してから、コミットするかロールバックするかを決定することができます。
  - **作業単位 ID** を使用して、他のシステム上にあるそのトランザクションの残りの部分を見付けることができます。
  - トランザクションの状態から、コミットするかロールバックするかを決定することもできます。例えば、データベース・トランザクションが通信障害時に 2 フェーズ・コミットを実行していて、障害後のその状態が「準備済み」または「最終エージェントの保留」であれば、そのトランザクションに対して強制コミットを選択することができます。
6. 未確定のトランザクションに対してコミットまたはロールバックを強制した後で、その特定されたトランザクションについて、障害が起きた接続の再同期を停止します。

#### 関連タスク:

71 ページの『コミットメント制御情報の表示』

IBM Navigator for i は、システム上のすべてのトランザクション (作業論理単位) に関する情報を表示することができます。IBM Navigator for i は、トランザクションに関連付けられたジョブの情報がある場合は、それを表示することができます。

122 ページの『コミットとロールバック操作の強制時点および再同期の取り消し時点』

コミットまたはロールバック操作の強制の決定は、**ヒューリスティック判定** と呼ばれます。この処置により、オペレーターは、準備済み状態にあるトランザクションのリソースを手動でコミットまたはロールバックできるようになります。

## コミットとロールバック操作の強制時点および再同期の取り消し時点

コミットまたはロールバック操作の強制の決定は、**ヒューリスティック判定** と呼ばれます。この処置により、オペレーターは、準備済み状態にあるトランザクションのリソースを手動でコミットまたはロールバックできるようになります。

ヒューリスティック判定を行って、その決定がトランザクション内の他のロケーションの結果と矛盾している場合、トランザクションの状態はヒューリスティック混合になります。トランザクションに参加している他のすべてのロケーションが行った処置を判別し、データベース・レコードを再同期化するのには、ユーザーの責任です。

ヒューリスティック判定を行う前に、トランザクションに関する情報をできるだけ多く集めてください。コミットメント定義に関連しているジョブを表示し、関係しているジャーナルおよびファイルの記録を作成してください。その後、ジャーナル項目を表示してジャーナル処理された変更を手動で適用または除去するときに、この情報を使用することができます。

トランザクションに関する情報を探する場合、そのトランザクションのイニシエーターであったロケーションを探すのが最善です。しかし、コミットまたはロールバックの決定権が、API リソースまたは最後のエージェントにある場合があります。

API リソースが最後のエージェント・リソースとして登録された場合、コミットまたはロールバックに関する最終的な決定権は API リソースにあります。コミットまたはロールバックするかを判別するには、アプリケーションに関する情報や、アプリケーションがどのように API リソースを使用しているかを調べる必要があります。

トランザクションに最終エージェントが選択されている場合、最終エージェントにはコミットするかまたはロールバックするか決定権があります。トランザクションに関する情報は、最後のエージェントの状況を調べてください。

修復できないシステムまたは通信の障害のために、ヒューリスティック判定をするか再同期を取り消さなければならぬときは、以下のステップを使用して疑わしいすべてのトランザクションを検索できます。

1. IBM Navigator for i で、処理対象のシステムを展開します。
2. 「データベース」およびシステム用のローカル・データベースを展開します。
3. 「トランザクション」を展開します。
4. 「データベース・トランザクション」または「グローバル・トランザクション」を展開します。

この表示ウィンドウには、トランザクションごとのコミットメント定義、再同期状況、現行作業単位 ID、および現行作業単位状態があります。以下の情報をもつトランザクションを探します。

- **論理作業単位状態**が「準備済み」または「最終エージェントの保留」のトランザクション。
- **進行中の再同期**の状況が「はい」を示すトランザクション。

このシステムのトランザクションに参加しているジョブを処理するには、トランザクションを右クリックしてから「ジョブ」を選択します。

トランザクションを右クリックした時に、「**強制コミット (Force Commit)**」、「**強制ロールバック (Force Rollback)**」、または「**再同期取り消し (Cancel Resynchronization)**」を選択することもできます。

ヒューリスティック判定や再同期化取り消しを行う前に、トランザクションに関連した他のシステムのジョブの状況をチェックできます。リモート・システム上のジョブをチェックすると、システム間のデータベースの矛盾が起こらないような決定を行うのに役立ちます。



1. 処理対象のトランザクションを右クリックします。
2. 「リソース状況 (Resource Status)」を選択します。
3. 「リソース状況 (Resource Status)」ダイアログで、SNA 接続の場合は「会話 (Conversation)」タブを、TCP/IP 接続の場合は「接続」を選択します。

各会話リソースは、トランザクションに参加しているリモート・システムを表しています。リモート・システムでは、IBM Navigator for i を使用して、そのトランザクションに関連付けられたトランザクションを表示することができます。

作業単位 ID の基本部分はすべてのシステムで同じです。リモート・システムでコミットメント制御情報の表示を行う時に、作業単位 ID の基本部分がローカル・システムのものと同じものを探します。

例えば、ローカル・システムの作業単位 ID が APPN.RCHASL97.X'112233445566 で始まる場合は、リモート・システムについて、同様に APPN.RCHASL97.X'112233445566 で始まる作業単位 ID を探します。

#### 関連概念:

46 ページの『コミットメント制御の XA トランザクション・サポート』

DB2 for i は、X/Open グローバル・トランザクションに参加することができます。

53 ページの『コミットメント制御の開始』

コミットメント制御を開始するには、コミットメント制御の開始 (STRCMTCTL) コマンドを使用します。

#### 関連タスク:

120 ページの『デッドロックの検出』

あるジョブがあるオブジェクト (オブジェクト A) のロックを保持しながら他のオブジェクト (オブジェクト B) のロックの獲得を待っており、同時に他のジョブまたはトランザクションがオブジェクト B のロックを保持しながらオブジェクト A のロックの獲得を待っている場合、デッドロック状態が発生することがあります。

121 ページの『通信障害後のトランザクションのリカバリー』

この手順を使用すると、リモート・システムとの通信の障害発生後に、そのシステム上で処理を実行するトランザクションを取り扱う際に役立ちます。

71 ページの『コミットメント制御情報の表示』

IBM Navigator for i は、システム上のすべてのトランザクション (作業論理単位) に関する情報を表示することができます。IBM Navigator for i は、トランザクションに関連付けられたジョブの情報がある場合は、それも表示することができます。

## 長時間を要するロールバックの終了

貴重なプロセッサ時間を消費したり、リソースをロックしたり、ストレージ・スペースを占有したりする、長時間を要するロールバックを終了することができます。

ロールバック操作は、以前のコミット操作またはロールバック操作以降にトランザクション内で行ったすべての変更を除去します。ロールバック操作時に、システムはトランザクションに関連したロックも解除します。システムに多数のトランザクションが含まれる場合、システムがロールバック操作を完了するのに数時間かかることがあります。このような長時間を要するロールバックは、貴重なプロセッサ時間を消費したり、リソースをロックしたり、ストレージ・スペースを占有したりすることがあります。

長時間を要するロールバックを終了する前に、どのコミットメント定義がロールバックしているか、およびそれらのコミットメント定義の現在の状態を知っておく必要があります。ロールバックしているコミットメント定義の「状態」フィールドは ROLLBACK IN PROGRESS に設定されます。

以下のステップに従って、コミットメント定義の処理 (WRKCMTDFN) コマンドを使用して、ロールバックの状態を調べます。

1. 文字ベース・インターフェースから WRKCMTDFN JOB(\*ALL) を入力します。
2. F11 を入力して「状態」フィールドを表示します。

長時間を要するロールバックを終了すると、トランザクション中に変更されたファイルは部分的なトランザクションを持ったままになります。ファイルが部分的なトランザクションを持つことができない場合にはロールバックを終了しないでください。トランザクション中にどのファイルが変更されたかを見るには、オプション 5 を選択して WRKCMTDFN リストから状況を表示します。F6 を押してリソース状況を表示し、レコード・レベルを選択します。

長時間を要するロールバックを終了するには、全オブジェクト (\*ALLOBJ) 特殊権限が必要です。長時間を要するロールバックを終了するには、以下のステップを行います。

1. 文字ベース・インターフェースから WRKCMTDFN JOB(\*ALL) を入力します。
2. 終了させたいコミットメント定義にオプション 20 (ロールバックの終了) を入力します。

部分的なトランザクションを持つファイルは、ファイル記述の表示 (DSPFD) コマンドからの出力で「部分的なトランザクションが存在する、ロールバック終了」フィールドを \*YES に設定します。ファイルを使用する前に、部分的なトランザクションを除去する必要があります。部分的なトランザクションを除去するには、ファイルを削除して、前に保管したファイルを復元します。前に保管したファイルがない場合には、ファイルをオープンできるように、ジャーナル処理済みオブジェクトの変更 (CHGJRNOBJ) コマンドを使用して、「部分的なトランザクションが存在する」状態をリセットします。CHGJRNOBJ を使用して、整合性のとれた状態にするためにファイルを編集する必要があります。CHGJRNOBJ コマンドは、前に保管したファイルがない場合にのみ使用してください。

## 長時間を要するロールバックの終了機能の使用不可化

デフォルトでは、\*ALLOBJ 特殊権限を持つユーザーはロールバックを終了できます。\*ALLOBJ 特殊権限を持つユーザーによるロールバックの終了を制限したい場合は、データ域 QGPL/QTNNOENDRB を作成して制限することができます。

### 関連資料:

データ域の作成 (CRTDTAARA) コマンド

## 大規模トランザクションまたは古いトランザクションを検出する

大規模トランザクションまたは古いトランザクションを検出するには、コミットメント定義の処理 (WRKCMTDFN) コマンドを使用します。

大規模トランザクションまたは古いトランザクションを検出するには、以下のステップを実行します。

1. 文字ベース・インターフェースから WRKCMTDFN JOB(\*ALL) OUTPUT(\*PRINT) を入力します。
2. スプール・ファイルを処理するには WRKSPLF コマンドを使用します。
3. オプション 5=Display を使用して、ステップ 1 の WRKCMTDFN コマンドで作成されたスプール・ファイルを表示します。
4. 大規模トランザクションを検出するには、ジャーナル状況の表示セクションでコミットメント定義ごとに、「レコード・ロック」フィールドおよび「保留中の変更」フィールドを検索します。これらのフィールドは、現在レコード・ロックおよび保留中の変更があるトランザクションの総数を表示します。

- 古いトランザクションを検出するには、コミットメント制御状況セクションでコミットメント定義ごとに、「日時スタンプ」フィールドを検索します。このフィールドはトランザクションが開始された時刻を表示します。

また、この情報は、System i® Navigator または IBM Navigator for i のいずれかを使用してトランザクションを表示しても示されますが、**WRKCMDFN** コマンドによって作成された単一プール・ファイルを検索することにより、各トランザクションごとにプロパティとリソース状況を表示するよりも、大規模トランザクションまたは古いトランザクションをより容易に検出できます。

#### 関連タスク:

120 ページの『デッドロックの検出』

あるジョブがあるオブジェクト (オブジェクト A) のロックを保持しながら他のオブジェクト (オブジェクト B) のロックの獲得を待っており、同時に他のジョブまたはトランザクションがオブジェクト B のロックを保持しながらオブジェクト A のロックの獲得を待っている場合、デッドロック状態が発生することがあります。

71 ページの『コミットメント制御情報の表示』

IBM Navigator for i は、システム上のすべてのトランザクション (作業論理単位) に関する情報を表示することができます。IBM Navigator for i は、トランザクションに関連付けられたジョブの情報がある場合は、それらも表示することができます。


---

## コミットメント制御の関連情報

製品マニュアル、IBM Redbooks 資料、Web サイト、およびその他の Information Center トピック・コレクションには、コミットメント制御トピック集に関連する情報が収められています。以下の PDF ファイルのいずれも表示または印刷できます。




### 資料

以下の資料は、V6R1 i5/OS Information Center には収められていません。ただし、これらの資料はお客様にとって有益な参考資料としてご利用いただけるもので、印刷ハードコピーでの入手をご希望の場合は


IBM Publications Center ([www.elink.ibm.link.ibm.com/publications/servlet/pbi.wss?](http://www.elink.ibm.link.ibm.com/publications/servlet/pbi.wss?))  にご注文いただけます。また、オンライン形式での入手を希望される場合は、無料でダウンロードすることも可能です。

- COBOL/400 使用者の手引き
- RPG/400® 使用者の手引き

### IBM Redbooks

- Advanced Functions and Administration on DB2 for i 
- Stored Procedures, Triggers, and User Defined Functions on DB2 for i 
- Striving for Optimal Journal Performance on DB2 for i 

### Web サイト

- The Open Group (英語) ([www.opengroup.org](http://www.opengroup.org)) 

## その他の情報

- データベース・プログラミング
- SQL プログラミング
- XA API
- ジャーナル管理

### 関連資料:

1 ページの『コミットメント制御の PDF ファイル』  
この情報の PDF ファイルを表示または印刷できます。

---

## コードに関するライセンス情報および特記事項

IBM は、お客様に、すべてのプログラム・コードのサンプルを使用することができる非独占的な著作使用权を許諾します。お客様は、このサンプル・コードから、お客様独自の特別のニーズに合わせた類似のプログラムを作成することができます。

強行法規で除外を禁止されている場合を除き、IBM、そのプログラム開発者、および供給者は「プログラム」および「プログラム」に対する技術的サポートがある場合にはその技術的サポートについて、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。

いかなる場合においても、IBM および IBM のサプライヤーならびに IBM ビジネス・パートナーは、その予見の有無を問わず発生した以下のものについて賠償責任を負いません。

1. データの喪失、または損傷。
2. 直接損害、特別損害、付随的損害、間接損害、または経済上の結果的損害
3. 逸失した利益、ビジネス上の収益、あるいは節約すべかりし費用

国または地域によっては、法律の強行規定により、上記の責任の制限が適用されない場合があります。

---

## 特記事項

本書は米国 IBM が提供する製品およびサービスについて作成したものです。

本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒103-8510

東京都中央区日本橋箱崎町19番21号

日本アイ・ビー・エム株式会社

法務・知的財産

知的財産権ライセンス渉外

以下の保証は、国または地域の法律に沿わない場合は、適用されません。IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態を提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

この情報には、技術的に不適切な記述や誤植を含む場合があります。本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム (本プログラムを含む) との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

IBM Corporation

Software Interoperability Coordinator, Department YBWA

3605 Highway 52 N

Rochester, MN 55901

U.S.A.



本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、IBM より提供されます。

この文書に含まれるいかなるパフォーマンス・データも、管理環境下で決定されたものです。そのため、他の操作環境で得られた結果は、異なる可能性があります。一部の測定が、開発レベルのシステムで行われた可能性があります。その測定値が、一般に利用可能なシステムのものと同じである保証はありません。さらに、一部の測定値が、推定値である可能性があります。実際の結果は、異なる可能性があります。お客様は、お客様の特定の環境に適したデータを確かめる必要があります。

IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。IBM は、それらの製品のテストは行っておりません。したがって、他社製品に関する実行性、互換性、またはその他の要求については確認できません。IBM 以外の製品の性能に関する質問は、それらの製品の供給者をお願いします。

IBM の将来の方向または意向に関する記述については、予告なしに変更または撤回される場合があります、単に目標を示しているものです。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれています。より具体性を与えるために、それらの例には、個人、企業、ブランド、あるいは製品などの名前が含まれている場合があります。これらの名称はすべて架空のものであり、名称や住所が類似する企業が実在しているとしても、それは偶然にすぎません。

#### 著作権使用許諾:

本書には、様々なオペレーティング・プラットフォームでのプログラミング手法を例示するサンプル・アプリケーション・プログラムがソース言語で掲載されています。お客様は、サンプル・プログラムが書かれているオペレーティング・プラットフォームのアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。これらのサンプル・プログラムは特定物として現存するままの状態を提供されるものであり、いかなる保証も提供されません。IBM は、お客様の当該サンプル・プログラムの使用から生ずるいかなる損害に対しても一切の責任を負いません。

それぞれの複製物、サンプル・プログラムのいかなる部分、またはすべての派生的創作物にも、次のように、著作権表示を入れていただく必要があります。

© (お客様の会社名) (西暦年). このコードの一部は、IBM Corp. のサンプル・プログラムから取られています。

© Copyright IBM Corp. \_年を入れる\_.

この情報をソフトコピーでご覧になっている場合は、写真やカラーの図表は表示されない場合があります。

---

## プログラミング・インターフェース情報

本書「コミットメント制御」には、プログラムを作成するユーザーが IBM i のサービスを使用するためのプログラミング・インターフェースが記述されています。

---

## 商標

IBM、IBM ロゴおよび [ibm.com](http://ibm.com) は、世界の多くの国で登録された International Business Machines Corporation の商標です。他の製品名およびサービス名等は、それぞれ IBM または各社の商標である場合があります。現時点での IBM の商標リストについては、『[www.ibm.com/legal/copytrade.shtml](http://www.ibm.com/legal/copytrade.shtml)』をご覧ください。

Adobe、Adobe ロゴ、PostScript、PostScript ロゴは、Adobe Systems Incorporated の米国およびその他の国における登録商標または商標です。

UNIX は The Open Group の米国およびその他の国における登録商標です。

他の製品名およびサービス名等は、それぞれ IBM または各社の商標である場合があります。

---

## 使用条件

これらの資料は、以下の条件に同意していただける場合に限りご使用いただけます。

**個人使用:** これらの資料は、すべての著作権表示その他の所有権表示をしていただくことを条件に、非商業的な個人による使用目的に限り複製することができます。ただし、IBM の明示的な承諾をえずに、これらの資料またはその一部について、二次的著作物を作成したり、配布（頒布、送信を含む）または表示（上映を含む）することはできません。

**商業的使用:** これらの資料は、すべての著作権表示その他の所有権表示をしていただくことを条件に、お客様の企業内に限り、複製、配布、および表示することができます。ただし、IBM の明示的な承諾をえずにこれらの資料の二次的著作物を作成したり、お客様の企業外で資料またはその一部を複製、配布、または表示することはできません。

ここで明示的に許可されているもの以外に、資料や資料内に含まれる情報、データ、ソフトウェア、またはその他の知的所有権に対するいかなる許可、ライセンス、または権利を明示的にも黙示的にも付与するものではありません。

資料の使用が IBM の利益を損なうと判断された場合や、上記の条件が適切に守られていないと判断された場合、IBM はいつでも自らの判断により、ここで与えた許可を撤回できるものとさせていただきます。

お客様がこの情報をダウンロード、輸出、または再輸出する際には、米国のすべての輸出入関連法規を含む、すべての関連法規を遵守するものとします。

IBM は、これらの資料の内容についていかなる保証もしません。これらの資料は、特定物として現存するままの状態を提供され、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任なしで提供されます。







プログラム番号: 5770-SS1

Printed in Japan

**日本アイ・ビー・エム株式会社**

〒103-8510 東京都中央区日本橋箱崎町19-21