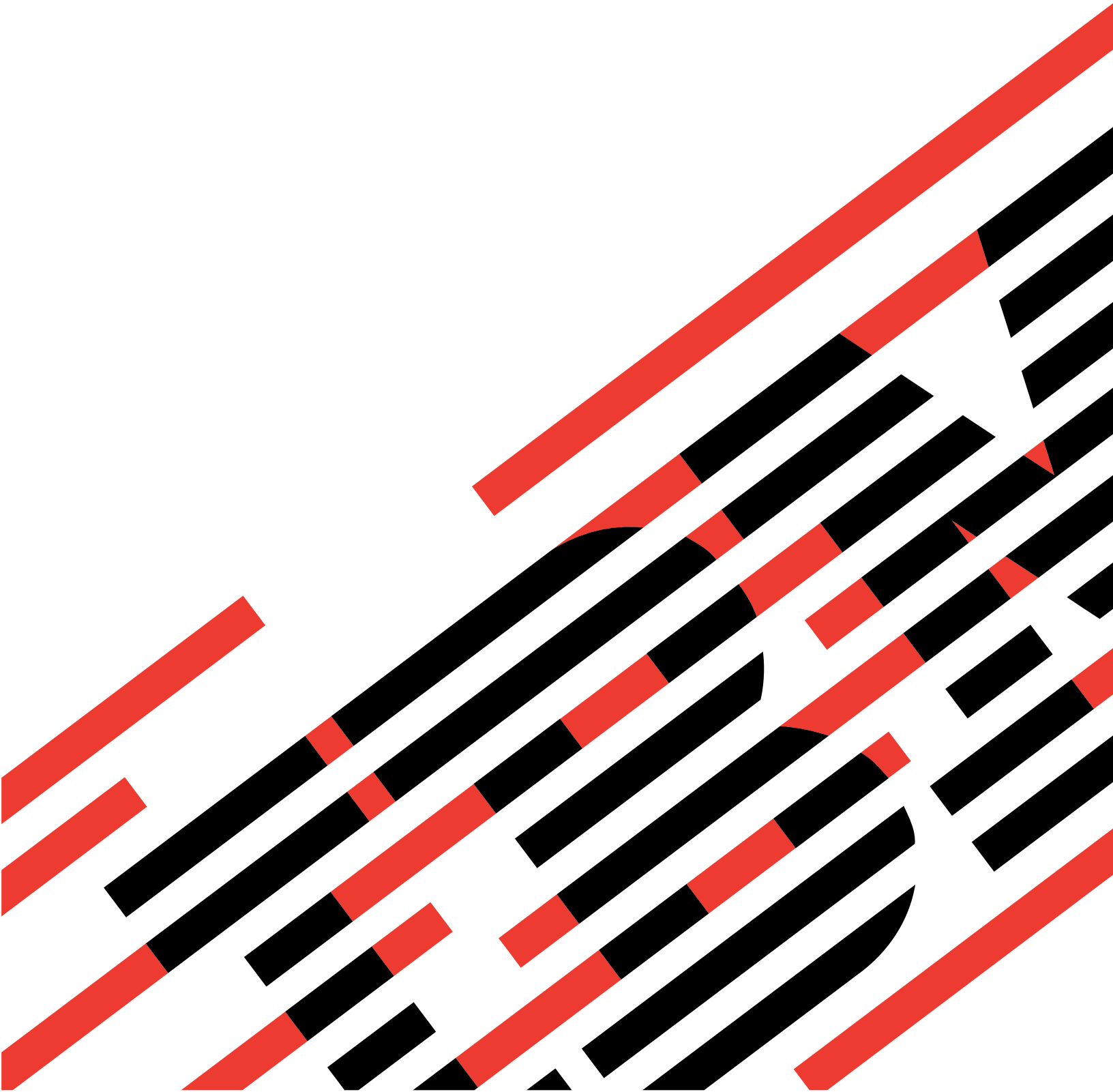
IBM

eserver

iSeries

コミットメント制御



IBM

@server

iSeries

コミットメント制御

本マニュアルに関するご意見やご感想は、次の URL からお送りください。今後の参考にさせていただきます。

http://www.ibm.com/jp/manuals/main/mail.html

なお、日本 IBM 発行のマニュアルはインターネット経由でもご購入いただけます。詳しくは

http://www.ibm.com/jp/manuals/ の「ご注文について」をご覧ください。

(URL は、変更になる場合があります)

原　典： RZAK-J000-00
iSeries
Commitment control

発　行： 日本アイ・ビー・エム株式会社

担　当： ナショナル・ランゲージ・サポート

第1刷 2002.8

目次

コミットメント制御

コミットメント制御は、データ保全性を保証するための機能です。これにより、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することが可能になります。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、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。DB2 Universal Database for iSeries™ は、コミットメント制御機能を使用して、分離レベル *NONE (コミットなし) 以外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。

iSeries サーバーでコミットメント制御を立ち上げて実行するために、以下の情報を参照してください。

V5R2 の新機能
このトピックでは、コミットメント制御についての新規の情報がリストされています。

このトピックの印刷
この情報の全体を印刷してください。

コミットメント制御の概念
コミットメント制御の機能を理解するには、この情報をお読みください。

コミットメント制御の開始
コミットメント制御の開始について理解するには、この情報をお読みください。

コミットメント制御の終了
コミットメント制御を終了させるための前提条件およびコミットメント制御の終了方法を理解するには、この情報をお読みください。

コミットメント制御のシステム起動による終了
システムがコミットメント制御の終了を開始する場合にユーザーが実施する必要のある作業を理解してください。

トランザクションおよびコミットメント制御の管理
コミットメント制御を使用してシステムを管理するために実施する必要のある作業を理解してください。

コミットメント制御のシナリオおよび例
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理解するには、これらのシナリオおよび例をお読みください。コミットメント制御を使用するプログラムのコード例をお読みください。

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
コミットメント制御のトラブルシューティングが必要な場合、この情報をお読みください。

コミットメント制御の関連情報
コミットメント制御に関係するトピック、マニュアル、IBM レッドブック、および外部の Web サイトを参照してください。

注: 重要な法的情報については、コード例の特記事項をお読みください。

V5R2 の新機能

V5R2 には、コミットメント制御に対する多くの改善および追加があります。次の項目には、これらの改善および追加が含まれています。

- **コミットメント制御および独立ディスク・プール**
  新規のライブラリーに対応した独立ディスク・プールにより、コミットメント制御がサポートされます。
- **コミットメント制御の XA トランザクション・サポート**
  コミットメント制御の XA トランザクション・サポートにより、XA 仕様への準拠が改善されます。
- **コミットメント制御の iSeries ナビゲーター・サポート**
  iSeries ナビゲーターにより、コミットメント制御のサポートが提供されます。次のトピックで詳細を示します。
  - コミットメント制御情報の表示
  - コミットメント制御の XA トランザクション・サポート
  - 通信障害後のトランザクションのリカバリー
  - コミットとロールバックの強制時点および再同期の取り消し時点

当リリースの新機能および変更点に関するその他の情報は、ユーザーへのお知らせ  を参照してください。

このトピックの印刷

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 (約 2453 KB、126 ページ) を選択してください。

PDF ファイルの保管

PDF を表示または印刷用にワークステーションに保管するには、以下の手順を実施します。

1. ブラウザーで、該当の PDF を右クリックします (上記のリンクの右クリック)。
2. **「対象をファイルに保存... (Save Target As...)」**をクリックします。
3. PDF を保管しようとするディレクトリーにナビゲートします。
4. **「保存 (Save)」**をクリックします。

Adobe Acrobat Reader のダウンロード

これらの PDF を表示または印刷するために Adobe Acrobat Reader が必要な場合は、Adobe Web サイト (www.adobe.com/products/acrobat/readstep.html)  からコピーをダウンロードすることができます。

コミットメント制御の概念

以下のページでは、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つ情報が提供されています。

- コミットメント制御の機能
- コミット操作およびロールバック操作の機能
- コミットメント定義
- コミットメント制御下でのリソース
- コミットメント制御および独立ディスク・プール
- コミットメント制御に関する考慮事項と制約事項
- バッチ・アプリケーションのコミットメント制御
- 2 フェーズ・コミットメント制御
- コミットメント制御の XA トランザクション・サポート
- 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

コミットメント制御の機能

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することを許可する機能です。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、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。たとえば、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、次のことが可能になります。

- 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: iSeries ナビゲーター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次のものがあります。

- 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。このデータベース・ファイルでは以下のものが可能です。
  - 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

次の 2 つの操作は、コミットメント制御下で行われた変更に影響します。

- 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放します。
- ロールバック操作
  以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。トランザクションに関連したロックもすべてリリースされます。

次のプログラム言語および 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
ILE RPG/400	COMIT 操作コード	ROLBK 操作コード
ILE COBOL/400(R)	COMMIT verb	ROLLBACK verb
ILE C/400(R)	_Rcommit 機能	_Rrollbck 機能
PL/I	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() function (コミット・トランザクションおよびロールバック・トランザクションに使用)	
XA API	db2xa_commit() API	db2xa_rollback() API

次のリンクでは、これらのプログラム言語および API に関する詳細が提供されています。

- AS/400 V3.0.5 COBOL/400 使用者の手引き 
- AS/400 RPG/400 使用者の手引き 
- ILE C/C++ Programmer's Guide 

- CL プログラミング 
- System API Programming 
- DB2 UDB for iSeries SQL 呼び出しレベル・インターフェース
- DB2 UDB for iSeries プログラミング概念

コミット操作

以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。トランザクションに関連したロックもすべてリリースされます。

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。
- 次の両方を満たす場合は、コミット操作の実行の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ーは変更されません。
- 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。

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放します。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。

- 次の両方に該当する場合は、入出力バッファー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、入出力フィードバック域と入出力バッファー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 (事前イメージ) 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事前イメージと事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かった場合は、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。たとえば、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

コミットメント制御開始 (STRCMTCTL) コマンドを使用して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作成されます。また、「コミットなし」以外の分離レベルの場合は、DB2 UDB for iSeries 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には、そのジョブ内でコミットメント制御下に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ものが含まれます。

- 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。

コミットメント定義の詳細については、以下に示されています。

- コミットメント定義の範囲
- コミットメント定義名
- 例 : ジョブおよびコミットメント定義

コミットメント定義および独立ディスク・プールについての規則は、コミットメント制御および独立ディスク・プールを参照してください。

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、「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。

iSeries ナビゲーターのコミットメント定義処理 (WRKCMTDFN) 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、以下のフィールドが表示されます。

- 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 *DFTACTGRP 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すで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。

iSeries ナビゲーターのコミットメント定義処理 (WRKCMTDFN) 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、以下のフィールドが表示されます。

- 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。

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すが、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、プログラムの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応じて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コミットメント制御要求または操作の中で、活動化グループ・レベルのコミットメント定義の開始を防ぐものには以下が含まれます。

- 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・ファイルをオープンするとき、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オープンしていて、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。

iSeries ナビゲーターのコミットメント定義処理 (WRKCMTDFN) 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、以下のフィールドが表示されます。

- 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 プロトコルに従ってトランザクション作業を実行するスレッドに接続します。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。

iSeries ナビゲーターのコミットメント定義処理 (WRKCMTDFN) 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、以下のフィールドが表示されます。

- 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)。

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 (たとえば、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 (たとえば、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

統合言語環境 (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。統合言語環境の詳細については、ILE 概念  を参照してください。

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。

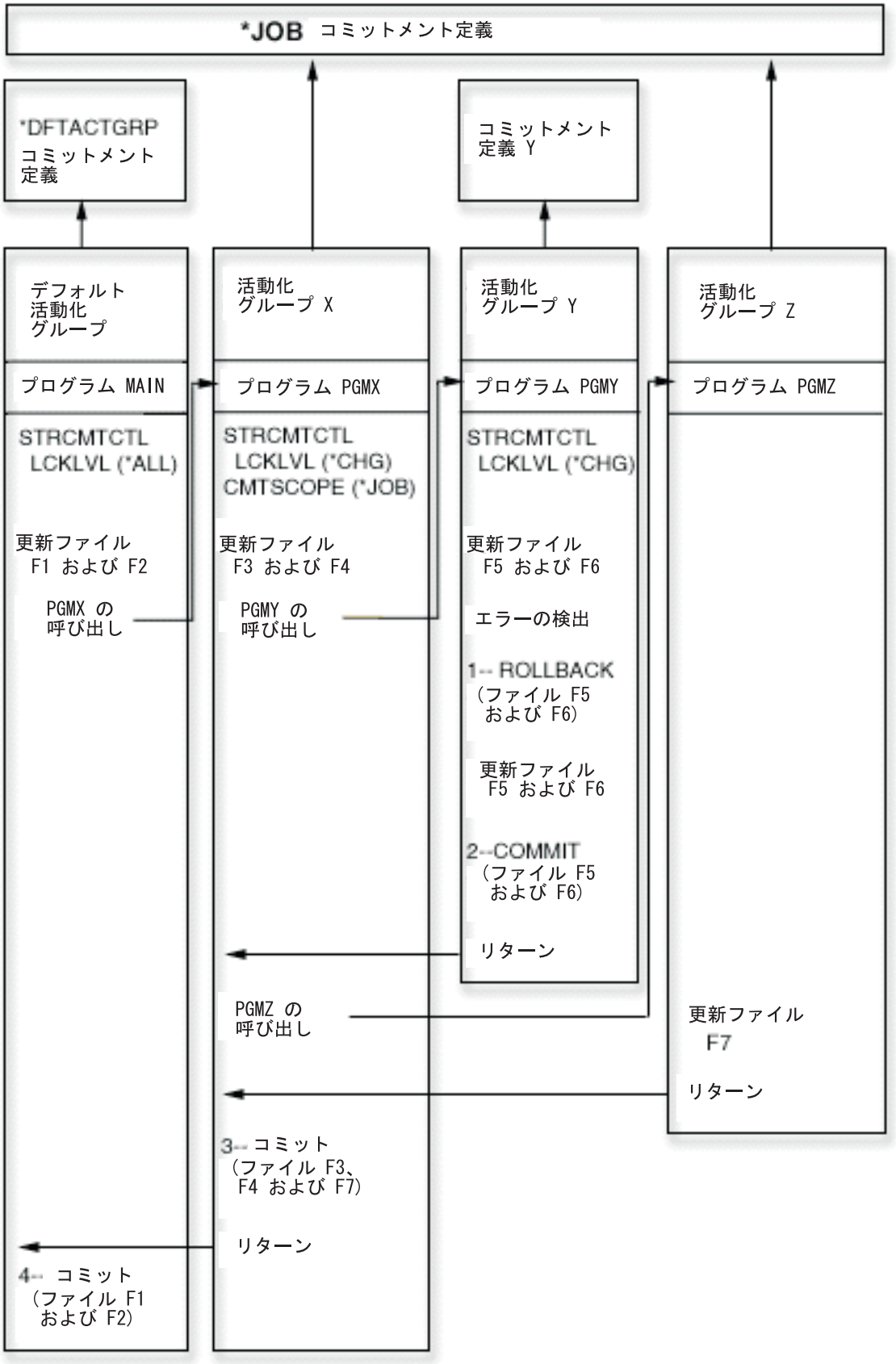
アプリケーション・プログラムがコミットメント制御要求を出すときには、個々のコミットメント定義を識別するのにコミットメント定義名を使用しません。コミットメント定義名は、主として、ジョブの特定のコミットメント定義を識別するため、メッセージ内で使用されます。

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。

例 : ジョブおよびコミットメント定義

次の図は、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。それは、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。


*JOB コミットメント定義
*DFTACTGRP
コミットメント
定義
コミットメント
定義 Y
デフォルト
活動化
グループ
活動化
グループ X
活動化
グループ Y
活動化
グループ Z
プログラム MAIN
プログラム PGMX
プログラム PGMY
プログラム PGMZ
STRCMTCTL
LCKLVL (*ALL)
STRCMTCTL
LCKLVL (*CHG)
CMTSCOPE (*JOB)
STRCMTCTL
LCKLVL (*CHG)
更新ファイル
F1 および F2
更新ファイル
F3 および F4
更新ファイル
F5 および F6
PGMX の
呼び出し
PGMY の
呼び出し
エラーの検出
1-- ROLLBACK
(ファイル F5
および F6)
更新ファイル
F5 および F6
2--COMMIT
(ファイル F5
および F6)
リターン
PGMZ の
呼び出し
更新ファイル
F7
リターン
3-- コミット
(ファイル F3、
F4 および F7)
リターン
4-- コミット
(ファイル F1
および F2)

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	すでにコミット済み	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	すでにコミット済み	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	すでにコミット済み	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にない	すでにコミット済み	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

次のリンクに、コミットメント制御下でのリソースに関する詳細情報があります。
- コミット可能なリソースのタイプ
- 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のアクセス・インテント
- コミット可能リソースのコミット・プロトコル
- ジャーナル・ファイルおよびコミットメント制御
- コミットメント制御下でのジャーナル項目の順序
- コミット・サイクル ID
- レコードのロック

コミット可能リソースのタイプ

次の表では、以下を示します。
- 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[1]	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トランザクション[2] 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 • SQL パッケージの作成 • SQL 表の作成 • SQL 表の除去	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[3]	会話の終了		

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
DRDA(R)- 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		
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る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:

[1]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。

[2]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。説明については、トランザクション・サイズの管理を参照してください。

[3]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 LU6.2 リソースが追加されます。

DRDA 接続が開始されたときに、以下の両方が当てはまる場合には、DRDA リソースに LU6.2 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・ロケーションを使って定義された RDB に対するものである。保護会話の開始の詳細については、APPC Programming を参照してください。

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常駐します。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、別個のローカル・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・プールにある場合、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とは見なされません。コミット可能リソースおよび独立ディスク・プールの詳細については、コミットメント制御および独立ディスク・プールを参照してください。

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常駐します。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をアクセスするには、DRDA を使用する必要があります。

以下には、コミット可能リソースのタイプとロケーションが示されています。

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

コミット可能リソースのアクセス・インテント

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

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。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

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
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ければ、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が判別されます。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、すで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、すで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、すで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・プールにある場合、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。コミット可能リソースおよび独立ディスク・プールの詳細については、コミットメント制御および独立ディスク・プールを参照してください。

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。

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「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にファイルをオープンする場合は、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・ファイルをオープンしようとしたが、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オープンさ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オープンされたときに、ファイルの事後イメージのみがジャーナル処理されている場合、システムは事前イメージと事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。事前イメージは、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、事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、ジャーナル以外のメカニズムを使用して、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。

ジャーナル処理の詳細については、ジャーナル管理のトピックを参照してください。

コミットメント制御下でのジャーナル項目の順序

次の表には、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・コード・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ジャーナル (ローカルまたはリモート) に書き込まれるのは、少なくとも以下のうちの 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。
- ジャーナルに割り当てら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 コマンドが出された時。
		各ローカル位置のジャーナル	ジャーナルに割り当てら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。

項目タイプ	説明	書き込まれる場所	書き込まれる時
C SC	コミット・サイクルの開始	各ローカル位置のジャーナル	このジャーナル[1] に割り当てら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。
		API リソースのジャーナル	QJOSJRNE API をコミット・サイクル *ID* 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	アプリケーション・プログラムが SNDJRNE コマンドまたは QJOSJRNE API を使用した場合。
C CM	コミット	各ローカル位置のジャーナル	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	各ローカル位置のジャーナル	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。
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。

項目タイプ	説明	書き込まれる場所	書き込まれる時
C EC	コミットメント制御終了	各ローカル位置のジャーナル	コミットメント制御の終了 (ENDCMTCTL) コマンドが完了した時。
		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。

注:
[1] 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) が含められます。同様に、XA トランザクションの場合、FIXLENDTA(*LUW) 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 が含められます。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。

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) が含められます。同様に、XA トランザクションの場合、FIXLENDTA(*LUW) 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 が含められます。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適用 (APYJRNCHG) コマンドまたはジャーナル処理済み変更除去 (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と、要求側のジョブは待機させられ、次のいずれかが起こるまでアクティブな処理から除去されます。

- 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・オーバーライド (OVRDBF)

待機時間を指定する時には、次のこと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これらの API がリストされています。
- 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こりました。たとえば、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をアクセスし、キーがすでに存在するかどうかを判別しようと試みます。キーがすで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の詳細については、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。アクティブ中のサーバーの保管のトピックには、活動時保管機能に関する詳細情報があります。

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 OS/400 SQL 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ので、以下の考慮事項に従う必要があります。

コミットメント定義の考慮事項

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。たとえば、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・プールのみに 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。

コミットメント定義についてのその他の考慮事項は、以下のとおりです。

- 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 CPFB8E9 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。
  - 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。

- 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。
- 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。
- LU6.2 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。

XA トランザクションの考慮事項

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。

独立ディスク・プールの詳細については、独立ディスク・プールのトピックを参照してください。

コミットメント制御に関する考慮事項と制約事項

以下は、コミットメント制御に関するその他の考慮事項および制約事項です。

データベース・ファイルの考慮事項

- 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に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) が使用されます。
- 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適用 (APYJRNCHG) コマンドまたはジャーナル処理済み変更除去 (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。
- ファイルの事前イメージと事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。ファイルの事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事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事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・レベルの変更の考慮事項

- 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。ただし、ジャーナル項目送信 (QJOSJRNE) API を使って他のイベントのジャーナル項目を送ることはできます。リカバリー時に、ユーザー作成プログラムを使って、これらの項目を検索し処理することができます。
- 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。SQL を使用したコミットメント制御下で行われる変更に関する詳細は、SQL プログラミング 概念を参照してください。

1 フェーズ・コミットおよび 2 フェーズ・コミットの考慮事項

- 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・システム上でまず始動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。

保管の考慮事項

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 つまたは複数ある場合には、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。

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。詳細は、ソフトウェア・インストール前の 2 フェーズ・コミット保全性の確認のトピックを参照してください。
- 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、次のいずれかの状況が起こり得ます。
  - 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にボートした。コミットを起動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ボートした。コミットを起動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。

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。

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。

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。OS/400 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。OS/400 のすべてのサポート済みリリースでは、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) 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るようになりました。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 の詳細については、 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 Open Group Web サイト 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、**ボート**といいます。
- **コミット・ウェーブ**時には、コミット要求を初期化する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・ボ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。起こり得る状態の詳細については、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。

詳細については、以下を参照してください。

- コミット処理での役割
- 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
- 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。以下に、リソース・マネージャーをリソース・タイプごとに示します。

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

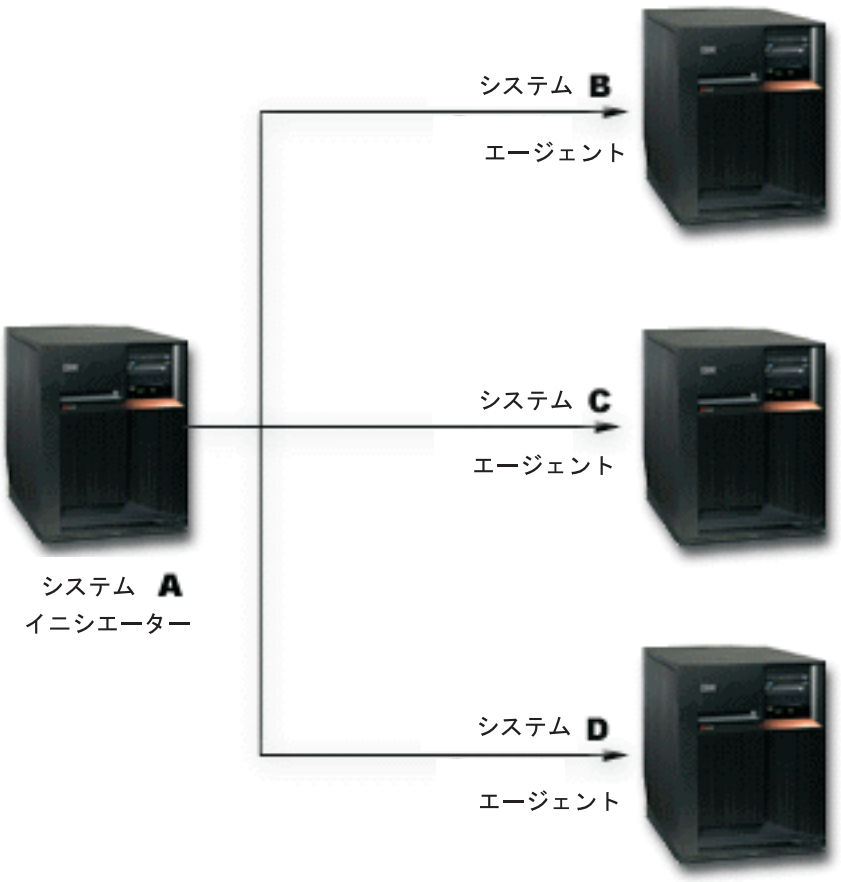
API
API 出口プログラム

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 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。

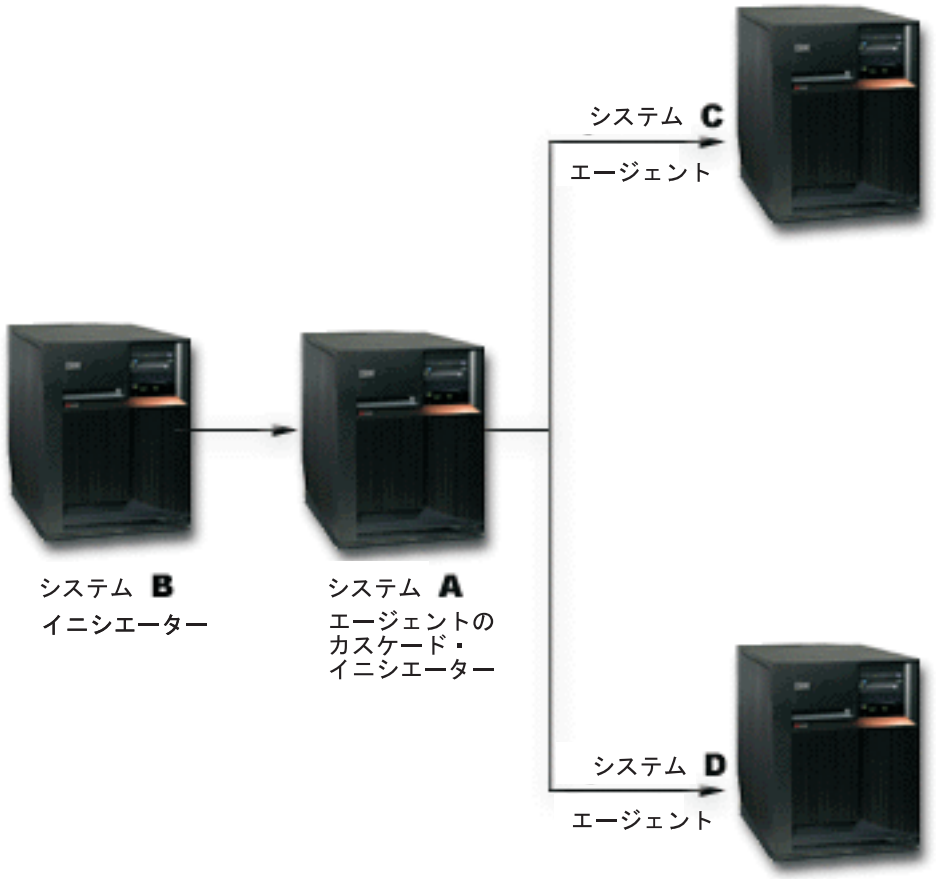
他の 3 つのシステム (B、C、および D) のリソース・マネージャーは、このトランザクションの**エージェント**になります。TCP/IP 上の DRDA 分散作業単位の場合、エージェントは**参加者**と呼ばれることもあります。



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ことがあります。 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ことがあります。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。

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。

次の表では、以下を示します。

- 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
2 フェーズ・コミット処理時の基本状態			
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ボートしていません。	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。
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、コミットのボートをしました。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る。

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
コミット進行中 (**CIP**)	すべてのロケーションがコミットをボートしました。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
2 フェーズ・コミット処理時の上記以外の状態			
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る。
読み取り専用のボート (**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。読み取り専用のボ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
ロールバック必須 (**RBR**)	次のどれかが生じました。 • 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。 • トランザクションに障害が発生した。 • 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。 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。	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。
オペレーターの処置またはエラーが原因で起きる条件			

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
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	保留中の変更はすで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	保留中の変更はすで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。

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を開始した後、QTNCHGCO (コミットメント変更オプション) API を使用して、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ことができます。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。

次のリンクで、コミットメント・オプションおよびそれらを使用する理由が説明されています。

- 読み取り専用ボートの許可
- 結果の待機なし
- 「流用可能」の指示
- 最後のエージェントを選択しない
- ボート信頼性

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「読み取り専用ボートの許可」のみです。

2 フェーズ・コミットのコミットメント定義 : 読み取り専用ボートの許可: 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ボートできるように、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ボートします。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な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、コミットメント変更オプション (QTNCHGCO) API を使用して、**許可された読み取り専用ボート**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。

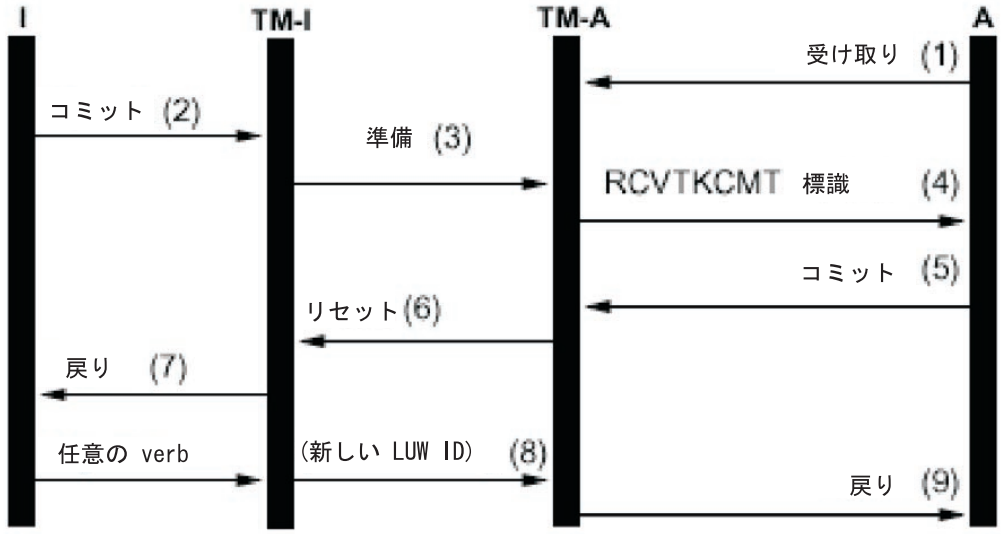
- 1 つまたは複数のリモート・システムにトランザクションのコミット可能な変更がない場合が多い。
- トランザクションが、前のトランザクションによって設定されたファイル・カーソル (次のレコード) の位置に依存していない。ロケーションが読み取り専用をボートした場合、トランザクションがロールバックされてもアプリケーションには通知されません。ロケーションは、データベース・ファイルに対する読み取り操作をコミットしているので、カーソル位置を移動しました。ファイル・カーソルの位置が重要なのは、順次処理を行っている場合だけです。

コミットメント定義が読み取り専用をボ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ボートオプションは、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なければ、プログラム A に**読み取り専用ボート**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ボートするときの、最終エージェントの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ボートするとき、最終エージェントの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（コミット要求を受け取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ボートするときの最終エージェントの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、エラーが起こると、イベントの順序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、それがプログラム 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なわれます。
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・プログラムのボートです。
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、許可された読み取り専用ボート・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・ウエーブはありません。
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。

2 フェーズ・コミットメント制御の詳細については、コミット処理での役割および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。

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 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。TCP/IP 接続で DRDA 分散作業単位 (DUW) は、出力待ちを全く行いません。

次の条件が当てはまる場合には、この方式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、QTNCHGCO (コミットメント・オプション変更) 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、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) のみ記述しています。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。QTNCHGCO (コミットメント・オプションの変更) 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。

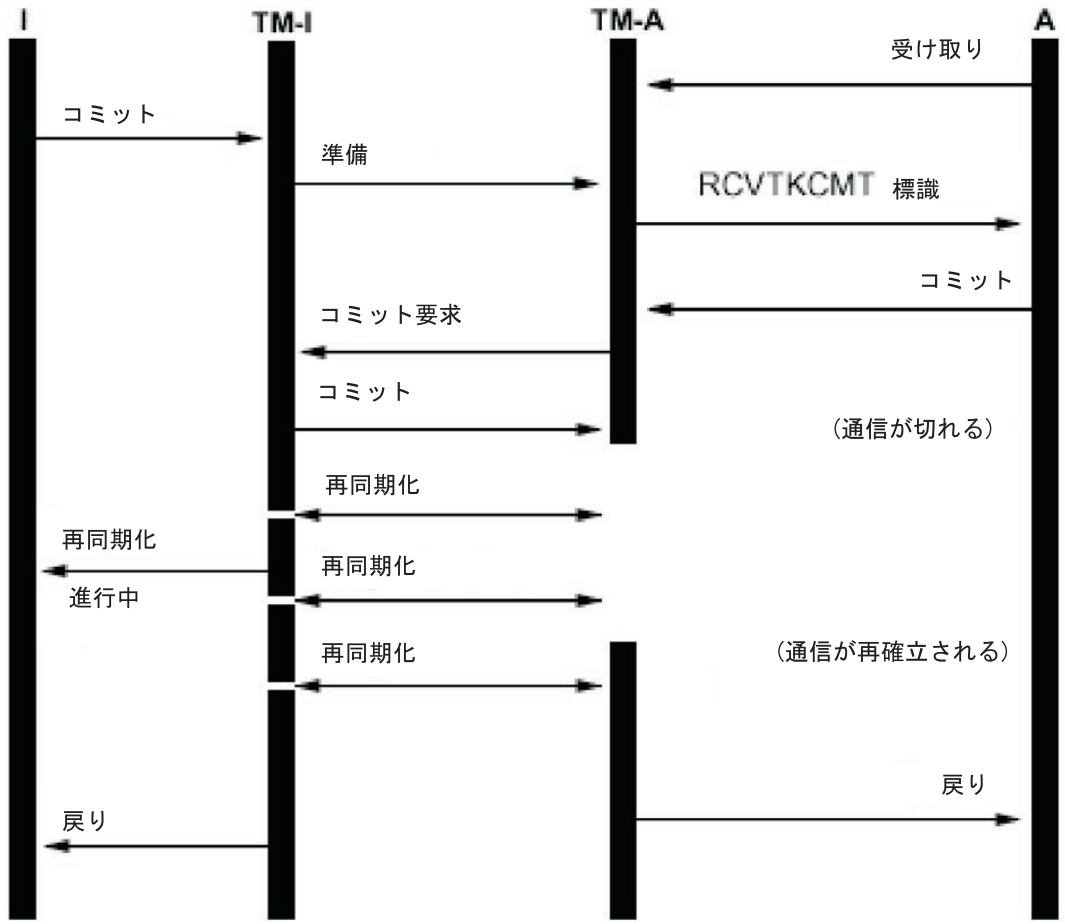
- 解決した WFO オプションが Y (Yes) の場合、アプリケーションは再同期の結果を待ちます。
- 解決した WFO オプションが N (No) であり、presumed abort プロトコルをサポートしているロケーションのウェーブまたはロールバックを準備している間に通信障害が発生する場合、再同期は実行されず、コミットメント定義はロールバックされます。

- 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。 疑問のあるコミットメント定義については、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。
- 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 - Yes

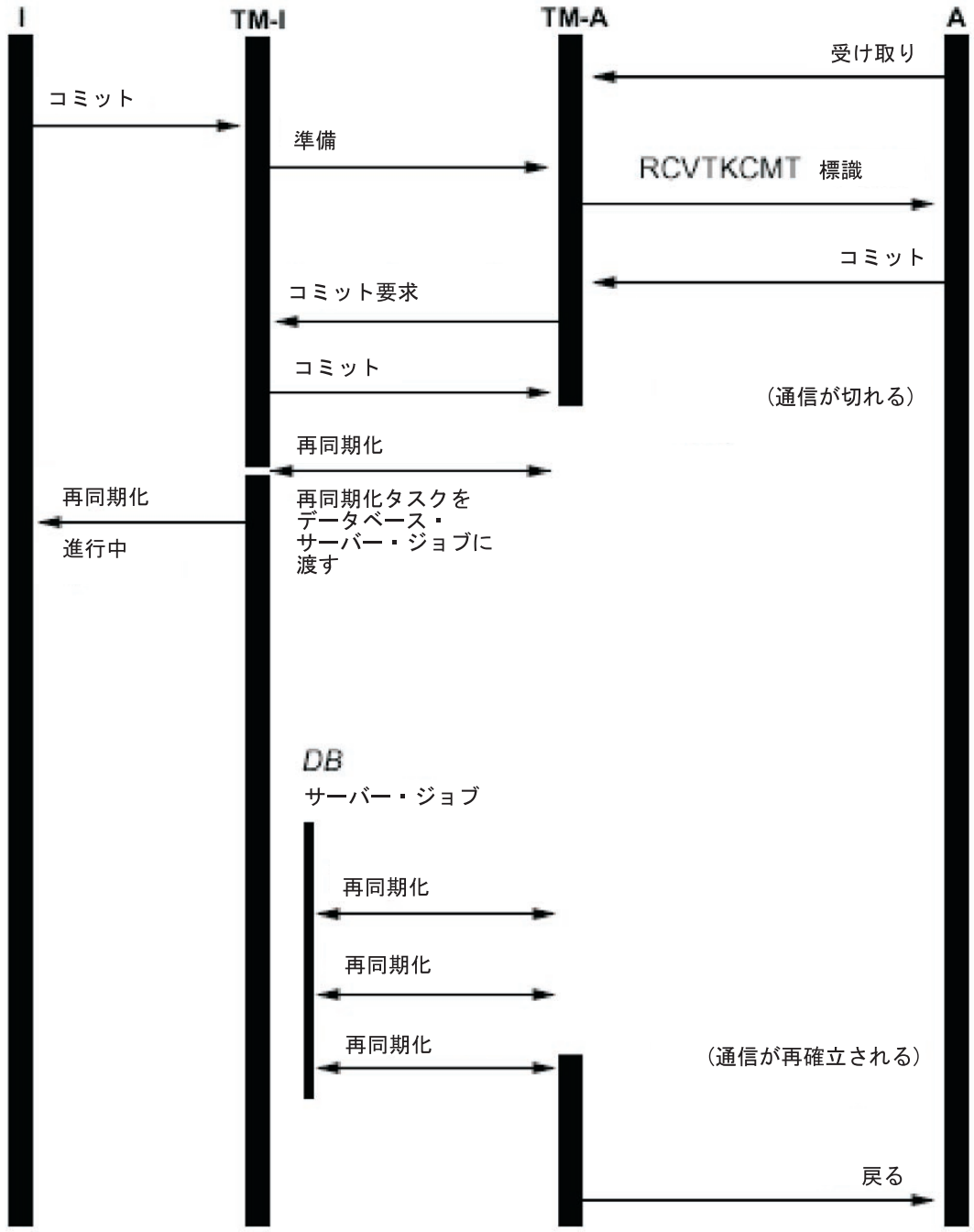
次の図では、イニシエーター (I) のコミットメント定義は、結果の待機オプションのデフォルト値である Y (Yes) を使用します。TM-I と TM-A との通信が失われると、アプリケーション A とアプリケーション I の両方が、トランザクションが再同期化されるまで待機します。



結果の待機 - 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い場合には無視され、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こと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。

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。

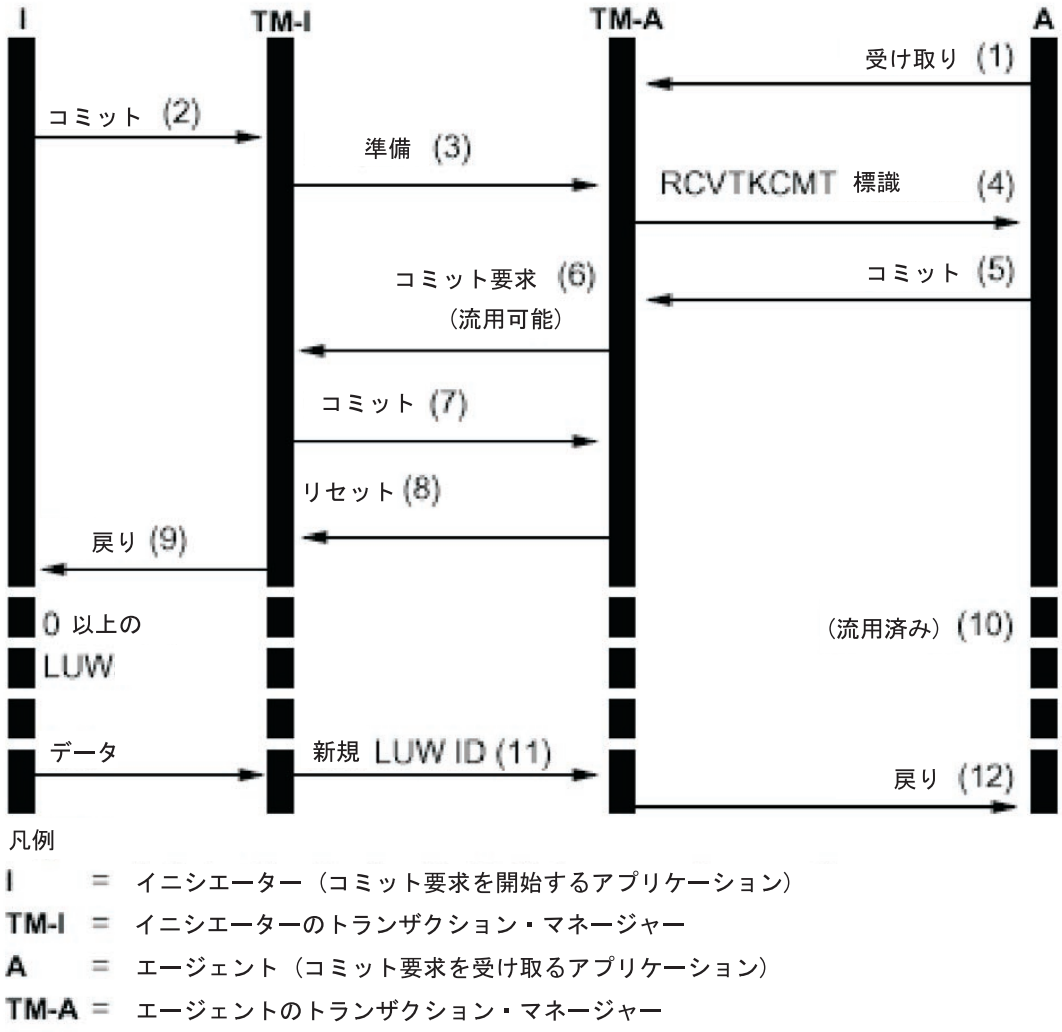
コミットメント制御を開始した後、コミットメント・オプション変更 (QTNCHGO) API を使用して「流用可能 (OK to leave out)」オプションを Y (Yes) に変更できます。この変更は、1 つ以上のリモート・システムが頻繁にトランザクションに関係しない場合に行えます。

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なければ、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、最終エージェントの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。



次に、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 がプログラム I からの要求を受け取る準備ができたかどうかを示す要求を受け取ります。
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、テーク・コミット受信 (RCVTKCMT) ICF 標識が送られることにより通知されます。
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・プログラムのボートです。
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-1) 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) がすべてのボートを受け取ると、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。

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登録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、これは可用性の問題でしょう。

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、エラーが生じたためにイニシエーターが待機するときの時間枠はとても小さいので、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を開始した後、コミットメント変更オプション (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 に変更することができます。

コミット処理のフローに影響するボート信頼性: ボート信頼性は、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と、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 DRDA 分散作業単位では、明示的なボート信頼性の最適化は行われません。しかし、OS/400 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を開始した後、コミットメント変更オプション (QTNCHGCO) API を使用して、ボ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。

ボート信頼性は、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。ボ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。イニシエーターもボ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、ボート信頼性の最適化の使用を考慮してください。

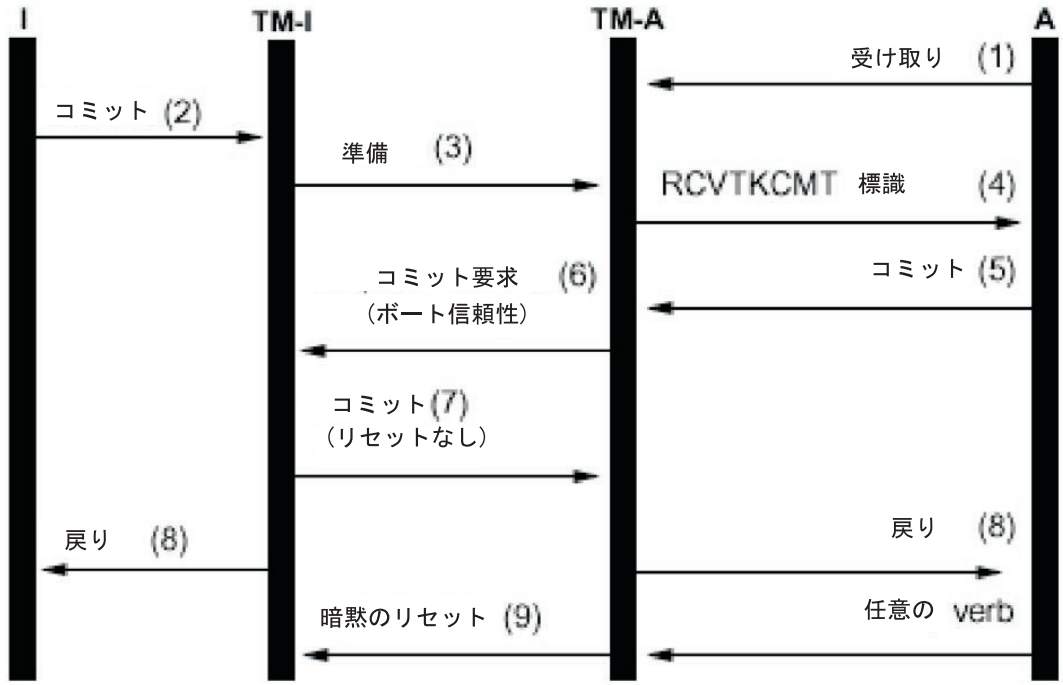
- 障害を修復できない場合を除いて、システムまたは通信障害が発生した場合に疑わしいエージェントでヒューリスティック判定が行われそうもない。
- データベース・ファイルの同期を確実に維持するのに、プログラム・ロジックは直前のトランザクションの結果を必要としない。

ボート信頼性の最適化は、次の条件に該当する場合に限り、OS/400 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ボート信頼性の指示を受け付ける。OS/400 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)。
  - ボ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ボート信頼性を行なう。OS/400 エージェントは、常にボ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。

ボ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、ボート信頼性の最適化が使用されるときの、アプリケーション・プログラムとトランザクション・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（コミット要求を受け取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、エージェントがボート信頼性を行な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 がプログラム I からの要求を受け取る準備ができたかどうかを示す要求を受け取ります。
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、テーク・コミット受信 (RCVTKCMT) ICF 標識が送られることにより通知されます。
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・プログラムのボートです。
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。OS/400 システムは、ボ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) がすべてのボートを受け取ると、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、ボ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ることになります。
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。

コミットメント制御の XA トランザクション・サポート

DB2 UDB for iSeries 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 UDB for iSeries 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 Open Group Web サイト  で見付けることができます。

DTP モデルには、5 つの構成要素があります。

アプリケーション・プログラム (AP)
データベースなどのリソースに関連する操作の順序を指定することによって、エンド・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。

トランザクション・マネージャー (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。

リソース・マネージャー (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、提供された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。

通信リソース・マネージャー (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 UDB for iSeries 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
CRM が相互に通信するために使用するプロトコル。これは DB2 UDB for iSeries 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 UDB for iSeries は、これらのインターフェースを UNIX 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 UDB for iSeries の使用法の詳細については、XA API を参照してください。

iSeries ナビゲーターおよび XA トランザクション

iSeries ナビゲーターは、XA トランザクションの管理を**グローバル・トランザクション**としてサポートします。

グローバル・トランザクションには、DB2 UDB for iSeries の外部と内部の両方の変更が含まれます。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 UDB for iSeries 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。

ローカル・システム上にあるデータベースに対して XA トランザクションを実行する場合、トランザクション範囲ロック用の XA API を使用する必要があります。これらの API では、ジョブ範囲ロックの場合の XA API よりも制約事項が少なく、次の状態であれば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。

リモート・システム上にあるデータベースに対して実行する場合、ジョブ範囲ロック用の XA API を使用する必要があります。

XA トランザクションの考慮事項

DB2 UDB for iSeries 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く必要があります。用語「スレッド」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 UDB for iSeries の考慮事項

- XA トランザクションは、SQL サーバー・モードで実行されているジョブでのみ実行できます。このことの影響の 1 つは、XA トランザクションの処理中に DB2 UDB for iSeries に変更を加える場合、アプリケーションは SQL インターフェースに限定されることです。SQL サーバー・モードで実行されていないジョブで 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。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を参照してください。
- XA API 呼び出し中に DB2 UDB for iSeries によって検出されたエラーは、XA 仕様に従って、戻りコードによって報告されます。エラーの意味が戻りコードだけでは明確にならない可能性があるときには、ジョブ・ログに診断メッセージが残されます。

組み込み SQL の考慮事項

- 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 が作成される前に db2xa_open() アプリケーション・プログラム・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 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使用されるユーザー・プロファイルとパスワードを db2xa_open() API に渡すことができます。それが渡されない場合は、デフォルトにより、接続の試行時に指定されたかまたはデフォルトが設定されたユーザー・プロファイルが使用されます。
- 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 iSeries セキュリティー機構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になります。

- 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 の考慮事項

- 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。

  注: 次の考慮事項は、トランザクション範囲ロックおよびジョブ範囲ロックのトランザクションに適用されます。

  CLI 結合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・データベースの考慮事項

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 LU6.2 上で DRDA 会話を実行するシステムが含まれます。また、TCP/IP 接続を使用する DRDA が実行されている場合には、V5R1 を使用するシステムも含まれます。
- 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。詳細は、コミットとロールバックの強制時点および再同期の取り消し時点を参照してください。

トランザクション・ブランチの考慮事項

- XA トランザクション・ブランチに関する情報は、iSeries ナビゲーターおよびジョブ処理 (WRKJOB)、ジョブ表示 (DSPJOB)、およびコミットメント定義の処理 (WRKCMTDFN) 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 iSeries ナビゲーターで**グローバル・トランザクション**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。

- 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、別のトランザクション・ブランチの開始よりも前に使用されており、スレッドとそのトランザクション・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になります。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 により最初のトランザクション・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。新しいトランザクション・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 によりトランザクション・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、次の状況が生じたときに、システムによってロールバックのみとマークされます。
  - スレッドがトランザクション・ブランチと関連したまま、そのスレッドが終了する。
  - db2xa_close() API が、トランザクション・ブランチとアクティブに関連したまま、スレッドで使用されている。
- 次のいずれかの状況が起こったときにトランザクション・ブランチと関連したままのスレッドがある場合、そのトランザクション・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。
- 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、トランザクション・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。

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。

- 次のいずれかの状況が起こったとしても、トランザクション・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、そのトランザクション・ブランチは影響を受けません。TM は、トランザクション・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 db2xa_open() API を使用した任意のスレッドから、トランザクション・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が、そのトランザクション・ブランチとはすでに関連していないスレッドまたはジョブが db2xa_close() API を使用する。

– トランザクション・ブランチの処理を実行したが、そのトランザクション・ブランチとはすでに関連していないスレッドまたはジョブが db2xa_close() API を使用する。
– システムに障害が起こる。この場合、トランザクション・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、システムはそのトランザクション・ブランチをロールバックします。

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。

これは、DB2 UDB for iSeries で、ジョブの変更 (QWTCHGJB) 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サポートされます。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 操作も、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。SQL サーバー・モードについての詳細は、データベースのトピック内にある次のページを参照してください。

- SQL サーバー・モードで DB2 CLI を実行する理由
- 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

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、STRCMTCTL (コミットメント制御開始) 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、次のことを指定できます。

上記のパラメーターについては、続く事項で説明されています。

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。詳細については、ロック・レベルのコミットを参照してください。

コミット通知オブジェクト
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。 詳細については、通知オブジェクトのコミットを参照してください。

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・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に、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てることができます。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 テキスト・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。

ジャーナル項目除外パラメーター
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。

これらの状態で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。

ジャーナル項目の可変長部分のトピックの 表 15 から表 21 までは、トランザクション・ジャーナル項目の項目特定データのレイアウトを示しています。

以下では、コミットメント制御の開始の詳細について説明しています。

- 通知オブジェクトのコミット
- ロック・レベルのコミット

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できるにはどこでアプリケーションの処理を終了するかの判別を助け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[1]
ILE RPG*	COMIT 操作コード	4000[1]
PLI	PLICOMMIT サブルーチン	4000[1]
ILE C*	_Rcommit 機能	4000[1]
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: [1]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。同時にレコードを変更できるユーザーまたはジョブがいくつかあるため、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を含める必要があります。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られます。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には各コミット 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できなければなりません。

通知オブジェクトの使用の例は、例 : アプリケーションを開始するための通知オブジェクトの使用を参照してください。

ロック・レベルのコミット

コミットメント制御開始 (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。 コミットメント制御に関する考慮事項と制約事項では、オブジェクト・レベルおよびレコード・レベルの変更の考慮事項が説明されています。

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応じ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コミットメント制御下に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、コミットメント制御下に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

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更新または削除する [1]	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 [2]	*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 [2]	*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 [2]	
更新用に読み取ってから解放する [1]	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
	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
	*UPDATE		
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

要求	LCKLVL パラメーター	ロックの持続期間	ロックのタイプ
注: [1]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。 [2]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。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行う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すが、 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制御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すで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。

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終了 (ENDCMTCTL) コマンドを使っ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すで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を行わなければなりません。

- 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。システムによる通知オブジェクトの更新に関しての詳細は、通知オブジェクトの更新を参照してください。

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終了しますが、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し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。

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。

以下のページでは、システムがコミットメント制御を暗黙的に終了させる状態、および、もしあれば必要な処置を説明しています。

- 活動化グループの終了時のコミットメント制御
- 暗黙コミットおよびロールバック操作
- 経路指定ステップの正常終了時のコミットメント制御
- 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
- コミットメント制御終了後の通知オブジェクトの更新
- 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

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 *DFTACTGRP 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 *DFTACTGRP 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、ENDCMTCTL 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、入出力バッファー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。システムによる通知オブジェクトの更新についての詳細は、通知オブジェクトの更新を参照してください。

暗黙コミットおよびロールバック操作

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・プログラムから始動されます。 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。ただし、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始動することがあります。システムによって始動されるコミット操作や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。

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イベントが生じたときに、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どうか。

状態	最後のエージェント API	ジョブ終了の場合の処置[1] 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	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。 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。 • 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。 • 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。
PIP	適用外	適用外	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。 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。

状態	最後のエージェント API	ジョブ終了の場合の処置[1] 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	コミットメント定義が X/Open[2] 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。 • 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。 • 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。

状態	最後のエージェント API	ジョブ終了の場合の処置[1] オプション	コミットまたはロールバック操作
PRP	N/A	C	コミットメント定義が X/Open[2] 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す。
		R	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。 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。 • 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。
CIP	適用外	適用外	明示コミット操作が実行されます。

状態	最後のエージェント API	ジョブ終了の場合の処置[1] オプション	コミットまたはロールバック操作
LAP	いいえ	WAIT	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。
			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。
LAP	はい	WAIT	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。
			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。
LAP	N/A	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すでに完了しています。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、読み取り専用にボートしました。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にボ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。

状態	最後のエージェント API	ジョブ終了の場合の処置[1] オプション	コミットまたはロールバック操作
注: [1] 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。 [2]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。 • 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。
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。

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通常、次の 1 つによって終了されます。

- 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先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を更新できます。システムによる通知オブジェクトの更新についての詳細は、通知オブジェクトの更新を参照してください。

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

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。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。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、次回の 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。この不確実さは、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ことによります。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

て行われた変更に悪影響が出ることがあります。たとえば、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すで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を実行します。

- 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。暗黙コミットおよびロールバック操作を参照してください。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りなく行われます。
  - *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りなく行われます。
  - *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。システムによる通知オブジェクトの更新についての詳細は、通知オブジェクトの更新を参照してください。

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。

通知オブジェクトの更新

通知オブジェクトの目的で、次のことが未コミット変更と考えられます。

- 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: *DFTACTGRP 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。
- 以下のすべてが当てはまる場合は、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。
  - デフォルト以外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 を指定しなかった場合には、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。ジョブの正常終了時に行われる機能についての詳細は、経路指定ステップの正常終了時のコミットメント制御を参照してください。
- 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で通知オブジェクトは更新されません。

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。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。たとえば、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 (ジョブ範囲かトランザクション範囲ロックかにかかわらず) に関連付けられたコミットメント定義を、このトピックで説明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。

iSeries ナビゲーターを使用するか、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 (WRKCMTDFN) 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。iSeries ナビゲーターおよび「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、データベースの整合性が保持されます。

トランザクションおよびコミットメント制御の管理

以下は、コミットメント制御を管理するために実施できるタスクです。

コミットメント制御情報の表示
この情報には、システム上のすべてのトランザクションに関する情報およびトランザクションに関連付けられたジョブに関する情報を表示できるタスクが含まれています。

コミットメント制御のパフォーマンスの最適化
この情報には、コミットメント制御がシステム・パフォーマンスに与える影響を最小化することができるタスクが含まれています。

コミットメント制御情報の表示

iSeries ナビゲーターを使用して、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。もしあれば、トランザクションに関連付けられたジョブの情報を参照することもできます。

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. **「iSeries ナビゲーター」**ウィンドウで、使用するサーバーを展開します。
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 UDB 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。

iSeries ナビゲーターを使用して、次の情報を表示することもできます。

- トランザクションのロックされたオブジェクトの表示
- トランザクションに関連付けられたジョブの表示
- トランザクションのリソース状況の表示
- トランザクションのプロパティーの表示

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. **「iSeries ナビゲーター」**ウィンドウで、使用するサーバーを展開します。
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 (Locked Objects)」**を選択します。

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. **「iSeries ナビゲーター」**ウィンドウで、使用するサーバーを展開します。
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。

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. **「iSeries ナビゲーター」**ウィンドウで、使用するサーバーを展開します。
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 (Resource status)」**を選択します。

トランザクションのプロパティーの表示

トランザクションのプロパティーを表示するには、次の手順に従います。

1. **「iSeries ナビゲーター」**ウィンドウで、使用するサーバーを展開します。
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、**「プロパティー」**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。以下は、パフォーマンスに影響のない要因、パフォーマンスを低下させる要因、およびパフォーマンスを向上させる要因です。

パフォーマンスに影響のない要因

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

ジャーナル処理

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。事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事前イメージと事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。ジャーナル処理の詳細については、ジャーナル管理のトピックを参照してください。

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

STRCMTCTL コマンドと ENDCMTCTL コマンドをそれぞれ使用してコミットメント定義を開始および終了するごとに、システムは、さらにオーバーヘッドを必要とします。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ので、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。

活動時保管処理がアクティブのときのレコード・レベルのデータベース・ファイル変更要求
コミットメント定義がコミットメント境界にあって、活動時保管操作が別のジョブで実行されている場合は、コミットメント定義下でレコード・レベルのデータベース・ファイルの変更を要求すると遅らされることがあります。保管要求のオブジェクトの特定のものと同じジャーナルにファイルがジャーナル処理されると、遅らされます。

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)」画面の状況欄には、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
活動時保管操作が別のジョブで実行されているときには、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、QTNADDCR API を使って API リソースを追加した場合、および*正常保管処理可能*フィールドに Y の値がある場合は別です。

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・リソースを追加された複数のコミットメント定義を持つジョブの場合、常に活動時保管操作は完了しません。

活動時保管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って、活動時保管操作が別のジョブで実行されている場合には、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して活動時保管操作が稼働しているときに、オブジェクト・レベル変更が行われたときに起こります。たとえば、ライブラリー MYSQLLIB に対して活動時保管操作が実行されているときには、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って、活動時保管操作が別のジョブで実行されている場合には、QTNADDCR API を使って 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. このことは、*正常保管処理可能*フィールドに Y の値がある場合、QTNADDCR API を使って追加された 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の使用
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の選択
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する必要があります。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の不使用
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」オプションの選択
「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。このオプションの詳細については、2 フェーズ・コミットのコミットメント定義 : 「流用可能」の指示を参照してください。

読み取り専用ボート・オプションの選択
読み取り専用ボート・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。このオプションの詳細については、2 フェーズ・コミットのコミットメント定義 : 読み取り専用ボートの許可を参照してください。

パフォーマンスを改善するために実施できるタスクは次のとおりです。

- ロックの最小化
- トランザクション・サイズの管理

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)。たとえば、単一のファイルをメインテナンスするアプリケーションは一般に次のことを行います。

- 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。
- 要求されたレコードを検索する。
- レコードを表示する。
- ワークステーション・ユーザーが変更を行うのを認める。
- 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・ユーザーの変更は重ね書きされるので、 (RPG での) 同じフィールド名の使用は適切ではありません。

レコード・データをデータ構造に移すことでこの問題を解決するか、または DDS キーワード RTNDTA を使用する場合には、外部記述のデータの使用を続けることができます。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次のことを行うことができます。

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。

ロックの詳細については、デッドロックの検出を参照してください。

トランザクション・サイズの管理

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。

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。サイズは、トランザクションが終了するまで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。

たとえば、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 (たとえば在庫マスター・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をアクセスできるようにしてください。 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、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。たとえば、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。たとえば、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ている場合には、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。

コミットメント制御のシナリオおよび例

以下は、コミットメント制御のシナリオおよび例です。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の概要が示されています。

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。

以下の 3 つの例では、システム異常終了後にアプリケーションを開始させるためのコミットメント制御の使用例が示されています。

シナリオ

- シナリオ : コミットメント制御

例

- コミットメント制御の実習問題
- コミットメント制御の実習問題のロジック・フロー
- 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
- 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
- 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

注: 重要な法的情報については、コード例の特記事項をお読みください。

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。このシナリオでは、JKL Toy Company で、部品が在庫部門から製造部門に転送されるときにコミットメント制御を使用する方法の概要が示されます。

JKL Toy Company のネットワーク環境の説明については、シナリオ : ジャーナル管理を参照してください。次のシナリオでは、コミットメント制御が本番用サーバー JKLPROD を操作する方法を示します。

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、サーバー 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、倉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なしのプログラム 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 は倉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)。
- 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 は倉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。たとえば、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、問題が起こります。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、ダイオードが倉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。

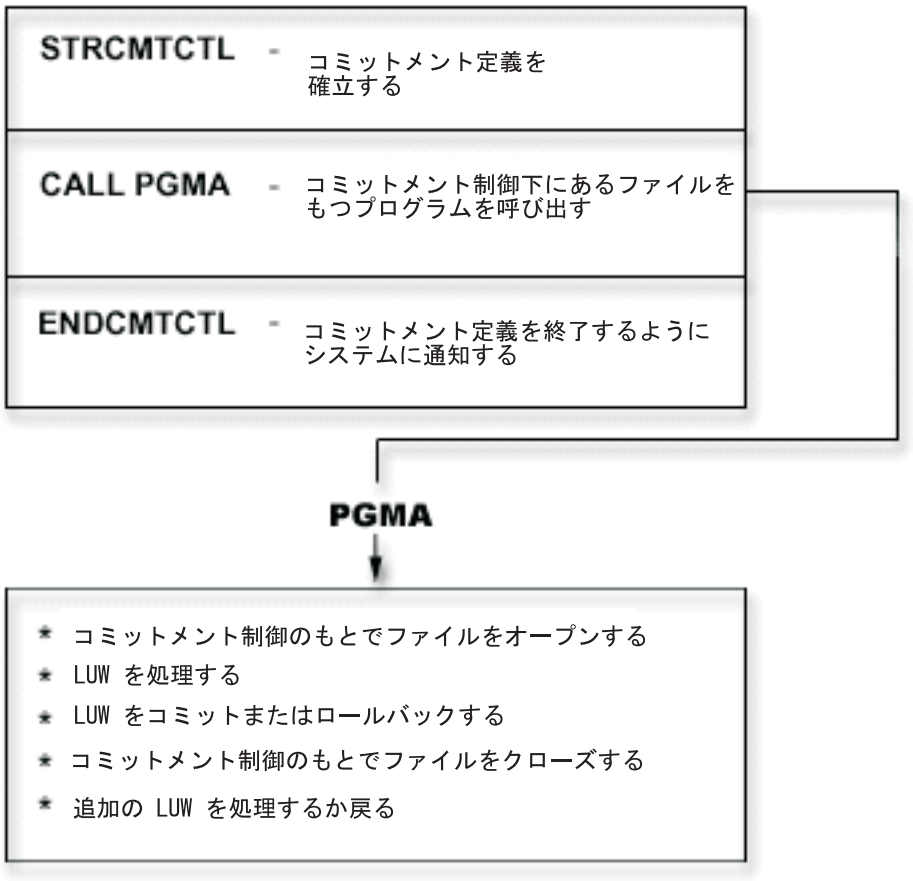
コミットメント制御を使用したプログラム A

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 は倉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 は倉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。倉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す。



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。

たとえば、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、事前イメージ (100) と事後イメージ (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、最初の項目が事前イメージ (80) で、2 番目の項目が事後イメージ (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、システムは倉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ないからです。この場合には、倉庫レコードの事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。

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。この実習問題は、OS/400 ライセンス・プログラムおよびデータ・ファイル・ユーティリティー (DFU) について読者がよく理解しており、このトピックをすでにここまで読んでいることが前提となっています。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。

注: 重要な法的情報については、コード例の特記事項をお読みください。

この問題を開始する前に、次のことを行ってください。

- 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。
- ソース・ファイルおよびジョブ記述の作成。

以下のステップを実行してください。

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 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・プログラム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

6. 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。

7. 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。

8. 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                                      CA03(93 'End of program')
  3.00      A                                      CA04(94 'Review last')
  4.00      A                                      SETOFF(64 'No rcd to rvw')
  5.00      A                                    1  2'INVENTORY TRANSACTIONS'
  6.00      A                                    3  2'Quantity'
  7.00      A            QTY            5  0I    +1
  8.00      A  61                                  ERRMSG('Invalid +
  9.00      A                                      quantity' 61)
 10.00      A                                   +5'ITEM'
 11.00      A            ITEM           2    I   +1
 12.00      A  62                                  ERRMSG('Invalid +
 13.00      A                                      Item number' 62)
 14.00      A  63                                  ERRMSG('Rollback +
 15.00      A                                      occurred' 63)
 16.00      A  64                               24  2'CF4 was pressed and +
 17.00      A                                       there are no +
 18.00      A                                       transactions for +
 19.00      A                                       this user'
 20.00      A                                       DSPATR(HI)
 21.00      A                                   23  2'CF4 Review last +
 22.00      A                                       transaction'
 23.00      A          R REVW
 24.00      A                                    1  2'INVENTORY TRANSACTIONS'
 25.00      A                                   +5'REVIEW LAST TRANSACTION'
 26.00      A                                    3  2'Quantity'
 27.00      A            QTY            5  0     +1EDTCDE(Z)
 28.00      A                                   +5'Item'
 29.00      A            ITEM           2        +1
```

9. 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。

10. 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*                                             KCOMIT
  3.00      FTRNP    O   E                    DISK
  4.00      F*                                             KCOMIT
  5.00      FTRNL    IF  E           K        DISK
  6.00      F            TRNR                              KRENAMETRNR1
  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                   EXFMTREVW
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. CRTRPGPGM 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を示すメッセージが表示されるはずです。

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。直前に入力された BB 品目を示す検討画面が表示されるはずです。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。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 CMRLIB ライブラリー内に作成します。少なくとも 5000KB 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できるだけ長くして、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、JRNTEST 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に使用されるので、MNGRCV(*SYSTEM) DLTRCV(*YES) を指定してください。JRNRCV パラメーターには、ステップ 16 (82ページ参照) 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/JRNTEST) を指定した物理ファイル・ジャーナル開始 (STRJRNPF) 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、レコードの事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、ジャーナル処理された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 バッファーに保持されていることによります。

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。

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。

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、事前イメージと事後イメージ (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

数量	品目
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 操作が行われるはずです。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。
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。

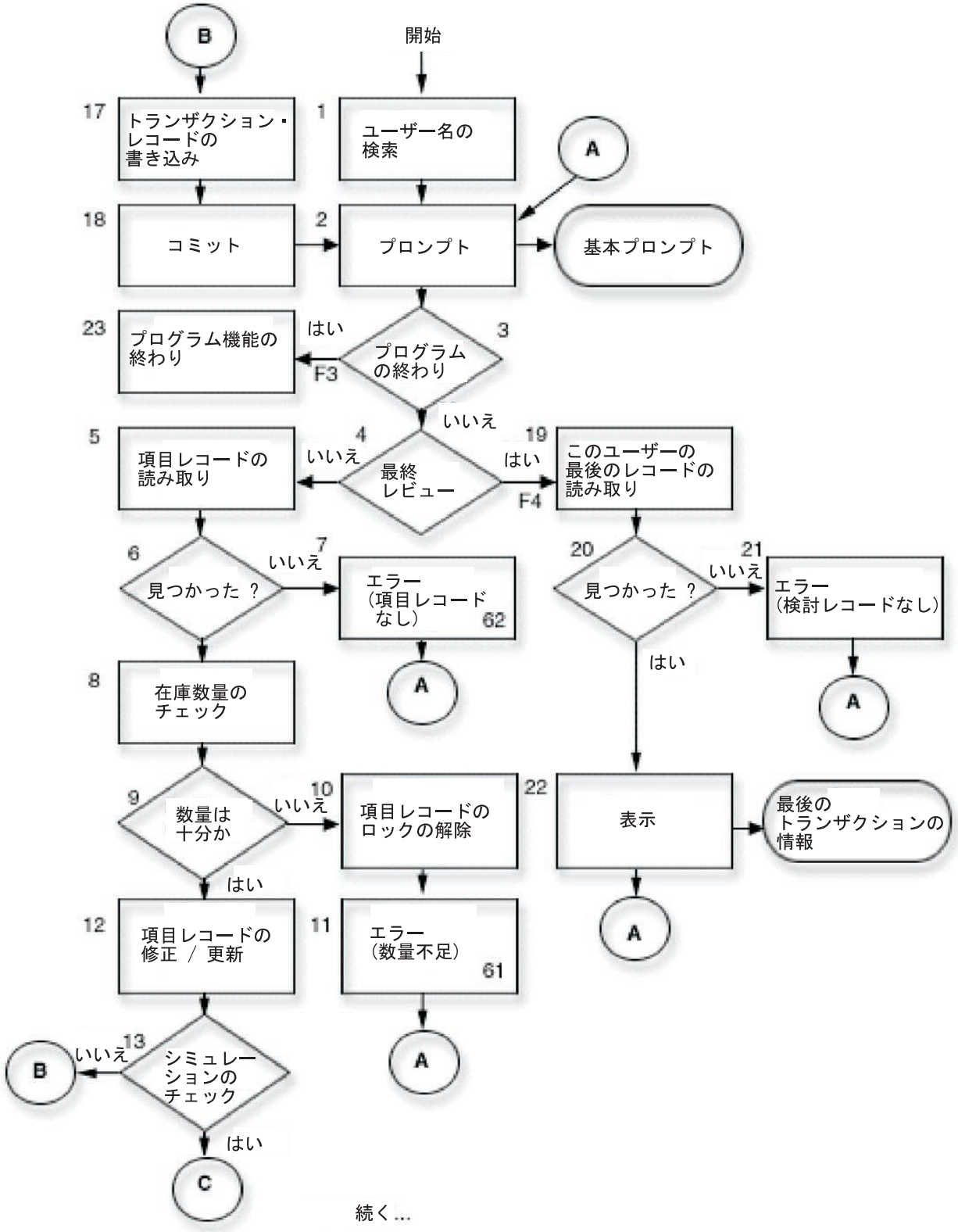
    同じタイプのロールバック処理が行われますが、今回のロールバックはプログラムではなく、システムによって行われます。RB 項目は、実行管理機能異常終了プログラムの QWTPITPP によって書き出されます。

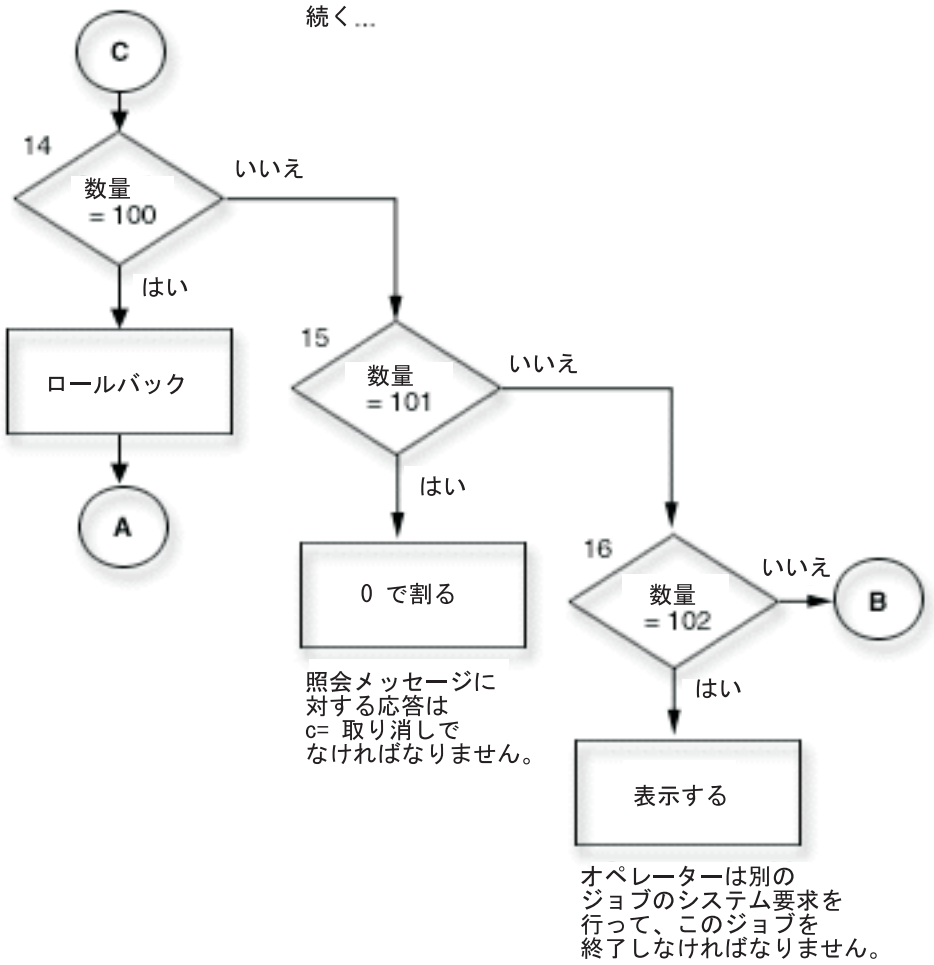
ここではコミットメント制御の 2 つの基本的な機能を使用しました。アプリケーションのコミットメント制御を続けるか、あるいは次のような他のいくつかの機能を試みることができます。

- 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

以下のイメージは、コミットメント制御の実習問題のフローを示しています。フローを進めながら、番号が付いたブロックをクリックして、ロジックのフローに対応した各ステップを参照してください。ステップのすべてを参照するために、実習プログラムのロジックのフローに対応したステップに進むこともできます。


B
開始
17
トランザクション・
レコードの
書き込み
1
ユーザー名の
検索
A
18
コミット
2
プロンプト
基本プロンプト
23
プログラム機能の
終わり
はい
F3
3
プログラム
の終わり
いいえ
5
項目レコードの
読み取り
4
いいえ
最終
レビュー
19
はい
F4
このユーザーの
最後のレコードの
読み取り
6
見つかった ?
7
いいえ
エラー
(項目レコード
なし)
62
20
見つかった ?
21
いいえ
エラー
(検討レコードなし)
A
A
はい
8
在庫数量の
チェック
9
数量は
十分か
10
いいえ
項目レコードの
ロックの解除
22
表示
最後の
トランザクションの
情報
はい
12
項目レコードの
修正 / 更新
11
エラー
(数量不足)
61
A
A
13
シミュレー
ションの
チェック
いいえ
B
はい
C
続く...



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 (87ページ参照) 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 (87ページ参照) 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、ステップ 2 (87ページ参照) 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 (87ページ参照)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 (87ページ参照) に戻ります。
23. プログラム終了機能を実行します。

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: 重要な法的情報については、コード例の特記事項をお読みください。

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。

たとえば、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すで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すでに存在していることを想定しています。プログラムは次のことを行います。

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。ワークステーション・ユーザーは、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てから、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 PRODCT
```

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            PRODCT         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
```

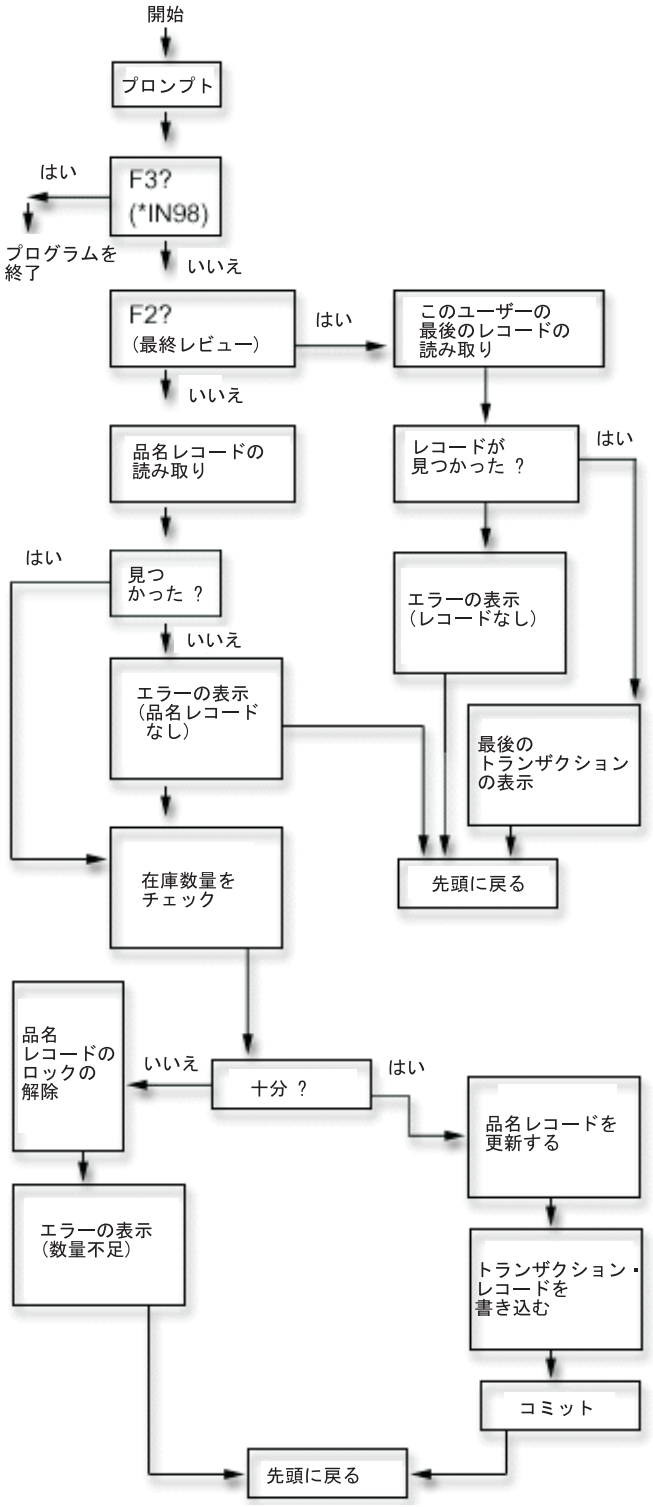
プログラムで使用される表示装置ファイル PRDISSD の DDS

```
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            PRODCT    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            PRODCT    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。

プログラムのフロー



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、データ入力が停止した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 UDATE     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            PRODCT    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                      COMIT
  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す。)

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 が使用された場合には、不足した数量があると、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 : アプリケーションを開始する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。

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。

通知オブジェクトを使用するためには、次の方法があります。

- 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、次のこと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。
- 通知オブジェクト情報を削除するときには、次の情報の使用直後に障害が起こるような状況を処理するように計画する必要があります。
  - 情報が即時に削除された場合、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ります。
  - 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ことになる。
- 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。

通知オブジェクトを使用するためのコード例は、以下を参照してください。

注: 重要な法的情報については、コード例の特記事項をお読みください。

- プログラムごとに固有の通知オブジェクト
- 全プログラム用の単一通知オブジェクト

例 : プログラムごとに固有の通知オブジェクト

このトピックでは、各プログラムを再始動するための固有の通知オブジェクトの使用法について、サンプル・コードと説明を提供しています。

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。

物理ファイル PRDMSTP の DDS を知るには、物理ファイル PRDMSTP の DDS (89ページ参照) を参照してください。

注: 重要な法的情報については、コード例の特記事項をお読みください。

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            PRODCT         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 PRODCT
  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            PRODCT   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(PRODCT)
 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            PRODCT    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。

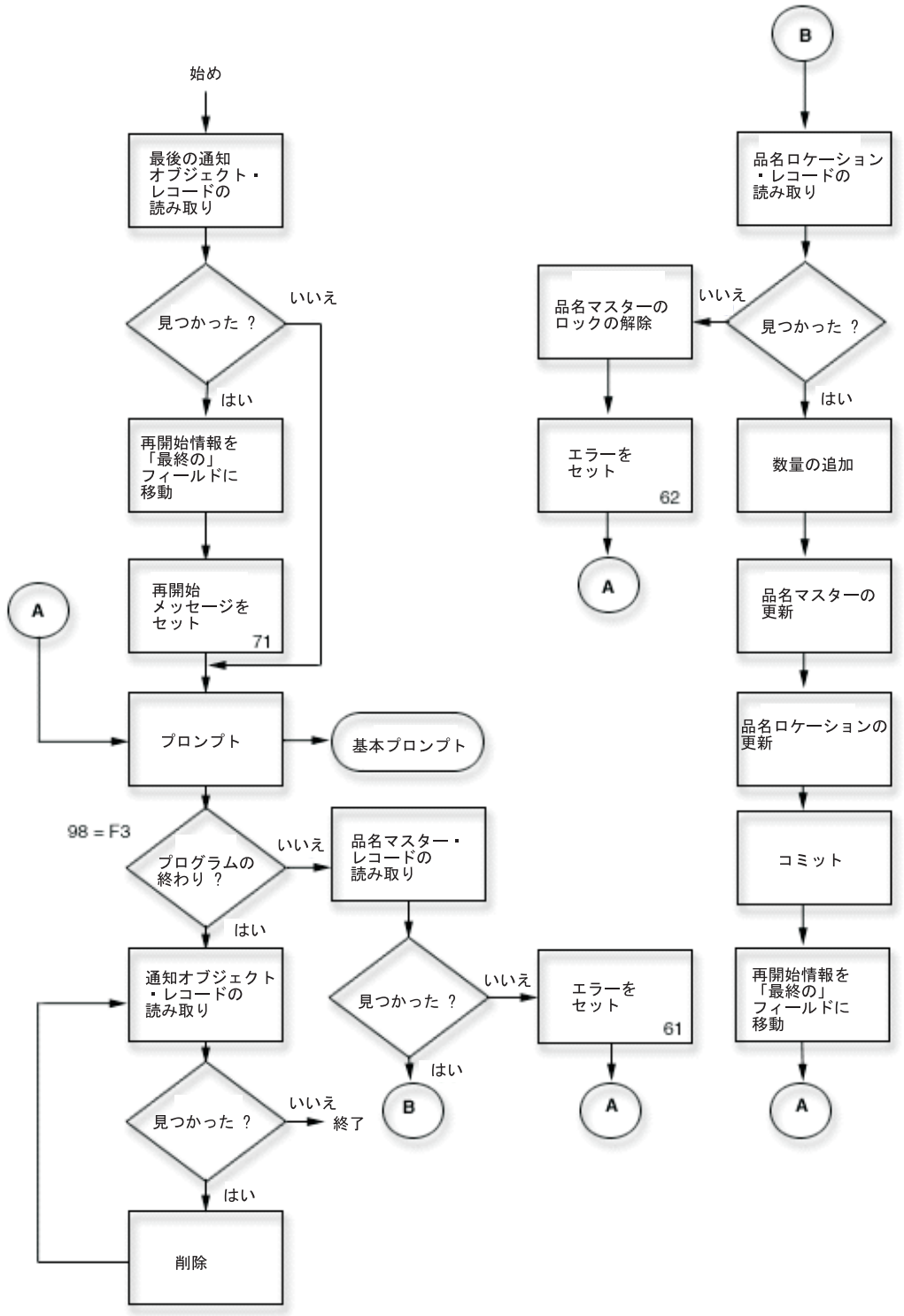
- 開始時に、プログラムは通知オブジェクトをランダムに処理し、特定のキーのレコードが存在する場合にはそのレコードを表示します。
  - 複数のレコードが存在する場合には、PRDRCTP ファイルが LIFO の順序になっているため、このキーの最後のレコードが使用される。
  - レコードが存在しない場合は、トランザクションが割り込まれなかったので再開する必要はない。
  - 最初のコミット操作が成功する前にプログラムが正常に実行されなかった場合には、プログラムは再開が必要であるとは見なされない。
- 通知オブジェクトを消去するためのルーチンは、プログラムの終わりにあります。
  - 複数の障害があった場合には、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。

プログラムのフロー



この例の RPG ソース・コードを以下に示します。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           L00P      TAG
 28.00      C                     EXFMTPROMPT
 29.00      C   98                GOTO END                         End of pgm
 30.00      C           PRODCT    CHAINPRDMSTR             61     Not found
 31.00      C   61                GOTO L00P
 32.00      C           KEY       KLIST
 33.00      C                     KFLD           PRODCT
 34.00      C                     KFLD           LOCATN
 35.00      C           KEY       CHAINPRDLOCR             62     Not found
 36.00      C   62                DO
 37.00      C                     EXCPTRLSMST                      Release lck
 38.00      C                     GOTO L00P
 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 L00P
 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 PRODCT    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
```

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と、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という利点があります。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、先のプログラムは、外部ではなくプログラム内で記述された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      PRODCT
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。

例 : プログラムごとに固有の通知オブジェクトには、通知オブジェクトの使用について、その他の例が説明されています。

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。

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	10 文字
プログラム識別コード	10 文字
再開のための情報	文字フィールド (これには、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、長さが 200 であると想定しています。)

このファイルは、SHARE(*YES) として作成されます。ファイルの始めの 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。)

注: 重要な法的情報については、コード例の特記事項をお読みください。

以下では、標準処理プログラムのコード例が示されています。

- 例 : 標準処理プログラムのコード
- 例 : 標準コミット処理プログラム
- 例 : アプリケーション再始動を決定するための標準処理プログラムの使用

例 : 標準処理プログラムのコード

次に標準処理プログラムの使用例を示します。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。

このプログラムのフローは、処理のフローを参照してください。

注: 重要な法的情報については、コード例の特記事項をお読みください。

アプリケーション・プログラム例

```
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 PRODCT
 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                     MOVEL'PRDRC2'  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            PRODCT   CHAINPRDMSTR              61    Not found
 48.00      C   61                GOTO LOOP
 49.00      C            KEY      KLIST
 50.00      C                     KFLD           PRODCT
 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 PRODCT    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
```

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、再開時にすべての情報を除去するために、呼び出しプログラムが正常に完了した場合に実行されます。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。

注: 重要な法的情報については、コード例の特記事項をお読みください。

以下には、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

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

  1.00      FNFYOBJP UF  E           K        DISK                    A
  2.00      ICMTID       DS
  3.00      I                                       1  10 UNQUSR
  4.00      I                                      11  20 UNQPGM
```

```
 5.00     I                                       21 220 BIGFLD
 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*  'O' for Open
14.00     C*
15.00     C            RQSCOD   IFEQ 'O'                        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'       RTNCOD            Found
37.00     C            LOOPl    TAG
38.00     C            KEY      READENFYOBJR                  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                     WRITENFYOBJR
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       CHAINNFYOBJR              51    Not found
54.00     C   51                MOVE '0'       RTNCOD
55.00     C   51                GOTO END
56.00     C                     MOVE '1'       RTNCOD            Found
57.00     C           LOOP2     TAG
58.00     C                     DELETNFYOBJR
59.00     C           KEY       READENFYOBJR                  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    SETLLNFYOBJR                  20 If equal
69.00     C  N20                MOVE '0'       RTNCOD
70.00     C  N20                GOTO END
71.00     C                     MOVE '1'       RTNCOD            Found
```

```
 72.00      C           LOOP3     TAG
 73.00      C           UNQUSR    READENFYOBJR                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
```

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

このトピックでは、異常 IPL の後で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に標準処理プログラムを使用する CL コード例を示します。

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。 たとえば、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の例です。

注: 重要な法的情報については、コード例の特記事項をお読みください。

初期プログラムの例

```
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            RTVJOBA     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
```

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

以下のページでは、コミットメント制御のトラブルシューティングを行うのに必要なタスクを示しています。

コミットメント制御エラー
この項では、エラーを引き起こす条件、コミットメント制御エラーをリストする条件、およびエラーに対処するメソッドを提供する条件を説明しています。

デッドロックの検出
このタスクのゴールは、デッドロック状態を検出することです。

通信障害後のトランザクションのリカバリー
このタスクのゴールは、リモート・システムとの通信の障害発生後に、そのリモート・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ことです。

コミットとロールバックの強制時点および再同期の取り消し時点
この項では、ロールバックまたはコミットをいつどのように強制するか、再同期をいつ取り消すかについて説明しています。

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、たと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。

コミット処理中に生じるエラー

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。詳細は、2 フェーズ・コミットのコミットメント定義 : 結果の待機なしを参照してください。
- トランザクションの状態。詳細は、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。

障害が、完全に修復不能か、またはその場に適した方法で修復できないような大災害によるものであるとき、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決定をする必要があります。ヒューリスティック決定は、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・キューに送られます。

以下の情報では、コミットメント制御エラーの処理の詳細について示されています。

- エラー状態
- 非エラー状態
- 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
- CALL コマンド後にモニターすべきエラー・メッセージ
- 通常のコミットまたはロールバック処理の障害

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は、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・レベルの定義をいったん開始すれば、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ので、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、すで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・レベルの定義をいったん開始すれば、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

以下の状態はエラー・メッセージを発生させると思われるかもしれません。しかし、コミットメント制御ではこれらの状態が許容されます。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にない。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分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にファイルをオープンする。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。

これらのメッセージは次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や、ジョブの終了時のコミットメント定義を終了する時に起きたエラーは、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、またはその両方がすで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 がロールバックをボートした。

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なければなりません。 たとえば、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。 たとえば、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。	• 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。 • 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。	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。	• 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。 • 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。 – CPD8353 – CPD8354 – CPD8355 – CPD8361 メッセージ CPF8363 がコミット処理の終了時に送られます。	メッセージをモニターし、必要に応じて処理する。
レコード・レベル入出力**ロールバック**に障害が起きた。	1. CPD8356 を戻す。 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。 3. 処理の終わりに CPF8359 を戻す。	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。	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。 • CPD8358 • CPD8359 • CPD835A • CPD8362 2. 処理を続行する。 3. 処理の終わりに CPF8359 を戻す。	メッセージをモニターし、必要に応じて処理する。

ジョブ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。

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いて、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します。

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. アクティブなジョブのリスト内で、ハングしているジョブを探し出します。ジョブの状況の判別を参照して、ジョブがハングしているかどうかを判別してください。
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。トランザクション範囲ロックをもつトランザクションの場合は、調べるステップについては、トランザクションのロックされたオブジェクトの表示を参照してください。ジョブ範囲ロックをもつトランザクションの場合は、詳細 : アクティブ・ジョブのプロパティーを参照してください。
3. ジョブがロックを待っ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、ハング・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は、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っているかどうかを調べます。

5. この待っているジョブが最初のハング・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。このジョブが最初のハング・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ことがわかれば、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ーを調べて、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ーにあるコミット・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. 関連情報を入手したら、ロールバックまたはコミットの強制操作を選択できます。

通信障害後のトランザクションのリカバリー

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 (たとえば、通信回線が切断される場合)、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、解放の必要なロックを保持していることもあります。

1. iSeries ナビゲーターで、処理中のトランザクションについて、コミットメント制御情報の表示を実施します。
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したら、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ーを調査してから、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。たとえば、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。

リカバリーの詳細については、コミットとロールバックの強制時点および再同期の取り消し時点を参照してください。

コミットとロールバックの強制時点および再同期の取り消し時点

コミットの強制、ロールバックの強制、または再同期の取り消しの決定は、**ヒューリスティック判定**と呼ばれます。ヒューリスティック判定とは、システムにトランザクションのコミットまたはロールバックを強制実行させるときに行う処置です。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して最終的な決定権は 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の決定権があります。 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る必要があります。

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ないときは、以下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. iSeries ナビゲーターで、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。次の状態のトランザクションを探します。

- **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、iSeries ナビゲーターを使用して、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。

たとえば、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。

コミットメント制御の関連情報

以下にリストしているのは、コミットメント制御のトピックに関連した iSeries 資料および IBM レッドブック (PDF 形式)、Web サイト、および Information Center トピックです。どの PDF も表示または印刷することができます。

資料

- AS/400 V3.0.5 COBOL/400 使用者の手引き  (約 425 ページ)
- AS/400 V3.0.5 RPG/400 使用者の手引き  (約 580 ページ)

レッドブック

- Connecting WebSphere to DB2 UDB Server  (約 458 ページ)
- Advanced Functions and Administration for DB2 Universal Database for iSeries  (約 372 ページ)
- Stored Procedures and Triggers on DB2 Universal Database for iSeries  (約 424 ページ)
- Striving for Optimal Journal Performance on DB2 Universal Database for iSeries  (約 184 ページ)

Web サイト

The Open Group (www.opengroup.org) 

その他の情報

- DB2 UDB for iSeries データベース・プログラミング
- SQL プログラミング 概念
- XA API
- ジャーナル管理

PDF を表示または印刷用にワークステーションに保管するには、以下の手順を実施します。

1. ブラウザーで、該当の PDF を右クリックします (上記のリンクの右クリック)。
2. 「**対象をファイルに保存... (Save Target As...)**」をクリックします。
3. PDF を保管しようとするディレクトリーにナビゲートします。
4. 「**保存 (Save)**」をクリックします。

これらの PDF を表示または印刷するために Adobe Acrobat Reader が必要な場合は、 Adobe Web サイト (www.adobe.com/products/acrobat/readstep.html)  からコピーをダウンロードすることができます。

Printed in Japan

日本アイ・ビー・エム株式会社
〒106-8711　東京都港区六本木3-2-12