



ノンストップデータベース

**HiRDB Version 10 構造型データベース機能 (UAP
開発編)**

解説・手引・文法・操作書

3020-6-579-70

前書き

■ 対象製品

●適用 OS : Red Hat(R) Enterprise Linux(R) Server 7 (64-bit x86_64), Red Hat(R) Enterprise Linux(R) Server 8 (64-bit x86_64), Red Hat(R) Enterprise Linux(R) Server 9 (64-bit x86_64)

P-8462-C5A1 HiRDB Structured Data Access Facility Version 10 10-09

P-8462-AEA1 HiRDB Structured Data Access Facility/Developer's Kit Version 10(64) 10-09

P-F8462-C5A11 HiRDB Structured Data Access Facility Extension for XDM/SD type Version 10 10-04

■ 輸出時の注意

本製品を輸出される場合には、外国為替及び外国貿易法の規制並びに米国輸出管理規則など外国の輸出関連法規をご確認の上、必要な手続きをお取りください。

なお、不明な場合は、弊社担当営業にお問い合わせください。

■ 商標類

記載の会社名、製品名などは、それぞれの会社の商標もしくは登録商標です。

■ 発行

2025年4月 3020-6-579-70

■ 著作権

All Rights Reserved. Copyright (C) 2019, 2025, Hitachi, Ltd.

変更内容

変更内容 (3020-6-579-70) HiRDB Structured Data Access Facility Version 10 10-09

追加・変更内容	変更個所
DML を使用してアクセスするレコードのデータ項目を基に、埋込み変数を作成する機能として#CBLFORM コマンドをサポートしました。	2.4.2

単なる誤字・脱字などはお断りなく訂正しました。

はじめに

このマニュアルは、HiRDB Structured Data Access Facility（以降、HiRDB/SD と略します）の構造型データベースを操作するインターフェース（DML）を使用して、COBOL 言語のユーザアプリケーションプログラム（UAP）を開発する方法について説明しています。

■ 対象読者

HiRDB/SD を使用して COBOL 言語の UAP を作成する方、UAP を実行する方（HiRDB クライアントを使用する方）を対象としています。

このマニュアルの記述は、次に示す知識があることを前提にしています。

- COBOL 言語のプログラミングの知識
- HiRDB の基礎的な知識
- Linux のシステム管理の基礎的な知識

■ マニュアルの構成

このマニュアルは、次に示す章と付録から構成されています。

第 1 章 UAP 開発の概要

UAP の形式、UAP の開発環境、UAP の実行環境、および UAP の開発の流れなどについて説明しています。

第 2 章 UAP の作成

DML による SDB データベースにアクセスする部分の COBOL ソースプログラムの作成方法について説明しています。

第 3 章 UAP の実行前準備（UAP のプリプロセス、コンパイル、リンクエージ）

UAP のプリプロセス、コンパイル、およびリンクエージの方法について説明しています。

第 4 章 UAP の実行環境の構築

HiRDB クライアントの環境設定方法、UAP をテストする際の UAP の実行方法、およびテスト環境から本番環境への UAP の移行方法について説明しています。

第 5 章 UAP の運用・保守

UAP の実行方法、UAP の再プリプロセスが必要なケース、および UAP の障害対策について説明しています。

第6章 DML プリプロセサ (pdsdbcbl コマンド)

DML プリプロセサ (pdsdbcbl コマンド) の機能と使い方について説明しています。

■ 関連マニュアル

このマニュアルの関連マニュアルを次に示します。必要に応じてお読みください。

HiRDB マニュアル

- HiRDB Version 10 構造型データベース機能 (3020-6-578)
- HiRDB Version 10 解説 (3020-6-551)
- HiRDB Version 10 システム導入・設計ガイド (UNIX(R)用) (3020-6-552)
- HiRDB Version 10 システム定義 (UNIX(R)用) (3020-6-554)
- HiRDB Version 10 システム運用ガイド (UNIX(R)用) (3020-6-556)
- HiRDB Version 10 コマンドリファレンス (UNIX(R)用) (3020-6-558)
- HiRDB Version 10 UAP 開発ガイド (3020-6-560)
- HiRDB Version 10 SQL リファレンス (3020-6-561)
- HiRDB Version 10 メッセージ (3020-6-562)

以降、HiRDB Version 10 のマニュアル名は、Version 10, (UNIX(R)用) を省略して表記しています。

関連製品

- COBOL2002 使用の手引 手引編 (3000-3-D08)
- COBOL85 使用の手引 (3000-3-354)
- COBOL85 言語 (3020-3-782)
- OpenTP1 Version 7 分散トランザクション処理機能 OpenTP1 プログラム作成リファレンス COBOL 言語編 (3000-3-D55)

OpenTP1 のマニュアルを本文中で参照させる場合は、「OpenTP1 Version 7 分散トランザクション処理機能 OpenTP1 プログラム作成リファレンス COBOL 言語編」を「OpenTP1 プログラム作成リファレンス COBOL 言語編」と表記します。

■ このマニュアルでの表記

このマニュアルでは製品名称および名称について次のように表記しています。ただし、それぞれのプログラムについての表記が必要な場合はそのまま表記しています。

製品名称または名称	表記
HiRDB Version 10	HiRDB または HiRDB サーバ
HiRDB Structured Data Access Facility/Run Time Version 10(64)	HiRDB Structured Data Access Facility/Run Time
HiRDB Structured Data Access Facility/Developer's Kit Version 10(64)	HiRDB Structured Data Access Facility/Developer's Kit
Linux(R)	Linux
Red Hat(R) Enterprise Linux(R) Server 7 (64-bit x86_64)	
Red Hat(R) Enterprise Linux(R) Server 8 (64-bit x86_64)	
Red Hat(R) Enterprise Linux(R) Server 9 (64-bit x86_64)	

- HiRDB 運用ディレクトリのパスを\$PDDIR と表記します。
- TCP/IP が規定する hosts ファイル（/etc/hosts ファイルも含む）を hosts ファイルと表記します。

■ このマニュアルで使用する略語

このマニュアルで使用する英略語の一覧を次に示します。

英略語	英字の表記
AFM	Attached File Management Program
Amazon EC2	Amazon Elastic Compute Cloud
API	Application Programming Interface
AWS	Amazon Web Services
COBOL	Common Business Oriented Language
DB	Database
DML	Data Manipulate Language
DNS	Domain Name System
FMB	File Manager for Banks
JIS	Japanese Industrial Standard code
OLTP	On-Line Transaction Processing
OS	Operating System
RD	Relational Database
SJIS	Shift JIS

英略語	英字の表記
SPP	Service Providing Program
SUP	Service Using Program
UAP	User Application Program

■ このマニュアルで使用する記号

形式および説明で使用する記号を次に示します。ここで説明する文法記述記号は、説明のための記号なので実際には記述しないでください。

文法記述記号	意味
[]	この記号で囲まれている項目は省略できます。 (例) pdsdbcbl [-Xb] これは、pdsdbcbl と指定するか、または pdsdbcbl -Xb と指定できることを意味しています。
...	この記号直前の項目を繰り返して指定できます。 (例) SDB データベース名 [,SDB データベース名] ... これは、「SDB データベース名」を繰り返し指定できることを意味しています。
~	この記号のあとにユーザ指定値の属性を示します。
< >	ユーザ指定値の構文要素記号を示します。
(())	ユーザ指定値の指定範囲を示します。

■ このマニュアルで使用する構文要素記号

このマニュアルで使用する構文要素記号を次に示します。

構文要素記号	意味
<識別子>	指定できる文字の規則については、マニュアル「HiRDB 構造型データベース機能」の「名前の規則」を参照してください。
<パス名>*	/, 英数字, ピリオド (.), #, @で構成される文字列

注

すべて半角文字を使用してください。

注※

パス名は使用している OS に依存します。

■ KB (キロバイト) などの単位表記について

1KB (キロバイト), 1MB (メガバイト), 1GB (ギガバイト), 1TB (テラバイト) はそれぞれ 1,024 バイト, $1,024^2$ バイト, $1,024^3$ バイト, $1,024^4$ バイトです。

目次

前書き 2

変更内容 3

はじめに 4

1 UAP 開発の概要 12

1.1 UAP の記述言語と形式 13

1.2 UAP の開発環境 14

1.3 UAP の実行環境および運用形態 15

1.3.1 UAP の実行環境 15

1.3.2 UAP の運用形態 15

1.4 UAP の開発の流れ 17

1.4.1 UAP の設計から実行までの流れ 17

1.4.2 目的別の参照先一覧 20

2 UAP の作成 23

2.1 COBOL ソースプログラムの基本構成 24

2.1.1 見出し部 (IDENTIFICATION DIVISION) 25

2.1.2 環境部 (ENVIRONMENT DIVISION) 25

2.1.3 データ部 (DATA DIVISION) 25

2.1.4 手続き部 (PROCEDURE DIVISION) 30

2.1.5 プログラム終わり見出し (END PROGRAM) 31

2.2 COBOL ソースプログラムを作成する際の考慮点 32

2.3 SDB データベース節の記述 33

2.3.1 SDB データベース節の記述例 33

2.3.2 SDB データベース節の記述内容と構文規則 33

2.3.3 SDB データベース節で指定する埋込み変数の宣言 35

2.4 埋込み変数の宣言 38

2.4.1 埋込み変数とは 38

2.4.2 埋込み変数の宣言方法 38

2.4.3 埋込み変数の使用例 44

2.5 DML によるレコードの検索 53

2.5.1 レコードの検索 53

2.6 DML によるレコードの更新, 格納, または削除 55

2.6.1 レコードの更新 55

2.6.2 レコードの格納 56

2.6.3	レコードの削除	57
2.7	DML の実行結果の判定処理	59
2.7.1	DML の実行結果の判定処理の例	59
2.7.2	SQLCODE の値と意味	60
2.7.3	DML のエラーを検出したときの対処方法	61
2.7.4	SQL 連絡領域の構成と内容	61
2.8	トランザクション制御	64
2.9	排他制御	65
2.10	COBOL ソースプログラムの記述規則	66
2.10.1	文字コードと改行コード	66
2.10.2	ソースプログラムの正書法	67
2.10.3	翻訳単位（最外側のプログラム）	67
2.10.4	プログラムの入れ子の上限	68
2.10.5	名前の記述規則	68
2.10.6	宣言が必要な節	68
2.10.7	登録集原文の制限	69
2.10.8	DML の記述規則	70
2.10.9	SDB データベースの特定規則	72
2.11	COBOL ソースプログラムのコーディング例	74
2.11.1	PAD チャート	74
2.11.2	コーディング例	79
2.12	DML と SQL の両方を実行する UAP を作成する場合の考慮点	89
2.12.1	UAP ソースファイルの構成	89
2.12.2	DML と SQL の両方を実行する UAP のトランザクション制御	89
2.12.3	COBOL ソースプログラムのコーディング例（DML と SQL の両方を実行する UAP の場合）	90
2.13	2 進項目をビッグエンディアン形式にする UAP を作成する場合の注意事項	108
2.14	性能向上、操作性向上に関する機能	109

3	UAP の実行前準備（UAP のプリプロセス、コンパイル、リンクエージ）	110
3.1	プリプロセス、コンパイル、およびリンクエージの実行環境の構築	111
3.2	UAP のプリプロセス、コンパイル、リンクエージの流れ	112
3.3	プリプロセスの実行	114
3.3.1	プリプロセスを実行するための準備作業	114
3.3.2	プリプロセスの実行例	114
3.3.3	プリプロセスエラーが発生した場合の対処	116
3.4	コンパイルおよびリンクエージの実行	117
3.4.1	コンパイルおよびリンクエージを実行するための準備作業	117
3.4.2	ccbl2002 コマンドの指定形式	118
3.4.3	コンパイルおよびリンクエージの実行例	119

3.4.4	コンパイルエラーまたはリンクエラーが発生した場合の対処	119
3.5	DML と SQL を実行する UAP をプリプロセス、コンパイル、およびリンクする場合	120
3.5.1	UAP のプリプロセス、コンパイル、リンクの流れ	120
3.5.2	プリプロセス、コンパイル、リンクの実行例	120

4 UAP の実行環境の構築 123

4.1	HiRDB クライアントの環境設定	124
4.1.1	HiRDB クライアントのインストール	124
4.1.2	環境変数の設定	124
4.1.3	クライアント環境定義の設定	124
4.2	UAP のテストの実行	126
4.3	テスト環境から本番環境への UAP の移行	127

5 UAP の運用・保守 129

5.1	UAP の実行	130
5.2	UAP の再プリプロセス、再コンパイル、再リンクが必要なケース	131
5.3	UAP の障害対策	132

6 DML プリプロセサ (pdsdbcbl コマンド) 133

6.1	機能	134
6.1.1	UAP のプリプロセス	134
6.1.2	プリプロセス時にチェックされない項目	136
6.2	コマンドの形式	137
6.3	プリプロセス実行前の準備作業	139
6.3.1	環境変数の設定	139
6.3.2	SDB ディレクトリ情報ファイルの準備	140
6.4	注意事項	141
6.5	リターンコード	142
6.6	トラブルシューティング	143
6.7	使用例	144
6.8	pdsdbcbl コマンドが解析する COBOL 命令	146

索引 147

1

UAP 開発の概要

この章では、UAP の形式、UAP の開発環境、UAP の実行環境、および UAP の開発の流れなどについて説明します。

1.1 UAP の記述言語と形式

SDB データベース種別が SD FMB の SDB データベースは、UAP を使用して操作できます。SDB データベースにアクセスする UAP の記述言語と形式を次に示します。

- UAP の記述言語

COBOL85

- UAP の形式

埋込み型 UAP

COBOL ソースプログラム中に、SDB データベースを操作する DML を直接記述する形式の UAP を、埋込み型 UAP といいます。COBOL ソースプログラム中に、次の表に示す DML を記述できます。

表 1-1 SDB データベースを操作する DML の一覧

項目番	分類	DML	機能
1	操作系 DML	ERASE	レコード実現値を削除します。
2		FETCH	レコードを検索して、レコード実現値を取得します。また、検索したレコードに対して位置づけを行います。
3		FIND	レコード実現値に位置指示子を位置づけます。
4		MODIFY	1 レコード実現値を更新します。
5		STORE	1 レコード実現値を格納します。
6		GET	1 レコード実現値を取得します。

各 DML の機能説明や記述形式については、マニュアル「HiRDB 構造型データベース機能」の「DML リファレンス」を参照してください。

注意事項

DML を使用して操作できる SDB データベースは、SDB データベース種別が SD FMB の SDB データベースです。SDB データベース種別が 4V FMB または 4V AFM の SDB データベースは、DML を使用して操作することはできません。

埋込み型 UAP と HiRDB/SD の間では、次に示すインターフェース領域を使用して情報を受け渡しします。

- SQL 連絡領域

SQL 連絡領域には、DML の実行結果の詳細情報が格納されます。DML の実行結果の判定処理をする際に使用します。

- 埋込み変数

埋込み変数には、DML に指定する値や、DML の実行結果が格納されます。DML 中に埋込み変数を指定することで、埋込み型 UAP と HiRDB/SD との間で値の受け渡しをします。

1.2 UAP の開発環境

DML を記述した UAP を開発する際、UAP のプリプロセス、コンパイル、およびリンクエージを実行します。そのためには、次に示す環境のマシンが必要になります。

- **OS**

次のどれかの OS が必要です。

- Red Hat Enterprise Linux Server 7 (64-bit x86_64)
- Red Hat Enterprise Linux Server 8 (64-bit x86_64)
- Red Hat Enterprise Linux Server 9 (64-bit x86_64)

- **HiRDB クライアント**

次の製品が必要です。

- HiRDB Structured Data Access Facility/Developer's Kit

- **COBOL 製品**

次の製品が必要です。COBOL2002 の COBOL コンパイラを使用します。

- COBOL2002 Net Server Suite(64)

上記の製品は、「Linux 版 COBOL2002 サポートサービス」の「レガシ文字コード Shift JIS サポートオプション」に対応しています。

COBOL2002 レガシ文字コード Shift-JIS サポートオプションを適用し、Shift-JIS のオブジェクトを生成してください。

- **OLTP 製品**

OLTP 環境下で実行する UAP を開発する場合、次の製品が必要です。

- OpenTP1

- **文字コード**

使用する文字コードは、シフト JIS 漢字コード (SJIS) です。

HiRDB サーバで使用する文字コードは、プリプロセス時の文字コードと一致させてください。

1.3 UAP の実行環境および運用形態

ここでは、UAP の実行環境および運用形態について説明します。

1.3.1 UAP の実行環境

UAP の実行環境は、次に示す条件をすべて満たす必要があります。

- OS

次のどれかの OS が必要です。

- Red Hat Enterprise Linux Server 7 (64-bit x86_64)
- Red Hat Enterprise Linux Server 8 (64-bit x86_64)
- Red Hat Enterprise Linux Server 9 (64-bit x86_64)

- HiRDB サーバ

HiRDB Structured Data Access Facility

- HiRDB クライアント

次のどちらかの製品が必要です。

- HiRDB Structured Data Access Facility/Run Time
- HiRDB Structured Data Access Facility/Developer's Kit

- COBOL 製品

次のどちらかの製品が必要です。

- COBOL2002 Net Server Suite(64)
- COBOL2002 Net Server Runtime(64)

上記の製品は、「Linux 版 COBOL2002 サポートサービス」の「レガシ文字コード Shift JIS サポートオプション」に対応しています。

- OLTP 製品

OLTP 環境下で UAP を実行する場合、次の製品が必要です。

- OpenTP1

- 文字コード

使用する文字コードは、シフト JIS 漢字コード (SJIS) です。

1.3.2 UAP の運用形態

UAP の運用形態は、次のどちらかになります。

- **OpenTP1 環境下で UAP を実行する**

OpenTP1 環境下で UAP を実行する場合、HiRDB サーバへの接続および切り離しは OpenTP1 が制御し、トランザクション制御は OpenTP1 の API で行います。

OpenTP1 環境下での UAP の運用形態については、マニュアル「HiRDB UAP 開発ガイド」の「UAP の動作環境」の「OLTP 下の UAP をクライアントとする運用形態」を参照してください。

- **UAP の実行可能ファイルを直接起動して UAP を実行する**

UAP の実行可能ファイルを直接起動して UAP を実行する場合、HiRDB サーバへの接続および切り離し、トランザクション制御は SQL で行います。

UAP の実行可能ファイルを直接起動する UAP の運用形態については、マニュアル「HiRDB UAP 開発ガイド」の「UAP の動作環境」の次の箇所を参照してください。

- 「サーバマシンとは別のマシンをクライアントとする運用形態」
- 「HiRDB サーバと同一のサーバマシンでクライアントを実行する運用形態」

1.4 UAP の開発の流れ

ここでは、UAP の開発の流れについて説明します。

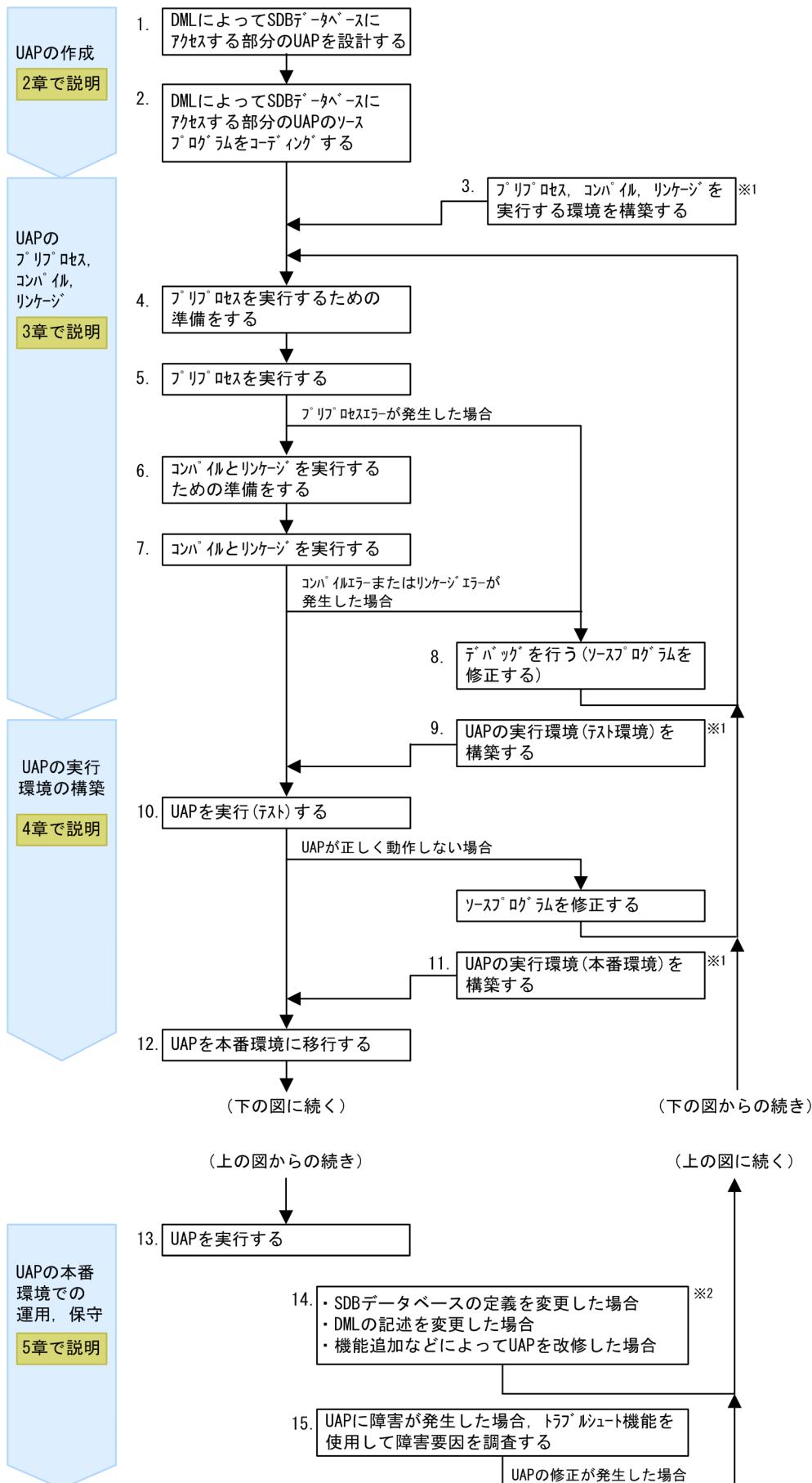
1.4.1 UAP の設計から実行までの流れ

UAP の設計から実行までの流れを次の図に示します。

■ ポイント

このマニュアルでは、DML によって SDB データベースにアクセスする部分の UAP の設計方法と作成方法について説明しています。そのため、下記の UAP の開発の流れの図は、DML によって SDB データベースにアクセスする部分の UAP 開発の流れを示しています。

図 1-1 UAP の開発の流れ



注※1

これらの環境の構築作業は1回だけ実施します。

注※2

これらの操作を実行した場合、SDB データベースにアクセスする部分の UAP の再プリプロセス、再コンパイル、および再リンクエージが必要になります。

■各作業項目のマニュアル中の参照先

「[図 1-1 UAP の開発の流れ](#)」の各作業項目は、次の表に示す参照先で説明しています。「[図 1-1 UAP の開発の流れ](#)」で示している項番は、次の表の項番と対応しています。

表 1-2 UAP 開発時の各作業項目の参照先

項番	作業項目	参照先
1	DML によって SDB データベースにアクセスする部分の UAP を設計する	「 2. UAP の作成 」
2	DML によって SDB データベースにアクセスする部分の UAP のソースプログラムをコーディングする	<ul style="list-style-type: none">「2.10 COBOL ソースプログラムの記述規則」「2.11 COBOL ソースプログラムのコーディング例」「2.12.3 COBOL ソースプログラムのコーディング例 (DML と SQL の両方を実行する UAP の場合)」
3	プリプロセス、コンパイル、リンクエージを実行する環境を構築する	「 3.1 プリプロセス、コンパイル、およびリンクエージの実行環境の構築 」
4	プリプロセスを実行するための準備をする	「 3.3.1 プリプロセスを実行するための準備作業 」
5	プリプロセスを実行する	「 3.3.2 プリプロセスの実行例 」
6	コンパイルとリンクエージを実行するための準備をする	「 3.4.1 コンパイルおよびリンクエージを実行するための準備作業 」
7	コンパイルとリンクエージを実行する	<ul style="list-style-type: none">「3.4.2 ccbl2002 コマンドの指定形式」「3.4.3 コンパイルおよびリンクエージの実行例」
8	デバッグを行う (ソースプログラムを修正する)	<ul style="list-style-type: none">プリプロセスエラーが発生した場合 「3.3.3 プリプロセスエラーが発生した場合の対処」コンパイルエラー、またはリンクエージエラーが発生した場合 「3.4.4 コンパイルエラーまたはリンクエージエラーが発生した場合の対処」
9	UAP の実行環境 (テスト環境) を構築する	「 4.1 HiRDB クライアントの環境設定 」
10	UAP を実行 (テスト) する	「 4.2 UAP のテストの実行 」
11	UAP の実行環境 (本番環境) を構築する	「 4.1 HiRDB クライアントの環境設定 」
12	UAP を本番環境に移行する	「 4.3 テスト環境から本番環境への UAP の移行 」
13	UAP を実行する	「 5.1 UAP の実行 」

項目番	作業項目	参照先
14	次のことが発生した場合 <ul style="list-style-type: none"> • SDB データベースの定義を変更した場合 • DML の記述を変更した場合 • 機能追加などによって UAP を改修した場合 	「5.2 UAP の再プリプロセス、再コンパイル、再リンクが必要なケース」
15	UAP に障害が発生した場合、トラブルシュート機能を使用して障害要因を調査する	「5.3 UAP の障害対策」

1.4.2 目的別の参照先一覧

UAP の設計から実行までの作業の目的別の参照先一覧を次の表に示します。

表 1-3 目的別の参照先一覧

分類	知りたいこと	参照先
UAP の設計および作成	SDB データベース節	SDB データベース節に記述する内容や、記述例を知りたい
	埋込み変数	埋込み変数の宣言方法、宣言例、および宣言する際の規則を知りたい
	DML	レコードを検索する方法を知りたい
		レコードを更新、格納、または削除する方法を知りたい
		DML の記述規則または文法を知りたい
		• 「2.10.8 DML の記述規則」 • マニュアル「HiRDB 構造型データベース機能」の「DML リファレンス」
		DML の実行結果の判定処理について知りたい
		「2.7 DML の実行結果の判定処理」
		SQLCODE の値とその意味を知りたい
		「2.7.2 SQLCODE の値と意味」
		SQL 連絡領域について知りたい
	トランザクション制御	トランザクションのコミット、ロールバックについて知りたい
	排他制御	排他制御について知りたい
	COBOL ソースプログラムの構成、記述規則、注意事項	COBOL ソースプログラムの構成、記述規則について知りたい
		• 「2.1 COBOL ソースプログラムの基本構成」 • 「2.10 COBOL ソースプログラムの記述規則」

分類	知りたいこと	参照先
	COBOL ソースプログラムを作成する際の考慮点、注意事項について知りたい	<ul style="list-style-type: none"> 「2.2 COBOL ソースプログラムを作成する際の考慮点」 「2.13 2 進項目をビッグエンディアン形式にする UAP を作成する場合の注意事項」
COBOL ソースプログラムの例	OpenTP1 環境下で実行する UAP の COBOL ソースプログラムの記述例を知りたい	「2.11 COBOL ソースプログラムのコーディング例」
	DML と SQL の両方を実行する UAP の COBOL ソースプログラムの記述例を知りたい	「2.12.3 COBOL ソースプログラムのコーディング例 (DML と SQL の両方を実行する UAP の場合)」
性能向上、操作性向上	<p>性能向上、操作性向上に関する次の機能を使用できるかどうかを知りたい</p> <ul style="list-style-type: none"> 自動再接続機能 ロック転送機能 複数接続機能 マルチスレッド対応 	「2.14 性能向上、操作性向上に関する機能」
UAP のプリプロセス、コンパイル、リンクエージ	プリプロセスを実行する前の準備作業を知りたい	<ul style="list-style-type: none"> 「3.1 プリプロセス、コンパイル、およびリンクエージの実行環境の構築」 「3.3.1 プリプロセスを実行するための準備作業」
	プリプロセスの実行方法を知りたい	<ul style="list-style-type: none"> 「3.3.2 プリプロセスの実行例」 「3.5.2 プリプロセス、コンパイル、リンクエージの実行例」
	プリプロセスエラーが発生したときの対処方法を知りたい	「3.3.3 プリプロセスエラーが発生した場合の対処」
コンパイル、リンクエージ	コンパイル、リンクエージを実行する前の準備作業を知りたい	<ul style="list-style-type: none"> 「3.1 プリプロセス、コンパイル、およびリンクエージの実行環境の構築」 「3.4.1 コンパイルおよびリンクエージを実行するための準備作業」
	コンパイル、リンクエージの実行方法を知りたい	<ul style="list-style-type: none"> 「3.4.2 ccbl2002 コマンドの指定形式」 「3.4.3 コンパイルおよびリンクエージの実行例」 「3.5.2 プリプロセス、コンパイル、リンクエージの実行例」
	コンパイルエラーまたはリンクエージエラーが発生したときの対処方法を知りたい	「3.4.4 コンパイルエラーまたはリンクエージエラーが発生した場合の対処」

分類	知りたいこと	参照先
UAP の実行環境の構築 (HiRDB クライアントの環境設定)	インストール, 環境設定方法	「4.1 HiRDB クライアントの環境設定」
	クライアント環境定義	「4.1.3 クライアント環境定義の設定」
UAP のテスト, 運用, 保守	UAP の実行環境	「4.3 テスト環境から本番環境への UAP の移行」
	UAP の実行	UAP の実行時に設定する環境変数, およびクライアント環境定義について知りたい
		UAP の実行方法を知りたい
	UAP の再プリプロセス	UAP の再プリプロセスが必要となるケースを知りたい
	トラブルシュート	UAP に障害が発生した場合に, 障害要因を調査するために使用できる機能について知りたい
		「5.1 UAP の実行」
		「5.2 UAP の再プリプロセス, 再コンパイル, 再リンクが必要なケース」
		「5.3 UAP の障害対策」

2

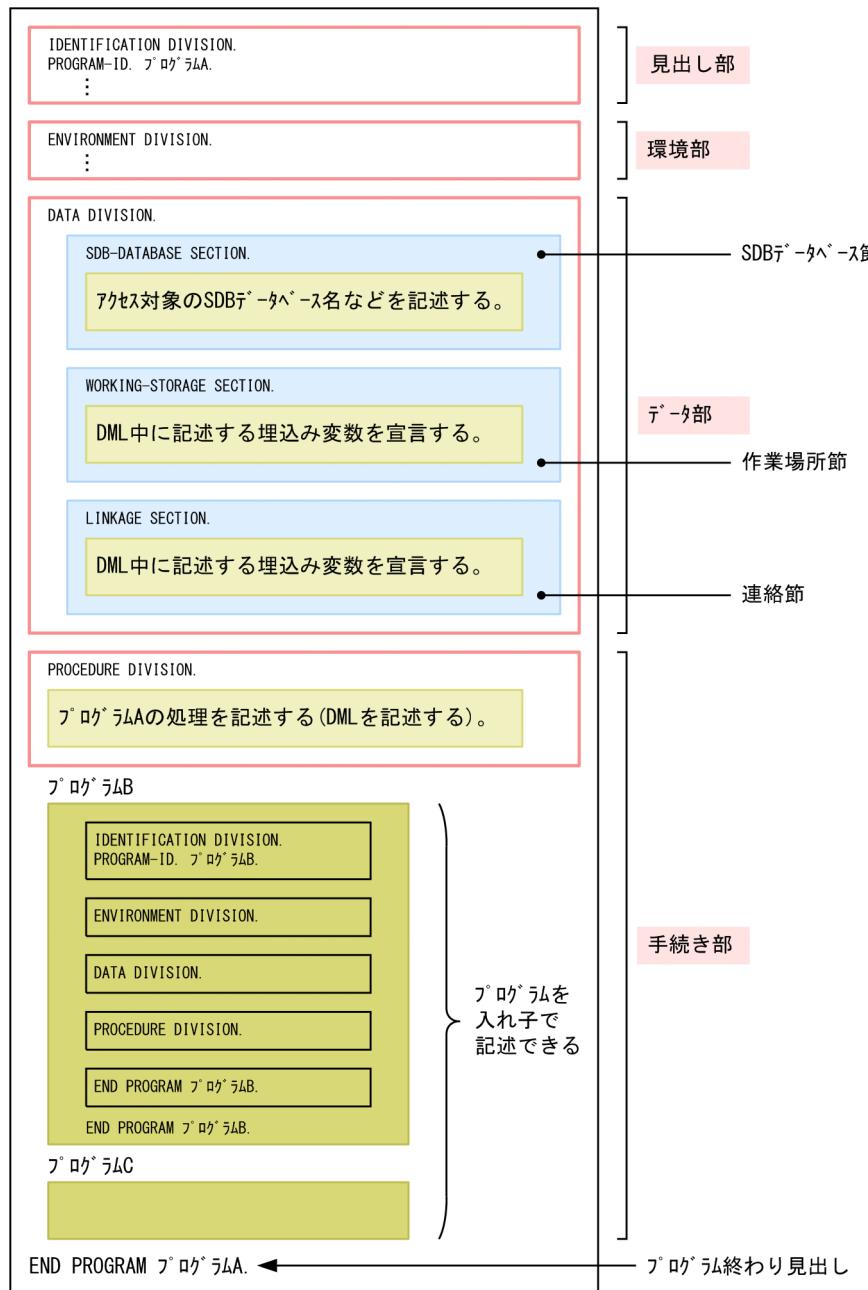
UAP の作成

この章では、DML による SDB データベースにアクセスする部分の COBOL ソースプログラムの作成方法について説明します。

2.1 COBOL ソースプログラムの基本構成

COBOL ソースプログラム中に DML を記述して SDB データベースにアクセスする場合、データ部でアクセス対象の SDB データベース名の指定と埋込み変数の宣言を行い、手続き部に DML を記述します。SDB データベースにアクセスする部分の COBOL ソースプログラムの基本構成を次の図に示します。

図 2-1 SDB データベースにアクセスする部分の COBOL ソースプログラムの基本構成



上記の図で示している見出し部、環境部、データ部、手続き部、およびプログラム終わり見出しについて説明します。

2.1.1 見出し部 (IDENTIFICATION DIVISION)

見出し部には、COBOL ソースプログラムの見出しどとなる情報を記述します。見出し部の記述規則については、マニュアル「COBOL85 言語」の「見出し部」を参照してください。

「IDENTIFICATION」を「ID」と省略できます。

(1) プログラム名段落の見出し (PROGRAM-ID.)

「PROGRAM-ID.」をプログラム名段落の見出しとして記述します。

プログラム名には、数字、または下線で始まるプログラム名を指定できません。

2.1.2 環境部 (ENVIRONMENT DIVISION)

環境部には、構成節と入出力節を記述します。環境部の記述規則については、マニュアル「COBOL85 言語」の「環境部」を参照してください。

2.1.3 データ部 (DATA DIVISION)

データ部には次に示す節を指定します。

- SDB データベース節 (SDB-DATABASE SECTION)
- 作業場所節 (WORKING-STORAGE SECTION)
- 連絡節 (LINKAGE SECTION)

(1) SDB データベース節 (SDB-DATABASE SECTION)

この節には、アクセス対象の SDB データベース名などを記述します。SDB データベース節の記述規則については、「[2.3 SDB データベース節の記述](#)」を参照してください。

(2) 作業場所節 (WORKING-STORAGE SECTION)

この節では、DML 中に記述する埋込み変数を宣言します。埋込み変数の宣言は、LINKAGE SECTION (連絡節) でも行うことができます。

埋込み変数の宣言方法については、「[2.4 埋込み変数の宣言](#)」を参照してください。

作業場所節の記述規則については、マニュアル「COBOL85 言語」の「データ部」の「作業場所節」を参照してください。

(3) 連絡節 (LINKAGE SECTION)

この節では、DML 中に記述する埋込み変数を宣言します。埋込み変数の宣言は、連絡節か作業場所節のどちらかで行ってください。

連絡節の記述規則については、マニュアル「COBOL85 言語」の「データ部」の「連絡節」を参照してください。

(4) 埋込み変数のデータ記述項

埋込み変数、埋込み変数の下位項目として使用する変数のデータ記述には、次に示すデータ記述項が指定できます。そのほかのデータ記述項を指定した変数は埋込み変数として使用できません。そのほかの埋込み変数についての記述規則については、「[2.4.2\(4\) 埋込み変数の規則](#)」を参照してください。



項目番	データ記述項	指定内容
1	レベル番号	01~49 (集団項目) または 77 (独立項目)
2	データ名	<div style="border: 1px solid black; padding: 2px;">{データ名 FILLER}</div> <p>埋込み変数名を指定します。</p> <p>データ名は 30 文字以下の名前が指定できます。※1</p> <p>FILLER は埋込み変数の名前としては使用できません。下位項目のデータ名には指定できます。</p> <p>数字、または下線で始まるデータ名は指定できません。</p>
3	REDEFINES 句	<div style="border: 1px solid black; padding: 2px;">REDEFINES データ名</div> <p>データ名は 30 文字以下の名前が指定できます。※1</p> <p>埋込み変数または埋込み変数の下位項目に指定できます。</p>
4	EXTERNAL 句	<div style="border: 1px solid black; padding: 2px;">EXTERNAL</div> <p>補助語の IS は指定できません。</p>
5	GLOBAL 句	<div style="border: 1px solid black; padding: 2px;">GLOBAL</div> <p>補助語の IS は指定できません。</p>

項目番	データ記述項	指定内容
6	PICTURE 句	「表 2-3 DML のデータ型と COBOL 言語のデータ記述項の対応」のレコード型の構成要素のデータ型に対応するデータ記述に記載されている記述以外は指定できません。PICTURE 句に指定する文字列は 30 バイトまでです。
7	USAGE 句	「表 2-3 DML のデータ型と COBOL 言語のデータ記述項の対応」のレコード型の構成要素のデータ型に対応するデータ記述に記載されている記述以外は指定できません。
8	OCCURS 句	「表 2-4 データ記述項の句の記述形式」に記載されている記述以外は指定できません。
9	SYNCHRONIZED 句	<p>{SYNCHRONIZED SYNC}</p> <p>埋込み変数が独立項目（レベル番号 77）の場合だけ指定できます。</p> <p>LEFT, RIGHT の指定はできません。</p>
10	VALUE 句	<p>VALUE [IS] 定数</p> <p>次の定数を指定できます。</p> <ul style="list-style-type: none"> 文字定数^{※2, ※3} 数字定数^{※2} 表意定数^{※4} 16 進文字定数^{※2}

注※1

DML プリプロセサ (pdsdbcbl) は、名前の最大長を 60 バイトとしてチェックします。

注※2

ポストソースのコンパイルに使用する COBOL コンパイラで指定できる長さ・桁数で指定してください。

注※3

文字列を囲む記号には、アポストロフィ (')、引用符 (") のどちらも使用できます。

注※4

記号文字を指定した表意定数は指定できません。

(5) 埋込み変数以外のデータ記述項

埋込み変数以外のデータ記述項には、数字、または下線で始まるデータ名は指定できません。

(6) COPY 文

COPY 文は、次の構文で記述します。

COPY 文を使用することで、登録集原文を COBOL ソースプログラムに取り込むことができます。

```
COPY 原文名 [OF  
    | IN] 登録集名] [SUPPRESS] .
```

次に示す節に記述された構文が、COPY 文として解析されます。

- ・作業場所節 (WORKING-STORAGE SECTION)
- ・連絡節 (LINKAGE SECTION)

上記の節に記述された COPY 文が、登録集原文内に入れ子の形で記述されている場合、その COPY 文も解析対象となります。

COPY 文の入れ子の形については、「2.10.7 登録集原文の制限」の「図 2-10 COPY 文の入れ子のレベルの数え方」を参照してください。

COPY 文の記述規則については、COBOL の仕様に従います。ただし、HiRDB/SD での記述規則で COBOL の仕様と異なる点があります。COBOL の仕様と異なる点については、次の表を参照してください。

表 2-1 COPY 文の記述規則および COBOL の仕様と異なる点

項目番	オペランド	指定内容
1	原文名	<ul style="list-style-type: none"> 登録集原文が登録されているファイルの名称を、拡張子を付けないで指定します。 30 文字以下の文字列で指定します。※2 次の文字で始まる原文名は指定できません。 数字 (0~9) 下線 (_) 英小文字は使用できません。 登録集原文のファイルが登録されているディレクトリ、およびファイルの検索順位は次のとおりです。そのため、次に示すどれかのファイルに登録しておいてください。 <p>ディレクトリの検索順位※1</p> <ul style="list-style-type: none"> 環境変数 PDCBLLIB に設定したディレクトリ（ディレクトリは絶対パスで指定します※2、※3） カレントディレクトリ <p>ファイルの検索順位※1</p> <ul style="list-style-type: none"> ファイル名.cbl ファイル名.CBL プリプロセス時とコンパイル時で、参照する登録集原文を一致させる必要があります。そのため、上記で登録した登録集原文と同じファイルを、コンパイル時に登録集原文を検索するディレクトリに格納しておいてください。 <p>コンパイル時に登録集原文を検索するディレクトリについては、マニュアル「COBOL85 使用の手引」の「登録集原文の使用方法」を参照してください。</p> <ul style="list-style-type: none"> COPY 文の入れ子は最大 10 レベルまで記述できます。※4 登録集原文は、COBOL の固定形式正書法で記述してください。
2	登録集名	<div style="border: 1px solid black; padding: 2px; display: inline-block;"> {OF IN} 登録集名 </div> <p>任意の登録集名を指定します。</p> <p>なお、プリプロセス時は、登録集名の指定に関係なく、次に示す順位でディレクトリが検索されます。</p> <ol style="list-style-type: none"> 環境変数 PDCBLLIB に設定したディレクトリ

項番	オペランド	指定内容
		2. カレントディレクトリ
3	SUPPRESS	<p>SUPPRESS</p> <p>原始プログラムの出力印刷時に、該当する COPY 文に複写される原文の印刷を抑止する場合に指定します。</p>

注※1

登録集原文のファイルの検索順位の方が、ディレクトリの検索順位より優先されます。

(例)

- 登録集原文名 : LIB1
- 環境変数 PDCBLLIB に設定したディレクトリ : /USER/A:/USER/B
- カレントディレクトリ : C

この場合、次の順序で登録集原文名が検索されます。

検索順序	検索時に使用するファイル名	検索対象ディレクトリ
1	LIB1.cbl	/USER/A
2	LIB1.cbl	/USER/B
3	LIB1.cbl	C
4	LIB1.CBL	/USER/A
5	LIB1.CBL	/USER/B
6	LIB1.CBL	C

注※2

登録集原文名長と、環境変数 PDCBLLIB で指定したディレクトリ名の最大長の合計が、1,018 バイト以下になるように指定してください。1,018 バイトを超えるとエラーになります。

注※3

環境変数 PDCBLLIB で指定するディレクトリ名の末尾にスラッシュ (/) を記述しないでください。記述した場合、登録集原文ファイルのオープン時にエラーになるおそれがあります。

注※4

UAP ソースファイルに記述した COPY 文で直接展開する登録集原文内に記述した COPY 文をレベル 1 と数えます。

COPY 文の入れ子のレベルの数え方の詳細については、「[2.10.7 登録集原文の制限](#)」の「[図 2-10 COPY 文の入れ子のレベルの数え方](#)」を参照してください。

2.1.4 手続き部 (PROCEDURE DIVISION)

手続き部には、プログラムの処理を記述します。手続き部に DML を記述して SDB データベースにアクセスすることができます。手続き部の記述規則については、マニュアル「COBOL85 言語」の「手続き部」を参照してください。

手続き部に記述する、SDB データベースにアクセスする部分のプログラムの処理の流れの例を次の図に示します。

図 2-2 SDB データベースにアクセスする部分のプログラムの処理の流れの例

```
PROCEDURE DIVISION.  
MAIN SECTION.  
  •  
  • (中略)  
  
* ZAIKO-KOUSHIN SECTION.  
M-01.  
  PERFORM ICHIDUKE.  
  IF ERR-FLG = '0'  
  THEN  
    IF IC_IO-KUBUN = '1'  
    THEN  
      COMPUTE ZSURYO = ZSURYO + IC_SSURYO  
      ELSE  
      COMPUTE ZSURYO = ZSURYO - IC_SSURYO  
    END-IF  
    MOVE 'M-01:MODIFY ZAIKO' TO EM_ERRDML  
    EXEC DML  
      MODIFY ZAIKO FROM :ZAIKO  
    END-DML  
  
    IF SQLCODE NOT = 0  
    THEN  
      MOVE '2' TO ERR-FLG  
    ELSE  
      CONTINUE  
    END-IF  
  END-IF.  
  M-EXIT.  
  EXIT.  
  • (中略)  
  
D-02.  
  IF ERR-FLG = '0'  
  THEN  
    CALL 'UAPSQL01' USING REQSQL_COMT  
    IF RETURN-CODE NOT = 0  
    THEN  
      MOVE '3' TO ERR-FLG  
    ELSE  
      CONTINUE  
    END-IF  
  ELSE  
    CONTINUE  
  END-IF.  
  D-EXIT.  
  EXIT.  
  • (中略)  
  
F-02.  
  IF ERR-FLG NOT = '0'  
  THEN  
    CALL 'UAPSQL01' USING REQSQL_ROLB  
  ELSE  
    CONTINUE  
  END-IF.  
  F-EXIT.  
  EXIT.  
  • (中略)
```

図 2-2 の説明:
赤枠で囲まれた部分は、DML によるレコードの操作 (レコードの更新) です。
青枠で囲まれた部分は、DML の実行結果の判定です。
青枠で囲まれた部分は、トランザクションのコミットです。
青枠で囲まれた部分は、トランザクションの取り消しです。

[説明]

- DML によるレコードの操作

DML を記述して、レコードに対する操作を実行します。DML によるレコードに対する操作方法の詳細については、次の個所を参照してください。

- ・「[2.5 DML によるレコードの検索](#)」
- ・「[2.6 DML によるレコードの更新、格納、または削除](#)」

DML は、DML 先頭子と DML 終了子で囲む必要があります。

EXEC DML (DML 先頭子)

DML 先頭子は、DML の始まりを示します。

END-DML (DML 終了子)

DML 終了子は、DML の終わりを示します。

• DML の実行結果の判定

次の情報が DML の実行結果として、SQL 連絡領域 (SQLCA) に返されます。返された情報を基に DML の実行結果の判定処理を行います。

- SQLCODE (リターンコード)
- SQLWARN0~SQLWARNF (警告情報)

DML の実行結果の判定処理の詳細については、「[2.7 DML の実行結果の判定処理](#)」を参照してください。

• トランザクションのコミット、またはトランザクションの取り消し

DML の実行結果の判定に従って、トランザクションが更新したレコードの内容を有効にするか、または取り消します。トランザクションのコミット、または取り消しについては、「[2.8 トランザクション制御](#)」を参照してください。

2.1.5 プログラム終わり見出し (END PROGRAM)

「END PROGRAM」をプログラム終わり見出しとして記述します。

プログラム名には、数字、または下線で始まるプログラム名を指定できません。

2.2 COBOL ソースプログラムを作成する際の考慮点

SDB データベースにアクセスする部分の COBOL ソースプログラムを作成する際の考慮点を次に示します。

- COBOL ソースプログラム中のデータ部の SDB データベース節に、アクセス対象の SDB データベース名を記述する必要があります。次に示す理由のため、アクセス対象の SDB データベースごとに UAP ソースファイルを分けることを推奨します。

理由

COBOL ソースプログラム中に記述している SDB データベースの定義を変更した場合、UAP の再プリプロセスと再コンパイルが必要になります。再コンパイルをするとオブジェクトが変わるために、オブジェクトが変わった部分のテストを実施する必要があります。テストを実施する影響範囲を限定するために、アクセス対象の SDB データベースごとに UAP ソースファイルを分けることを推奨します。

2.3 SDB データベース節の記述

COBOL ソースプログラムの主プログラム部分のデータ部に、SDB データベース節を記述します。SDB データベース節には、UAP がアクセスする SDB データベースの名称と、DML を使用してアクセスしたレコードのレコード型名とレコード長を受け取る埋込み変数を記述します。

2.3.1 SDB データベース節の記述例

SDB データベース節の記述例を次に示します。

記述例

```
SDB-DATABASE SECTION.          ...1
  SDB      DATABASE01          ...2
  RECORD NAME  RECNAME        ...3
  RECORD LENGTH RECLENGTH    ...4
.
.
```

[説明]

1. SDB データベース節の開始を宣言します。
2. UAP がアクセスする SDB データベースの名称を指定します。この例では、DATABASE01 を指定しています。
3. FETCH 文、FIND 文、または GET 文で操作したレコードのレコード名を受け取る埋込み変数の名前を指定します。この例では、RECNAME を指定しています。
4. FETCH 文、MODIFY 文、STORE 文、または GET 文で操作したレコードのレコード長を受け取る埋込み変数の名前を指定します。この例では、RECLENGTH を指定しています。
5. SDB データベース節の終了を示す終止符を指定します。

2.3.2 SDB データベース節の記述内容と構文規則

SDB データベース節に記述する内容とその構文規則を説明します。

形式

```
SDB-DATABASE SECTION.
  SDB SDBデータベース名 [, SDBデータベース名] ...
  [RECORD NAME 埋込み変数]
  [RECORD LENGTH 埋込み変数]
.
.
```

SDB-DATABASE SECTION.

SDB データベース節の開始を宣言します。

SDB-DATABASE SECTION. は 1 行に記述してください。複数行にわたって記述した場合、SDB データベース節として認識されません。

SDB SDB データベース名 [,SDB データベース名] …

～<識別子>((1～30 文字))

UAP がアクセスする SDB データベースの名称を指定します。

SDB データベース名は、最大 64 個指定できます。

注意事項

- SDB データベース名に英小文字がある場合は、SDB データベース名を引用符 ("") で囲んでください。引用符で囲まない場合、SDB データベース名はすべて英大文字と見なされます。例えば、SDB データベース名に「Database01」を指定した場合、「DATABASE01」を指定したと見なされます。
- SDB データベース名が pdsdbcbl コマンドの予約語に該当する場合は、SDB データベース名を引用符 ("") で囲んでください。pdsdbcbl コマンドの予約語は、pdsdbcbl コマンドの機能拡張によって追加されることがあります。そのため、SDB データベース名は予約語に該当しない場合でも、あらかじめ引用符で囲んでおくことを推奨します。
- 次のどちらかの条件を満たす場合、「[2.10.9 SDB データベースの特定規則](#)」で説明している規則に従って、定義情報を参照する SDB データベースが決まります。よって、SDB データベース名の指定順序に注意が必要です。
 - DML に指定したレコード名と同一のレコード型名が、複数の SDB データベースに定義されている場合
 - DML に指定した親子集合型名と同一の親子集合型名が、複数の SDB データベースに定義されている場合また、「[2.10.9 SDB データベースの特定規則](#)」で説明しているように、異なる SDB データベースに定義された同一名称のレコード型は、1 つの UAP ソースファイルで同時に操作できません。親子集合型も同様に、DML にレコード名と同時に指定する場合を除いて、異なる SDB データベースに定義された同一名称の親子集合型は、1 つの UAP ソースファイルで同時に操作できません。

SDB データベース名の指定規則については、マニュアル「HiRDB 構造型データベース機能」の「HiRDB/SD 定義ユーティリティ (pdsdbdef)」の「名前の規則」を参照してください。

pdsdbcbl コマンドの予約語については、マニュアル「HiRDB 構造型データベース機能」の「pdsdbcbl コマンドの予約語」を参照してください。

RECORD NAME 埋込み変数

～<識別子>((1～30 文字))

FETCH 文, FIND 文, または GET 文で操作したレコードのレコード型名を受け取る埋込み変数を指定します。FETCH 文, FIND 文, または GET 文で操作したレコードのレコード型名を受け取る場合にこのオプションを指定してください。

留意事項を次に示します。

- レコード型名が返されるのは, 「SQLCODE ≥ 0 かつSQLCODE $\neq 100$ 」のときに限ります。
「SQLCODE < 0 または SQLCODE = 100」の場合は, 埋込み変数に空白が返されます。
SQLCODEについては, [「2.7.2 SQLCODE の値と意味」](#)を参照してください。

埋込み変数に付ける名前の規則については, マニュアル「COBOL85 言語」の「利用者語」を参照してください。ただし, 数字, または下線で始まる名前は指定できません。

RECORD LENGTH 埋込み変数

~<識別子>((1~30 文字))

FETCH 文, MODIFY 文, STORE 文, または GET 文で操作したレコードのレコード長を受け取る埋込み変数を指定します。FETCH 文, MODIFY 文, STORE 文, または GET 文で操作したレコードのレコード長を受け取る場合にこのオプションを指定してください。

留意事項を次に示します。

- レコード長が返されるのは, 「SQLCODE ≥ 0 かつSQLCODE $\neq 100$ 」のときに限ります。
「SQLCODE < 0 または SQLCODE = 100」の場合は, 埋込み変数に 0 が返されます。
SQLCODEについては, [「2.7.2 SQLCODE の値と意味」](#)を参照してください。

埋込み変数に付ける名前の規則については, マニュアル「COBOL85 言語」の「利用者語」を参照してください。ただし, 数字, または下線で始まる名前は指定できません。

(終止符)

SDB データベース節の終了を示す終止符を指定します。

注意事項

- SDB データベース節は, 主プログラム部分のデータ部に記述します。
- SDB データベース節は, データ部のほかの節より先に記述してください。
- SDB データベース節を記述した行に, ほかの命令を記述しないでください。
- SDB データベース節は第 8 欄から第 72 欄までの間に記述してください。字句の途中で改行する場合は, COBOL の行のつなぎの規則に従ってください。SDB データベース名を引用符で囲む場合は, COBOL の文字列定数の行のつなぎの規則に従ってください。

2.3.3 SDB データベース節で指定する埋込み変数の宣言

SDB データベース節の RECORD NAME および RECORD LENGTH で指定する埋込み変数を, 主プログラムの作業場所節で宣言する必要があります。作業場所節以外で宣言した場合, 動作は保証されません。

SDB データベース節の RECORD NAME および RECORD LENGTH で指定する埋込み変数と、COBOL 言語のデータ記述項の対応を次の表に示します。

表 2-2 SDB データベース節で指定する埋込み変数と、COBOL 言語のデータ記述項の対応

SDB データベース節で指定する埋込み変数	COBOL 言語のデータ記述項	項目の記述
RECORD NAME に指定する埋込み変数	L1 基本項目名※ PICTURE X(30) [[USAGE] DISPLAY]	• 基本項目 • 独立項目
RECORD LENGTH に指定する埋込み変数	INTEGER 型に対応する埋込み変数を宣言します。INTEGER 型の COBOL 言語のデータ記述項については、「 表 2-3 DML のデータ型と COBOL 言語のデータ記述項の対応 」を参照してください。	

(凡例)

L1：レベル番号 01～49、または 77

注※

基本項目名は、60 バイト以内で、COBOL コンパイラで使用できる名称にしてください。

SDB データベース節で指定する埋込み変数の宣言例を次に示します。

- SDB データベース節の記述例（データ部の SDB データベース節）

```
SDB-DATABASE SECTION.  
  SDB          DATABASE01  
  RECORD NAME  RECNAME  
  RECORD LENGTH RECLENGTH  
.
```

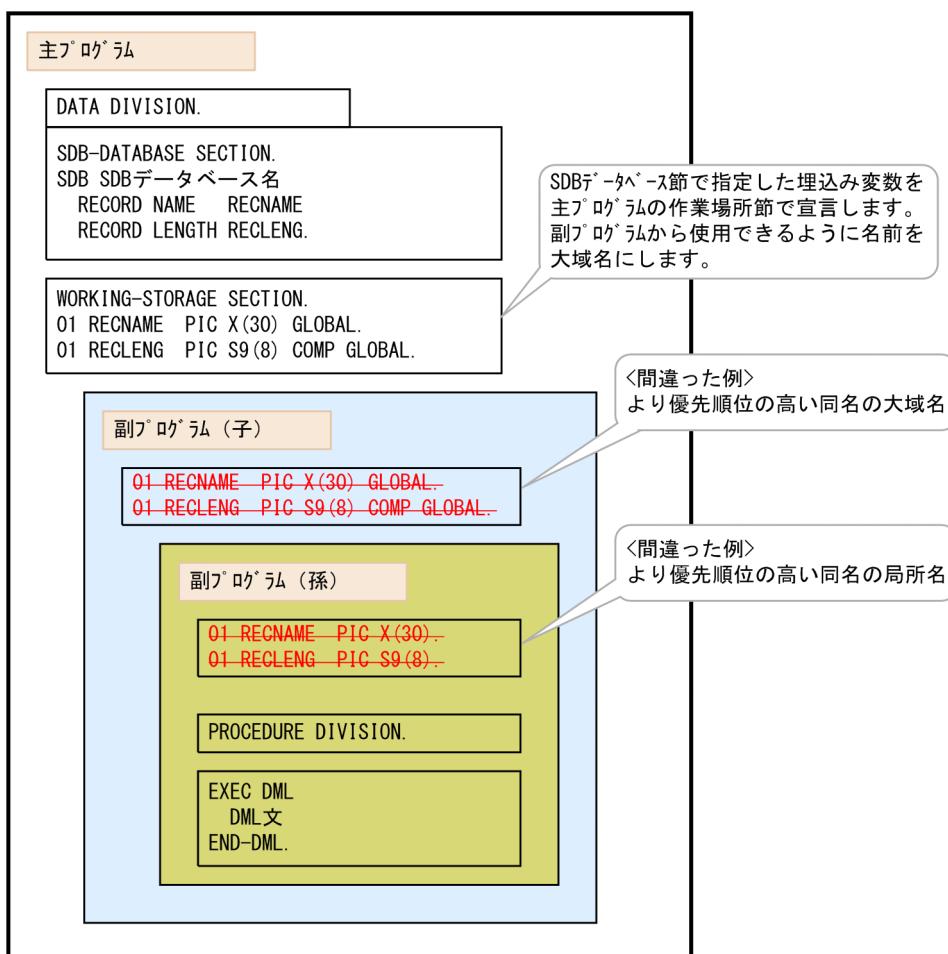
- 埋込み変数の宣言例（データ部の作業場所節）

```
WORKING-STORAGE SECTION.  
  77 RECNAME          PIC X(30)      VALUE SPACE.  
  77 RECLENGTH        PIC S9(8) COMP VALUE ZERO.
```

SDB データベース節の RECORD NAME および RECORD LENGTH で指定する埋込み変数の使用例については、「[2.4.3\(5\) SDB データベース節で指定する埋込み変数の使用例](#)」を参照してください。

注意事項

プログラムに DML を記述する場合は、SDB データベース節で指定した埋込み変数のデータ記述項に GLOBAL 句を指定して埋込み変数の名前を大域名にしてください。このとき、DML を実行する副プログラムから主プログラムまでの間に、より優先順位が高い同一名称のデータ項目がないようにしてください。例を次に示します。



2.4 埋込み変数の宣言

DML 中に記述する埋込み変数の宣言方法と、埋込み変数の使用方法について説明します。

2.4.1 埋込み変数とは

埋込み変数とは、UAP と HiRDB/SD の間で、DML を使用して値の受け渡しをする際に使用する変数です。埋込み変数は、次の用途で使用します。

- DML 中に条件値を指定する際に埋込み変数を使用する
- DML 中に更新値を指定する際に埋込み変数を使用する
- SDB データベースの検索結果のレコード実現値を受け取る際に埋込み変数を使用する
- レコード名を受け取る際 (FETCH 文、FIND 文、または GET 文の正常終了時) に埋込み変数を使用する
- レコード長を受け取る際 (FETCH 文、MODIFY 文、STORE 文、または GET 文の正常終了時) に埋込み変数を使用する

埋込み変数の使用例を次に示します。

```
EXEC DML
  MODIFY "REC01" FROM :REC01_DATA
END-DML.
```

[説明]

- 上記は、REC01 のレコードのレコード実現値を MODIFY 文で更新している例です。MODIFY 文中に記述している REC01_DATA が埋込み変数です。
- DML 中に埋込み変数を指定する場合、「:埋込み変数」の形式で指定します。指定形式の詳細については、マニュアル「HiRDB 構造型データベース機能」の「DML リファレンス」の「埋込み変数」を参照してください。

2.4.2 埋込み変数の宣言方法

DML 中に記述する埋込み変数は、データ部の次のどちらかの節で宣言する必要があります。

- 作業場所節 (WORKING-STORAGE SECTION)
- 連絡節 (LINKAGE SECTION)

上記以外の節で宣言した変数は、埋込み変数として使用できません。

DML を使用してアクセスするレコードのデータ項目を基に、埋込み変数を作成する機能として#CBLFORM コマンドをサポートしました。詳細については、マニュアル「HiRDB 構造型データベース機能」の「HiRDB/SD データベースアクセスユティリティ (pdsdbexe)」を参照してください。

(1) 埋込み変数の宣言例

埋込み変数はレコード型、構成要素のデータ型に合わせて宣言します。埋込み変数の宣言例を次に示します。

レコード型の定義

```
RECORD FMBDT_ROOT
 2 DBKEY
 3 KEYDATA1      XCHARACTER 1  TYPE  K, L
 2 CHR_L01       CHARACTER 1   TYPE  U, D
 2 CHR_L30       CHARACTER 30  TYPE  U, D
 2 XCHR_L01      XCHARACTER 1  TYPE  U, D
 2 XCHR_L30      XCHARACTER 30  TYPE  U, D
 2 PACK_38_0     PACKED 38, 0  TYPE  U, D
 2 PACK_10_5     PACKED 10, 5  TYPE  U, D
 2 PACK_0_38     PACKED 0, 38  TYPE  U, D
 2 INT           INTEGER      TYPE  U, D
 2 SINT          SMALLINT    TYPE  U, D

RECORD FMBDT_CHILD
 2 KEYDATA1      XCHARACTER 1  TYPE  K, L
 2 DBKEY          INTEGER      TYPE  K, N
 2 CHR_L01       CHARACTER 1   TYPE  U, D
 2 CHR_L30       CHARACTER 30  TYPE  U, D
 2 XCHR_L01      XCHARACTER 1  TYPE  U, D
 2 XCHR_L30      XCHARACTER 30  TYPE  U, D
 2 PACK_38_0     PACKED 38, 0  TYPE  U, D
 2 PACK_10_5     PACKED 10, 5  TYPE  U, D
 2 PACK_0_38     PACKED 0, 38  TYPE  U, D
 2 INT           INTEGER      TYPE  U, D
 2 SINT          SMALLINT    TYPE  U, D
```

埋込み変数の宣言例

```
WORKING-STORAGE SECTION.
*
77  RECNAME          PIC X(30)      VALUE SPACE.      ...1
77  RECLENG         PIC S9(8) COMP VALUE ZERO.     ...2
*
01  FMBDT_ROOT.
 02  PDBKEYRT.
    03  PKEYDATA1      PIC X          VALUE SPACE.
    02  PCHR_L01       PIC X          VALUE SPACE.
    02  PCHR_L30       PIC X(30)     VALUE SPACE.
    02  PXCHR_L01      PIC X          VALUE SPACE.
    02  PXCHR_L30      PIC X(30)     VALUE SPACE.
    02  PPACK_38_0     PIC S9(38)    COMP-3 VALUE 0.
    02  PPACK_10_5     PIC S9(10)V9(5) COMP-3 VALUE 0.
    02  PPACK_0_38     PIC SV9(38)   COMP-3 VALUE 0.
    02  PINT           PIC S9(5)     COMP  VALUE 0.
    02  PSINT          PIC S9(1)     COMP  VALUE 0.
*
01  FMBDT_CHILD.      ...4
```

```

02 CKEYDATA1      PIC X          VALUE SPACE.
02 CDBKEY        PIC S9(8)      COMP  VALUE 0.
02 CCHR_L01      PIC X          VALUE SPACE.
02 CCHR_L30      PIC X(30)      VALUE SPACE.
02 CXCHR_L01      PIC X          VALUE SPACE.
02 CXCHR_L30      PIC X(30)      VALUE SPACE.
02 CPACK_38_0      PIC S9(38)    COMP-3 VALUE 0.
02 CPACK_10_5      PIC S9(10)V9(5) COMP-3 VALUE 0.
02 CPACK_0_38      PIC SV9(38)   COMP-3 VALUE 0.
02 CINT          PIC S9(9)      COMP  VALUE 0.
02 CSINT          PIC S9(4)      COMP  VALUE 0.

```

[説明]

1. FETCH 文, FIND 文, または GET 文で操作したレコードのレコード名を受け取るための埋込み変数 (RECNAME) を宣言します。
2. FETCH 文, MODIFY 文, STORE 文, または GET 文で操作したレコードのレコード長を受け取るための埋込み変数 (RECLENG) を宣言します。
3. レコード型 FMBDT_ROOT とデータの受け渡しをする埋込み変数を宣言します。
4. レコード型 FMBDT_CHILD とデータの受け渡しをする埋込み変数を宣言します。

COBOL 言語のデータ記述項で埋込み変数を宣言します。データ記述項の指定形式および構文規則については、マニュアル「COBOL85 言語」の「データ記述項」を参照してください。

(2) DML のデータ型と COBOL 言語のデータ記述項の対応

埋込み変数を宣言する際は、埋込み変数のデータ型に合わせてデータ記述項を記述してください。埋込み変数のデータ型は、埋込み変数を使用する DML のデータ型によって決まります。DML のデータ型とは、レコード型の各構成要素のデータ型のことです。DML のデータ型によって埋込み変数のデータ型が決まり、埋込み変数のデータ型に従って埋込み変数を宣言する際のデータ記述項を記述します。

DML のデータ型と COBOL 言語のデータ記述項の対応を次の表に示します。

表 2-3 DML のデータ型と COBOL 言語のデータ記述項の対応

DML のデータ型	COBOL 言語のデータ記述項	項目の種類	備考
CHARACTER n	L1 基本項目名 PICTURE X(n) [[USAGE] DISPLAY]	• 基本項目 • 独立項目	$1 \leq n \leq 30,000$
XCHARACTER n	L1 基本項目名 PICTURE X(n) [[USAGE] DISPLAY]	• 基本項目 • 独立項目	$1 \leq n \leq 30,000$
PACKED [DECIMAL FIXED] m [,n]	• $m > 0$, かつ $n > 0$ の場合 L1 基本項目名 PICTURE S9(m)V9(n) [USAGE]	• 基本項目 • 独立項目	$1 \leq m + n \leq 38^{*1}$

DML のデータ型	COBOL 言語のデータ記述項	項目の種類	備考
	COMPUTATIONAL-3. • $m > 0$, かつ $n = 0$ の場合 L1 基本項目名 PICTURE S9(m) [USAGE] COMPUTATIONAL-3. • $m = 0$ の場合 L1 基本項目名 PICTURE SV9(n) [USAGE] COMPUTATIONAL-3.		
SMALLINT	L1 基本項目名 PICTURE S9(n) COMPUTATIONAL.* ²	• 基本項目 • 独立項目	$1 \leq n \leq 4$
INTEGER	L1 基本項目名 PICTURE S9(n) COMPUTATIONAL.* ²	• 基本項目 • 独立項目	$5 \leq n \leq 9$

(凡例)

L1 : レベル番号 01~49, または 77

注※1

HiRDB/SD で使用可能な範囲です。埋込み変数として使用可能な範囲は、COBOL コンパイラの仕様によって決まります。

注※2

2 進項目をビッグエンディアン形式にする UAP の場合、データ型が INTEGER または SMALLINT の埋込み変数を宣言するときは、COMPUTATIONAL-5 または COMP-5 を指定してください。

2 進項目をビッグエンディアン形式にする UAP を作成する場合の注意事項については、「[2.13 2 進項目をビッグエンディアン形式にする UAP を作成する場合の注意事項](#)」を参照してください。

記述規則

埋込み変数を宣言する際のデータ記述項の句は、次の表に示す形式で記述してください。

表 2-4 データ記述項の句の記述形式

データ記述項の句の記述形式	左記以外の記述形式
PICTURE	PIC
COMPUTATIONAL	COMP
COMPUTATIONAL-n	COMP-n
9(n)	999....9 (n 個の 9 の並び)
X(n)	XXX....X (n 個の X の並び)
OCCURS n TIMES	OCCURS 1 TO n TIMES

データ記述項の句の記述形式	左記以外の記述形式
	OCCURS 1 TO n
	OCCURS n

(3) 複数のレコード型の構成要素に対応するデータ記述項

レコード型や、複数のレコード型の構成要素から成る集団項目に対応する埋込み変数は、対象となる各レコード型の構成要素に対応する基本項目を持った集団項目で宣言します。埋込み変数の宣言例を次の図に示します。

図 2-3 ルートレコードのレコード型とレコード実現値の受け渡しの際に使用する埋込み変数の宣言例

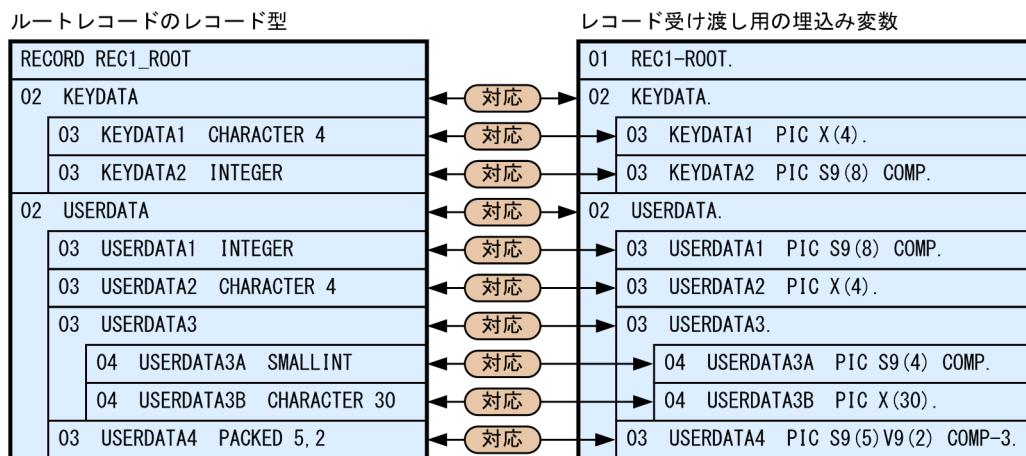
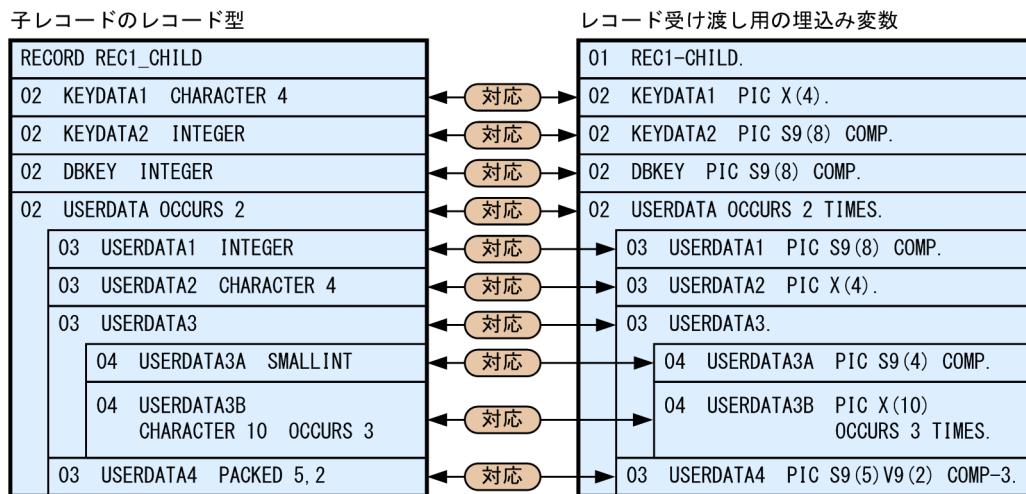


図 2-4 子レコードのレコード型とレコード実現値の受け渡しの際に使用する埋込み変数の宣言例



記述規則

- 埋込み変数に使用する集団項目は、この埋込み変数下の階層構造、基本項目のデータ型およびデータ長を、対応するレコード型、構成要素と一致させてください。なお、埋込み変数の長さに対応する、次の SDB データベース定義については、HiRDB/SD がチェックします。

- ・レコード長
- ・シーケンシャルインデクスの構成要素の合計長

集団項目を構成する基本項目のデータ記述については、「[2.1.3 データ部 \(DATA DIVISION\)](#)」の「埋込み変数のデータ記述項」を参照してください。

- ・埋込み変数に使用する集団項目は、ほかの集団項目の下位項目であってもかまいません。
- ・レベル番号、変数名は一致していなくてもかまいません。埋込み変数の下位項目については、変数名に FILLER を指定できます。

(4) 埋込み変数の規則

埋込み変数の規則を次に示します。

- ・埋込み変数は、次のどちらかの節で宣言してください。
 - ・作業場所節 (WORKING-STORAGE SECTION)
 - ・連絡節 (LINKAGE SECTION)

上記以外の個所で宣言した変数を埋込み変数として使用しないでください。

- ・プログラムを入れ子で記述している場合、外側のプログラムで宣言した埋込み変数を、内側のプログラムに記述した DML で使用するときは、埋込み変数の宣言に GLOBAL 句を指定してください。
- ・レベル番号に 66 は指定できません。
- ・レベル番号に 88 は指定できません。
- ・埋込み変数のデータ記述項には、JUSTIFIED 句および BLANK WHEN ZERO 句は指定できません。
- ・SYNCHRONIZED 句は独立項目（レベル番号 77）にだけ指定できます。
- ・FILLER は、埋込み変数として使用できません。ただし、下位項目には FILLER を使用できます。
- ・PICTURE 句、USAGE 句、および OCCURS 句は「[表 2-3 DML のデータ型と COBOL 言語のデータ記述項の対応](#)」のレコード型の構成要素のデータ型に対応するデータ記述に記載されている記述以外は指定できません。
- ・PICTURE 句を省略し、VALUE 句だけを指定したデータ項目は、埋込み変数として使用できません。
- ・REDEFINES 句は、集団項目に従属する一つ目の下位項目には指定できません。REDEFINES 句が指定された記述項が、そのほかの REDEFINES 句の規則に従っているかはチェックされません。
- ・埋込み変数の名称は、プログラム単位で一意にしてください。
- ・埋込み変数には OCCURS 句を指定できません。ただし、埋込み変数の下位項目に指定することはできます。下位項目に OCCURS 句を指定する場合、指定できる反復回数の最大値は 4,000 です。
- ・COBOL の文の文法に誤りのある変数宣言は、埋込み変数に指定できません。
- ・データ記述項に埋込み変数として指定できない項目は、指定できません。
- ・集団項目を構成する基本項目のデータ長の合計は、レコード長の最大値 (30,000 バイト) を超えることはできません。

上記以外にも埋込み変数の規則があります。詳細については、マニュアル「HiRDB 構造型データベース機能」の「埋込み変数」を参照してください。

2.4.3 埋込み変数の使用例

埋込み変数の使用例を次に示します。

(1) ルートレコードを検索する際に条件値の指定で埋込み変数を使用する例

ルートレコードを検索する際に、条件値の指定で埋込み変数を使用する例を説明します。

検索結果のレコード実現値を受け取る埋込み変数の使用例については、「[\(2\) 検索結果のレコード実現値を取得する際に埋込み変数を使用する例](#)」を参照してください。

レコード型（ルートレコード）の定義例

```
RECORD TENPO
  2 DBKEY
  3 TENPO_CD           XCHARACTER 1  TYPE K, L    ...1
  2 TENPO_NAME         CHARACTER  30 TYPE U, D  ...2
```

[説明]

1. ルートレコードのデータベースキーの集団項目
キーの条件の左辺に指定する構成要素です。
2. ルートレコードのデータベースキーの構成要素

埋込み変数の宣言例（データ部の作業場所節）

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01 X_TENPO.           ...1
  02 X_DBKEY.
    03 X_TENPO_CD  PIC X.
    02 X_TENPO_NAME PIC X(30).
01 Y_DBKEY.           ...2
  02 Y_TENPO_CD    PIC X.      ...3
01 Z_DBKEY.           ...2
  02 Z_TENPO_CD    PIC X.      ...3
```

[説明]

1. 検索対象のレコード型に対応する埋込み変数の集団項目を宣言します。
2. ルートレコードのデータベースキーの集団項目に対応する埋込み変数の集団項目を宣言します。
3. ルートレコードのデータベースキーの構成要素に対応する埋込み変数の基本項目を宣言します。

埋込み変数の使用例（手続き部）

```
PROCEDURE DIVISION.
MOVE X'01' TO Y_TENPO_CD.    ...1
```

```

EXEC DML
  FETCH FIRST TENPO
    INTO :X_TENPO
    WHERE ("DBKEY" = :Y_DBKEY)    ... 2
END-DML.

MOVE X'02' TO Y_TENPO_CD.    ... 3
MOVE X'05' TO Z_TENPO_CD.    ... 4
EXEC DML
  FETCH NEXT TENPO
  INTO :X_TENPO
    WHERE ("DBKEY" >= :Y_DBKEY    ... 5
          AND "DBKEY" <= :Z_DBKEY)
END-DML.

```

[説明]

下線の個所が埋込み変数です。

1. 検索するレコード実現値のデータベースキーを埋込み変数に設定します。
2. データベースキーが条件値に指定した埋込み変数と一致するレコード実現値を FETCH 文で検索して、レコード実現値を取得しています。条件式の右辺に、条件値を設定した埋込み変数を指定します。
3. 検索するレコード実現値の範囲の下限であるデータベースキーを埋込み変数に設定します。
4. 検索するレコード実現値の範囲の上限であるデータベースキーを埋込み変数に設定します。
5. データベースキーが条件値に指定した 2 つの埋込み変数の範囲にあるレコード実現値を FETCH 文で検索して、レコード実現値を取得しています。範囲を指定するそれぞれの条件式の右辺に、条件値を設定した埋込み変数を指定します。

(2) 検索結果のレコード実現値を取得する際に埋込み変数を使用する例

(a) 例 1

FETCH 文で検索結果のレコード実現値を取得する際に埋込み変数を使用する例を説明します。

レコード型（子レコード）の定義例

RECORD ZAIKO				... 1
2 TENPO_CD	XCHARACTER 1	TYPE K, L		... 2
2 DBKEY	INTEGER	TYPE K, N		... 3
2 SCODE	CHARACTER 4	TYPE U, D		... 4
2 SNAME	CHARACTER 30	TYPE U, D		... 4
2 TANKA	INTEGER	TYPE U, D		... 4
2 ZSURYO	INTEGER	TYPE U, D		... 4

[説明]

1. 検索対象の子レコード
 2. 親レコードのデータベースキーの構成要素
 3. 子レコードのデータベースキー（一連番号）
2. UAP の作成

4. ユーザデータの基本項目

埋込み変数の宣言例（データ部の作業場所節）

```
DATA DIVISION.  
WORKING-STORAGE SECTION.  
  
01 X_ZAIKO. ....1  
  02 CH_TENPO_CD    PIC X. ....2  
  02 CH_DBKEY      PIC S9(8) COMP. ....3  
  02 SCODE         PIC X(4). ....4  
  02 SNAME         PIC X(30). ....4  
  02 TANKA         PIC S9(8) COMP. ....4  
  02 ZSURYO        PIC S9(8) COMP. ....4
```

[説明]

1. 検索対象のレコード型に対応する埋込み変数の集団項目を宣言します。
2. 親レコードのデータベースキーの構成要素に対応する埋込み変数の基本項目を宣言します。
3. 子レコードのデータベースキー（一連番号）に対応する埋込み変数の基本項目を宣言します。
4. 子レコードのユーザデータに対応する埋込み変数の基本項目を宣言します。

埋込み変数の使用例（手続き部）

```
PROCEDURE DIVISION.  
  
EXEC DML  
  FETCH FIRST ZAIKO  
    INTO :X_ZAIKO ....1  
    WITHIN TENPO_ZAIKO  
END-DML.
```

[説明]

下線の個所が埋込み変数です。

1. FETCH 文でレコードを検索して、レコード実現値を取得しています。FETCH 文の INTO 句に、レコード実現値を受け取る埋込み変数を指定します。

(b) 例 2

GET 文でレコード実現値を取得する際に埋込み変数を使用する例を説明します。

レコード型（子レコード）の定義例

```
RECORD ZAIKO ....1  
  2 TENPO_CD      XCHARACTER 1  TYPE  K, L ....2  
  2 DBKEY         INTEGER      TYPE  K, N ....3  
  2 SCODE         CHARACTER   4   TYPE  U, D ....4  
  2 SNAME         CHARACTER   30  TYPE  U, D ....4  
  2 TANKA         INTEGER      TYPE  U, D ....4  
  2 ZSURYO        INTEGER      TYPE  U, D ....4
```

[説明]

1. 検索対象の子レコード
2. 親レコードのデータベースキーの構成要素
3. 子レコードのデータベースキー（一連番号）
4. ユーザデータの基本項目

埋込み変数の宣言例（データ部の作業場所節）

```
DATA DIVISION.
WORKING-STORAGE SECTION.

01 X_ZAIKO.           ...1
  02 CH_TENPO_CD      PIC X.      ...2
  02 CH_DBKEY         PIC S9(8) COMP. ...3
  02 SCODE            PIC X(4).    ...4
  02 SNAME            PIC X(30).   ...4
  02 TANKA            PIC S9(8) COMP. ...4
  02 ZSURYO           PIC S9(8) COMP. ...4
```

[説明]

1. 検索対象のレコード型に対応する埋込み変数の集団項目を宣言します。
2. 親レコードのデータベースキーの構成要素に対応する埋込み変数の基本項目を宣言します。
3. 子レコードのデータベースキー（一連番号）に対応する埋込み変数の基本項目を宣言します。
4. 子レコードのユーザデータに対応する埋込み変数の基本項目を宣言します。

埋込み変数の使用例（手続き部）

```
PROCEDURE DIVISION.

EXEC DML
  FIND FIRST ZAIKO
    WITHIN TENPO_ZAIKO
END-DML.

EXEC DML
  GET ZAIKO
    INTO :X_ZAIKO           ...1
END-DML.
```

[説明]

下線の個所が埋込み変数です。

1. FIND 文で位置づけたレコード実現値を、GET 文で取得しています。GET 文の INTO 句に、レコード実現値を受け取る埋込み変数を指定します。

(3) レコードの更新処理で埋込み変数を使用する例

レコードの更新処理で埋込み変数を使用する例を説明します。

レコード型（子レコード）の定義例

```
RECORD ZAIKO
 2 TENPO_CD      XCHARACTER 1 TYPE K, L    ... 1
 2 DBKEY         INTEGER      TYPE K, N    ... 2
 2 SCODE         CHARACTER   4 TYPE U, D    ... 3
 2 SNAME         CHARACTER   30 TYPE U, D   ... 4
 2 TANKA         INTEGER      TYPE U, D   ... 4
 2 ZSURYO        INTEGER      TYPE U, D   ... 4
```

[説明]

1. 更新対象の子レコード
2. 親レコードのデータベースキーの構成要素
3. 子レコードのデータベースキー（一連番号）
4. ユーザデータの基本項目

埋込み変数の宣言例（データ部の作業場所節）

```
DATA DIVISION.
WORKING-STORAGE SECTION.

01 X_ZAIKO.                                ... 1
 02 CH_TENPO_CD    PIC X.                   ... 2
 02 CH_DBKEY      PIC S9(8) COMP.          ... 3
 02 SCODE         PIC X(4).                ... 4
 02 SNAME         PIC X(30).               ... 4
 02 TANKA         PIC S9(8) COMP.          ... 4
 02 ZSURYO        PIC S9(8) COMP.          ... 4
```

[説明]

1. 更新対象のレコード型に対応する埋込み変数の集団項目を宣言します。
2. 親レコードのデータベースキーの構成要素に対応する埋込み変数の基本項目を宣言します。
3. 子レコードのデータベースキー（一連番号）に対応する埋込み変数の基本項目を宣言します。
4. 子レコードのユーザデータに対応する埋込み変数の基本項目を宣言します。

埋込み変数の使用例（手続き部）

```
PROCEDURE DIVISION.

EXEC DML
  FETCH FOR UPDATE FIRST ZAIKO
    INTO :X_ZAIKO          ... 1
    WITHIN TENPO_ZAIKO
  END-DML.

  COMPUTE ZSURYO = ZSURYO + 1.      ... 2

  EXEC DML
    MODIFY ZAIKO FROM :X_ZAIKO    ... 3
  END-DML.
```

[説明]

下線の個所が埋込み変数です。

1. FETCH 文で更新対象のレコード実現値を検索し、位置づけとレコード実現値の取得をしています。FETCH 文で検索したレコード実現値を受け取るために、埋込み変数 X_ZAIKO を指定します。
2. 更新する構成要素に対応する埋込み変数の基本項目に、更新値を設定します。このとき、埋込み変数の下位項目 ZSURYO を使用します。
3. MODIFY 文で位置づけしたレコード実現値を更新しています。MODIFY 文の FROM 句の更新値に埋込み変数を指定します。

(4) レコードの格納処理で埋込み変数を使用する例

レコードの格納処理で埋込み変数を使用する例を説明します。

レコード型の定義例

RECORD TENPO				... 1
2 DBKEY				
3 TENPO_CD	XCHARACTER	1	TYPE	K, L
2 TENPO_NAME	CHARACTER	30	TYPE	U, D
RECORD ZAIKO				... 4
2 TENPO_CD	XCHARACTER	1	TYPE	K, L
2 DBKEY	INTEGER		TYPE	K, N
2 SCODE	CHARACTER	4	TYPE	U, D
2 SNAME	CHARACTER	30	TYPE	U, D
2 TANKA	INTEGER		TYPE	U, D
2 ZSURYO	INTEGER		TYPE	U, D

[説明]

1. 格納するルートレコードのレコード型の定義
2. ルートレコードのレコード型のデータベースキーの構成要素
3. ルートレコードのレコード型のユーザデータの構成要素
4. 格納する子レコードのレコード型の宣言
5. 子レコードのレコード型のユーザデータの構成要素

埋込み変数の宣言例（データ部の作業場所節）

```
DATA DIVISION.  
WORKING-STORAGE SECTION.  
  
01 X_TENPO. ... 1  
  02 X_DBKEY.  
    03 X_TENPO_CD  PIC X. ... 2  
  02 X_TENPO_NAME  PIC X(30). ... 3  
01 X_ZAIKO. ... 4  
  02 CH_TENPO_CD  PIC X.  
  02 CH_DBKEY    PIC S9(8) COMP.  
  02 SCODE        PIC X(4). ... 5  
  02 SNAME        PIC X(30). ... 5
```

2. UAP の作成

02 TANKA	PIC S9(8) COMP.	...5
02 ZSURYO	PIC S9(8) COMP.	...5

[説明]

1. 格納するルートレコードのレコード型に対応する埋込み変数の集団項目を宣言します。
2. ルートレコードのデータベースキーの構成要素に対応する埋込み変数の基本項目を宣言します。
3. ルートレコードのユーザデータの構成要素に対応する埋込み変数の基本項目を宣言します。
4. 格納する子レコードのレコード型に対応する埋込み変数の集団項目を宣言します。
5. 子レコードのユーザデータに対応する埋込み変数の基本項目を宣言します。

埋込み変数の使用例（手続き部）

```

PROCEDURE DIVISION.
MOVE X'06' TO X TENPO_CD ...1
MOVE 'TOTSUKA SHITEN' TO X TENPO NAME ...2

EXEC DML
  STORE TENPO FROM :X TENPO ...3
END-DML.

MOVE 'A001' TO SCODE ...4
MOVE 'PEN CASE' TO SNAME ...4
MOVE 800 TO TANKA ...4
MOVE 20 TO ZSURYO ...4

EXEC DML
  STORE ZAIKO FROM :X ZAIKO ...5
END-DML.

```

[説明]

ルートレコード TENPO を格納後、子レコード ZAIKO を格納します。下線の箇所が埋込み変数です。

1. 格納するルートレコード実現値のデータベースキーを、ルートレコードのデータベースキーの構成要素に対応する埋込み変数の基本項目に設定します。
2. 格納するルートレコード実現値のユーザデータを、ルートレコードのユーザデータの構成要素に対応する埋込み変数の基本項目に設定します。
3. STORE 文でルートレコード TENPO にレコード実現値を格納しています。STORE 文の FROM 句の格納値に埋込み変数を指定します。
4. 格納する子レコード実現値のユーザデータを設定しています。子レコードのユーザデータの構成要素に対応する埋込み変数の基本項目に設定します。このとき、埋込み変数の下位項目 SCODE, SNAME, ~ を使用します。
5. STORE 文で子レコード ZAIKO にレコード実現値を格納しています。STORE 文の FROM 句の格納値に埋込み変数を指定します。

(5) SDB データベース節で指定する埋込み変数の使用例

SDB データベース節で指定する次の埋込み変数を使用する例を説明します。

- RECORD NAME 句で指定する埋込み変数
FETCH 文, FIND 文, GET 文の正常終了後にレコード型名を受け取ります。
- RECORD LENGTH 句で指定する埋込み変数
FETCH 文, MODIFY 文, STORE 文, GET 文の正常終了後にレコード長を受け取ります。

SDB データベース節の宣言例

```
SDB-DATABASE SECTION.
SDB DATABASE01
  RECORD NAME    RECNAME      ...1
  RECORD LENGTH  RECLENG     ...2
.
```

[説明]

1. FETCH 文, FIND 文, GET 文の正常終了後にレコード型名を受け取る埋込み変数を, RECORD NAME 句に指定します。
2. FETCH 文, MODIFY 文, STORE 文, GET 文の正常終了後にレコード長を受け取る埋込み変数を, RECORD LENGTH 句に指定します。

埋込み変数の宣言例 (主プログラムのデータ部の作業場所節)

```
DATA DIVISION.
WORKING-STORAGE SECTION.

 01 RECNAME    PIC X(30) .      ...1
 02 RECLENG   PIC S9(8) COMP.  ...2
```

[説明]

1. RECORD NAME 句に指定した埋込み変数を CHARCTER(30)のデータ型で宣言します。
2. RECORD LENGTH 句に指定した埋込み変数を INTEGER のデータ型で宣言します。

埋込み変数の使用例 (手続き部)

```
PROCEDURE DIVISION.

  EXEC DML
    FETCH FIRST TENPO INTO :X_TENPO
      WHERE ("DBKEY" = :Y_DBKEY)           ...1
  END-DML.
  IF SQLCODE >= 0 AND SQLCODE NOT = 100      ...2
  THEN
    DISPLAY 'FETCH RECORD NAME = "' RECNAME '"'
    UPON SYSOUT                                ...3
    DISPLAY 'FETCH RECORD LENGTH = ' RECLENG
    UPON SYSOUT                                 ...4
  ELSE
    DISPLAY SQLERRMC(1:SQLERRML) UPON SYSOUT
  END-IF.
```

[説明]

上記は、FETCH 文が正常終了した場合、FETCH 文の対象としたレコード型のレコード型名とレコード長を参照する処理です。下線の個所が埋込み変数です。

1. FETCH 文を実行します。
2. FETCH 文が正常終了したかを判定します。
3. FETCH 文が正常終了した場合に、埋込み変数に設定されたレコード型名を参照する処理を行います。
4. FETCH 文が正常終了した場合に、埋込み変数に設定されたレコード長を参照する処理を行います。

2.5 DML によるレコードの検索

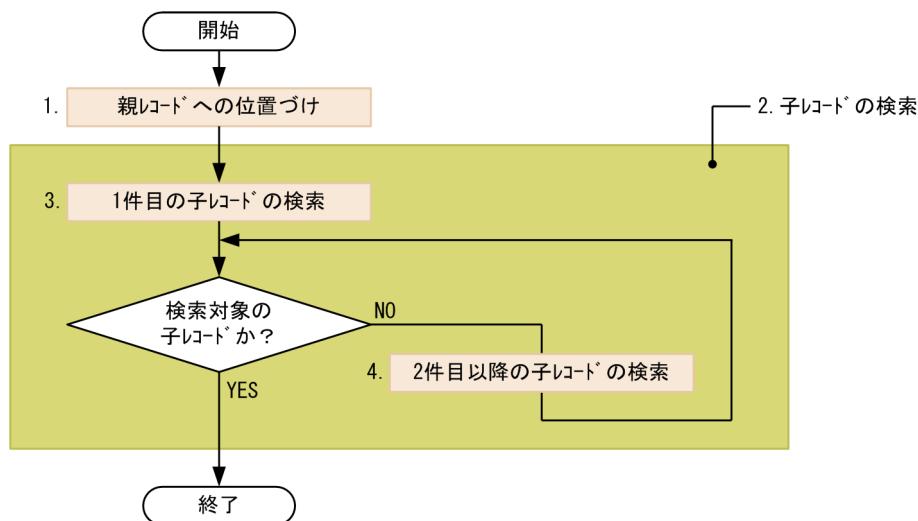
DML によるレコードの検索方法について説明します。

なお、ここでの説明は、マニュアル「HiRDB 構造型データベース機能」の「SDB データベースの操作」の内容を理解していることを前提としています。

2.5.1 レコードの検索

DML による子レコードの検索方法について説明します。子レコードを検索する際の処理の流れを次の図に示します。

図 2-5 子レコードを検索する際の処理の流れ



[説明]

1. 親レコードへの位置づけ

検索対象の子レコード実現値の親レコード実現値に、位置指示子を位置づけます。

```
FIND FIRST 親レコード名  
WHERE キーの条件
```

「親レコードへの位置づけ」は、「[2.11.2 コーディング例](#)」に記載されている COBOL ソースプログラムの例の、行番号 160～162 の処理が該当します。

2. 子レコードの検索

子レコード実現値を検索します。先頭の子レコード実現値から順に、検索対象の子レコード実現値が見つかるまで FETCHを行います。

3. 1 件目の子レコードの検索

```
FETCH FIRST 子レコード名 ←1件目の子レコードを検索します。  
INTO :埋込み変数A  
WITHIN 親子集合名
```

「1件目の子レコードの検索」は、「[2.11.2 コーディング例](#)」に記載されている COBOL ソースプログラムの例の、行番号 305~308 の処理が該当します。

4.2 件目以降の子レコードの検索

検索対象の子レコード実現値と埋込み変数 A の値が一致するまで検索を繰り返します。

```
FETCH NEXT 子レコード名 ←位置づけしている位置から、次のレコード実現値を検索します。  
INTO :埋込み変数A  
WITHIN 親子集合名
```

「2件目以降の子レコードの検索」は、「[2.11.2 コーディング例](#)」に記載されている COBOL ソースプログラムの例の、行番号 319~322 の処理が該当します。

FIND 文および FETCH 文の記述形式および規則については、マニュアル「HiRDB 構造型データベース機能」の「操作系 DML」を参照してください。

なお、レコードの検索は、FIND 文でレコード実現値に位置づけを行い、その後に GET 文でレコード実現値を取得することでも実現できます。GET 文の記述形式および規則については、マニュアル「HiRDB 構造型データベース機能」の「操作系 DML」を参照してください。

2.6 DML によるレコードの更新, 格納, または削除

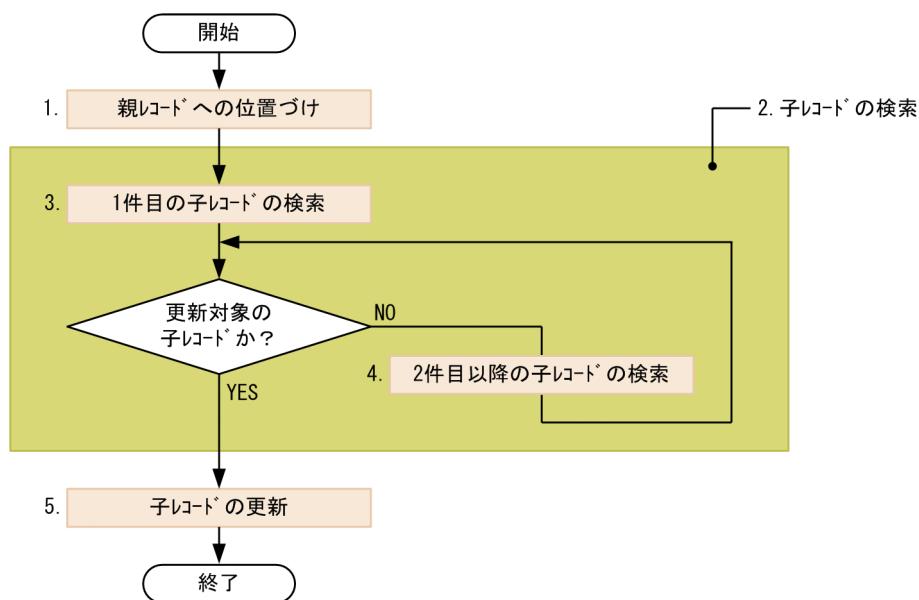
DML によるレコードの更新方法, 格納方法, および削除方法について説明します。

なお, ここでの説明は, マニュアル「HiRDB 構造型データベース機能」の「SDB データベースの操作」の内容を理解していることを前提としています。

2.6.1 レコードの更新

DML による子レコードの更新方法について説明します。子レコードを更新する際の処理の流れを次の図に示します。

図 2-6 子レコードを更新する際の処理の流れ



[説明]

1. 親レコードへの位置づけ

更新対象の子レコード実現値の親レコード実現値に, 位置指示子を位置づけます。

```
FIND FIRST 親レコード名  
WHERE キーの条件
```

「親レコードへの位置づけ」は, 「2.11.2 コーディング例」に記載されている COBOL ソースプログラムの例の, 行番号 160~162 の処理が該当します。

2. 子レコードの検索

更新指定 (FOR UPDATE 指定) で, 更新対象の子レコード実現値を検索します。先頭の子レコード実現値から順に, 更新対象の子レコード実現値が見つかるまで (更新対象の子レコード実現値と埋込み変数 A の値が一致するまで) FETCH を行います。

3. 1 件目の子レコードの検索

2. UAP の作成

FETCH FOR UPDATE	←更新目的であることを指定します。
FIRST 子レコード名	←1件目の子レコードを検索します。
INTO :埋込み変数A	
WITHIN 親子集合名	

「1件目の子レコードの検索」は、「[2.11.2 コーディング例](#)」に記載されている COBOL ソースプログラムの例の、行番号 305~308 の処理が該当します。

4.2 件目以降の子レコードの検索

更新対象の子レコード実現値と埋込み変数 A の値が一致するまで検索を繰り返します。

FETCH FOR UPDATE	←更新目的であることを指定します。
NEXT 子レコード名	←位置づけしている位置から、次のレコード実現値を検索します。
INTO :埋込み変数A	
WITHIN 親子集合名	

「2件目以降の子レコードの検索」は、「[2.11.2 コーディング例](#)」に記載されている COBOL ソースプログラムの例の、行番号 319~322 の処理が該当します。

5. 子レコードの更新

埋込み変数 A に、更新後の子レコード実現値を格納します。その後に、MODIFY 文を実行して、埋込み変数 A の値で子レコード実現値を更新します。

MODIFY 子レコード名 FROM :埋込み変数A	
----------------------------	--

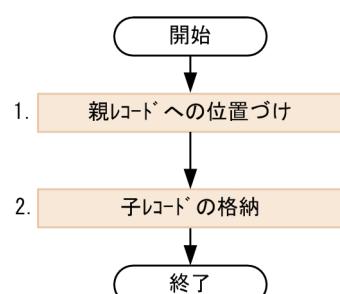
「子レコードの更新」は、「[2.11.2 コーディング例](#)」に記載されている COBOL ソースプログラムの例の、行番号 193~195 の処理が該当します。

FIND 文、FETCH 文、MODIFY 文の記述形式および規則については、マニュアル「HiRDB 構造型データベース機能」の「操作系 DML」を参照してください。

2.6.2 レコードの格納

DML による子レコードの格納方法について説明します。子レコードを格納する際の処理の流れを次の図に示します。

図 2-7 子レコードを格納する際の処理の流れ



[説明]

1. 親レコードへの位置づけ

格納対象の子レコード実現値の親レコード実現値に、位置指示子を位置づけます。

```
FIND FIRST 親レコード名  
WHERE キーの条件
```

「親レコードへの位置づけ」は、「[2.11.2 コーディング例](#)」に記載されている COBOL ソースプログラムの例の、行番号 160~162 の処理が該当します。

2. 子レコードの格納

埋込み変数 A に子レコード実現値を格納します。その後に、STORE 文を実行して、埋込み変数 A の値を子レコードに格納します。

```
STORE 子レコード名 FROM :埋込み変数A
```

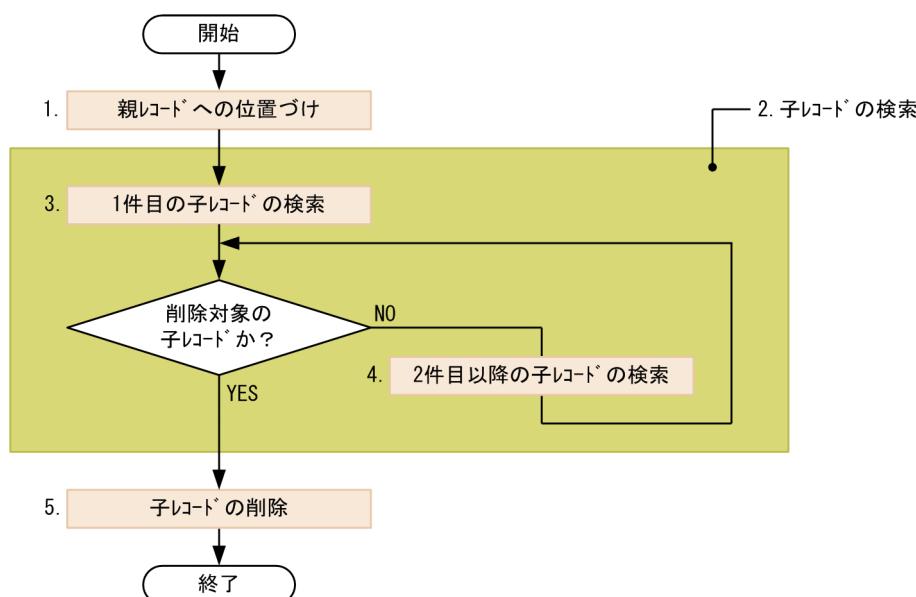
「子レコードの格納」の処理は、「[2.11.2 コーディング例](#)」に記載されている COBOL ソースプログラムの例の、行番号 219~221 の処理が該当します。

FIND 文および STORE 文の記述形式および規則については、マニュアル「HiRDB 構造型データベース機能」の「操作系 DML」を参照してください。

2.6.3 レコードの削除

DML による子レコードの削除方法について説明します。子レコードを削除する際の処理の流れを次の図に示します。

図 2-8 子レコードを削除する際の処理の流れ



[説明]

1. 親レコードへの位置づけ

2. UAP の作成

削除対象の子レコード実現値の親レコード実現値に、位置指示子を位置づけます。

```
FIND FIRST 親レコード名  
WHERE キーの条件
```

「親レコードへの位置づけ」は、「[2.11.2 コーディング例](#)」に記載されている COBOL ソースプログラムの例の、行番号 160～162 の処理が該当します。

2. 子レコードの検索

更新指定 (FOR UPDATE 指定) で、削除対象の子レコード実現値を検索します。先頭の子レコード実現値から順に、削除対象の子レコード実現値が見つかるまで FETCH を行います。

3.1 1 件目の子レコードの検索

```
FETCH FOR UPDATE      ←更新目的であることを指定します。  
FIRST 子レコード名    ←1 件目の子レコードを検索します。  
INTO :埋込み変数A  
WITHIN 親子集合名
```

「1 件目の子レコードの検索」は、「[2.11.2 コーディング例](#)」に記載されている COBOL ソースプログラムの例の、行番号 305～308 の処理が該当します。

4.2 2 件目以降の子レコードの検索

削除対象の子レコード実現値と埋込み変数 A の値が一致するまで検索を繰り返します。

```
FETCH FOR UPDATE      ←更新目的であることを指定します。  
NEXT 子レコード名    ←位置づけしている位置から、次のレコード実現値を検索します。  
INTO :埋込み変数A  
WITHIN 親子集合名
```

「2 件目以降の子レコードの検索」は、「[2.11.2 コーディング例](#)」に記載されている COBOL ソースプログラムの例の、行番号 319～322 の処理が該当します。

5. 子レコードの削除

ERASE 文を実行して、削除対象の子レコード実現値を削除します。

```
ERASE 子レコード名
```

「子レコードの削除」は、「[2.11.2 コーディング例](#)」に記載されている COBOL ソースプログラムの例の、行番号 239～241 の処理が該当します。

FIND 文、FETCH 文、および ERASE 文の記述形式および規則については、マニュアル「HiRDB 構造型データベース機能」の「操作系 DML」を参照してください。

2.7 DML の実行結果の判定処理

DML の実行結果の判定処理について説明します。

2.7.1 DML の実行結果の判定処理の例

次の情報が DML の実行結果として、SQL 連絡領域 (SQLCA) に返されます。返された情報を基に DML の実行結果の判定処理を行います。

- SQLCODE (リターンコード)
- SQLWARN0～SQLWARNF (警告情報)

DML の実行結果の判定処理の例

```
EXEC DML
  MODIFY ZAIKO FROM :ZAIKO          ... 1
END-DML
IF SQLCODE < 0                      ... 2
THEN
  MOVE 8 TO RETURN-CODE
ELSE
  CONTINUE
END-IF
IF SQLWARN0 = 'W' OR
  (SQLCODE > 0 AND SQLCODE NOT = 100) ... 3
THEN
  MOVE 4 TO RETURN-CODE
ELSE
  CONTINUE
END-IF
```

[説明]

1. レコード実現値を更新する DML (MODIFY 文) を実行します。

2. DML の実行結果の判定処理を行います。

SQLCODE が 0 未満の場合は、DML がエラーとなっているため、エラー時の処理を行います。

SQLCODE が 0 以上の場合は、DML が正常終了しているため、処理を続行します。

3. DML の実行結果の判定処理を行います。

DML が警告付きで正常終了した場合は、SQLCODE に 100 以外の正の値が返されるか、または SQLWARN1～SQLWARNF に警告情報が返されます。SQLWARN1～SQLWARNF に警告情報が返された場合は、SQLWARN0 に 'W' が返されます。

警告が発生していない場合は処理を続行してください。警告が発生している場合は、DML に警告が発生したときの処理を行います。

COBOL ソースプログラムのコーディング例については、次の箇所を参照してください。コーディング例中に、DML の実行結果の判定処理の例が記述されています。

- ・「2.11 COBOL ソースプログラムのコーディング例」
- ・「2.12.3 COBOL ソースプログラムのコーディング例 (DML と SQL の両方を実行する UAP の場合)」

なお、SQL 連絡領域 (SQLCA) は、宣言を収めた登録集原文の COPY 文がポストソースに展開されます。そのため、COBOL ソースプログラム中に SQL 連絡領域を宣言しないでください。

2.7.2 SQLCODE の値と意味

返却される SQLCODE の値と意味について説明します。

(1) SQLCODE = 0 の場合

DML が正常終了した場合、SQLCODE に 0 が返されます。

ただし、SQLWARN0 に 'W' が返された場合、警告付きの正常終了になります。この場合、警告情報が SQLWARN1～SQLWARNF に返されるため、SQLWARN1～SQLWARNF の値を確認してください。

SQLWARN0～SQLWARNF については、「2.7.4 SQL 連絡領域の構成と内容」を参照してください。

(2) SQLCODE > 0 かつ SQLCODE ≠ 100 の場合

DML が警告付きで正常終了した場合、SQLCODE に正の値（100 を除く）が返されます。この場合、警告情報が SQLWARN1～SQLWARNF に返されるため、SQLWARN1～SQLWARNF の値を確認してください。

(3) SQLCODE=100 の場合

位置づけるレコードがなくなった場合、SQLCODE に 100 が返されます。次の項目を判定する際に利用します。

- ・ FIND 文で位置づけるレコードがなくなった
- ・ FETCH 文で位置づけるレコードがなくなった

(4) SQLCODE < 0 の場合

DML の処理でエラーが発生した場合、SQLCODE に負の値が返されます。

なお、DML の処理でエラーが発生した際、暗黙的ロールバックが発生する場合と、暗黙的ロールバックが発生しない場合があります。暗黙的ロールバックが発生した場合は、SQLWARN0 と SQLWARN6 に 'W' が返されます。

エラーが発生した DML を特定したい場合は、SQL トレース情報を利用してください。SQL トレース情報については、マニュアル「HiRDB 構造型データベース機能」の「SQL トレース機能」を参照してください。

2.7.3 DML のエラーを検出したときの対処方法

DML のエラーの発生を示す SQLCODE が返却された場合、次の手順で対処します。

手順

1. リターンコードを出力または表示します。
2. リターンコードだけではエラーの内容が判別できない場合は、各コードの付加情報を表示または出力します。また、必要に応じて、エラーになった DML、またはエラーになった DML を識別するための情報表示します。

リターンコードの付加情報と参照先を次の表に示します。

表 2-5 リターンコードの付加情報と参照先

付加情報	参照先
SQLCODE に対するメッセージ	SQL 連絡領域中の SQLERRML フィールド、および SQLERRMC フィールドの内容

3. トランザクションを取り消します (ROLLBACK、または UAP を異常終了させます)。

デッドロックによって暗黙的にロールバックされた UAP は次のようにになります。

- 通常の UAP の場合

暗黙的にロールバックされると、次に実行した DML または SQL が新たなトランザクション開始となります (ROLLBACK、または DISCONNECT もできます)。

- OLTP 下の UAP の場合

暗黙的にロールバックされると、OLTP 下の UAP からは DISCONNECT、または ROLLBACK 以外は受け付けられません。また、OLTP 環境で X/Open に従ったアプリケーションプログラムをクライアントとした場合に、実行したアプリケーションプログラムがデッドロックになったときもトランザクションの終了が必要です。

4. UAP の終了、またはトランザクションの開始（別のトランザクションの新規実行、または同じトランザクションの再実行）をします。

なお、同じトランザクションを再実行する場合、実行前にエラーの対策をしてください。エラーの原因が取り除かれないとトランザクションを再実行すると、無限ループになるおそれがあります。また、再実行しても同じエラーが発生する場合は、UAP の終了を考慮する必要があります。

2.7.4 SQL 連絡領域の構成と内容

DML を実行すると、HiRDB/SD は DML が正常に実行されたかどうかを示す情報を UAP に返します。これらの情報を受け取るための領域を SQL 連絡領域といいます。SQLCODE、SQLWARN1～SQLWARNF は、SQL 連絡領域に格納されます。

(1) SQL 連絡領域の構成

SQL 連絡領域の構成を次の図に示します。

図 2-9 SQL 連絡領域の構成

SQLCA (368)						
SQLCAID (8)			SQLCABC (8)	SQLCODE (8)	SQLERRM (256)	
SQLCAIDC (5)	SQLCAIDS (2)	SQLCAIDE (1)			SQLERRML (2)	SQLERRMC (254)
SQLERRP (8)						
SQLERRD (8 × 6)	SQLWARNO (1)	SQLWARN1 (1)	SQLWARN2 (1)	SQLWARN3 (1)	SQLWARN4 (1)	
SQLWARN5 (1)						
SQLWARN6 (1)	SQLWARN7 (1)	SQLWARN8 (1)	SQLWARN9 (1)	SQLWARNA (1)	SQLWARNB (1)	
SQLWARNC (1)						
SQLWARND (1)	SQLWARNE (1)	SQLWARNF (1)	SQLCASYS (16)			

注

- （ ）内は領域の長さ（単位：バイト）を示しています。
- SQLCABC, SQLCODE, および SQLERRD の長さは、long 型のサイズになります。

(2) SQL 連絡領域の内容

SQL 連絡領域の内容については、マニュアル「HiRDB UAP 開発ガイド」の「SQL 連絡領域の構成と内容」の「SQL 連絡領域の内容」を参照してください。

マニュアル「HiRDB UAP 開発ガイド」の記載内容と異なる部分だけ、次の表に示します。

表 2-6 SQL 連絡領域の内容

レベル番号※1	連絡領域名	データ型	長さ（バイト）	内容
2	SQLCODE	long	8	DML の実行結果を示す SQLCODE が返される領域です。
2	SQLERRM	—	256	DML の実行結果を示すメッセージの情報が返される領域です。

レベル番号※1	連絡領域名	データ型	長さ (バイト)	内容
3	SQLERRML	short	2	SQLERRMC 領域に返されるメッセージの長さが返される領域です。
3	SQLERRMC	char	254	SQLCODE に対応するメッセージが返される領域です。 SQLCODE とメッセージの対応については、マニュアル「HiRDB メッセージ」の「メッセージの記述形式」の「メッセージに関する注意事項」を参照してください。
2	SQLERRD	long	8×6	HiRDB の内部状態を示す領域で、データ型が long の 6 個の配列です。 <ul style="list-style-type: none"> SQLERRD[0] : 未使用 SQLERRD[1] : 未使用 SQLERRD[2] : SQLCODE の値が 0, または正の場合: SDB データベース操作のアクセス件数 ※2 SQLCODE の値が負の場合: この領域は使用しないでください。 SQLERRD[3] : システムが使用します。 SQLERRD[4] : システムが使用します。 SQLERRD[5] : 未使用

(凡例)

– : 該当しません。

注※1

レベル番号は、SQL 連絡領域の包含関係を示しています。例えば、レベル番号 1 の連絡領域はレベル番号 2 の連絡領域で構成されることを示しています。

注※2

SDB データベース操作のアクセス件数については、マニュアル「HiRDB 構造型データベース機能」の「監査証跡表の列に格納される情報についての留意事項 (SDB データベース操作イベントの場合)」の「SDB データベース操作イベントのアクセス件数」の表を参照してください。

(3) SQL 連絡領域の展開

SQL 連絡領域のデータ記述項は、pdsdbcbl コマンドによるプリプロセスの際に、登録集原文の COPY 文がポストソースに自動的に出力されます。そのため、SQL 連絡領域のデータ記述項を、COBOL ソースプログラム中に記述する必要はありません。

登録集原文によって展開されるデータ記述項については、マニュアル「HiRDB UAP 開発ガイド」の「SQL 連絡領域の展開」を参照してください。

2.8 トランザクション制御

トランザクション制御については、マニュアル「HiRDB UAP 開発ガイド」の「トランザクション制御」を参照してください。

なお、マニュアル「HiRDB UAP 開発ガイド」の「トランザクション制御」をお読みいただく際、次のこと留意してください。

- **OpenTP1 環境下で実行する UAP の場合**

トランザクションの開始、トランザクションのコミット、およびトランザクションの取り消しは、OpenTP1 の X/Open に準拠した API である TXBEGIN、TXCOMMIT、TXROLLBACK を使用します。これらの API の文法の詳細については、マニュアル「OpenTP1 プログラム作成リファレンス COBOL 言語編」の「X/Open に準拠したアプリケーションプログラミングインタフェース」を参照してください。

- **COBOL の実行可能ファイルとして実行する UAP の場合**

トランザクション制御は SQL で行います。また、HiRDB サーバへの接続、および HiRDB サーバからの切り離しも SQL で行います。詳細については、「[2.12.2 DML と SQL の両方を実行する UAP のトランザクション制御](#)」を参照してください。

2.9 排他制御

HiRDB/SD では、複数の UAP（トランザクション）が SDB データベースに同時にアクセスしても、SDB データベースの整合性を保つように排他制御を行っています。HiRDB/SD の排他制御については、マニュアル「HiRDB 構造型データベース機能」の「排他制御」を参照してください。

また、SDB 用 UAP 環境定義を使用すると、UAP グループ（OpenTP1 環境下のサービスグループ）単位に、UAP の目的に応じて排他の粒度と強さを制御することができます。なお、DML でレコードを更新、格納、または削除する場合は、SDB 用 UAP 環境定義を使用する必要があります。SDB 用 UAP 環境定義については、マニュアル「HiRDB 構造型データベース機能」の「SDB 用 UAP 環境定義」を参照してください。

2.10 COBOL ソースプログラムの記述規則

COBOL ソースプログラムの記述規則について説明します。

2.10.1 文字コードと改行コード

- COBOL ソースプログラムは、Shift-JIS 漢字コードで記述してください。
- 改行は、LF (X'0A') を使用してください。
- 区切り文字は、次のどれかを使用してください。
 - 空白 (X'20')
 - タブ (X'09')
タブは 1 文字で数えます。
 - 全角空白 (X'8140')

■使用できる文字

COBOL ソースプログラムを記述する際に使用できる文字を次の表に示します。

表 2-7 COBOL ソースプログラムを記述する際に使用できる文字

種別	使用できる文字
半角文字	<ul style="list-style-type: none">• 英大文字 (A~Z, ¥, @, #)• 英小文字 (a~z)• 数字 (0~9)• 空白• 下線文字 (_)• カタカナ文字• 次に示す特殊記号<ul style="list-style-type: none">タブ (X'09')改行 (X'0A')'!'''''%''&'''''('')''*''+'',''_''.'

種別	使用できる文字
	'/' '.' ';' '<' '=' '>' '?' '[' ']' '^' ' ' '..' * '{' * '}' * '-' * ' ' * ' ' * '.' *
全角文字	全角文字コードのすべての文字（外字を含みません）

注※

文字列定数または注釈行だけで使用できる文字です。

2.10.2 ソースプログラムの正書法

ソースプログラムの正書法について説明します。

- COBOL ソースプログラムは、固定形式正書法で記述します。
- 1UAP ソースファイル中の COBOL ソースプログラムに記述できる行数は、最大 999,999 行です。
- UAP ソースファイルにコンパイルリスト出力制御（EJECT, SKIP1, SKIP2, SKIP3, または TITLE）は記述できません。

2.10.3 翻訳単位（最外側のプログラム）

1UAP ソースファイル中の COBOL ソースプログラムに記述できる埋込み型 UAP の翻訳単位は 1 つです。

2.10.4 プログラムの入れ子の上限

プログラムの入れ子の上限について説明します。

- プログラム中にプログラムを入れ子で記述できます。最大 16 階層のプログラムを入れ子で記述できます。
- 1UAP ソースファイル中に記述できるプログラムの最大数は 64 個です。

2.10.5 名前の記述規則

名前の記述規則は、COBOL の規則に従います。ただし、次の規則があります。なお、ここでいう名前とは、埋込み変数の名前のことを意味しています。

- 名前の長さ

名前の最大長は 60 バイトです。ただし、COBOL コンパイラが定める名前の最大長以下にしてください。

- 文字の扱い

- 「全角の英字、数字、記号、カタカナ、空白」と「半角の英字、数字、記号、カタカナ、空白」は、異なる文字と見なされます。

- 使用できない名前

- 次のどれかの条件を満たす名前は、外部属性を持つため使用できません。

- 大文字の「SQL」で始まる名前

- 小文字の「p_」で始まる名前

- 小文字の「pd」で始まる名前

- pdsdb または PDSDDB で始まる名前は使用できません。

- DML の予約語と同じ名前は使用できません。DML の予約語については、マニュアル「HiRDB 構造型データベース機能」の「DML の予約語」を参照してください。

- pdsdbcbl コマンドの予約語と同じ名前は使用できません。pdsdbcbl コマンドの予約語については、マニュアル「HiRDB 構造型データベース機能」の「dsdbcbl コマンドの予約語」を参照してください。

- 数字、または下線で始まる名前は使用できません。

2.10.6 宣言が必要な節

DML を記述する場合に宣言が必要な節を次に示します。

- DML を記述する場合、UAP ソースファイル中の主プログラムに SDB データベース節の宣言をする必要があります。SDB データベース節は、主プログラムのデータ部の先頭の節として宣言してください。

- DML を記述する場合、UAP ソースファイル中の主プログラムと DML を記述した副プログラムに作業場所節 (WORKING-STORAGE SECTION) の宣言をする必要があります。
- 作業場所節の見出しを記述した行では、続けてほかの命令を記述しないでください。

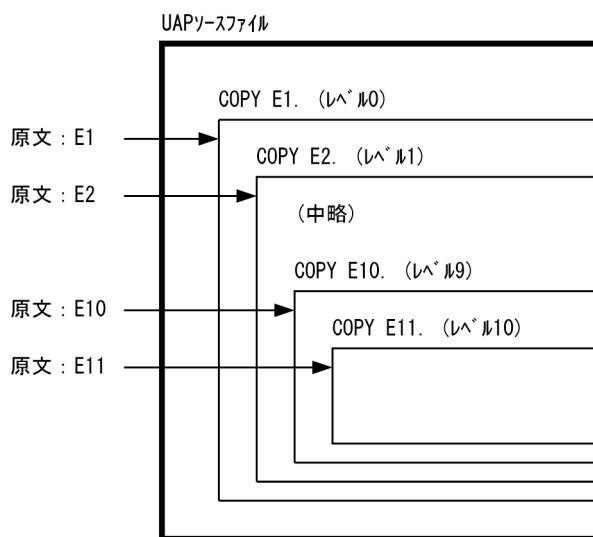
2.10.7 登録集原文の制限

登録集原文の制限を次に示します。

- 登録集原文内には、次の記述はできません。
 - DML
 - SDB データベース節
- 次の節に記述した COPY 文の登録集原文内に、埋込み変数の宣言を記述できます。埋込み変数の宣言以外の文を記述した場合の動作は保証しません。
 - 作業場所節 (WORKING-STORAGE SECTION)
 - 連絡節 (LINKAGE SECTION)
- 次の節に記述した COPY 文で、COPY 文の入れ子は最大 10 レベルまで記述できます。
 - 作業場所節 (WORKING-STORAGE SECTION)
 - 連絡節 (LINKAGE SECTION)

COPY 文の入れ子のレベルの数え方を次の図に示します。

図 2-10 COPY 文の入れ子のレベルの数え方



- 1 つの文を UAP ソースファイル、または登録集原文ファイルに分けて記述することはできません。
- 登録集原文ファイルでの埋込み変数の宣言で、終止符の指定の前にファイルが終了している場合、当該埋込み変数の宣言は DML プリプロセサ (pdsdbcbl) に認識されないため、埋込み変数に指定できません。

- 次の節に記述した COPY 文に、原文名定数指定、PREFIXING 指定、SUFFIXING 指定、および REPLACING 指定はできません。指定した場合は構文エラーになります。
 - 作業場所節 (WORKING-STORAGE SECTION)
 - 連絡節 (LINKAGE SECTION)
- なお、上記の節に記述した COPY 文が、登録集原文内に入れ子の形で記述されている場合、その COPY 文も対象となります。COPY 文の入れ子の形については、「[図 2-10 COPY 文の入れ子のレベルの考え方](#)」を参照してください。
- プリプロセス時とコンパイル時で、登録集原文の内容を変えないでください。
 - DML 中には COPY 文を記述できません。また、SDB データベース節にも COPY 文は記述できません。記述した場合は構文エラーになります。
 - 次の COBOL 命令の途中に COPY 文は記述できません。
 - 見出し部の見出し (IDENTIFICATION DIVISION.)
 - プログラム名段落の見出し (PROGRAM-ID. プログラム名)
 - 環境部の見出し (ENVIRONMENT DIVISION.)
 - 構成節の見出し (CONFIGURATION SECTION.)
 - 入出力節の見出し (INPUT-OUTPUT SECTION.)
 - データ部の見出し (DATA DIVISION.)
 - ファイル節の見出し (FILE SECTION.)
 - 作業場所節の見出し (WORKING-STORAGE SECTION.)
 - 連絡節の見出し (LINKAGE SECTION.)
 - 手続き部の見出し (PROCEDURE DIVISION~)
 - 宣言部分の終わり指示 (END DECLARATIVES.)
 - プログラムの終わりの見出し (END PROGRAM プログラム名.)
 - データ記述項 (レベル番号から、終止符まで)

2.10.8 DML の記述規則

DML の記述規則を次に示します。

- 1UAP ソースファイル中に記述できる DML の数は、最大 1,024 です。
- DML は手続き部 (PROCEDURE DIVISION) に記述します。
- UAP ソースファイル中に DML の記述がなくてもかまいません。DML の記述がない場合、pdsdbcbl コマンドは作業場所節に SQL 連絡領域の登録集原文と、ハンドラ用共通エリアとハンドラ用 SD 固有エリアの登録集原文の COPY 命令だけを展開します。

- 1UAP ソースファイル中に DML と SQL を混在して記述できません。
 - COBOL の命令と DML は同一行に記述できません。記述した場合は、構文エラーになります。
 - DML 先頭子は 1 行に記述してください。DML 先頭子を複数行にわたって記述した場合、DML 先頭子として認識されません。この場合、DML は COBOL 命令に置換されません。
 - DML は DML 先頭子と DML 終了子を含めて、12~72 欄の間に記述してください。12~72 欄の外に DML を記述した場合、構文エラーになります。または、DML と認識されないので、COBOL 命令に置換されません。
 - DML 先頭子の記述にエラーがある場合、pdsdbcbl コマンドは DML と認識できないため、COBOL 命令に置換されません。
 - DML 先頭子がある行から DML 終了子がある行の間のデバッグ行は、注釈行に置換されてポストソースに出力されます。それ以外のデバッグ行は、そのままポストソースに出力されます。
 - 一連番号領域（第 1~第 6 欄の範囲）にタブ文字は記述できません。
 - 文字列定数と 16 進文字列定数の文字列を囲む記号には、アポストロフィ (') を使用してください。COBOL85 の-Xc コンパイラオプション、COBOL2002 の-DoubleQuote コンパイラオプションを指定している場合でも、引用符 ("") は使用できません。
 - DML の継続規則は、COBOL の「行のつなぎ」の規則に従います。
DML で空白を必ず挿入する個所、または空白を挿入できる個所であれば、自由に行を変えて記述できます。また、複数行にわたって記述することもできます。
 - DML で空白を挿入できない個所で行を変える場合は、任意の欄まで記述したあとに次の行の標識領域にハイフン (-) を記述し、プログラム原文領域の 12 欄以降から行の続きを記述してください。
 - 文字列定数の途中で行を変える場合は、72 欄まで記述したあとに次の行の標識領域にハイフン (-) を記述し、プログラム原文領域の 12 欄以降からアポストロフィ (') に続けて、文字列の続きを記述してください。※
 - 引用符 ("") で囲んだ識別子の途中で行を変える場合は、72 欄まで記述したあとに次の行の標識領域にハイフン (-) を記述し、プログラム原文領域の 12 欄以降から引用符 ("") に続けて、文字列の続きを記述してください。※
 - 16 進文字列定数の途中で行を変える場合は、先頭の X の直後で改行しないでください。72 欄まで記述したあとに次の行の標識領域にハイフン (-) を記述し、プログラム原文領域の 12 欄以降からアポストロフィ (') に続けて、16 進文字列の続きを記述してください。※
 - 段落の見出しを DML と同一行に記述できません。記述した場合は、構文エラーになります。
 - 表意定数は DML 中に指定できません。指定した場合、表意定数の示す値として解釈されないため、構文エラーになるおそれがあります。
 - 文字列定数は、COBOL 処理系の定める最大長と DML の定める最大長のうち、短い方が使用可能な長さの上限になります。
- DML の定める最大長を超えた場合、UAP のプリプロセス時に構文エラーになります。プリプロセスが正常終了した場合でも、COBOL 処理系の定める最大長を超えていたときは、UAP のコンパイル時にエラーになります。

DML で使用可能な文字列定数の最大長については、マニュアル「HiRDB 構造型データベース機能」の「DML のデータ型」の「変換（比較）できるデータ型」を参照してください。

- ・ 標識領域（第 7 欄）に全角文字は記述できません。

注※

72 欄以前で改行した場合、pdsdbcbl コマンドは 72 欄までの空白を補います。

2.10.9 SDB データベースの特定規則

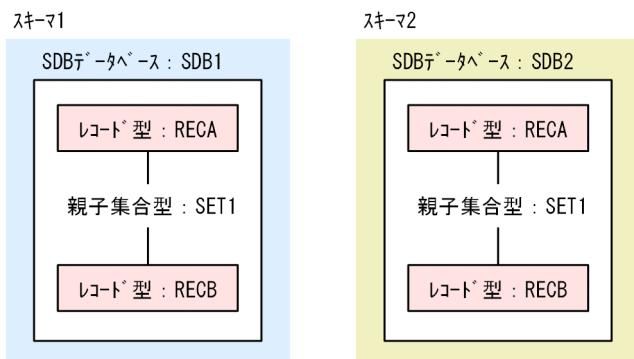
pdsdbcbl コマンドは、DML に指定されたレコード名からレコード型に関する定義情報を取得しています。情報を取得する際、SDB データベース節に指定された順に SDB データベースの定義情報を参照します。同じレコード型名が定義されている SDB データベースが複数指定されている場合は、最初に参照した SDB データベースの定義情報からレコード型に関する定義情報を取得します。

なお、DML に指定された親子集合型名について定義情報を取得する際にも、レコード型名と同様に SDB データベースの定義情報を参照します。ただし、DML にレコード名と親子集合型名の両方が指定されている場合は、参照する SDB データベースの特定はレコード名によって行われます。

DML のレコード名、親子集合型名から、SDB データベースを特定する例を次に示します。

(例)

SDB データベースの定義



- ・ UAP ソースファイル (SRC01)

```
SDB-DATABASE SECTION.
SDB SDB1, SDB2.                                ←SDB1が先に指定されている。
:
EXEC DML
    FETCH NEXT RECA INTO :XRECA    ←SDB1に定義されたレコード型RECAと解釈される。
END-DML.
EXEC DML
    FETCH NEXT RECB INTO :XRECB    ←SDB1に定義されたレコード型RECBと解釈される。
    WITHIN SET1
END-DML.
EXEC DML
```

```
FIND CURRENT OWNER OF SET1 ←SDB1に定義された親子集合型SET1と解釈される。  
END-DML.
```

- UAP ソースファイル (SRC02)

```
SDB-DATABASE SECTION.  
  SDB SDB2, SDB1.          ←SDB2が先に指定されている。  
  :  
  EXEC DML  
    FETCH NEXT RECA INTO :XRECA ←SDB2に定義されたレコード型RECAと解釈される。  
  END-DML.  
  EXEC DML  
    FETCH NEXT RECB INTO :XRECB ←SDB2に定義されたレコード型RECBと解釈される。  
    WITHIN SET1  
  END-DML.  
  EXEC DML  
    FIND CURRENT OWNER OF SET1 ←SDB2に定義された親子集合型SET1と解釈される。  
  END-DML.
```

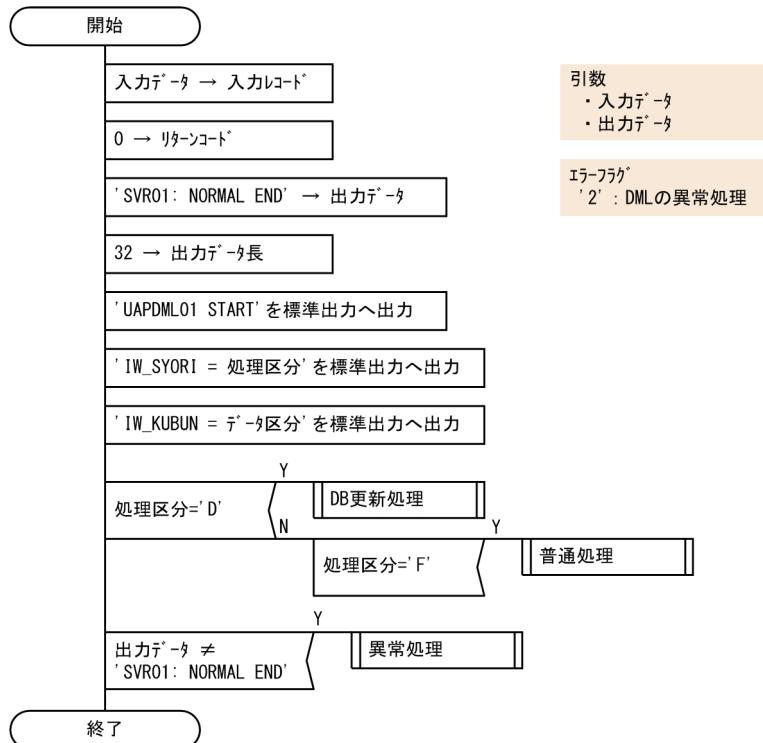
2.11 COBOL ソースプログラムのコーディング例

DML を記述した埋込み型 UAP (SDB データベースにアクセスする部分の UAP) の COBOL ソースプログラムの PAD チャートとコーディング例を示します。OpenTP1 環境下で実行する UAP (SPP) の例です。

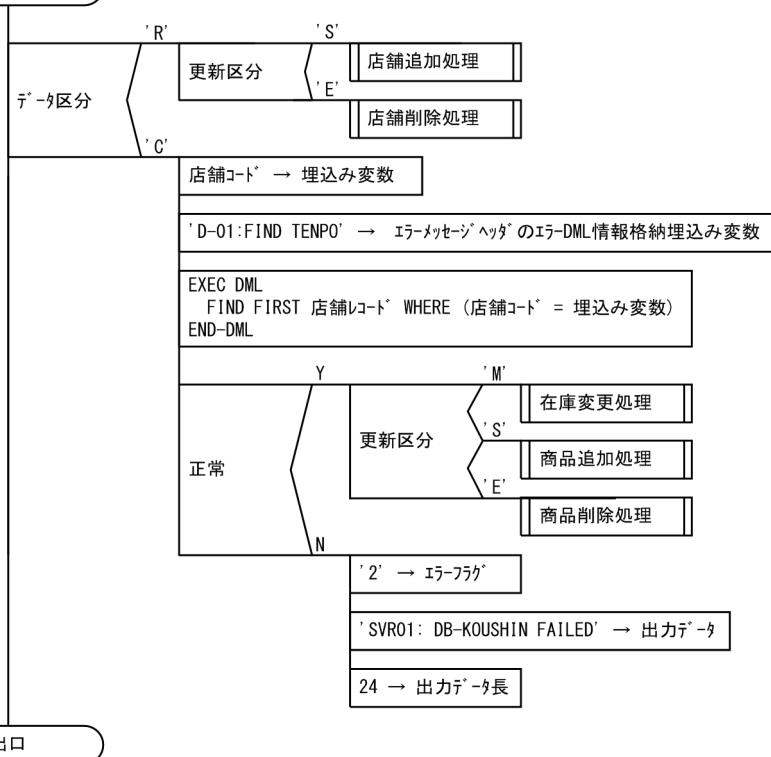
2.11.1 PAD チャート

UAP の PAD チャートを次の図に示します。

図 2-11 UAP の PAD チャート

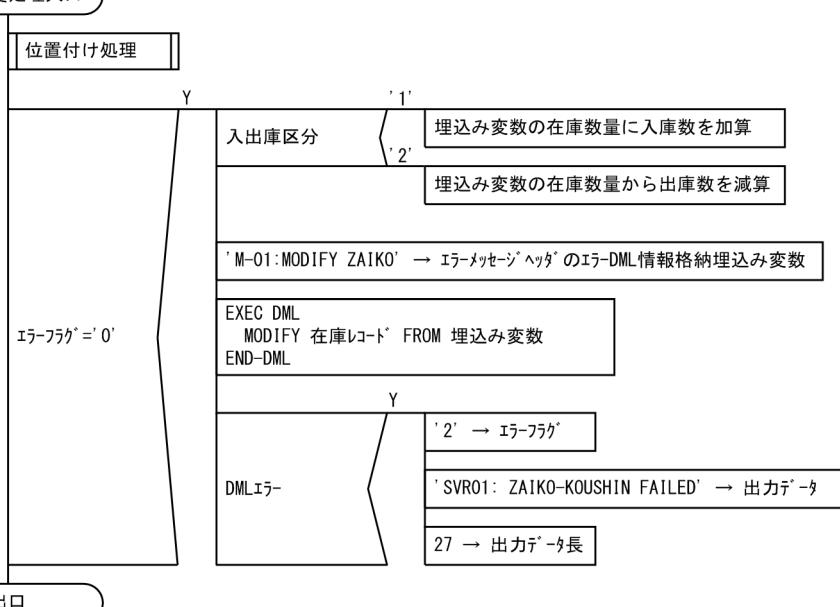


DB更新処理入口

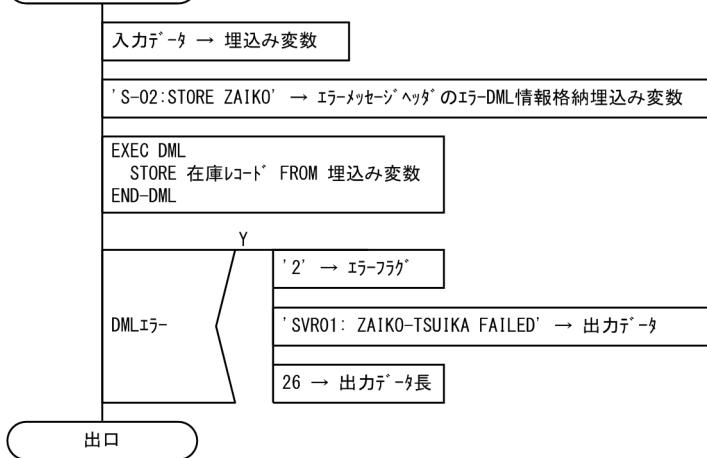


出口

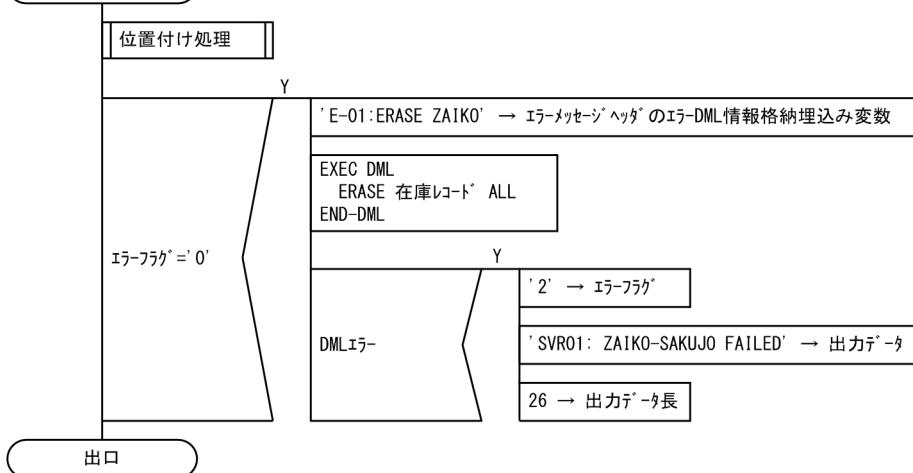
在庫変更処理入口



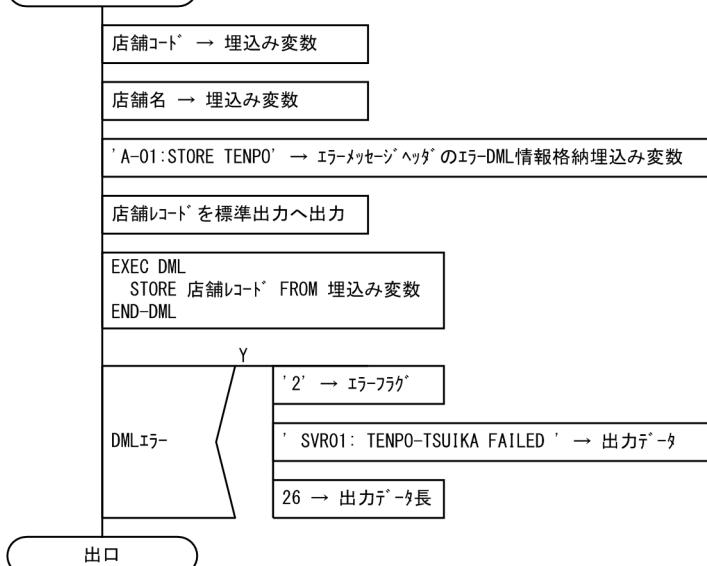
商品追加処理入口



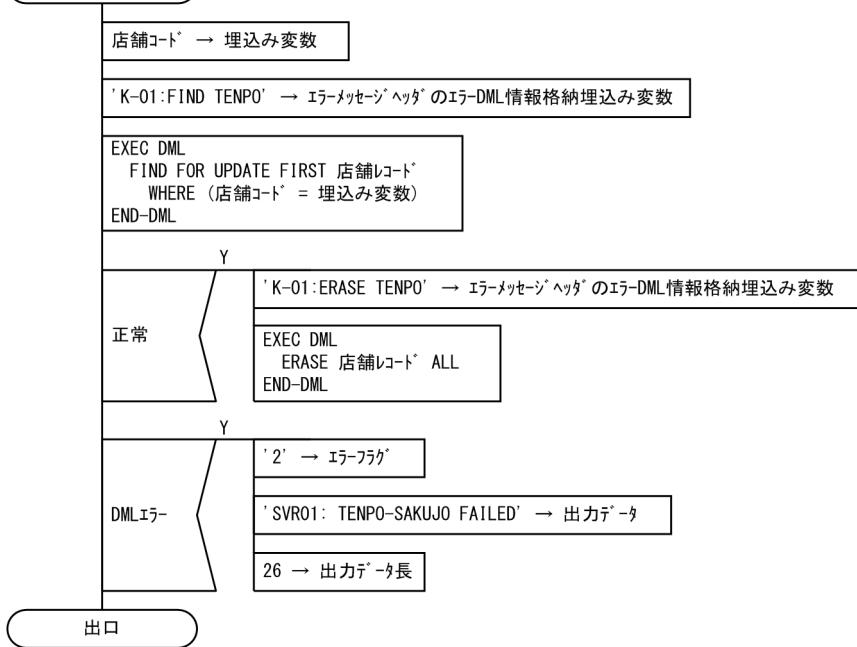
商品削除処理入口



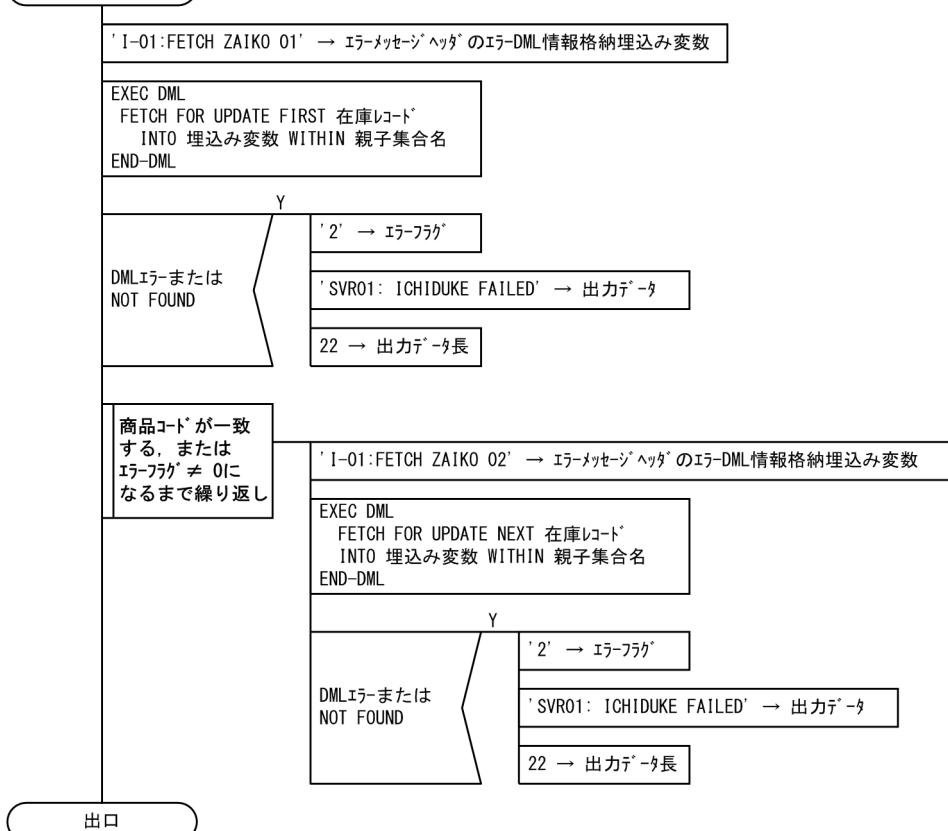
店舗追加処理入口



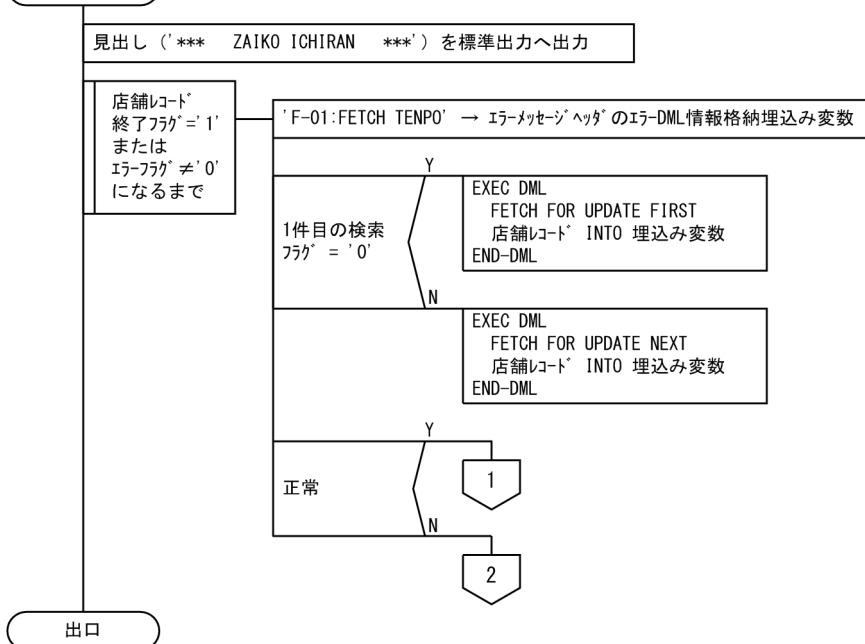
店舗削除処理入口



位置付け処理入口

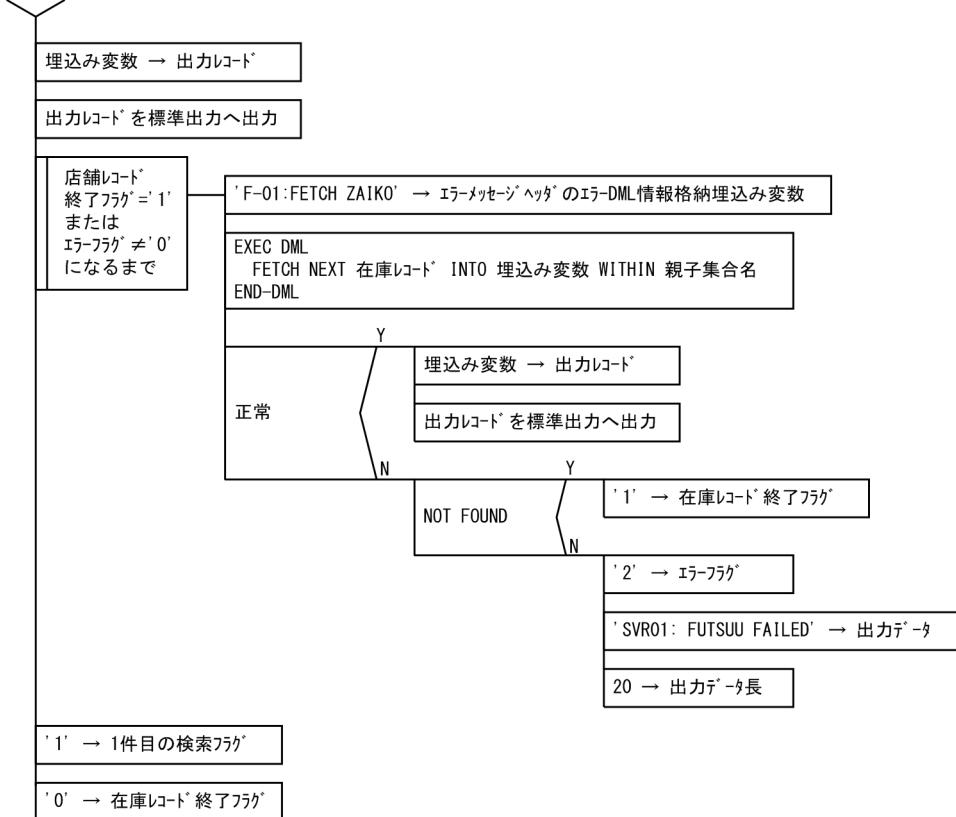


普通処理入口

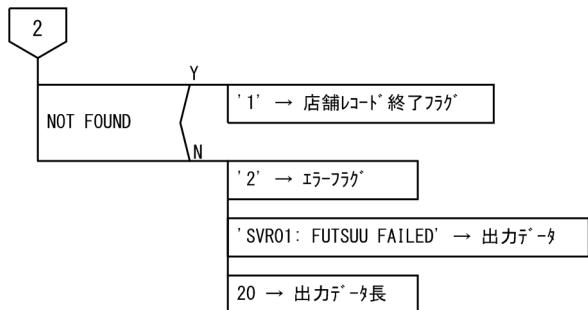


出口

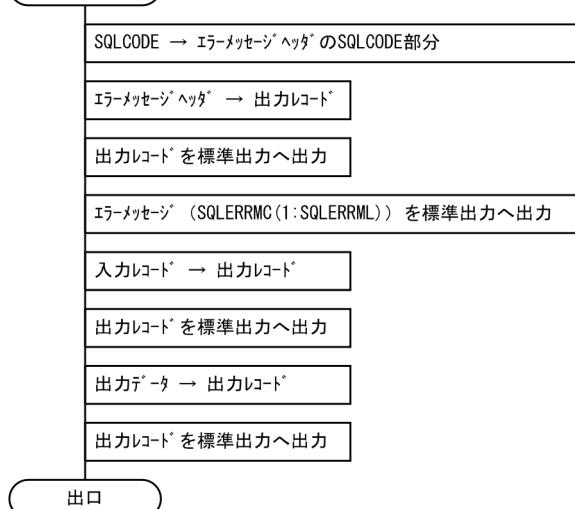
1



2. UAP の作成



異常処理



2.11.2 コーディング例

COBOL ソースプログラムのコーディング例を次に示します。

左端の番号は行番号を示しています。

```

1      IDENTIFICATION DIVISION.
2      PROGRAM-ID. UAPDML01.
3      *
4      ENVIRONMENT DIVISION.
5      *
6      INPUT-OUTPUT SECTION.
7      *
8      DATA DIVISION.
9      *
10     SDB-DATABASE SECTION.
11     SDB          DATABASE01
12     RECORD NAME  RECNAME
13     RECORD LENGTH RECLENG
14     .
15     *
16     WORKING-STORAGE SECTION.
17     *
18     77 EOF          PIC X VALUE '0'.
19     77 ERR-FLG        PIC X VALUE '0'.

```

```

20    77 TENPO_END          PIC X VALUE '0'.
21    77 ZAIKO_END         PIC X VALUE '0'.
22    77 TENPO_NOT_1ST     PIC X VALUE '0'.
23    *
24    77 RECNAME           PIC X(30) VALUE SPACE.      ←2.
25    77 RECLENG           PIC S9(8) COMP-5 VALUE ZERO. ←2.
26    *
27    01 OUTREC IS GLOBAL PIC X(132).
28    *
29    01 INREC IS GLOBAL PIC X(80).
30    01 INREC_W           REDEFINES INREC.
31      02 IW_SYORI        PIC X.
32      02 FILLER          PIC X.
33      02 IW_KUBUN        PIC X.
34      02 FILLER          PIC X(77).
35    *
36    01 INREC_R           REDEFINES INREC_W.
37      02 IR_SYORI        PIC X.
38      02 FILLER          PIC X.
39      02 IR_KUBUN        PIC X.
40      02 FILLER          PIC X.
41      02 IR_KSN-KUBUN    PIC X.
42      02 FILLER          PIC X.
43      02 IR_TENPO_CD     PIC X.
44      02 FILLER          PIC X.
45      02 IR_TENPO_NAME   PIC X(30).
46    01 INREC_C           REDEFINES INREC_W.
47      02 IC_SYORI        PIC X.
48      02 FILLER          PIC X.
49      02 IC_KUBUN        PIC X.
50      02 FILLER          PIC X.
51      02 IC_KSN-KUBUN    PIC X.
52      02 FILLER          PIC X.
53      02 IC_IO-KUBUN    PIC X.
54      02 FILLER          PIC X.
55      02 IC_TENPOID      PIC X.
56      02 FILLER          PIC X.
57      02 IC_SCODE        PIC X(4).
58      02 FILLER          PIC X.
59      02 IC_SNAME        PIC X(30).
60      02 FILLER          PIC X.
61      02 IC_SCOLOR       PIC X(10).
62      02 FILLER          PIC X.
63      02 IC_STANKA       PIC 9(10).
64      02 FILLER          PIC X.
65      02 IC_SSURYO       PIC 9(10).
66    *
67    01 TENPO.
68      02 RT_DBKEY.        -
69        03 TENPO_CD       PIC X.          3.
70      02 TENPO_NAME      PIC X(30).      -
71    *
72    01 ZAIKO.
73      02 CH_TENPO_CD     PIC X.
74      02 CH_DBKEY        PIC S9(8) COMP-5.      -
75      02 SCODE            PIC X(4).          4.
76      02 SNAME            PIC X(30).
77      02 SCOLOR           PIC X(10).

```

```

78      02 TANKA          PIC S9(8) COMP-5.      |
79      02 ZSURYO        PIC S9(8) COMP-5.      -
80      *
81      01 MIDASHI.      PIC X(80) VALUE
82          02 FILLER      '*** ZAIKO ICHIRAN ***'.
83      *
84      01 0_TENPO.
85          02 FILLER      PIC X(15) VALUE ' TENPO CODE : '.
86          02 0_TENPO_CD  PIC X.
87          02 FILLER      PIC X(10) VALUE ', NAME : "'".
88          02 0_TENPO_NAME PIC X(30).
89          02 FILLER      PIC X      VALUE "'".
90      *
91      01 0_ZAIKO.
92          02 FILLER      PIC X(15) VALUE ' ZAIKO CODE : '.
93          02 0_ZAIKO_CODE PIC X(4).
94          02 FILLER      PIC X(10) VALUE ', NAME : "'".
95          02 0_ZAIKO_NAME PIC X(30).
96          02 FILLER      PIC X(11) VALUE '", COLOR : '.
97          02 0_ZAIKO_COLOR PIC X(10).
98          02 FILLER      PIC X(10) VALUE ', TANKA : '.
99          02 0_ZAIKO_TANKA PIC ZZZZZZZ9.
100         02 FILLER      PIC X(11) VALUE ', SUURYO : '.
101         02 0_ZAIKO_SUURYO PIC ZZZZZZZ9.
102      *
103      01 ERR_MSG.
104          02 FILLER      PIC X(22) VALUE '*** ERROR SQLCODE : '.
105          02 EM_SQLCODE    PIC -ZZZZZZZZ9.
106          02 FILLER      PIC X(15) VALUE ', ERROR DML : "'".
107          02 EM_ERRDML    PIC X(30).
108          02 FILLER      PIC X(7) VALUE '" ***'.
109      *
110      *
111      LINKAGE SECTION.
112      *
113      77 IN_DATA        PIC X(80).           ←5.
114      77 IN LENG        PIC S9(9) COMP-5.    ←6.
115      77 OUT_DATA       PIC X(32).          ←7.
116      77 OUT LENG       PIC S9(9) COMP-5.    ←8.
117      *
118      PROCEDURE DIVISION USING IN_DATA IN LENG OUT DATA OUT LENG.
119      MAIN SECTION.
120      * test
121          MOVE IN_DATA TO INREC.
122      *
123      M-01.
124          MOVE 0 TO RETURN-CODE.
125          MOVE 'SVR01: NORMAL END' TO OUT DATA           ←9.
126          MOVE 32 TO OUT LENG           ←9.
127          DISPLAY 'UAPDML01 START'.
128          DISPLAY 'IW_SYORI =' IW_SYORI.
129          DISPLAY 'IW_KUBUN =' IW_KUBUN.
130      *
131      M-02.
132          IF IW_SYORI = 'D'
133          THEN
134              PERFORM DB-KOUSHIN
135          ELSE IF IW_SYORI = 'F'

```

```

136          PERFORM FUTSUU
137      END-IF.
138      *
139          IF OUT_DATA NOT = 'SVR01: NORMAL END'
140          THEN
141              PERFORM IJYOU
142          END-IF.
143      M-EXIT.
144          EXIT PROGRAM.
145      *
146          DB-KOUSHIN SECTION.
147          D-01.
148          * MOVE INREC TO INREC_W
149          IF IW_KUBUN = 'R'
150          THEN
151              EVALUATE IR_KSN-KUBUN
152                  WHEN 'S'
153                      PERFORM TENPO-TSUIKA
154                  WHEN 'E'
155                      PERFORM TENPO-SAKUJO
156          END-EVALUATE
157      ELSE
158          MOVE IC_TENPOID TO TENPO_CD
159          MOVE 'D-01:FIND TENPO' TO EM_ERRDML
160          EXEC DML
161              FIND FIRST "TENPO" WHERE ( "DBKEY" = :RT_DBKEY )      | 10.
162          END-DML
163          IF SQLCODE = 0                                ←11.
164              EVALUATE IC_KSN-KUBUN
165                  WHEN 'M'
166                      PERFORM ZAIKO-KOUSHIN
167                  WHEN 'S'
168                      PERFORM ZAIKO-TSUIKA
169                  WHEN 'E'
170                      PERFORM ZAIKO-SAKUJO
171          END-EVALUATE
172      ELSE
173          MOVE '2' TO ERR-FLG
174          MOVE 'SVR01: DB-KOUSHIN FAILED' TO OUT_DATA      ←12.
175          MOVE 24 TO OUT LENG
176          END-IF
177      END-IF.
178      D-EXIT.
179          EXIT.
180      *
181          ZAIKO-KOUSHIN SECTION.
182          M-01.
183              PERFORM ICHIDUKE.          ←13.
184              IF ERR-FLG = '0'
185              THEN
186                  IF IC_IO-KUBUN = '1'
187                      THEN
188                          COMPUTE ZSURYO = ZSURYO + IC_SSURYO      ←14.
189                      ELSE
190                          COMPUTE ZSURYO = ZSURYO - IC_SSURYO      ←14.
191                      END-IF
192                  MOVE 'M-01:MODIFY ZAIKO' TO EM_ERRDML
193                  EXEC DML

```

```

194      MODIFY ZAIKO FROM :ZAIKO           ←15.
195      END-DML
196      IF SQLCODE NOT = 0                ←11.
197      THEN
198          MOVE '2' TO ERR-FLG
199          MOVE 'SVR01: ZAIKO-KOUSHIN FAILED' TO OUT_DATA   ←16.
200          MOVE 27 TO OUT LENG
201      ELSE
202          CONTINUE
203      END-IF
204      END-IF.
205      M-EXIT.
206      EXIT.
207      *
208      ZAIKO-TSUIKA SECTION.
209      S-01.
210          MOVE IC_TENPOID TO TENPO_CD.      ←17.
211          MOVE 0 TO CH_DBKEY.           ←18.
212          MOVE IC_SCODE TO SCODE.        -
213          MOVE IC_SNAME TO SNAME.        |
214          MOVE IC_SCOLOR TO SCOLOR.      17.
215          MOVE IC_STANKA TO TANKA.        |
216          MOVE IC_SSURYO TO ZSURYO.      -
217      S-02.
218          MOVE 'S-02:STORE ZAIKO' TO EM_ERRDML.
219          EXEC DML
220              STORE ZAIKO FROM :ZAIKO           ←19.
221          END-DML.
222          IF SQLCODE NOT = 0                ←11.
223          THEN
224              MOVE '2' TO ERR-FLG
225              MOVE 'SVR01: ZAIKO-TSUIKA FAILED' TO OUT_DATA   ←12.
226              MOVE 26 TO OUT LENG
227          ELSE
228              CONTINUE
229          END-IF.
230          S-EXIT.
231          EXIT.
232      *
233      ZAIKO-SAKUJO SECTION.
234      E-01.
235          PERFORM ICHIDUKE.           ←20.
236          IF ERR-FLG = '0'
237          THEN
238              MOVE 'E-01:ERASE ZAIKO' TO EM_ERRDML
239              EXEC DML
240                  ERASE ZAIKO ALL           ←21.
241              END-DML
242              IF SQLCODE NOT = 0           ←11.
243              THEN
244                  MOVE '2' TO ERR-FLG
245                  MOVE 'SVR01: ZAIKO-SAKUJO FAILED' TO OUT_DATA   ←12.
246                  MOVE 26 TO OUT LENG
247              ELSE
248                  CONTINUE
249              END-IF
250          END-IF.
251      E-EXIT.

```

```

252      EXIT.
253      *
254      TENPO-TSUIKA SECTION.
255      A-01.
256      MOVE IR_TENPO_CD TO TENPO_CD.           ←22.
257      MOVE IR_TENPO_NAME TO TENPO_NAME.       ←17.
258      MOVE 'A-01:STORE TENPO' TO EM_ERRDML
259      DISPLAY TENPO.
260      EXEC DML
261      STORE TENPO FROM :TENPO               ←23.
262      END-DML.
263      IF SQLCODE NOT = 0                   ←11.
264      THEN
265          MOVE '2' TO ERR-FLG
266          MOVE 'SVR01: TENPO-TSUIKA FAILED' TO OUT_DATA   ←12.
267          MOVE 26 TO OUT LENG
268      ELSE
269          CONTINUE
270      END-IF.
271      A-EXIT.
272      EXIT.
273      *
274      TENPO-SAKUJO SECTION.
275      K-01.
276      MOVE IR_TENPO_CD TO TENPO_CD.           ←24.
277      MOVE 'K-01:FIND TENPO' TO EM_ERRDML
278      EXEC DML
279      FIND FOR UPDATE FIRST TENPO           ←25.
280          WHERE ( "DBKEY" = :RT_DBKEY )      ←25.
281      END-DML.
282      IF SQLCODE = 0                      ←11.
283      THEN
284          MOVE 'K-01:ERASE TENPO' TO EM_ERRDML
285          EXEC DML
286          ERASE TENPO ALL                 ←26.
287          END-DML
288      ELSE
289          CONTINUE
290      END-IF.
291      IF SQLCODE NOT = 0                   ←11.
292      THEN
293          MOVE '2' TO ERR-FLG
294          MOVE 'SVR01: TENPO-SAKUJO FAILED' TO OUT_DATA   ←12.
295          MOVE 26 TO OUT LENG
296      ELSE
297          CONTINUE
298      END-IF.
299      K-EXIT.
300      EXIT.
301      *
302      ICHIDUKE SECTION.
303      I-01.
304      MOVE 'I-01:FETCH ZAIKO 01' TO EM_ERRDML.
305      EXEC DML
306      FETCH FOR UPDATE FIRST ZAIKO          ←27.
307          INTO :ZAIKO WITHIN TENPO_ZAIKO
308      END-DML.
309      IF SQLCODE NOT = 0                   ←11.

```

```

310
311      MOVE '2' TO ERR-FLG
312      MOVE 'SVR01: ICHIDUKE FAILED' TO OUT_DATA      ←12.
313      MOVE 22 TO OUT LENG
314  ELSE
315      CONTINUE
316  END-IF.
317  PERFORM UNTIL ( IC_SCODE = SCODE OR ERR-FLG NOT = '0' )
318      MOVE 'I-01:FETCH ZAIKO 02' TO EM_ERRDML
319      EXEC DML
320          FETCH FOR UPDATE NEXT ZAIKO      ←28.
321          INTO :ZAIKO WITHIN TENPO_ZAIKO
322  END-DML
323  IF SQLCODE NOT = 0      ←11.
324  THEN
325      MOVE '2' TO ERR-FLG
326      MOVE 'SVR01: ICHIDUKE FAILED' TO OUT_DATA      ←12.
327      MOVE 22 TO OUT LENG
328  ELSE
329      CONTINUE
330  END-IF
331  END-PERFORM.
332  I-EXIT.
333      EXIT.
334  *
335  FUTSUU SECTION.
336  F-01.
337      MOVE MIDASHI TO OUTREC.
338      DISPLAY OUTREC UPON SYSOUT.
339      PERFORM UNTIL ( TENPO_END = '1' OR ERR-FLG NOT = '0' )
340          MOVE 'F-01:FETCH TENPO' TO EM_ERRDML
341          IF TENPO_NOT_1ST = '0'
342  THEN
343      EXEC DML
344          FETCH FOR UPDATE FIRST TENPO INTO :TENPO      ←29.
345  END-DML
346  ELSE
347      EXEC DML
348          FETCH FOR UPDATE NEXT TENPO INTO :TENPO      ←30.
349  END-DML
350  END-IF
351  IF SQLCODE = 0      ←11.
352  THEN
353      MOVE TENPO_CD TO O_TENPO_CD
354      MOVE TENPO_NAME TO O_TENPO_NAME
355      MOVE O_TENPO TO OUTREC
356      DISPLAY OUTREC UPON SYSOUT
357      PERFORM UNTIL ( ZAIKO_END = '1' OR ERR-FLG NOT = '0' )
358          MOVE 'F-01:FETCH ZAIKO' TO EM_ERRDML
359          EXEC DML
360              FETCH NEXT ZAIKO INTO :ZAIKO WITHIN TENPO_ZAIKO      ←31.
361  END-DML
362  IF SQLCODE = 0      ←11.
363  THEN
364      MOVE SCODE TO O_ZAIKO_CODE
365      MOVE SNAME TO O_ZAIKO_NAME
366      MOVE SCOLOR TO O_ZAIKO_COLOR
367      MOVE TANKA TO O_ZAIKO_TANKA

```

```

368      MOVE ZSURYO TO O_ZAIKO_SUURYO          |
369      MOVE O_ZAIKO TO OUTREC                -
370      DISPLAY OUTREC UPON SYSOUT
371      ELSE
372          IF SQLCODE = 100
373          THEN
374              MOVE '1' TO ZAIKO_END
375          ELSE
376              MOVE '2' TO ERR-FLG
377              MOVE 'SVR01: FUTSUU FAILED' TO OUT_DATA    ←12.
378              MOVE 20 TO OUT LENG
379          END-IF
380      END-IF
381      END-PERFORM
382      MOVE '1' TO TENPO_NOT_1ST
383      MOVE '0' TO ZAIKO_END
384      ELSE
385          IF SQLCODE = 100
386          THEN
387              MOVE '1' TO TENPO_END
388          ELSE
389              MOVE '2' TO ERR-FLG
390              MOVE 'SVR01: FUTSUU FAILED' TO OUT_DATA    ←12.
391              MOVE 20 TO OUT LENG
392          END-IF
393      END-IF
394      END-PERFORM.
395      F-EXIT.
396      EXIT.
397      *
398      IJYOU SECTION.
399      J-01.
400          MOVE SQLCODE TO EM_SQLCODE.          ←35.
401          MOVE ERR_MSG TO OUTREC.
402          DISPLAY OUTREC UPON SYSOUT.        ←36.
403          DISPLAY SQLERRMC(1:SQLERRML) UPON SYSOUT. ←37.
404          MOVE INREC TO OUTREC.
405          DISPLAY OUTREC UPON SYSOUT.        ←38.
406          MOVE OUT_DATA TO OUTREC.
407          DISPLAY OUTREC UPON SYSOUT.        ←39.
408      J-EXIT.
409      EXIT.

```

[説明]

1. SDB データベース節を指定します。

- UAP 内の DML でアクセスする SDB データベースを指定します。
- DML の実行後にレコード名を受け取る埋込み変数の名前を指定します。
- DML の実行後にレコード長を受け取る埋込み変数の名前を指定します。

SDB データベース節については、「[2.3 SDB データベース節の記述](#)」を参照してください。

2. 次の埋込み変数を宣言します。

- SDB データベース節の RECORD NAME で指定した埋込み変数を宣言します。

2. UAP の作成

- SDB データベース節の RECORD LENGTH で指定した埋込み変数を宣言します。
3. レコード型 TENPO とデータの受け渡しを行う埋込み変数を宣言します。
埋込み変数の宣言については、「[2.4 埋込み変数の宣言](#)」を参照してください。
4. レコード型 ZAIKO とデータの受け渡しを行う埋込み変数を宣言します。
5. クライアント UAP から値が渡されるデータ領域：入力パラメタ
6. クライアント UAP から値が渡されるデータ領域：入力パラメタ長
7. UAP で値を設定するデータ領域：サービスプログラムの応答
8. UAP で値を設定するデータ領域：サービスプログラムの応答の長さ
9. SUP に返すメッセージを正常時の値で初期設定します。
10. ルートレコードのデータベースキーの一致する TENPO レコードに位置指示子を位置づけます。
DML 先頭子 (EXEC DML) に続けて DML を記述します。DML の直後に DML 終了子 (END-DML) を記述します。
11. SQLCODE を参照して DML の実行結果を判定します。
DML の実行結果の判定については、「[2.7 DML の実行結果の判定処理](#)」を参照してください。
12. SUP に返すメッセージにエラーを示す値を設定します。
13. レコード実現値の変更前に、変更するレコードに対して位置指示子を位置づけます。
14. 変更する構成要素に対応する埋込み変数に更新値を設定します。
15. 位置づけした ZAIKO レコードのユーザデータを埋込み変数の値に変更します。
16. SUP 側でチェックするメッセージです。
17. 構成要素に対応する埋込み変数に格納するデータを設定します。
18. 一連番号は HiRDB/SD が割り当てます。
19. 親レコード TENPO への位置づけ後に、子レコード ZAIKO を格納します。
20. レコードの削除をする前に、削除するレコードに位置指示子を位置づけます。
21. 位置づけした ZAIKO レコードを削除します。
22. TENPO レコードはルートレコードのため、格納前にルートレコードのデータベースキーの値を埋込み変数に設定します。
23. ルートレコードの TENPO レコードを格納します。
24. キーの検索条件に指定する埋込み変数に、削除対象のルートレコードのデータベースキーの値を設定します。
25. レコード実現値の削除前に、削除するレコードに対して位置指示子を更新指定で位置づけます。削除するレコードはキーの検索条件で指定します。
26. 位置指示子が位置づけられているレコードを削除します。下位レコードがある場合は、下位レコードも同時に削除されます。

27. 子レコード ZAIKO の検索は、親レコード TENPO の位置づけ後に行います。更新指定で先頭の ZAIKO レコードへの位置づけを行い、レコード実現値を埋込み変数 ZAIKO に取得します。
28. 現在位置づけられている ZAIKO レコードの次のレコードに更新指定で位置づけを行い、レコード実現値を埋込み変数 ZAIKO に取得します。
29. 先頭の TENPO レコードへの位置づけを行い、レコード実現値を埋込み変数 TENPO に取得します。
30. 現在位置づけられている TENPO レコードの次のレコードに位置づけを行い、レコード実現値を埋込み変数 TENPO に取得します。
31. ZAIKO レコードに位置づけがない場合は、先頭の ZAIKO レコードに位置づけを行い、レコード実現値を埋込み変数 ZAIKO に取得します。
位置づけられている場合、位置づけられている ZAIKO レコードの次のレコードに位置づけを行い、レコード実現値を埋込み変数 ZAIKO に取得します。
32. 埋込み変数からレコード実現値を取り出します。
33. SQLCODE が 100 かどうかを判定します。100 の場合は、TENPO レコード下のすべての ZAIKO レコードの検索が完了しています。
34. SQLCODE が 100 かどうかを判定します。100 の場合は、すべての TENPO レコードの検索が完了しています。
35. エラー要因を取得するため、SQLCODE をエラーメッセージに含めます。
36. エラーメッセージを出力します。
37. HiRDB のエラーメッセージを出力します。
38. エラーが発生した入力データを出力します。
39. エラーが発生した手続きを出力します。

2.12 DML と SQL の両方を実行する UAP を作成する場合の考慮点

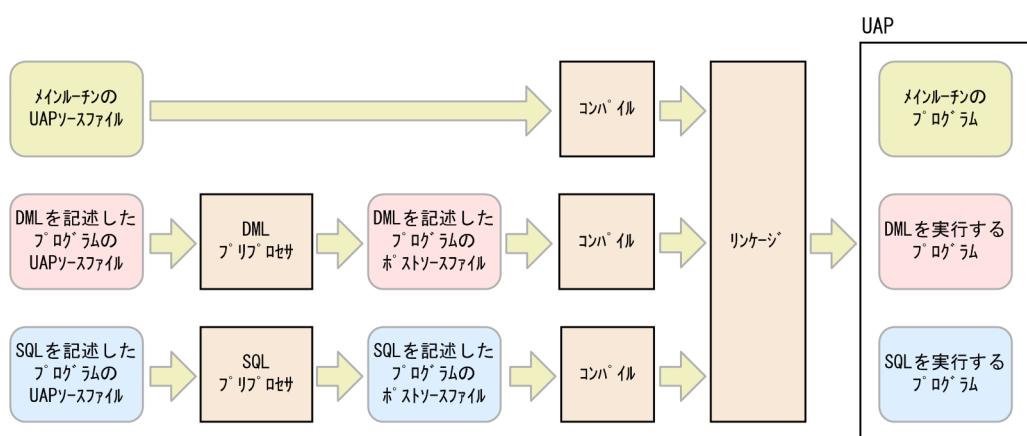
DML と SQL の両方を実行する UAP を作成する場合の考慮点について説明します。

2.12.1 UAP ソースファイルの構成

1 つの UAP ソースファイル中に、DML と SQL の両方を記述することはできません。DML を記述する UAP ソースファイルと、SQL を記述する UAP ソースファイルを別々に作成してください。各 UAP ソースファイルをプリプロセスしたあとに、コンパイルとリンクエージを実行します。

DML と SQL の両方を実行する UAP の実行可能ファイル作成までの流れを次の図に示します。

図 2-12 DML と SQL の両方を実行する UAP の実行可能ファイル作成までの流れ



2.12.2 DML と SQL の両方を実行する UAP のトランザクション制御

DML と SQL の両方を実行する UAP を作成する場合、HiRDB サーバへの接続、HiRDB サーバからの切り離し、およびトランザクション制御は SQL で行います。

HiRDB サーバに接続する場合は CONNECT 文を、HiRDB サーバから切り離す場合は DISCONNECT 文を実行します。トランザクションをコミットする場合は COMMIT 文を、トランザクションを取り消す場合は ROLLBACK 文を実行します。

「図 2-12 DML と SQL の両方を実行する UAP の実行可能ファイル作成までの流れ」で示す「SQL を記述したプログラムの UAP ソースファイル」中に、CONNECT 文や COMMIT 文などを記述します。

COBOL ソースプログラムのコーディング例については、「2.12.3 COBOL ソースプログラムのコーディング例 (DML と SQL の両方を実行する UAP の場合)」を参照してください。

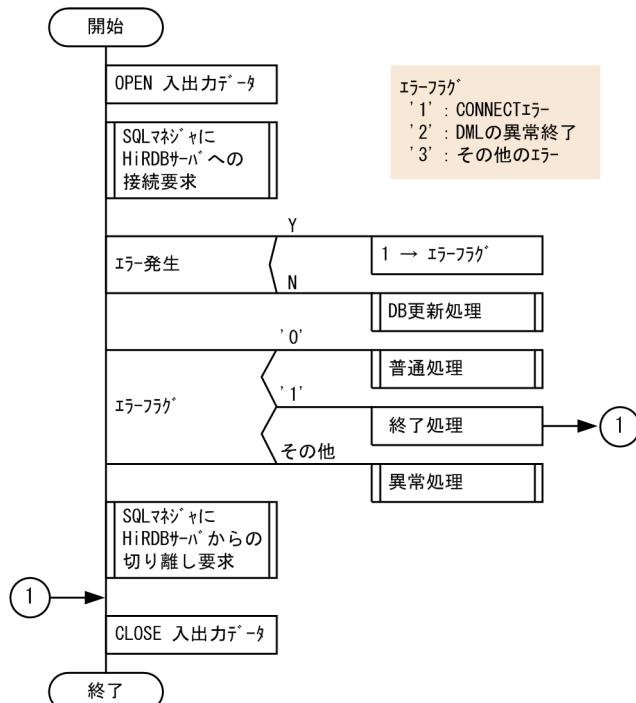
2.12.3 COBOL ソースプログラムのコーディング例 (DML と SQL の両方を実行する UAP の場合)

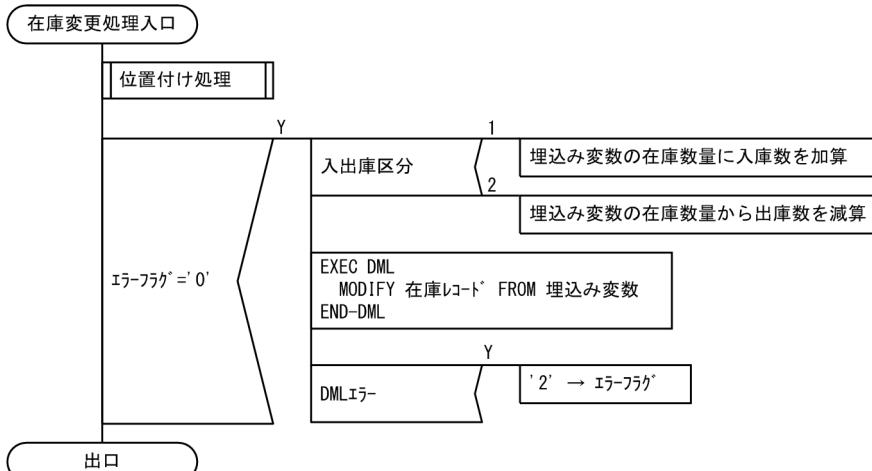
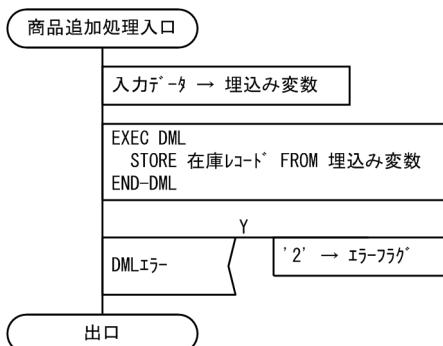
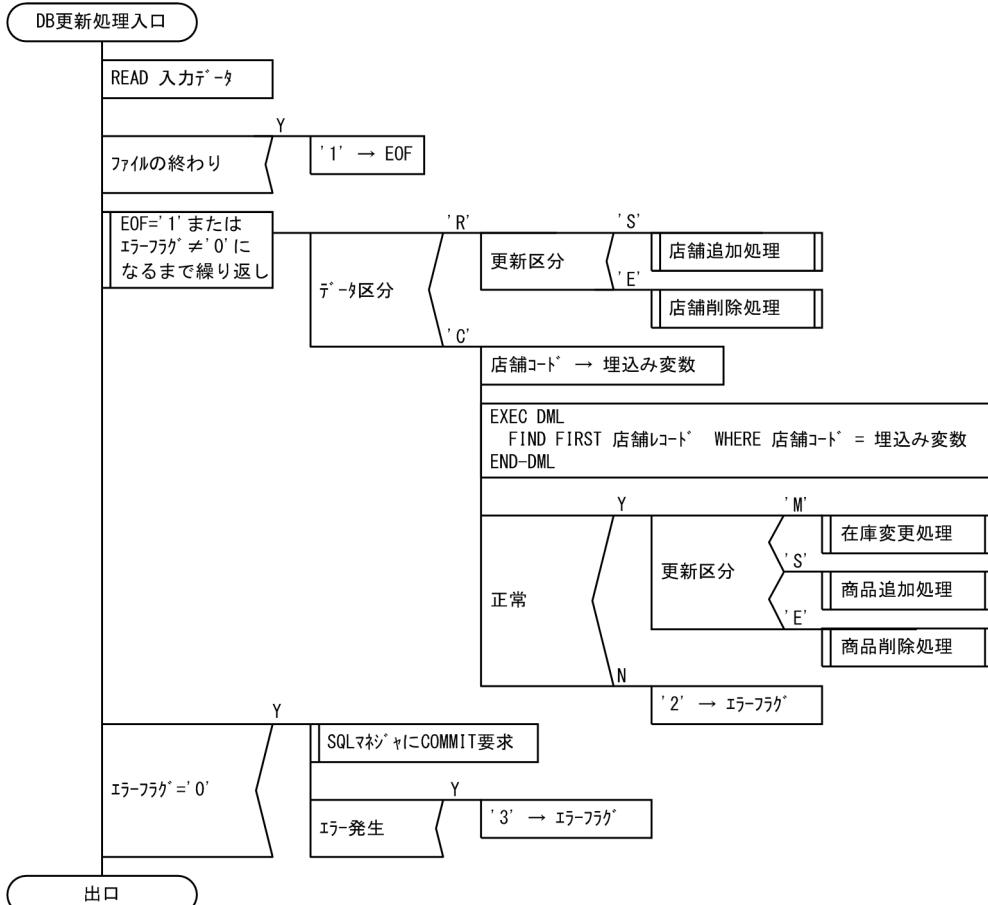
DML を記述した埋込み型 UAP (SDB データベースにアクセスする部分の UAP) の COBOL ソースプログラムの PAD チャートとコーディング例を示します。DML と SQL の両方を実行する UAP の例です。

(1) PAD チャート

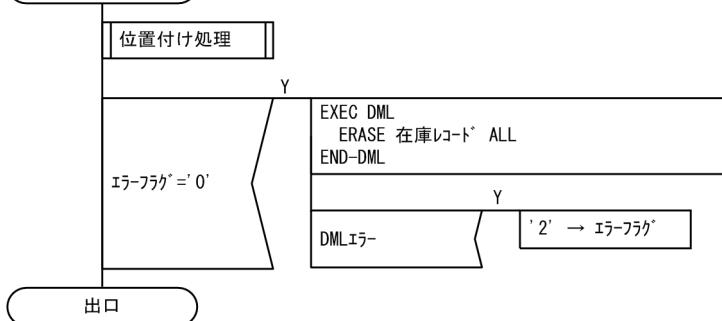
UAP の PAD チャートを次の図に示します。

図 2-13 UAP の PAD チャート





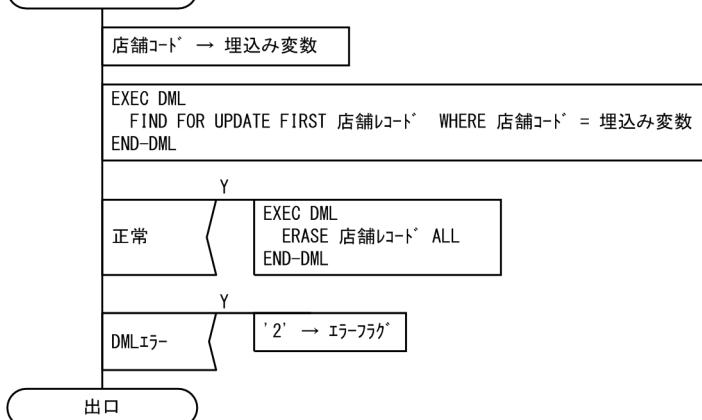
商品削除処理入口



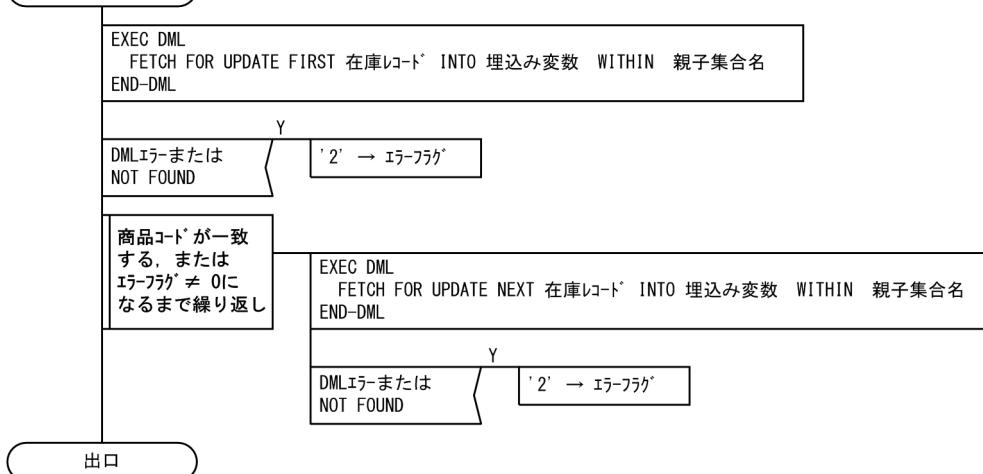
店舗追加処理入口



店舗削除処理入口

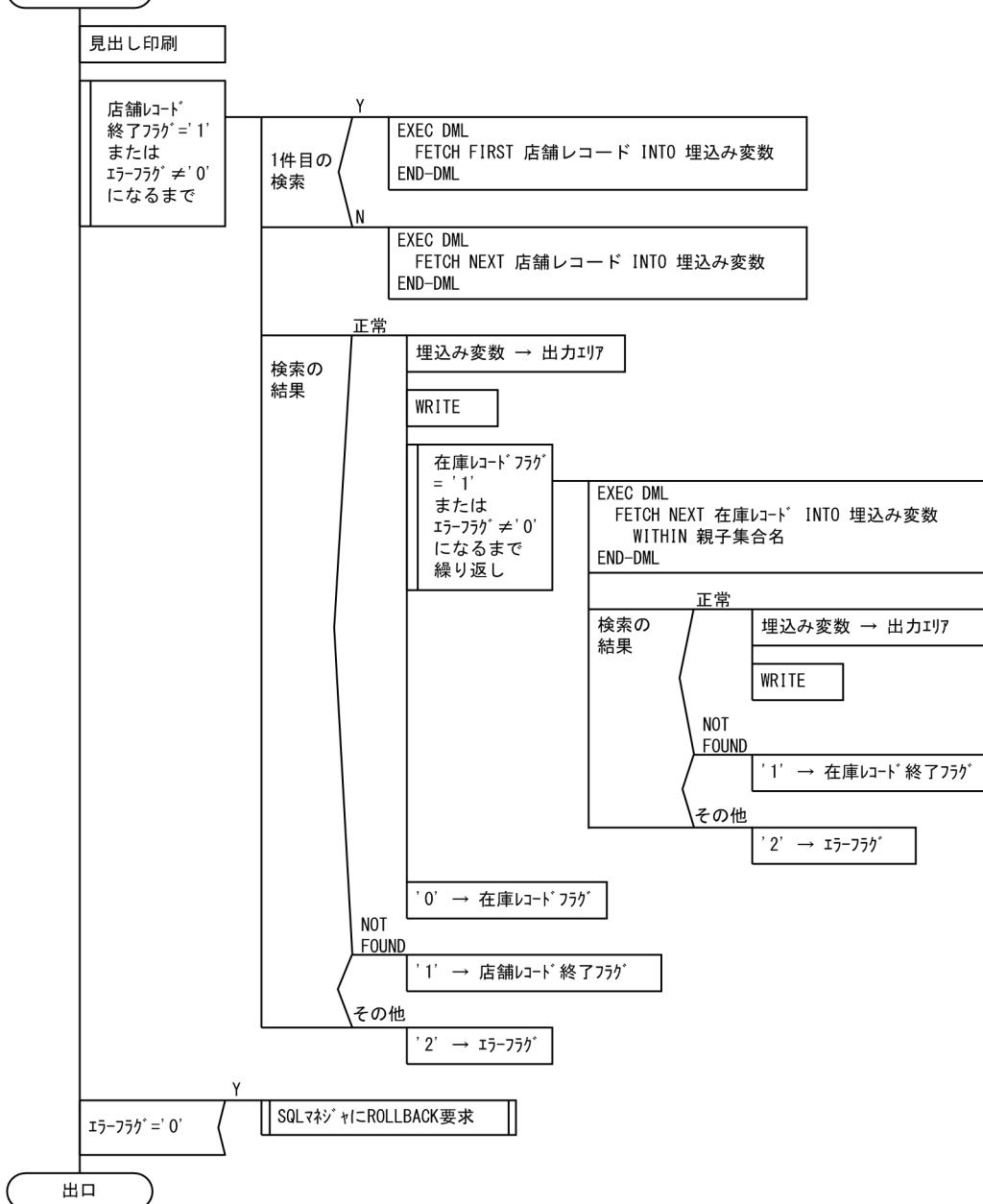


位置付け処理入口

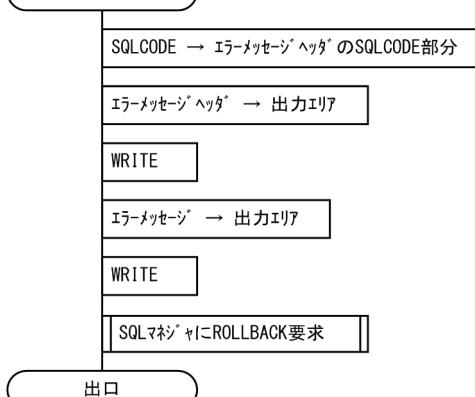


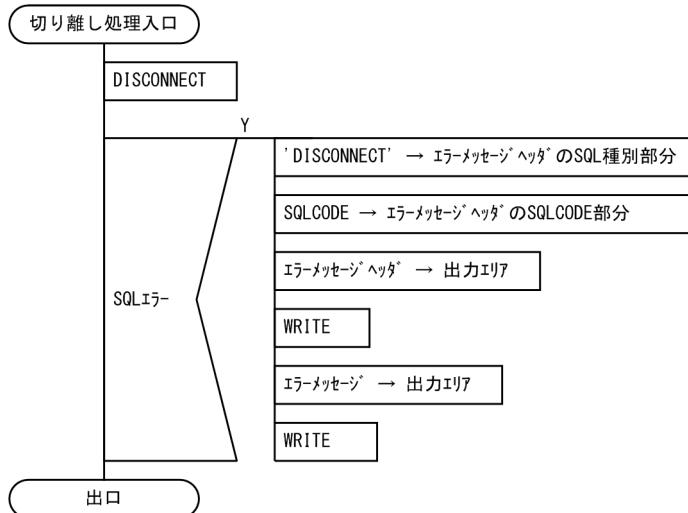
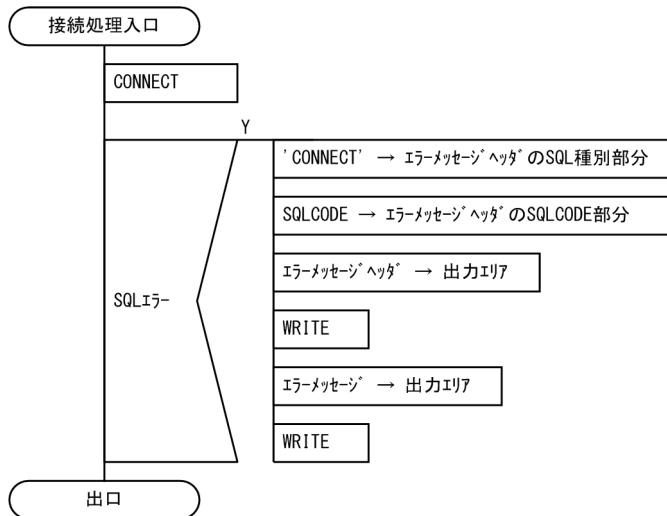
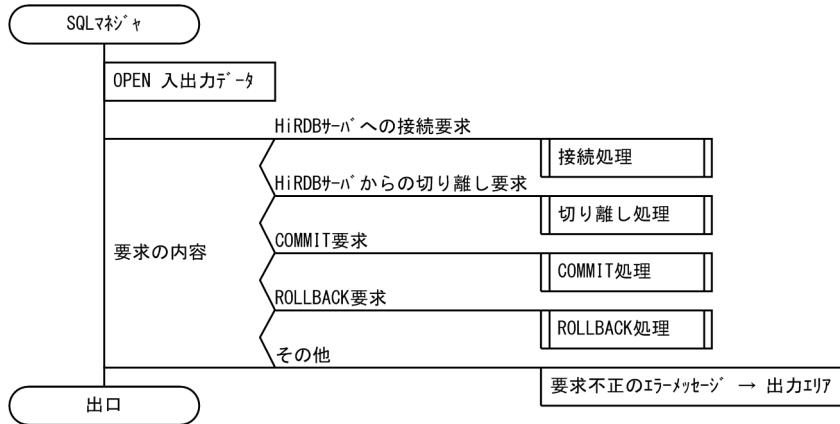
2. UAP の作成

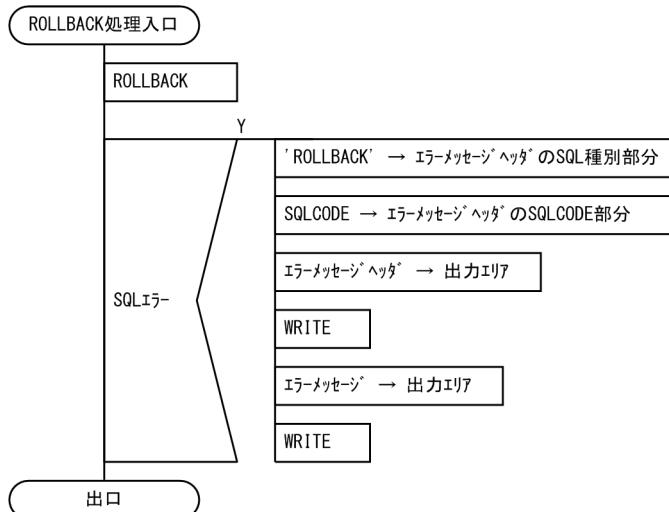
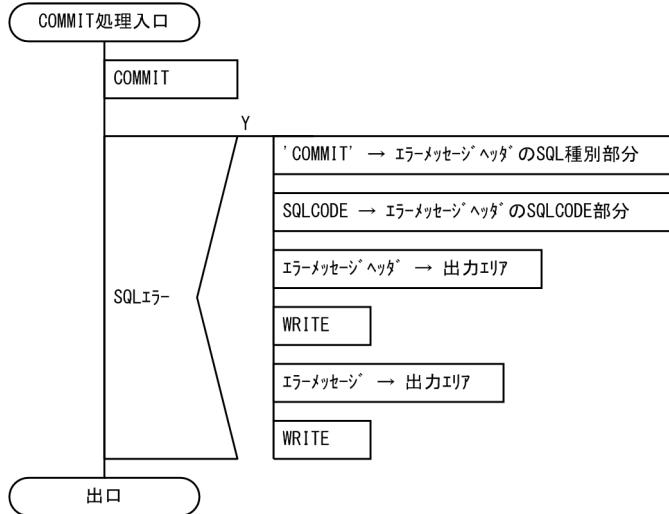
普通処理入口



異常処理







(2) コーディング例

次の COBOL ソースプログラムのコーディング例を説明します。

- DML を記述した COBOL ソースプログラム (UAPDML01)
- SQL を記述した COBOL ソースプログラム (UAPSQL01)

左端の番号は行番号を示しています。

■DML を記述した COBOL ソースプログラム (UAPDML01) のコーディング例

```

1 IDENTIFICATION DIVISION.
2   PROGRAM-ID. UAPDML01.
3 *
4   ENVIRONMENT DIVISION.
5 *
6   INPUT-OUTPUT SECTION.
7     FILE-CONTROL.
8     SELECT 0-FILE
9       ASSIGN TO './UAPDML01.log'
  
```

```

10      LINE SEQUENTIAL.
11      SELECT I-FILE
12          ASSIGN TO './UAPDML01.txt'
13          LINE SEQUENTIAL.
14      DATA DIVISION.
15      FILE SECTION.
16      FD 0-FILE          DATA RECORD OUTREC.
17          01 OUTREC          PIC X(132).
18      FD I-FILE          DATA RECORD INREC.
19          01 INREC          PIC X(80).
20      *
21      SDB-DATABASE SECTION.
22          SDB             DATABASE01
23          RECORD NAME    RECNAME
24          RECORD LENGTH   RECLENG
25      -
26      *
27      WORKING-STORAGE SECTION.
28      *
29          77 EOF            PIC X VALUE '0'.
30          77 ERR-FLG          PIC X VALUE '0'.
31          77 TENPO_END        PIC X VALUE '0'.
32          77 ZAIKO_END        PIC X VALUE '0'.
33          77 TENPO_NOT_1ST    PIC X VALUE '0'.
34      *
35          77 REQSQL_CNCT      PIC X(4) VALUE 'CNCT'.
36          77 REQSQL_DISC      PIC X(4) VALUE 'DISC'.
37          77 REQSQL_COMT      PIC X(4) VALUE 'COMT'.
38          77 REQSQL_ROLB      PIC X(4) VALUE 'ROLB'.
39      *
40          77 RECNAME          PIC X(30)      VALUE SPACE.      ←2.
41          77 RECLENG          PIC S9(8) COMP VALUE ZERO.      ←2.
42      *
43          01 INREC_W.
44              02 IW_KUBUN        PIC X.
45              02 FILLER          PIC X(79).
46          01 INREC_R.
47              02 IR_KUBUN        PIC X.
48              02 FILLER          PIC X.
49              02 IR_KSN-KUBUN    PIC X.
50              02 FILLER          PIC X.
51              02 IR_TENPO_CD     PIC X.
52              02 FILLER          PIC X.
53              02 IR_TENPO_NAME   PIC X(30).
54          01 INREC_C.
55              02 IC_KUBUN        PIC X.
56              02 FILLER          PIC X.
57              02 IC_KSN-KUBUN    PIC X.
58              02 FILLER          PIC X.
59              02 IC_IO-KUBUN     PIC X.
60              02 FILLER          PIC X.
61              02 IC_TENPOID      PIC X.
62              02 FILLER          PIC X.
63              02 IC_SCODE         PIC X(4).
64              02 FILLER          PIC X.
65              02 IC_SNAME         PIC X(30).
66              02 FILLER          PIC X.
67              02 IC_SCOLOR        PIC X(10).

```

```

68      02 FILLER          PIC X.
69      02 IC_STANKA       PIC 9(10).
70      02 FILLER          PIC X.
71      02 IC_SSURYO       PIC 9(10).
72      *
73      01 TENPO.          ←3.
74          02 RT_DBKEY.
75          03 TENPO_CD     PIC X.
76          02 TENPO_NAME    PIC X(30).
77      *
78      01 ZAIKO.          ←4.
79          02 CH_TENPO_CD   PIC X.
80          02 CH_DBKEY      PIC S9(8) COMP.
81          02 SCODE         PIC X(4).
82          02 SNAME         PIC X(30).
83          02 SCOLOR        PIC X(10).
84          02 TANKA         PIC S9(8) COMP.
85          02 ZSURYO        PIC S9(8) COMP.
86      *
87      01 MIDASHI.
88          02 FILLER        PIC X(80) VALUE
89          '*** ZAIKO ICHIRAN ***'.
90      *
91      01 O_TENPO.
92          02 FILLER        PIC X(15) VALUE ' TENPO CODE : '.
93          02 O_TENPO_CD     PIC X.
94          02 FILLER        PIC X(10) VALUE ', NAME : "'.
95          02 O_TENPO_NAME    PIC X(30).
96          02 FILLER        PIC X      VALUE "'.
97      *
98      01 O_ZAIKO.
99          02 FILLER        PIC X(15) VALUE ' ZAIKO CODE : '.
100         02 O_ZAIKO_CODE  PIC X(4).
101         02 FILLER        PIC X(10) VALUE ', NAME : "'.
102         02 O_ZAIKO_NAME   PIC X(30).
103         02 FILLER        PIC X(11) VALUE '", COLOR : '.
104         02 O_ZAIKO_COLOR  PIC X(10).
105         02 FILLER        PIC X(10) VALUE ', TANKA : '.
106         02 O_ZAIKO_TANKA  PIC ZZZZZZZ9.
107         02 FILLER        PIC X(11) VALUE ', SUURYO : '.
108         02 O_ZAIKO_SSURYO PIC ZZZZZZZ9.
109      *
110      01 ERR_MSG.
111          02 FILLER        PIC X(22) VALUE '*** ERROR SQLCODE : '.
112          02 EM_SQLCODE     PIC -ZZZZZZZZ9.
113          02 FILLER        PIC X(15) VALUE ', ERROR DML : "'.
114          02 EM_ERRDML      PIC X(30).
115          02 FILLER        PIC X(7) VALUE '" ***'.
116      *
117      PROCEDURE DIVISION.
118      MAIN SECTION.
119      M-01.
120          MOVE 0 TO RETURN-CODE.
121          OPEN OUTPUT O-FILE.
122          OPEN INPUT I-FILE.
123      M-02.
124          CALL 'UAPSQL01' USING REQLSQL_CNCT.      ←5.
125          IF RETURN-CODE NOT = 0

```

```

126      THEN
127          MOVE '1' TO ERR-FLG
128      ELSE
129          PERFORM DB-KOUSHIN
130      END-IF.
131
M-03.
132      EVALUATE ERR-FLG
133          WHEN '0'
134              PERFORM FUTSUU
135          WHEN '1'
136              GO TO M-EXIT
137          WHEN '2'
138              PERFORM IJYOU           ←6.
139      END-EVALUATE.
140
M-04.
141      CALL  'UAPSQL01' USING REQSQL_DISC.    ←7.
142
M-EXIT.
143      CLOSE O-FILE.
144      CLOSE I-FILE.
145      GOBACK.
146
*  

147      DB-KOUSHIN SECTION.
148
D-01.
149      PERFORM UNTIL ( EOF = '1' OR ERR-FLG NOT = '0' )
150          READ I-FILE
151              AT END MOVE '1' TO EOF
152          END-READ
153          IF EOF = '0'
154          THEN
155              MOVE INREC TO INREC_W
156              IF IW_KUBUN = 'R'
157              THEN
158                  EVALUATE IR_KSN-KUBUN
159                      WHEN 'S'
160                          PERFORM TENPO-TSUIKA
161                      WHEN 'E'
162                          PERFORM TENPO-SAKUJO
163                  END-EVALUATE
164              ELSE
165                  MOVE IC_TENPOID TO TENPO_CD
166                  MOVE 'D-01:FIND TENPO' TO EM_ERRDML
167                  EXEC DML
168                      FIND FIRST "TENPO" WHERE ( "DBKEY" = :RT_DBKEY )    |8.
169                  END-DML
170                  IF SQLCODE = 0           ←9.
171                      EVALUATE IC_KSN-KUBUN
172                          WHEN 'M'
173                              PERFORM ZAIKO-KOUSHIN
174                          WHEN 'S'
175                              PERFORM ZAIKO-TSUIKA
176                          WHEN 'E'
177                              PERFORM ZAIKO-SAKUJO
178                      END-EVALUATE
179                  ELSE
180                      MOVE '2' TO ERR-FLG
181                  END-IF
182              END-IF
183          END-IF

```

```

184      END-PERFORM.
185  D-02.
186      IF ERR-FLG = '0'
187      THEN
188          CALL 'UAPSQL01' USING REQLSQL_COMT      ←10.
189          IF RETURN-CODE NOT = 0
190          THEN
191              MOVE '3' TO ERR-FLG
192          ELSE
193              CONTINUE
194          END-IF
195      ELSE
196          CONTINUE
197      END-IF.
198  D-EXIT.
199      EXIT.
200  *
201  ZAIKO-KOUSHIN SECTION.
202  M-01.
203      PERFORM ICHIDUKE.                      ←11.
204      IF ERR-FLG = '0'
205      THEN
206          IF IC_IO-KUBUN = '1'
207          THEN
208              COMPUTE ZSURYO = ZSURYO + IC_SSURYO  ←12.
209          ELSE
210              COMPUTE ZSURYO = ZSURYO - IC_SSURYO  ←12.
211          END-IF
212          MOVE 'M-01:MODIFY ZAIKO' TO EM_ERRDML
213          EXEC DML
214              MODIFY ZAIKO FROM :ZAIKO          ←13.
215          END-DML
216          IF SQLCODE NOT = 0                ←9.
217          THEN
218              MOVE '2' TO ERR-FLG
219          ELSE
220              CONTINUE
221          END-IF
222      END-IF.
223  M-EXIT.
224      EXIT.
225  *
226  ZAIKO-TSUIKA SECTION.
227  S-01.
228      MOVE IC_TENPOID TO TENPO_CD.          ←14.
229      MOVE 0          TO CH_DBKEY.          ←15.
230      MOVE IC_SCODE  TO SCODE.             -
231      MOVE IC_SNAME  TO SNAME.             |
232      MOVE IC_SCOLOR  TO SCOLOR.          14.
233      MOVE IC_STANKA  TO TANKA.            |
234      MOVE IC_SSURYO  TO ZSURYO.          -
235  S-02.
236      MOVE 'S-02:STORE ZAIKO' TO EM_ERRDML.
237      EXEC DML
238          STORE ZAIKO FROM :ZAIKO          ←16.
239      END-DML.
240      IF SQLCODE NOT = 0                ←9.
241      THEN

```

```

242      MOVE '2' TO ERR-FLG
243      ELSE
244          CONTINUE
245      END-IF.
246      S-EXIT.
247          EXIT.
248      *
249      ZAIKO-SAKUJO SECTION.
250      E-01.
251          PERFORM ICHIDUKE.          ←17.
252          IF ERR-FLG = '0'
253          THEN
254              MOVE 'E-01:ERASE ZAIKO' TO EM_ERRDML
255              EXEC DML
256                  ERASE ZAIKO ALL          ←18.
257              END-DML
258              IF SQLCODE NOT = 0          ←9.
259              THEN
260                  MOVE '2' TO ERR-FLG
261              ELSE
262                  CONTINUE
263                  END-IF
264              END-IF.
265          E-EXIT.
266          EXIT.
267      *
268      TENPO-TSUIKA SECTION.
269      A-01.
270          MOVE IR_TENPO_CD TO TENPO_CD.          ←19.
271          MOVE IR_TENPO_NAME TO TENPO_NAME.          ←14.
272          MOVE 'A-01:STORE TENPO' TO EM_ERRDML
273          EXEC DML
274              STORE TENPO FROM :TENPO          ←20.
275          END-DML.
276          IF SQLCODE NOT = 0          ←9.
277          THEN
278              MOVE '2' TO ERR-FLG
279          ELSE
280              CONTINUE
281              END-IF.
282          A-EXIT.
283          EXIT.
284      *
285      TENPO-SAKUJO SECTION.
286      K-01.
287          MOVE IR_TENPO_CD TO TENPO_CD.          ←21.
288          MOVE 'K-01:FIND TENPO' TO EM_ERRDML
289          EXEC DML
290              FIND FOR UPDATE FIRST TENPO          ←22.
291                  WHERE ( "DBKEY" = :RT_DBKEY )          ←22.
292          END-DML.
293          IF SQLCODE = 0          ←9.
294          THEN
295              MOVE 'K-01:ERASE TENPO' TO EM_ERRDML
296              EXEC DML
297                  ERASE TENPO ALL          ←23.
298              END-DML
299          ELSE

```

```

300      CONTINUE
301      END-IF.
302      IF SQLCODE NOT = 0                                ←9.
303      THEN
304          MOVE '2' TO ERR-FLG
305      ELSE
306          CONTINUE
307      END-IF.
308      K-EXIT.
309      EXIT.
310 *
311      ICHIDUKE SECTION.
312      I-01.
313          MOVE 'I-01:FETCH ZAIKO 01' TO EM_ERRDML.
314          EXEC DML
315              FETCH FOR UPDATE FIRST ZAIKO             ←24.
316                  INTO :ZAIKO WITHIN TENPO_ZAIKO
317              END-DML.
318          IF SQLCODE NOT = 0                          ←9.
319          THEN
320              MOVE '2' TO ERR-FLG
321          ELSE
322              CONTINUE
323          END-IF.
324          PERFORM UNTIL ( IC_SCODE = SCODE OR ERR-FLG NOT = '0' )
325              MOVE 'I-01:FETCH ZAIKO 02' TO EM_ERRDML
326              EXEC DML
327                  FETCH FOR UPDATE NEXT ZAIKO           ←25.
328                      INTO :ZAIKO WITHIN TENPO_ZAIKO
329                  END-DML
330          IF SQLCODE NOT = 0                          ←9.
331          THEN
332              MOVE '2' TO ERR-FLG
333          ELSE
334              CONTINUE
335          END-IF
336          END-PERFORM.
337      I-EXIT.
338      EXIT.
339 *
340      FUTSUU SECTION.
341      F-01.
342          WRITE OUTREC FROM MIDASHI.
343          PERFORM UNTIL ( TENPO_END = '1' OR ERR-FLG NOT = '0' )
344              MOVE 'F-01:FETCH TENPO' TO EM_ERRDML
345              IF TENPO_NOT_1ST = '0'
346              THEN
347                  EXEC DML
348                      FETCH FIRST TENPO INTO :TENPO          ←26.
349                  END-DML
350              ELSE
351                  EXEC DML
352                      FETCH NEXT TENPO INTO :TENPO         ←27.
353                  END-DML
354              END-IF
355              IF SQLCODE = 0                          ←9.
356              THEN
357                  MOVE TENPO_CD    TO 0_TENPO_CD

```

```

358      MOVE TENPO_NAME TO O_TENPO_NAME
359      WRITE OUTREC FROM O_TENPO
360      PERFORM UNTIL ( ZAIKO_END = '1' OR ERR-FLG NOT = '0' )
361          MOVE 'F-01:FETCH ZAIKO' TO EM_ERRDML
362          EXEC DML
363              FETCH NEXT ZAIKO INTO :ZAIKO WITHIN TENPO_ZAIKO      ←28.
364          END-DML
365          IF SQLCODE = 0                                ←9.
366          THEN
367              MOVE SCODE  TO O_ZAIKO_CODE
368              MOVE SNAME  TO O_ZAIKO_NAME
369              MOVE SCOLOR  TO O_ZAIKO_COLOR
370              MOVE TANKA  TO O_ZAIKO_TANKA
371              MOVE ZSURYO TO O_ZAIKO_SUURYO
372              WRITE OUTREC FROM O_ZAIKO
373          ELSE
374              IF SQLCODE = 100
375              THEN
376                  MOVE '1' TO ZAIKO_END
377              ELSE
378                  MOVE '2' TO ERR-FLG
379                  END-IF
380                  END-IF
381          END-PERFORM
382          MOVE '1' TO TENPO_NOT_1ST
383          MOVE '0' TO ZAIKO_END
384          ELSE
385              IF SQLCODE = 100
386              THEN
387                  MOVE '1' TO TENPO_END
388              ELSE
389                  MOVE '2' TO ERR-FLG
390                  END-IF
391                  END-IF
392          END-PERFORM.
393 F-02.
394     IF ERR-FLG NOT = '0'
395     THEN
396         CALL 'UAPSQL01' USING REQLSQL_ROLB      ←32.
397     ELSE
398         CONTINUE
399     END-IF.
400 F-EXIT.
401     EXIT.
402 *
403 IJYOU SECTION.
404 J-01.
405     MOVE SQLCODE TO EM_SQLCODE.          ←33.
406     WRITE OUTREC FROM ERR_MSG.          ←34.
407     WRITE OUTREC FROM SQLERRMC.         ←35.
408     WRITE OUTREC FROM INREC.            ←36.
409     CALL 'UAPSQL01' USING REQLSQL_ROLB. ←37.
410 J-EXIT.
411     EXIT.

```

[説明]

1. SDB データベース節を指定します。

- UAP 内の DML でアクセスする SDB データベースを指定します。
- DML の実行後にレコード名を受け取る埋込み変数の名前を指定します。
- DML の実行後にレコード長を受け取る埋込み変数の名前を指定します。

SDB データベース節については、「[2.3 SDB データベース節の記述](#)」を参照してください。

2. 次の埋込み変数を宣言します。

- SDB データベース節の RECORD NAME で指定した埋込み変数
- SDB データベース節の RECORD LENGTH で指定した埋込み変数

SDB データベース節で指定する埋込み変数の宣言については、「[2.3.3 SDB データベース節で指定する埋込み変数の宣言](#)」を参照してください。

3. レコード型 TENPO とデータの受け渡しを行う埋込み変数を宣言します。

埋込み変数の宣言については、「[2.4 埋込み変数の宣言](#)」を参照してください。

4. レコード型 ZAIKO とデータの受け渡しを行う埋込み変数を宣言します。

5. HiRDB サーバに接続します。

SQL の CONNECT 文を実行して HiRDB サーバに接続します。SQL の CONNECT 文は、DML を記述した UAP ソースプログラムとは別の UAP ソースプログラム (UAPSQL01) に記述します。

6. エラーが発生した場合、トランザクションの取り消し処理を行います。

7. HiRDB サーバから切り離します。

SQL の DISCONNECT 文を実行して HiRDB サーバから切り離します。SQL の DISCONNECT 文は、DML を記述した UAP ソースプログラムとは別の UAP ソースプログラム (UAPSQL01) に記述します。

8. ルートレコードのデータベースキーの一致する TENPO レコードに位置指示子を位置づけます。

DML 先頭子 (EXEC DML) に続けて DML を記述します。DML の直後に DML 終了子 (END-DML) を記述します。

9. SQLCODE を参照して DML の実行結果を判定します。

DML の実行結果の判定については、「[2.7 DML の実行結果の判定処理](#)」を参照してください。

10. データベースの更新がすべて正常に終了した場合は、SQL の COMMIT 文でトランザクションをコミットします。DML を記述した UAP ソースプログラムとは別の UAP ソースプログラム (UAPSQL01) に COMMIT 文を記述します。

11. レコード実現値の変更前に、変更するレコードに対して位置指示子を位置づけます。

12. 変更する構成要素に対応する埋込み変数に更新値を設定します。

13. 位置づけした ZAIKO レコードのユーザデータを埋込み変数の値に変更します。

14. 構成要素に対応する埋込み変数に格納するデータを設定します。

15. 一連番号は HiRDB/SD が割り当てます。

16. 親レコード TENPO への位置づけ後に、子レコード ZAIKO にレコード実現値を格納します。
17. レコード実現値の削除前に、削除するレコードに対して位置指示子を位置づけます。
18. 位置づけした ZAIKO レコードのレコード実現値を削除します。
19. TENPO レコードはルートレコードのため、格納前にルートレコードのデータベースキーの値を埋込み変数に設定します。
20. ルートレコードの TENPO レコードを格納します。
21. キーの検索条件に指定する埋込み変数に、削除対象のルートレコードのデータベースキーの値を設定します。
22. レコード実現値の削除前に、削除するレコードに対して位置指示子を更新指定で位置づけます。削除するレコードはキーの検索条件で指定します。
23. 位置指示子が位置づけられているレコード実現値を削除します。下位レコードがある場合は、下位レコードも同時に削除されます。
24. 子レコード ZAIKO の検索は、親レコード TENPO の位置づけ後に行います。更新指定で先頭の ZAIKO レコードへの位置づけを行い、レコード実現値を埋込み変数 ZAIKO に取得します。
25. 現在位置づけられている ZAIKO レコードの次のレコードに更新指定で位置づけを行い、レコード実現値を埋込み変数 ZAIKO に取得します。
26. 先頭の TENPO レコードへの位置づけを行い、レコード実現値を埋込み変数 TENPO に取得します。
27. 現在位置づけられている TENPO レコードの次のレコードに位置づけを行い、レコード実現値を埋込み変数 TENPO に取得します。
28. ZAIKO レコードに位置づけがない場合は、先頭の ZAIKO レコードに位置づけを行い、レコード実現値を埋込み変数 ZAIKO に取得します。
位置づけられている場合は、位置づけられている ZAIKO レコードの次のレコードに位置づけを行い、レコード実現値を埋込み変数 ZAIKO に取得します。
29. 埋込み変数からレコード実現値を取り出します。
30. SQLCODE が 100 かどうかを判定します。100 の場合は、TENPO レコード下のすべての ZAIKO レコードの検索が完了しています。
31. SQLCODE が 100 かどうかを判定します。100 の場合は、すべての TENPO レコードの検索が完了しています。
32. SQL の ROLLBACK 文でトランザクションを取り消します。DML を記述した UAP ソースプログラムとは別の UAP ソースプログラム (UAPSQL01) に ROLLBACK 文を記述します。
33. エラー要因を取得するため、SQLCODE をエラーメッセージに含めます。
34. エラーメッセージを出力します。
35. HiRDB のエラーメッセージを出力します。
36. エラーが発生した入力データを出力します。

37. トランザクションを取り消します。SQL の ROLLBACK 文は、DML を記述した UAP ソースプログラムとは別の UAP ソースプログラム (UAPSQL01) に記述します。

■SQL を記述した COBOL ソースプログラム (UAPSQL01) のコーディング例

```
1      IDENTIFICATION DIVISION.
2      PROGRAM-ID. UAPSQL01.
3      *
4      ENVIRONMENT DIVISION.
5      *
6      INPUT-OUTPUT SECTION.
7      FILE-CONTROL.
8      SELECT 0-FILE
9          ASSIGN TO './CT0.log'
10         LINE SEQUENTIAL.
11      DATA DIVISION.
12      FILE SECTION.
13      FD 0-FILE          DATA RECORD OUTREC.
14      01 OUTREC          PIC X(132).
15      *
16      WORKING-STORAGE SECTION.
17      *
18      ****
19      ***  REQCODE FOR UAPSQL01
20      ****
21      77 REQSQL_CNCT      PIC X(4) VALUE 'CNCT'.
22      77 REQSQL_DISC      PIC X(4) VALUE 'DISC'.
23      77 REQSQL_COMT      PIC X(4) VALUE 'COMT'.
24      77 REQSQL_ROLB      PIC X(4) VALUE 'ROLB'.
25      *
26      *
27      ****
28      ***  MESSAGE
29      ****
30      01 MSG-ERRREQ.
31          02 FILLER          PIC X(80) VALUE
32                      '">>>> INVALID REQSQL "'.
33          02 ERRREQ          PIC X(4).
34          02 FILLER          PIC X(80) VALUE ''' SPECIFIED'.
35      *
36      01 ERRSQL.
37          02 FILLER          PIC X(28) VALUE
38                      '">>>> SQL ERROR, SQLCODE = "'.
39          02 ERRCODE          PIC -ZZZZZZZZ9.
40          02 FILLER          PIC X(12) VALUE '''', SQLSTMT "'.
41          02 ERRSTMT          PIC X(4).
42          02 FILLER          PIC X(1) VALUE ''''.
43      *
44      LINKAGE SECTION.
45      77 REQSQL          PIC X(4).
46      *
47      PROCEDURE DIVISION    USING REQSQL. ← 実行するSQLのリクエストを引数で受け取ります
48      *
49          OPEN OUTPUT 0-FILE.
50      *
51          MOVE REQSQL TO ERRSTMT.
52          MOVE 0 TO RETURN-CODE.
```

```

53  *
54      EVALUATE REQSQL           ←リクエストで実行するSQLの振り分け
55      WHEN REQSQL_CNST
56          PERFORM PROC-CNST      ←CONNECT処理の実行
57      WHEN REQSQL_DISC
58          PERFORM PROC-DISC      ←DISCONNECT処理の実行
59      WHEN REQSQL_COMT
60          PERFORM PROC-COMT      ←COMMIT処理の実行
61      WHEN REQSQL_ROLB
62          PERFORM PROC-ROLB      ←ROLLBACK処理の実行
63      WHEN OTHER
64          MOVE REQSQL TO ERRREQ
65          DISPLAY MSG-ERRREQ UPON SYSOUT
66          MOVE 99 TO RETURN-CODE
67          GO TO OWARI
68      END-EVALUATE.
69  *
70      OWARI.
71          CLOSE O-FILE.
72          GOBACK.
73  *
74  ****
75      *** CONNECT           ***
76  ****
77      PROC-CNST SECTION.
78          EXEC SQL
79              CONNECT
80          END-EXEC.
81          IF SQLCODE < 0           ←SQLの実行結果の判定
82          THEN
83              MOVE SQLCODE TO ERRCODE
84              DISPLAY ERRSQL UPON SYSOUT
85              MOVE SQLCODE TO RETURN-CODE
86          ELSE
87              CONTINUE
88          END-IF.
89  ****
90      *** DISCONNECT           ***
91  ****
92      PROC-DISC SECTION.
93          EXEC SQL
94              DISCONNECT
95          END-EXEC.
96          IF SQLCODE < 0           ←SQLの実行結果の判定
97          THEN
98              MOVE SQLCODE TO ERRCODE
99              DISPLAY ERRSQL UPON SYSOUT
100             MOVE SQLCODE TO RETURN-CODE
101          ELSE
102              CONTINUE
103          END-IF.
104  ****
105      *** COMMIT           ***
106  ****
107      PROC-COMT SECTION.
108          EXEC SQL
109              COMMIT
110          END-EXEC.

```

```
111      IF SQLCODE < 0           ←SQLの実行結果の判定
112      THEN
113          MOVE SQLCODE TO ERRCODE
114          DISPLAY ERRSQL UPON SYSOUT
115          MOVE SQLCODE TO RETURN-CODE
116      ELSE
117          CONTINUE
118      END-IF.
119      ****
120      *** ROLLBACK           ***
121      ****
122      PROC-ROLB SECTION.
123          EXEC SQL
124              ROLLBACK
125          END-EXEC.
126          IF SQLCODE < 0           ←SQLの実行結果の判定
127          THEN
128              MOVE SQLCODE TO ERRCODE
129              DISPLAY ERRSQL UPON SYSOUT
130              MOVE SQLCODE TO RETURN-CODE
131          ELSE
132              CONTINUE
133          END-IF.
```

2.13 2進項目をビッグエンディアン形式にする UAP を作成する場合の注意事項

COBOL2002 のコンパイラオプションに -BigEndian,Bin オプション (COBOL85 のコンパイラオプションの場合は -Bb オプション) を指定して、2進項目をビッグエンディアン形式にする場合、2進項目の埋込み変数をリトルエンディアン形式にする必要があります。そのため、埋込み変数の宣言時、UAP のプリプロセスおよびコンパイル時に次のことをしてください。

- 埋込み変数の宣言時のこと

データ型が INTEGER または SMALLINT の埋込み変数を宣言する場合、データ記述項の USAGE 句に COMPUTATIONAL-5 または COMP-5 を指定してください。詳細については、「[2.4.2\(2\) DML のデータ型と COBOL 言語のデータ記述項の対応](#)」を参照してください。

- UAP のプリプロセス時のこと

UAP のプリプロセスを実行する際、-Xb オプションを指定して pdsdbcbl コマンドを実行してください。-Xb オプションについては、「[6.2 コマンドの形式](#)」を参照してください。

- UAP のコンパイル時のこと

UAP のコンパイルを実行する際、COBOL2002 のコンパイラオプションに -Comp5 オプション (COBOL85 のコンパイラオプションの場合は -X5 オプション) を指定してください。詳細については、「[3.4.1\(2\) コンパイラオプションの指定を確認する](#)」を参照してください。

2.14 性能向上, 操作性向上に関する機能

性能向上, 操作性向上に関する次の機能を使用できます。

- **自動再接続機能**

自動再接続機能とは, サーバプロセスダウン, 系切り替え, ネットワーク障害などの要因による HiRDB サーバとの接続障害を検知した場合に, 自動的に UAP の再接続を行う機能です。自動再接続機能を使用すると, HiRDB サーバとの接続の切断を意識しないで, UAP の実行を継続できます。自動再接続機能については, マニュアル「HiRDB UAP 開発ガイド」の「自動再接続機能」を参照してください。

なお, 性能向上, 操作性向上に関する次の機能は使用できません。

- ブロック転送機能
- 複数接続機能
- マルチスレッド対応

上記の機能については, マニュアル「HiRDB UAP 開発ガイド」を参照してください。

3

UAP の実行前準備 (UAP のプリプロセス, コンパイル, リンケージ)

この章では、UAP のプリプロセス、コンパイル、およびリンクエージの方法について説明します。

3.1 プリプロセス, コンパイル, およびリンクエージの実行環境の構築

DML を記述した UAP をプリプロセス, コンパイル, およびリンクエージする実行環境の構築について説明します。

プリプロセスを実行するマシンには, 次の製品をインストールして環境設定をする必要があります。

- HiRDB Structured Data Access Facility/Developer's Kit

上記製品のインストールおよび環境設定方法は, HiRDB クライアントのインストールおよび環境設定方法と同じです。HiRDB クライアントのインストールおよび環境設定方法については, マニュアル「HiRDB UAP 開発ガイド」の「クライアントの環境設定」を参照してください。

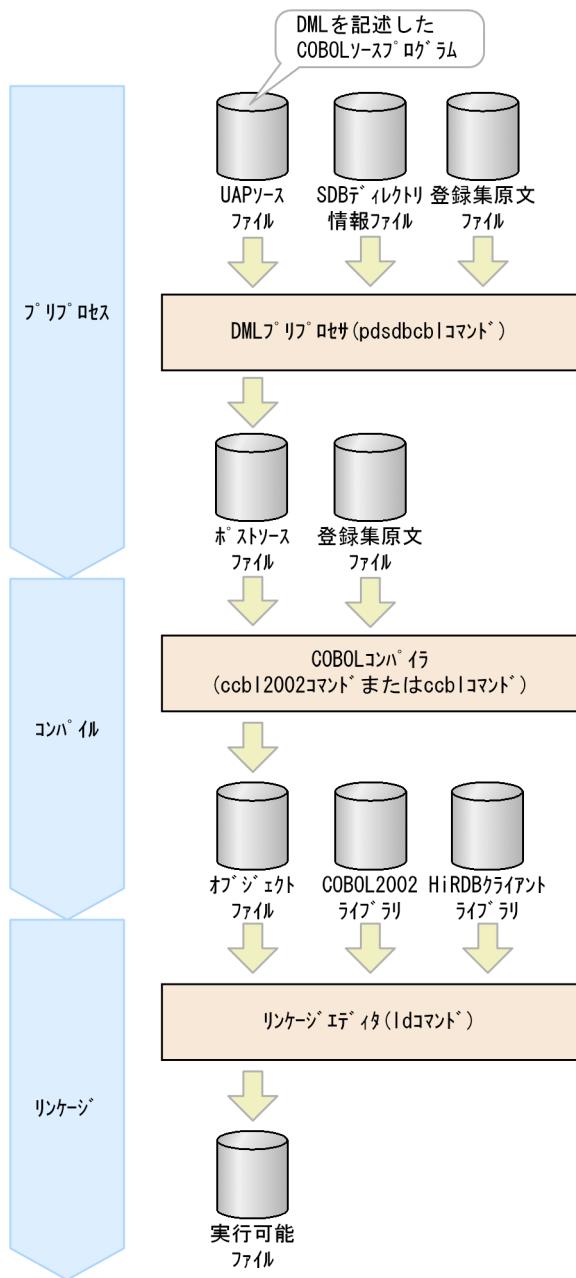
また, コンパイルおよびリンクエージを実行するマシンには, 次の製品をインストールして環境設定をする必要があります。

- HiRDB Structured Data Access Facility/Developer's Kit
- COBOL2002 Net Server Suite(64)

3.2 UAP のプリプロセス, コンパイル, リンケージの流れ

DML を記述した COBOL ソースプログラムを, DML プリプロセサ (pdsdbcbl コマンド) でポストソースに変換します。そのポストソースを COBOL コンパイラでコンパイルおよびリンクエージすると, UAP の実行可能ファイルが作成されます。UAP の実行可能ファイル作成までの流れを次の図に示します。

図 3-1 UAP の実行可能ファイル作成までの流れ



各工程の説明を次に示します。

プリプロセス

COBOL ソースプログラム中に記述されている DML を, COBOL コンパイラでコンパイルできる COBOL 言語の命令に置換し, その実行結果をポストソースとして出力します。

DML が記述された COBOL ソースプログラムは、そのままの状態では COBOL コンパイラでコンパイルできません。DML プリプロセサによるプリプロセスを実行してポストソースを出力し、そのポストソースを COBOL コンパイラでコンパイルします。

プリプロセスを実行する際には、次のファイルを DML プリプロセサの入力情報にします。

- COBOL ソースプログラムを格納した UAP ソースファイル
- SDB ディレクトリ情報ファイル
- 登録集原文ファイル

プリプロセスを実行するコマンドは、pdsdbcbl コマンドです。

コンパイル、リンクエージ

プリプロセスの結果、出力されたポストソースファイル、および登録集原文ファイルを入力情報にして、COBOL コンパイラでコンパイルおよびリンクエージを実行します。

ポストソースには登録集原文を取り込む COPY 文が展開されます。そのため、ポストソースをコンパイルする際には、これらの登録集原文を格納したディレクトリの絶対パスを環境変数 CBLLIB に追加してください。

コンパイルを実行すると、UAP のオブジェクトがオブジェクトファイルに出力されます。UAP のオブジェクトファイル、COBOL2002 ライブラリ、および HiRDB クライアントライブラリを入力情報にしてリンクエージを実行し、UAP の実行可能ファイルを作成します。

コンパイルを実行するコマンドは、COBOL の ccbl2002 コマンドまたは ccbl コマンドです。リンクエージを実行するコマンドは、ld コマンドです。

参考

COBOL2002 のコンパイラオプションで、リンクエージまで行うかを指定できます。リンクエージを行う場合は COBOL2002 がリンク (ld コマンド) を呼び出します。

3.3 プリプロセスの実行

DML を記述した UAP のプリプロセスの実行方法について説明します。

3.3.1 プリプロセスを実行するための準備作業

プリプロセスを実行する前に、ここで説明する準備作業を実施してください。

(1) 環境変数を設定する

HiRDB クライアントで次の環境変数を設定してください。

- **PDCLTLANG**

PDCLTLANG に SJIS (シフト JIS 漢字コード) を指定してください。

PDCLTLANG については、マニュアル「HiRDB UAP 開発ガイド」の「クライアント環境定義の設定内容」を参照してください。

- **PATH**

環境変数 PATH に次のディレクトリを追加してください。

- HiRDB クライアントのサーバマシンでプリプロセスを実行する場合

`/opt/HiRDB/client/utl/`

- HiRDB サーバのサーバマシンでプリプロセスを実行する場合

`$PDDIR/client/utl/`

(2) SDB ディレクトリ情報ファイルを準備する

プリプロセスを実行する際、COBOL ソースプログラム中の SDB データベース節に記述した SDB データベースの SDB ディレクトリ情報が必要になります。その SDB データベースの定義が格納されている SDB ディレクトリ情報ファイルを準備してください。

■ 注意事項

HiRDB Structured Data Access Facility のバージョン 09-66 以降で出力した SDB ディレクトリ情報ファイルを準備してください。

3.3.2 プリプロセスの実行例

pdsdbcbl コマンドでプリプロセスを実行します。プリプロセスの実行例を次に示します。

例 1

DML を記述した UAP (UAP ソースファイル名 : uap01.ecb) をプリプロセスして、ポストソースを作成します。

コマンドの実行例

```
pdldbcl /UAPsrc/DMLsrc/uap01.ecb -d /dirinf/pdldbdir
```

[説明]

/UAPsrc/DMLsrc/uap01.ecb :

プリプロセス対象の UAP ソースファイル名を指定します。

-d /dirinf/pdldbdir :

「3.3.1(2) SDB ディレクトリ情報ファイルを準備する」で準備した、SDB ディレクトリ情報ファイル名を指定します。

例 2

DML を記述した UAP (UAP ソースファイル名 : uap02.ecb) をプリプロセスして、ポストソースを作成します。

なお、uap02.ecb は、COBOL2002 のコンパイラオプションに-BigEndian,Bin オプション (COBOL85 のコンパイラオプションの場合は-Bb オプション) を指定して、2 進項目をビッグエンディアン形式にする UAP です。

コマンドの実行例

```
pdldbcl /UAPsrc/DMLsrc/uap02.ecb -d /dirinf/pdldbdir -Xb
```

[説明]

/UAPsrc/DMLsrc/uap02.ecb :

プリプロセス対象の UAP ソースファイル名を指定します。

-d /dirinf/pdldbdir :

「3.3.1(2) SDB ディレクトリ情報ファイルを準備する」で準備した、SDB ディレクトリ情報ファイル名を指定します。

-Xb :

COBOL2002 のコンパイラオプションに-BigEndian,Bin オプション (COBOL85 のコンパイラオプションの場合は-Bb オプション) を指定して、2 進項目をビッグエンディアン形式にする UAP を作成する場合に指定するオプションです。

2 進項目をビッグエンディアン形式にする UAP を作成する場合の注意事項については、「2.13 2 進項目をビッグエンディアン形式にする UAP を作成する場合の注意事項」を参照してください。

参考

- pdldbcl コマンドの機能詳細、各オプションについては、「6. DML プリプロセサ (pdldbcl コマンド)」を参照してください。

- SQL を記述した UAP のプリプロセス方法については、マニュアル「HiRDB UAP 開発ガイド」の「プリプロセス」を参照してください。

3.3.3 プリプロセスエラーが発生した場合の対処

プリプロセスエラーが発生した場合、標準エラー出力にエラーメッセージが出力されます。そのエラーメッセージの対処に従って、COBOL ソースプログラムのデバッグを実施してください。

プリプロセスエラーが発生した場合のエラーメッセージの出力例を次に示します。

エラーメッセージの出力例

```
/UAPsrc/DMLsrc/uap01.ecb:  
pdsdbcbl: /UAPsrc/DMLsrc/uap01.ecb, 23: KFPB65415-E: The specified SDB database name "DAT  
ABASE01" was not found in SDB directory information  
pdsdbcbl: /UAPsrc/DMLsrc/uap01.ecb, *: KFPB65000-I: DML preprocessing was ended, return c  
ode = 8
```

[説明]

- 下線部には、プリプロセスを実行した UAP ソースファイル名が出力されます。pdsdbcbl コマンドに指定したパスの形式で出力されます。
- 色が付いている部分（例中の 23）には、UAP ソースファイル内のエラーの原因となった行の行番号が出力されます。エラーの原因が特定の行に起因しない場合は、行番号にアスタリスク (*) が出力されます。
- KFPB65000-I メッセージには、pdsdbcbl コマンドのリターンコードが出力されます。
- pdsdbcbl コマンドのオプション指定誤りの場合や、コマンドの実行環境によるエラーの場合などは、メッセージ中に UAP ソースファイル名が出力されません。

3.4 コンパイルおよびリンクエージの実行

コンパイルおよびリンクエージの実行方法について説明します。

3.4.1 コンパイルおよびリンクエージを実行するための準備作業

コンパイルおよびリンクエージを実行する前に、ここで説明する準備作業を実施してください。

(1) 環境変数を設定する

次の環境変数を設定してください。

- CBLLIB

ポストソースには、HiRDB/SD が提供する登録集原文を取り込む COPY 命令が展開されます。そのため、登録集原文を格納したディレクトリの絶対パスを、環境変数 CBLLIB に追加してください。

(2) コンパイラオプションの指定を確認する

次のことを確認してください。

- UAP のプリプロセスの実行時、-Xb オプションを指定して pdsdbcbl コマンドを実行した場合、COBOL コンパイラのコンパイラオプションに-BigEndian,Bin オプションと-Comp5 オプション※を指定してください。

注※

COBOL85 のコンパイラオプションの場合は、-Bb オプションと-X5 オプションを指定してください。

オプションを指定する理由については、「[2.13 2 進項目をビッグエンディアン形式にする UAP を作成する場合の注意事項](#)」を参照してください。

- DML を記述した UAP をコンパイルする際、COBOL2002 のコンパイラオプションの-DynamicLink オプションの指定は次のどちらかにしてください。
 - -DynamicLink オプションを指定しない
 - -DynamicLink,IdentCall オプションを指定する（一意名指定の CALL 文だけを動的なリンクとする）

(3) コンパイル時に指定する HiRDB が提供するライブラリを確認する

コンパイルおよびリンクエージをする際、次に示す HiRDB が提供するライブラリを指定する必要があります。

- XA インタフェースを使用する場合
libzcltys64.so (シングルスレッド対応)
- XA インタフェースを使用しない場合

(4) トランザクションオブジェクトファイルを作成する

OpenTP1 環境下で実行する UAP の場合、トランザクションオブジェクトファイルを作成してください。UAP のコンパイルおよびリンクエージをする際に、作成したトランザクションオブジェクトファイルを指定します。

詳細については、マニュアル「HiRDB UAP 開発ガイド」の「X/Open に従った API (TX_関数) を使用した UAP の実行」の「OpenTP1 を使用する場合」の「COBOL 言語の場合」を参照してください。

3.4.2 ccbl2002 コマンドの指定形式

コンパイルおよびリンクエージをするには、ccbl2002 コマンドを実行します。ccbl2002 コマンドの指定形式を次に示します。

指定形式

```
ccbl2002 [オプション] ポストソースファイル名 ディレクトリ 提供ライブラリ
```

オプション：

ccbl2002 コマンドのオプションを指定します。

ccbl2002 コマンドのオプションについては、マニュアル「COBOL2002 使用の手引 手引編」の「ccbl2002 コマンド」を参照してください。

注意事項

UAP のプリプロセスの実行時、-Xb オプションを指定して pdsdbcbl コマンドを実行した場合、コンパイラオプションに-BigEndian,Bin オプションと-Comp5 オプション※を指定してください。

オプションを指定する理由については、「[2.13 2進項目をビッグエンディアン形式にする UAP を作成する場合の注意事項](#)」を参照してください。

注※

COBOL85 のコンパイラオプションの場合は、-Bb オプションと-X5 オプションを指定してください。

ポストソースファイル名：

コンパイルおよびリンクエージを実行するポストソースファイルの名称を指定します。

ディレクトリ：

インクルードディレクトリ (HiRDB が提供するライブラリのヘッダファイルがあるディレクトリ) を指定します。

提供ライブラリ：

HiRDB が提供するライブラリを指定します。「3.4.1(3) コンパイル時に指定する HiRDB が提供するライブラリを確認する」で確認したライブラリを指定します。

3.4.3 コンパイルおよびリンクエラーの実行例

例

ポストソースファイル (uap01.cbl) のコンパイルおよびリンクエラーを実行します。

```
CBLLIB=/HiRDB/include ...1  
export CBLLIB ...2  
ccbl2002 -Compati85,All uap01.cbl -L/HiRDB/client/lib -lzcltk64.so ...3
```

[説明]

1. 環境変数 CBLLIB に、登録集原文の検索先ディレクトリを指定します。下線部分は、HiRDB Structured Data Access Facility/Developer's Kit のインストールディレクトリです。
2. 環境変数 CBLLIB を設定します。
3. ccbl2002 コマンドを実行します。

3.4.4 コンパイルエラーまたはリンクエラーが発生した場合の対処

コンパイルエラーまたはリンクエラーが発生した場合、標準エラー出力にエラーメッセージが出力されます。出力されたメッセージに従って対処してください。

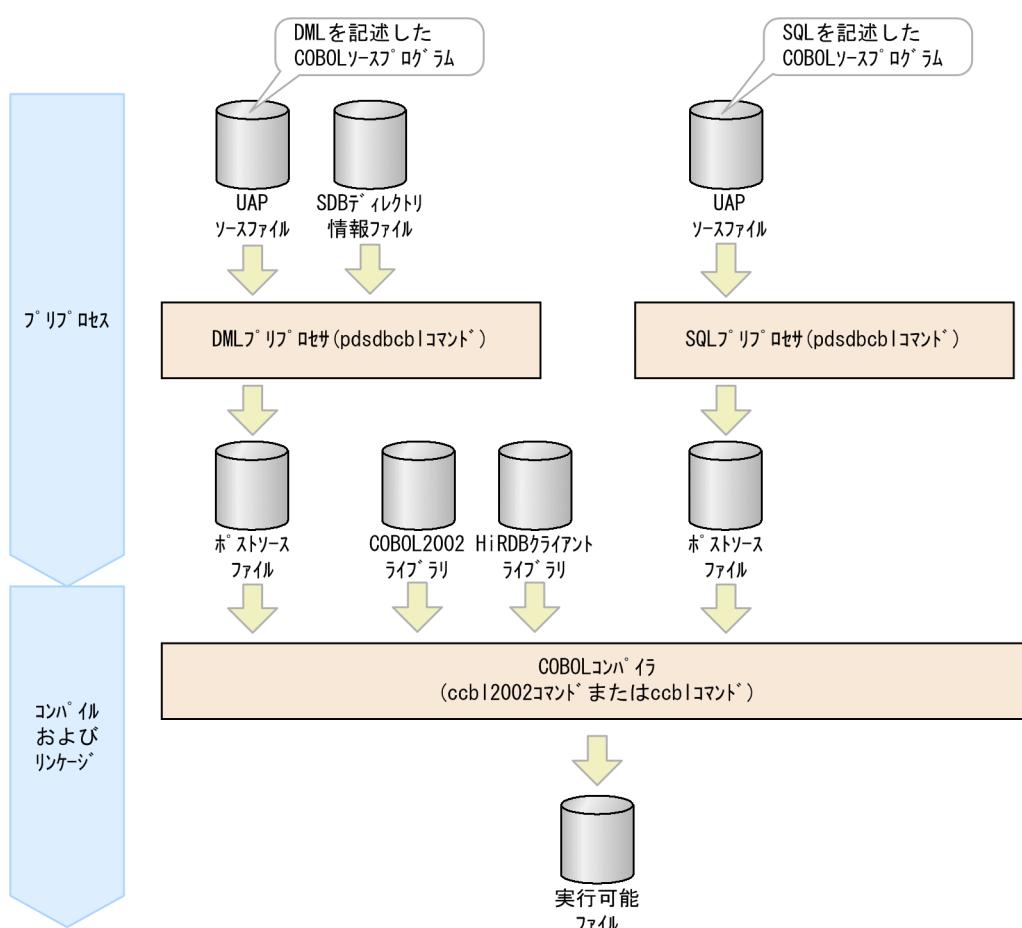
3.5 DML と SQL を実行する UAP をプリプロセス, コンパイル, およびリンクージする場合

DML と SQL の両方を実行する UAP をプリプロセス, コンパイル, およびリンクージする方法について説明します。

3.5.1 UAP のプリプロセス, コンパイル, リンケージの流れ

DML と SQL の両方を実行する UAP をプリプロセス, コンパイル, およびリンクージする際の流れを次の図に示します。

図 3-2 DML と SQL の両方を実行する UAP をプリプロセス, コンパイル, およびリンクージする際の流れ



3.5.2 プリプロセス, コンパイル, リンケージの実行例

例

DML と SQL の両方を実行する UAP の実行可能ファイルを作成します。

DML を記述した UAP の UAP ソースファイル名を uapdml01.ecb とします。SQL を記述した UAP の UAP ソースファイル名を uapsql01.ecb とします。

手順を次に示します。

(1) DML を記述した UAP のプリプロセスを実行する

pdsdbcbl コマンドで、DML を記述した UAP のプリプロセスを実行します。

コマンド実行例

```
pdsdbcbl /UAPsrc/DMLsrc/uapdml01.ecb -d /dirinf/pdsdbdir
```

[説明]

/UAPsrc/DMLsrc/uapdml01.ecb :

プリプロセス対象の UAP ソースファイル名を指定します。

-d /dirinf/pdsdbdir :

「3.3.1(2) SDB ディレクトリ情報ファイルを準備する」で準備した、SDB ディレクトリ情報ファイル名を指定します。

(2) SQL を記述した UAP のプリプロセスを実行する

pdcbl コマンドで、SQL を記述した UAP のプリプロセスを実行します。

コマンド実行例

```
pdcbl uapsql01.ecb -h64
```

[説明]

uapsql01.ecb :

プリプロセス対象の UAP ソースファイル名を指定します。

-h64 :

64 ビットモード用のポストソースを作成するために指定します。必ず指定する必要があります。

pdcbl コマンドの詳細については、マニュアル「HiRDB UAP 開発ガイド」の「UNIX 環境でのプリプロセス」の「COBOL 言語の場合」を参照してください。

(3) ポストソースのコンパイルおよびリンクエージを実行する

ccbl2002 コマンドで、(1)と(2)で作成したポストソース (uapdml01.cbl, uapsql01.cbl) のコンパイルおよびリンクエージを実行します。

コマンド実行例

```
CBLLIB=/HiRDB/include           ...1
export CBLLIB                   ...2
ccbl2002 -Compati85,All uapsql01.cbl uapdml01.cbl -L/HiRDB/client/lib
    -libzcltk.so -OutputFile uap01    ...3
```

[説明]

1. 環境変数 CBLLIB に、登録集原文の検索先ディレクトリを指定します。下線部分は、HiRDB Structured Data Access Facility/Developer's Kit のインストールディレクトリです。
2. 環境変数 CBLLIB を設定します。
3. ccbl2002 コマンドを実行します。

4

UAP の実行環境の構築

この章では、HiRDB クライアントの環境設定方法、UAP をテストする際の UAP の実行方法、およびテスト環境から本番環境への UAP の移行方法について説明します。

4.1 HiRDB クライアントの環境設定

DML を記述した UAP を実行するには、HiRDB クライアントの環境設定が必要になります。UAP の実行環境（テスト環境または本番環境）を構築する際は、HiRDB クライアントの環境設定をしてください。

4.1.1 HiRDB クライアントのインストール

UAP を実行するマシンに、次のどちらかの HiRDB クライアントをインストールしてください。

- HiRDB Structured Data Access Facility/Run Time
- HiRDB Structured Data Access Facility/Developer's Kit

HiRDB クライアントのインストール方法については、マニュアル「HiRDB UAP 開発ガイド」の「HiRDB クライアントのインストール」を参照してください。

HiRDB クライアントをインストールしたとのディレクトリおよびファイル構成については、マニュアル「HiRDB 構造型データベース機能」の「HiRDB クライアントの環境設定」の「HiRDB クライアントのディレクトリおよびファイル構成」を参照してください。

なお、DNS を利用していない場合は、HiRDB クライアントをインストールしたあとに hosts ファイルを設定する必要があります。hosts ファイルの設定方法については、マニュアル「HiRDB UAP 開発ガイド」の「hosts ファイルの設定」を参照してください。

4.1.2 環境変数の設定

HiRDB クライアントをインストールしたマシンで、次の環境変数を設定してください。

- LANG
ja_JP.SJIS を指定してください。
- LD_LIBRARY_PATH
\$PDDIR/client/lib を追加してください。

4.1.3 クライアント環境定義の設定

HiRDB クライアントをインストールしたマシンで、クライアント環境定義を設定してください。クライアント環境定義の設定方法については、マニュアル「HiRDB UAP 開発ガイド」の「クライアント環境定義（環境変数の設定）」を参照してください。

クライアント環境定義の各オペランドの指定内容については、次のマニュアルを参照してください。

- ・マニュアル「HiRDB UAP 開発ガイド」の「クライアント環境定義の設定内容」
- ・マニュアル「HiRDB 構造型データベース機能」の「クライアント環境定義（環境変数の設定）」の「そのほかのクライアント環境定義」

注意事項

- ・クライアント環境定義の PDLANG には、SJIS（シフト JIS 漢字コード）を指定してください。
- ・クライアント環境定義のオペランドのうち、指定が無効になるオペランドがあります。詳細については、マニュアル「HiRDB 構造型データベース機能」の「クライアント環境定義（環境変数の設定）」の「クライアント環境定義の一覧」を参照してください。

4.2 UAP のテストの実行

UAP の実行環境を構築したら、UAP を実行して UAP のテストを実施してください。

OpenTP1 環境下での UAP の実行方法については、マニュアル「OpenTP1 プログラム作成リファレンス COBOL 言語編」の「アプリケーションプログラムの開始と終了」を参照してください。

COBOL の実行可能ファイルとして実行する UAP の場合は、UAP の実行可能ファイルを直接起動して UAP を実行します。

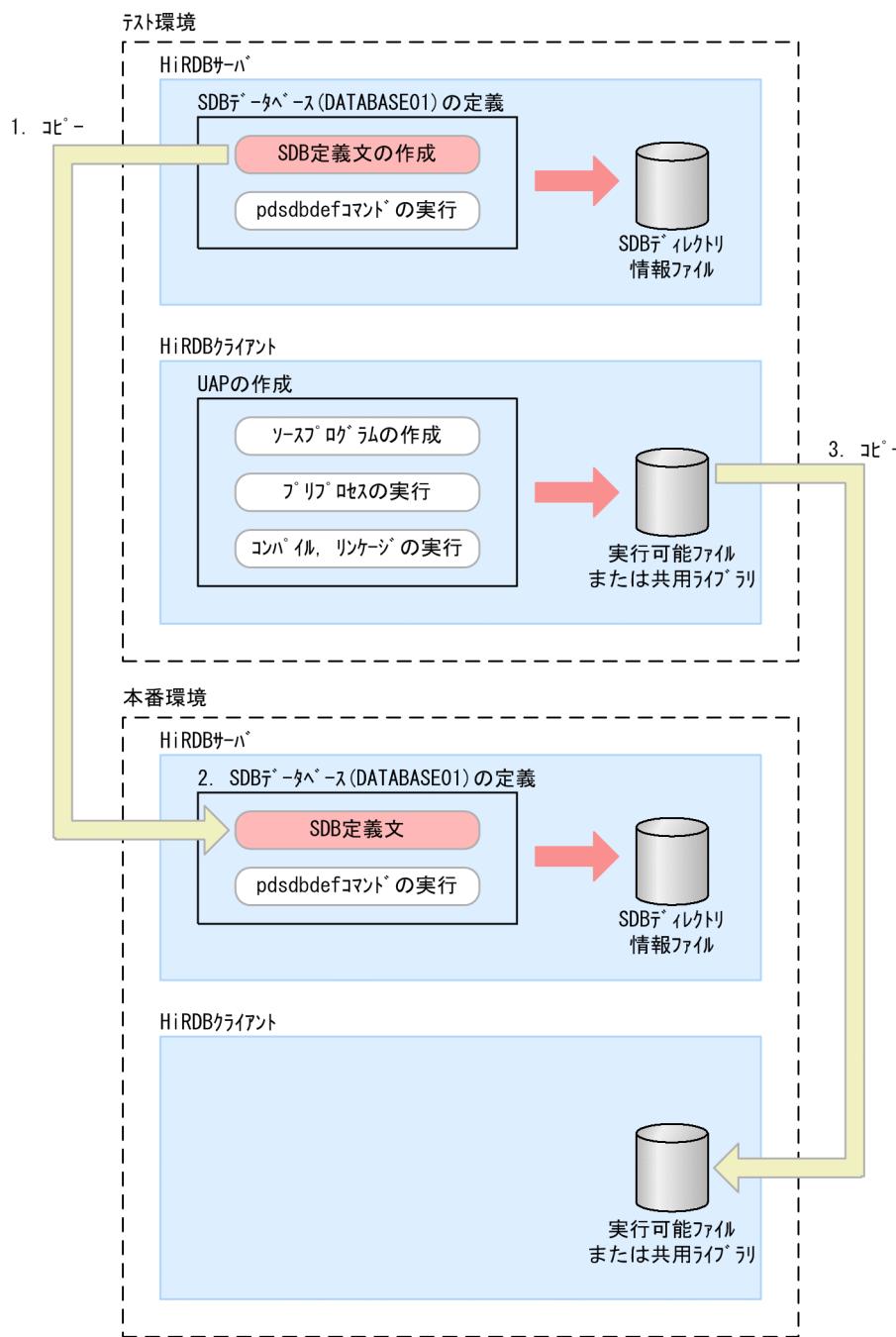
SDB データベースにアクセスする部分の UAP の COBOL ソースプログラムを修正した場合、その UAP ソースファイルの再プリプロセスと再コンパイルが必要になります。また、UAP の再リンクエージも必要になります。

4.3 テスト環境から本番環境への UAP の移行

テストが完了した UAP をテスト環境から本番環境に移行する際の手順を説明します。

例

SDB データベース (DATABASE01) にアクセスする UAP を、テスト環境から本番環境に移行します。DATABASE01 は、新たに定義した SDB データベースとします。



UAP をテスト環境から本番環境に移行する際の手順を次に示します。

手順

1. テスト環境で定義した SDB データベース (DATABASE01) の SDB 定義文を、テスト環境から本番環境にコピーします。
2. 1.でコピーした SDB 定義文を使用して、SDB データベース (DATABASE01) を本番環境で新たに定義します。
3. 実行可能ファイルまたは共用ライブラリをテスト環境から本番環境にコピーします。

5

UAP の運用・保守

この章では、UAP の実行方法、UAP の再プリプロセスが必要なケース、および UAP の障害対策について説明します。

5.1 UAP の実行

OpenTP1 環境下での UAP の実行方法については、マニュアル「OpenTP1 プログラム作成リファレンス COBOL 言語編」の「アプリケーションプログラムの開始と終了」を参照してください。

COBOL の実行可能ファイルとして実行する UAP の場合は、UAP の実行可能ファイルを直接起動して UAP を実行します。

5.2 UAP の再プリプロセス、再コンパイル、再リンクージが必要なケース

次に示す場合は、SDB データベースにアクセスする部分の UAP ソースファイルの再プリプロセスと再コンパイルが必要になります。また、UAP の再リンクージも必要になります。

- **SDB データベースの定義を変更した場合**

SDB データベース節に指定している SDB データベースの定義を変更した場合、その UAP ソースファイルの再プリプロセスと再コンパイルが必要になります。また、UAP の再リンクージも必要になります。

- **DML の記述を変更した場合**

DML の記述を変更した場合、その UAP ソースファイルの再プリプロセスと再コンパイルが必要になります。また、UAP の再リンクージも必要になります。

- **機能追加などによって、SDB データベースにアクセスする部分の UAP を改修した場合**

機能追加などによって、SDB データベースにアクセスする部分の UAP を改修した場合、その UAP ソースファイルの再プリプロセスと再コンパイルが必要になります。また、UAP の再リンクージも必要になります。

- **HiRDB/SD をバージョンアップして、HiRDB/SD の新機能を使用する場合**

HiRDB/SD が提供している API は上位互換性があります。そのため、基本的には HiRDB/SD をバージョンアップしても UAP の修正は必要ありません。ただし、HiRDB/SD の新機能を使用する場合は、UAP ソースファイルの再プリプロセスと再コンパイルが必要になります。また、UAP の再リンクージも必要になります。

5.3 UAP の障害対策

UAP に障害が発生した場合、次に示すトラブルシュート機能を使用して障害要因を調査してください。

- SQL トレース機能
- クライアントエラーログ機能
- 拡張 SQL エラー情報出力機能
- UAP 統計レポート機能

上記のトラブルシュート機能に出力される情報については、マニュアル「HiRDB 構造型データベース機能」の「UAP の障害対策」を参照してください。

■ ポイント

SDB データベースにアクセスする部分の UAP の COBOL ソースプログラムを修正した場合、その UAP ソースファイルの再プリプロセスと再コンパイルが必要になります。また、UAP の再リンクageも必要になります。

6

DML プリプロセサ (pdsdbcbl コマンド)

この章では、DML プリプロセサ (pdsdbcbl コマンド) の機能と使い方について説明します。

6.1 機能

DML プリプロセサを実行して、DML を記述した UAP のプリプロセスを行います。

DML プリプロセサを実行すると、COBOL ソースプログラム中に記述されている DML を COBOL 命令に置換し、その実行結果をポストソースとして出力します。これを UAP のプリプロセスといいます。

なお、この章では、DML プリプロセサを pdsdbcbl コマンドと表記します。

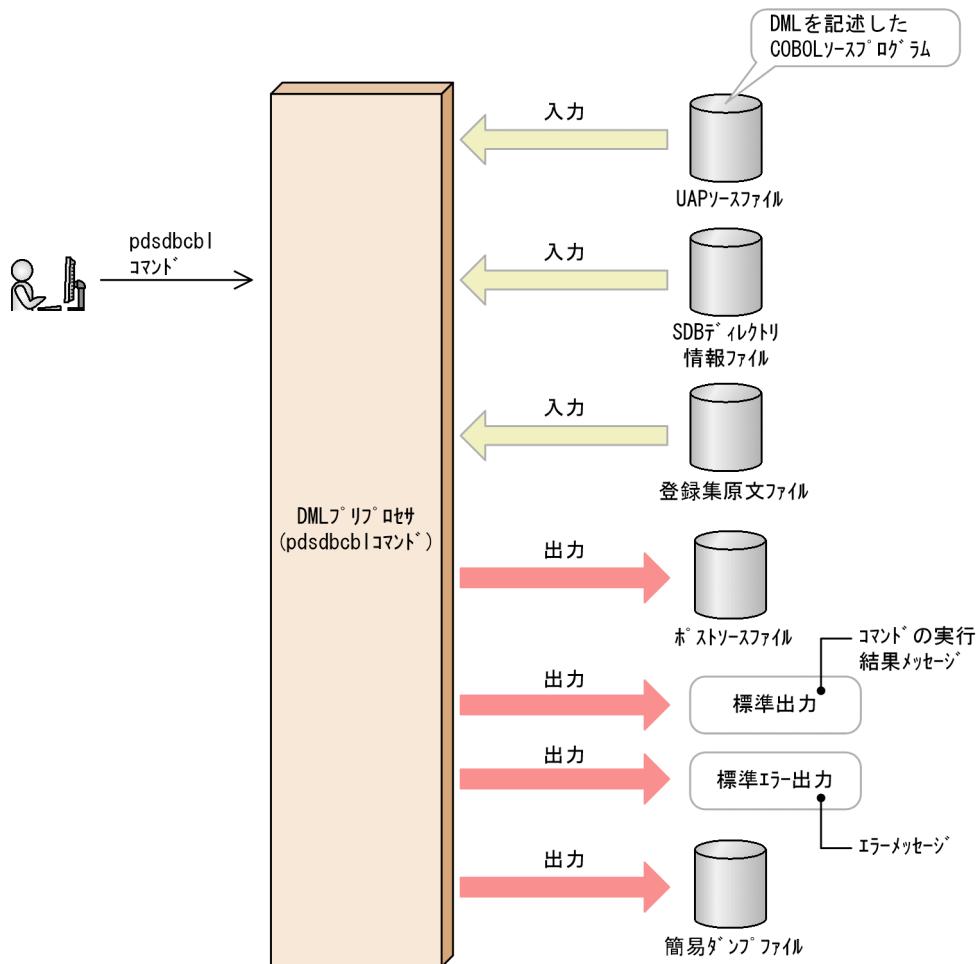
6.1.1 UAP のプリプロセス

DML が記述された COBOL ソースプログラムは、そのままの状態では COBOL コンパイラでコンパイルすることはできません。pdsdbcbl コマンドによって、UAP のプリプロセスを実行してポストソースを出力し、そのポストソースを COBOL コンパイラでコンパイルします。

なお、pdsdbcbl コマンドによるプリプロセスを実行できる UAP は、COBOL 言語で記述された埋込み型 UAP です。

pdsdbcbl コマンドによるプリプロセスの概要を次の図に示します。

図 6-1 pdsdbcbl コマンドによるプリプロセスの概要



[説明]

- SDB ディレクトリ情報ファイル中の SDB データベースの定義情報を参照し、DML の記述内容と SDB データベースの定義情報の整合性が取れているかをチェックします。
- DML を COBOL 命令に置換し、その実行結果（ポストソース）をポストソースファイルに出力します。
- エラーを検出した場合、エラーメッセージが標準エラー出力に出力されます。pdsdbcbl コマンドが異常終了した場合は、簡易ダンプが出力されます。

pdsdbcbl コマンドによるプリプロセス時の入出力ファイルを次に説明します。

入力ファイル

- UAP ソースファイル

DML が記述されている COBOL ソースプログラムを格納しているファイルです。UAP ソースファイルのファイル拡張子は、「.ecb」になります。

- SDB ディレクトリ情報ファイル

pdshdbcbl コマンドによるプリプロセスの際、COBOL ソースプログラム中に記述された DML と SDB データベース定義との整合性をチェックするために、SDB ディレクトリ情報ファイルを参照します。

- 登録集原文ファイル

pdshdbcbl コマンドによるプリプロセスの際、COBOL ソースプログラム中に記述された COPY 文に従って取り込まれる、登録集原文が登録されているファイルです。

注意事項

プリプロセスの実行時に使用する SDB データベースの定義情報と、UAP の実行時に使用する SDB データベースの定義情報には、同じ定義情報を使用してください。

出力ファイル

- ポストソースファイル

pdshdbcbl コマンドによるプリプロセスを実行すると、DML を COBOL 命令に置換したポストソースがポストソースファイルに出力されます。ポストソースファイルの名称は、UAP ソースファイルのファイル拡張子を「.cbl」に変更したファイル名になります。

(例) uap01.ecb → uap01.cbl

なお、プリプロセス時に同じファイル名のポストソースファイルがある場合、そのポストソースファイルが上書きされます。

- 簡易ダンプファイル

pdshdbcbl コマンドが異常終了した場合に、簡易ダンプが出力されるファイルです。簡易ダンプについては、「6.6 トラブルシューティング」を参照してください。

注意事項

プリプロセスで使用する入出力ファイル、およびディレクトリには、読み込み、書き込みができるように、用途ごとに権限を設定してください。

6.1.2 プリプロセス時にチェックされない項目

pdshdbcbl コマンドによるプリプロセス時に、次に示す内容はチェックされません。

- DML の実行順序に起因するエラー

例えば、MODIFY 文でレコード実現値を更新する際、更新対象のレコードに FIND 文などで位置づけていなくても、プリプロセス時にはチェックされません。この場合、UAP の実行時にエラーになります。

- 埋込み変数の内容に起因するエラー

例えば、埋込み変数の内容が、対応するレコード型の構成要素のデータ型のデータ形式に合っていなくても、プリプロセス時にはチェックされません。この場合、UAP の実行時にエラーになります。

6.2 コマンドの形式

UAP のプリプロセスを実行する pdsdbcbl コマンドの形式を次に示します。

形式

```
pdsdbcbl UAPソースファイル名
  -d SDBディレクトリ情報ファイル名
  [-Xb]
```

UAP ソースファイル名：

～<パス名>

プリプロセスを実行する COBOL ソースプログラムを格納している UAP ソースファイルの名称を、絶対パスまたは相対パスで指定します。

指定できる UAP ソースファイルのファイル拡張子は、「.ecb」になります。

指定規則を次に示します。

- UAP ソースファイルのパスの最大長は 1,023 バイトです。
- ファイル名、およびディレクトリ名の最大長は、OS の仕様に従います。
- UAP ソースファイル名を複数回指定した場合、最後に指定した UAP ソースファイル名が有効になります。

-d SDB ディレクトリ情報ファイル名：

～<パス名>

COBOL ソースプログラム中に指定している SDB データベースの定義情報が格納されている SDB ディレクトリ情報ファイルの名称を、絶対パスまたは相対パスで指定します。

注意事項

プリプロセスに使用する SDB ディレクトリ情報ファイルが、pdsdbdef コマンドの dirinf 文に指定しているディレクトリにある場合 (-d オプションに、pdsdbdef コマンドの dirinf 文に指定しているディレクトリと同じディレクトリを指定している場合)、pdsdbcbl コマンドと pdsdbdef コマンドを同時に実行しないでください。

指定規則を次に示します。

- SDB ディレクトリ情報ファイルのファイル名は、「pdsdbdir」にしてください。
- SDB ディレクトリ情報ファイルのパスの最大長は 1,023 バイトです。
- ファイル名、およびディレクトリ名の最大長は、OS の仕様に従います。
- -d オプションを複数回指定した場合、最後に指定した-d オプションの指定が有効になります。

-Xb :

COBOL2002 のコンパイラオプションに-BigEndian,Bin オプション (COBOL85 のコンパイラオプションの場合は-Bb オプション) を指定して、2進項目をビッグエンディアン形式にする UAP を作成する場合に、このオプションを指定してください。

2進項目をビッグエンディアン形式にする UAP を作成する場合の注意事項については、「[2.13 2進項目をビッグエンディアン形式にする UAP を作成する場合の注意事項](#)」を参照してください。

なお、-Xb オプションは複数回指定しても有効になります。

■コマンドの指定規則

- UAP ソースファイル名、およびオプションの指定順序は任意です。
- オプションは大文字、小文字を区別しません。
- pdsdbcbl コマンドに指定できる引数リストの長さの上限は、4,096 バイトです。

6.3 プリプロセス実行前の準備作業

プリプロセスを実行する前に、ここで説明する準備作業をしてください。

6.3.1 環境変数の設定

プリプロセスを実行する前に、次の環境変数を設定してください。

- **PDCLTLANG**

文字コード種別にシフト JIS 漢字を使用します。PDCLTLANG に SJIS を指定してください。

- **PDCBLLIB**

COPY 文を使用する場合、COPY 文に従って COBOL ソースプログラムに取り込む登録集原文を検索するディレクトリを指定します。環境変数 PDCBLLIB については、マニュアル「HiRDB UAP 開発ガイド」の「UNIX 環境でのプリプロセス」の「COBOL 言語の場合」を参照してください。その際、「SQL」は「DML」に読み替えてください。

環境変数 PDCBLLIB に複数のディレクトリを指定する場合、指定できるディレクトリのパス名の合計長は、最大で 4,095 バイトです（区切り文字のコロン（:）を含む）。なお、プリプロセス時に保証するディレクトリ数は、設定した順で最大 10 個となります。

4,095 バイトを超えた場合は、環境変数 PDCBLLIB の指定がない場合と同様に動作します。

ディレクトリのパス名の合計長が 4,095 バイトを超えていている場合、プリプロセスでの最初の COPY 文解析時に、環境変数 PDCBLLIB に設定したディレクトリのパス名の合計長が長過ぎる旨のメッセージが出力されます。

- **PDCLTPATH**

pdsdbcbl コマンドが異常終了した場合に、pdsdbcbl コマンドが出力する簡易ダンプの出力先ディレクトリのパス名を指定します。環境変数 PDCLTPATH については、マニュアル「HiRDB UAP 開発ガイド」の「クライアント環境定義の設定内容」を参照してください。

簡易ダンプについては、「[6.6 トラブルシューティング](#)」を参照してください。

- **PATH**

環境変数 PATH に次のディレクトリを追加してください。

- HiRDB クライアントのサーバマシンでプリプロセスを実行する場合

`/opt/HiRDB/client/utl/`

- HiRDB サーバのサーバマシンでプリプロセスを実行する場合

`$PDDIR/client/utl/`

6.3.2 SDB ディレクトリ情報ファイルの準備

プリプロセス対象の UAP がアクセスする SDB データベースの定義情報が格納されている SDB ディレクトリ情報ファイルを準備してください。pdsdbcbl コマンドの-d オプションに、SDB ディレクトリ情報ファイルのパスを指定します。

■ 注意事項

HiRDB Structured Data Access Facility のバージョン 09-66 以降で出力した SDB ディレクトリ情報ファイルを使用してください。

6.4 注意事項

- DML 中に記述している名前が引用符 ("") で囲まれていない場合、名前中の英小文字は英大文字に変換されます。英大文字に変換された識別子は、英大文字のままメッセージやポストソースに出力されます。例えば、SDB データベース名や、埋込み変数の指定などが該当します。これらの指定を引用符で囲まない場合、英小文字は英大文字に変換されます。
 - COBOL 登録集原文ファイルに次の記述はできません。
 - DML
 - SDB データベース節
 - COBOL 埋込み型 UAP のプリプロセスでは、登録集原文を埋込み変数の宣言に使用できます。登録集原文で埋込み変数を宣言する場合、COBOL ソースプログラムの次の節に COPY 文を記述して、COPY 文で指定した登録集原文のファイルの中に埋込み変数を宣言する命令を記述してください。
 - 作業場所節 (WORKING-STORAGE SECTION)
 - 連絡節 (LINKAGE SECTION)
 - pdsdbcbl コマンドでは、COBOL ソースプログラムの次の節に記述された COPY 文だけを解析します。そのほかの節に書かれた COPY 文は無視されます。
 - 作業場所節 (WORKING-STORAGE SECTION)
 - 連絡節 (LINKAGE SECTION)
- なお、上記の節に記述した COPY 文が、登録集原文内に入れ子の形で記述されている場合、その COPY 文も解析対象となります。COPY 文の入れ子の形については、「[2.10.7 登録集原文の制限](#)」の「[図 2-10 COPY 文の入れ子のレベルの数え方](#)」を参照してください。
- 登録集原文の検索先ディレクトリを、環境変数 PDCBLLIB に指定してください。PDCBLLIB については、「[6.3.1 環境変数の設定](#)」を参照してください。
 - pdsdbcbl コマンドを実行する際、カレントディレクトリは書き込みできる状態にしておいてください。

6.5 リターンコード

pdsdbcbl コマンドのリターンコードの意味および対処を次の表に示します。

表 6-1 pdsdbcbl コマンドのリターンコードの意味および対処

リターンコード	意味	対処
0	pdsdbcbl コマンドが正常終了しました。	なし。
8	pdsdbcbl コマンドの処理が完了しました。ただし、エラーを検知したため、ポストソースの出力を中止しました。	出力されたメッセージの対処に従ってエラー原因を取り除いてください。 エラーの原因を取り除いたあとで、再度プリプロセスを実行してください。
12	処理が続行できないエラーが発生し、pdsdbcbl コマンドが異常終了しました。	

6.6 トラブルシューティング

pdsdbcbl コマンドが異常終了した場合、KFPB65400-E メッセージが出力されます。KFPB65400-E メッセージ中に出力されたアボートコードを参照してエラーの対処をしてください。

また、pdsdbcbl コマンドが異常終了した場合、簡易ダンプが出力されます。

簡易ダンプの出力先

簡易ダンプは環境変数 PDCLTPATH で指定されたディレクトリに出力されます。次の場合はカレントディレクトリに出力されます。

- 環境変数 PDCLTPATH の定義がない場合
- 環境変数 PDCLTPATH で指定されたディレクトリに書き込みができなかった場合

簡易ダンプファイル名

次の形式で出力されます。

```
pdsdbcbl.abort.yyyymmddhhmmss. プロセスID
```

yyyymmddhhmmss : 簡易ダンプの出力日時

プロセス ID : 1~10 バイト

6.7 使用例

例題

DML を記述した UAP (UAP ソースファイル名 : uap01.ecb) をプリプロセスして、ポストソースを作成します。

コマンドの実行例

```
pdsdbcbl /UAPsrc/DMLsrc/uap01.ecb -d /dirinf/pdsdbdir
```

[説明]

/UAPsrc/DMLsrc/uap01.ecb :

プリプロセス対象の COBOL ソースプログラムを格納している UAP ソースファイル名を絶対パスで指定します。

-d /dirinf/pdsdbdir :

SDB ディレクトリ情報ファイル名を指定します。

■pdsdbcbl コマンドの実行結果の出力例

pdsdbcbl コマンドの実行結果の出力例を次に示します。

(例 1) pdsdbcbl コマンドが正常終了してプリプロセスが完了した場合

```
/UAPsrc/DMLsrc/uap01.ecb:  
pdsdbcbl: /UAPsrc/DMLsrc/uap01.ecb, *: KFPB65000-I: DML preprocessing was ended, return code = 0
```

[説明]

- 下線部には、プリプロセスを実行した UAP ソースファイル名が output されます。pdsdbcbl コマンドに指定したパスの形式で出力されます。
- KFPB65000-I メッセージには、pdsdbcbl コマンドのリターンコードが output されます。

(例 2) プリプロセスエラーが発生した場合

```
/UAPsrc/DMLsrc/uap01.ecb:  
pdsdbcbl: /UAPsrc/DMLsrc/uap01.ecb, 23: KFPB65415-E: The specified SDB database name "DATABASE01" was not found in SDB directory information  
pdsdbcbl: /UAPsrc/DMLsrc/uap01.ecb, *: KFPB65000-I: DML preprocessing was ended, return code = 8
```

[説明]

- 下線部には、プリプロセスを実行した UAP ソースファイル名が output されます。pdsdbcbl コマンドに指定したパスの形式で出力されます。
- 色が付いている部分 (例中の 23) には、UAP ソースファイル内のエラーの原因となった行の行番号が output されます。エラーの原因が特定の行に起因しない場合は、行番号にアスタリスク (*) が output されます。
- KFPB65000-I メッセージには、pdsdbcbl コマンドのリターンコードが output されます。

参考

pdsdbcbl コマンドのオプション指定誤りの場合や、コマンドの実行環境によるエラーの場合などは、メッセージ中に UAP ソースファイル名が出力されません。

6.8 pdsdbcbl コマンドが解析する COBOL 命令

pdsdbcbl コマンドが解析する COBOL 命令を次に示します。

見出し部

- 見出し部の見出し

次の構文を見出し部の見出しとして解析します。

- IDENTIFICATION DIVISION.

IDENTIFICATION は ID と省略できます。

プログラム名段落の見出し

次の構文をプログラム名段落の見出しとして解析します。

- PROGRAM-ID.

データ部

- データ部の見出し

次の構文をデータ部の見出しとして解析します。

- DATA DIVISION.

- SDB データベース節

SDB データベース節を解析します。

- 作業場所節の見出し

次の構文を作業場所節の見出しとして解析します。

- WORKING-STORAGE SECTION.

- 連絡節の見出し

次の構文を連絡節の見出しとして解析します。

- LINKAGE SECTION.

手続き部

- 手続き部の見出し

次の構文を手続き部の見出しとして解析します。

- PROCEDURE DIVISION

- DML

DML 先頭子と DML 終了子に囲まれた範囲を DML として解析します。DML に終止符を付加する場合は、DML 終了子に続けて終止符を記述してください。

DML 先頭子および DML 終了子については、マニュアル「HiRDB 構造型データベース機能」の「埋込み言語文法」を参照してください。

プログラム終わり見出し

次の構文をプログラム終わり見出しとして解析します。

- END PROGRAM

索引

記号

-Bb オプション 108
-BigEndian,Bin オプション 108
-d オプション [pdsdbcbl コマンド] 137
-Xb オプション [pdsdbcbl コマンド] 138

C

CBLLIB 117
ccbl2002 コマンドの指定形式 118
COBOL ソースプログラム
記述規則 66
基本構成 24
コーディング例 [DML と SQL の両方を実行する UAP の場合] 90
コーディング例 [OpenTP1 環境下の UAP の場合] 74
作成する際の考慮点 32
COPY 文 27

D

DATA DIVISION 25
DML 終了子 31
DML 先頭子 31
DML と SQL の両方を実行する UAP 89
DML の一覧 13
DML のエラーを検出したときの対処方法 61
DML の記述規則 70
DML の実行結果の判定処理 59
DML のデータ型 40
DML のデータ型と COBOL 言語のデータ記述項の対応 40
DML プリプロセサ 133

E

END PROGRAM 31
END-DML 31
ENVIRONMENT DIVISION 25

EXEC DML 31

H

HiRDB クライアントのインストール 124
HiRDB クライアントの環境設定 124

I

IDENTIFICATION DIVISION 25

L

LANG 124
LD_LIBRARY_PATH 124
LINKAGE SECTION 26

O

OpenTP1 環境下での UAP の実行 16

P

PATH 114, 139
PDCBLLIB 139
PDCLTLANG 114, 139
PDCLTPATH 139
pdsdbcbl コマンド 133
形式 137
準備作業 139
使用例 144
リターンコード 142
PROCEDURE DIVISION 30
PROGRAM-ID. 25

R

RECORD LENGTH 35
RECORD NAME 34

S

SDB データベース節 25
埋込み変数の宣言 35

記述形式 33
記述例 33
SDB データベースの特定規則 72
SDB-DATABASE SECTION 25, 34
SQLCODE の値と意味 60
SQLWARN1～SQLWARNF 61
SQL 連絡領域の構成 61

U

UAP
DML と SQL を実行する UAP 120
運用形態 15
開発環境 14
開発の流れ 17
記述形式 13
記述言語 13
実行環境 15
プリプロセス, コンパイル, リンケージの流れ 112
UAP ソースファイル 135
UAP の運用 129
UAP の実行 130
UAP の障害対策 132
UAP のテスト 126
UAP のプリプロセス 134
UAP の保守 129

W

WORKING-STORAGE SECTION 25

い

インストール [HiRDB クライアント] 124
インターフェース領域 13

う

埋込み型 UAP 13
埋込み変数 38
規則 43
使用例 44
埋込み変数以外のデータ記述項 27

埋込み変数の宣言 38
埋込み変数のデータ記述項 26

か

改行コード 66
簡易ダンプファイル 136
環境部 25
環境変数の設定
pdsdbcbl コマンド 139
UAP の実行前 124
コンパイルおよびリンクエージの実行前 117
プリプロセスの実行前 114

く

クライアント環境定義の設定 [HiRDB クライアント] 124

こ

コーディング例
DML と SQL の両方を実行する UAP の場合 95
OpenTP1 環境下の UAP の場合 79
コンパイルおよびリンクエージ
エラーが発生した場合の対処 119
実行例 119
準備作業 117
コンパイルおよびリンクエージの実行 118

さ

再プリプロセスが必要なケース 131
作業場所節 25

し

自動再接続機能 109
終止符 35
使用できる文字 66

せ

正書法 67

て

データ部 25
テスト環境から本番環境への UAP の移行 127
手続き部 30

と

登録集原文の制限 69
登録集原文ファイル 136
トランザクションオブジェクトファイルの作成 118
トランザクション制御
 DML と SQL の両方を実行する UAP の場合 89
 OpenTP1 環境下 64

な

名前の記述規則 68

は

排他制御 65

ひ

ビッグエンディアン形式
UAP を作成する際の注意事項 108

ふ

複数接続機能 109
プリプロセス
 エラーが発生した場合の対処 116
 実行例 [DML と SQL を実行する UAP の場合]
 120
 実行例 [OpenTP1 環境下の UAP の場合] 114
 準備作業 114
 プリプロセス時にチェックされない項目 136
 プログラム終わり見出し 31
 プログラムの入れ子の上限 68
 プログラム名段落の見出し 25
 ロック転送機能 109

ほ

ポストソースファイル 136

本番環境への移行作業 [UAP の移行作業] 127

翻訳単位 67

ま

マルチスレッド対応 109

み

見出し部 25

も

文字コード 66

り

リターンコード [pdsdbcbl コマンド] 142

れ

レコードの格納 56
レコードの検索 53
レコードの更新 55
レコードの削除 57
連絡節 26